Faculty of Engineering, Science and Computing
Department of Computing

A Software Based Mentor System

Andrew Marriott

This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University of Technology

September 2008

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

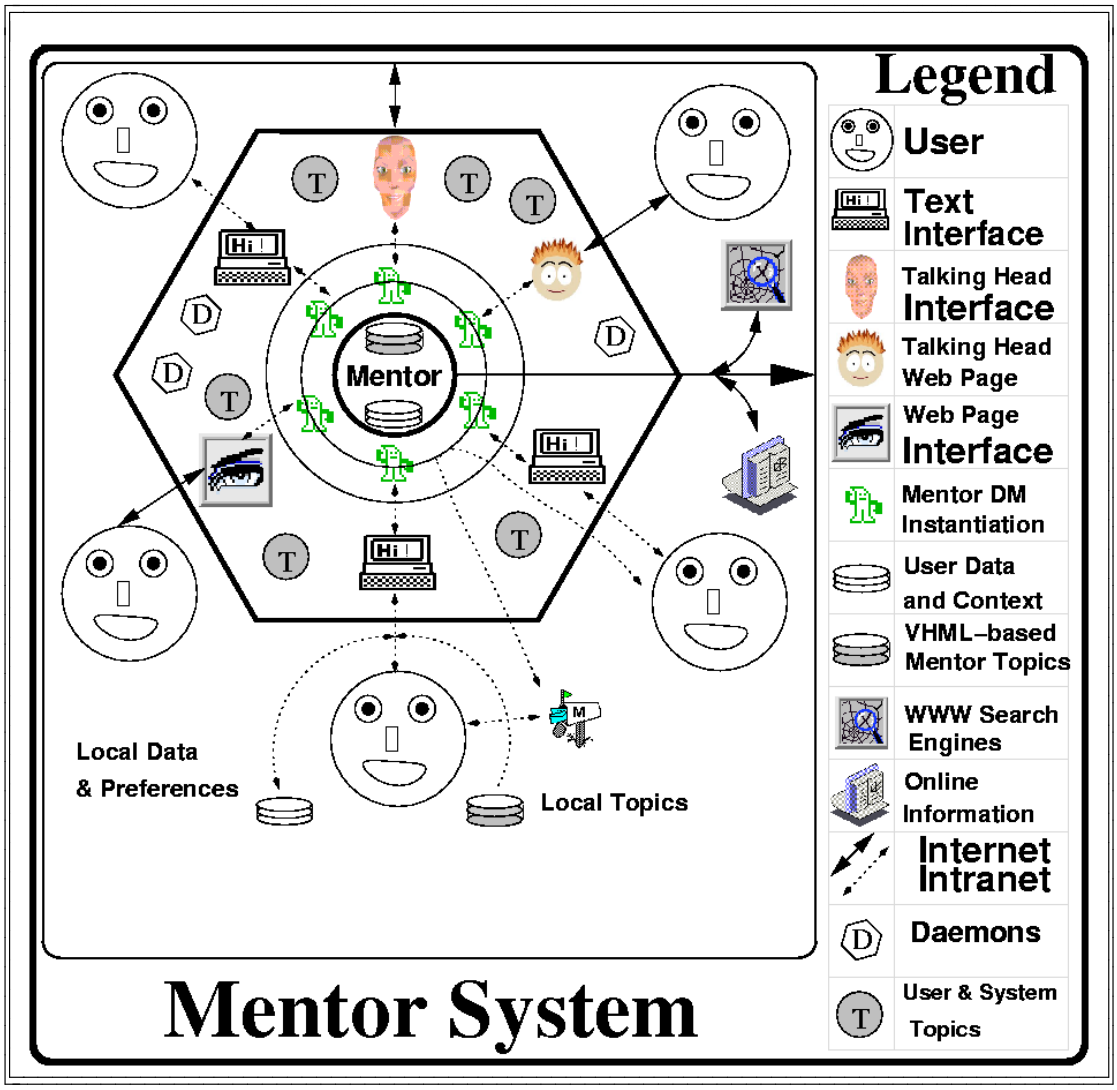
Signature: _____

Date : September 30, 2008

Mentor

Andrew Marriott

Department of Computing
Curtin University of Technology
Western Australia



-- September 30, 2008 --

This thesis has tried to conform to the style specified by Chad Perry in ‘A Structured Approach to Presenting Theses: Notes for Students and Their Supervisors’ (revised 05.09.02) (Perry 1998a).

It has also made extensive use of the guidance afforded by the ‘Guide to the Successful Thesis and Dissertation. A handbook for Students and Faculty’, Third edition by James E. Mauch and Jack W. Birch (Mauch & Birch 1993a).

The style has been modified in consultation with my supervisors to cater for the Science and Education multi-disciplinary nature of the research.

It also conforms to the style specified in the Australian Government Publishing Service publication ‘Style Manual for Authors, Editors and Printers, 5th Edition’ (NOIE 1994a).

It uses the Harvard Referencing System.

The latest copy of this document can always be found at:
http://www.mentor.computing.edu.au/phd/thesis

Conventions used in this thesis.

- A constant width font is used to indicate Java Method, Class and Package names:
  Example:
  `replaceXMLVariables` in class `MentorVariables` which can be found in the Java Package `local.mentor.mentord`.
  The ***Mentor*** Topic `adminTopic` makes use of the *String* class.

- A constant width font is used to indicate program names:
  Example:
  On Unix, we can use `vi`, `sed` or even `grep`.

- A constant width **bold** font is used to indicate XML and VHML tags:
  Example:
  `You <emph>said</emph> to me once`

- A constant width *italic* font is used to indicate XML and VHML attributes:
  Example:
  `and <emph affect="b" level="moderate">mere</emph> beauty`

- A constant width font within code examples is used to indicate user code:
  Example:

```
topicTrace(context,"checkTopic : " + i);
typed = typed.trim();   // get rid of extra white space
if(typed.equals("italian"))  // english spelling
```

- A **bold** font within code examples is used to indicate Java keywords:
  Example:
  public static final *String* `WHOIS_STRING =`

- An *italic* font within code examples is used to indicate Java predefined classes:
  Example:
  Vector `Prev_Query_State;`
  String `state_string = topic_string;`

- User input or ***Mentor*** response uses this font surrounded by normal double quotes - ".
  Example:
  "what problems did you find when ....."

- User quotes from evaluations use this font surrounded by a box
  Example:

  This part is quite impressive Good!

  Quote 7.3 Example: *'How the delayed answers helped'* comment

- Regular Expressions use this `font`.
  Example:

```
who\s*(built|made|constructed|created)\s*you
```

- An inline cited quotation uses an italic font surrounded by “ and ”, and a long quotation is indented and uses an italic font. A term used within the cited work simply has “ and ” surrounding it.
  Example:
  discuss the “*general positive trend*”
  Scriven (1991a) has indicated that “early-warning summative” evaluation ...

Table of Contents

Index of Figures

Index of Tables

Index of Quotes

Abstract

This thesis describes the architecture, implementation issues and evaluation of ***Mentor*** - an educational support system designed to mentor students in their university studies. Students can ask (by typing) natural language questions and ***Mentor*** will use several educational paradigms to present information from its Knowledge Base or from data-mined online Web sites to respond. Typically the questions focus on the student's assignments or in their preparation for their examinations. ***Mentor*** is also pro-active in that it prompts the student with questions such as "Have you started your assignment yet?". If the student responds and enters into a dialogue with ***Mentor***, then, based upon the student's questions and answers, it guides them through a Directed Learning Path planned by the lecturer, specific to that assessment.

The objectives of the research were to determine if such a system could be designed, developed and applied in a large-scale, real-world environment and to determine if the resulting system was beneficial to students using it. The study was significant in that it provided an analysis of the design and implementation of the system as well as a detailed evaluation of its use.

This research integrated the Computer Science disciplines of network communication, natural language parsing, user interface design and software agents, together with pedagogies from the Computer Aided Instruction and Intelligent Tutoring System fields of Education. Collectively, these disciplines provide the foundation for the two main thesis research areas of Dialogue Management and Tutorial Dialogue Systems.

The development and analysis of the ***Mentor*** System required the design and implementation of an easy to use text based interface as well as a hyper- and multi-media graphical user interface, a client-server system, and a dialogue management system based on an extensible kernel. The multi-user Java-based client-server system used Perl-5 Regular Expression pattern matching for Natural Language Parsing along with a state-based Dialogue Manager and a Knowledge Base marked up using the XML-based Virtual Human Markup Language. The kernel was also used in other Dialogue Management applications such as with computer generated Talking Heads. The system also enabled a user to easily program their own knowledge into the Knowledge Base as well as to program new information retrieval or management tasks so that the system could grow with the user.

The overall framework to integrate and manage the above components into a usable system employed suitable educational pedagogies that helped in the student's learning process. The thesis outlines the learning paradigms used in, and summarises the evaluation of, three course-based Case Studies of university students' perception of the system to see how effective and useful it was, and whether students benefited from using it. This thesis will demonstrate that ***Mentor*** met its objectives and was very successful in helping students with their university studies. As one participant indicated: '*I couldn't have done without it.*'

It all begins with the bright light,
the bright light and the noise.
The chaos of the senses
and the scream of desire.
Touching and being touched,
a million loose ends to tie up,
ticking of the clock
and the cradle rock.
The Big Lie
Marillion.

You gotta be crazy,
you gotta have a real need.
Dogs
Pink Floyd

Acknowledgement

Undertaking a PhD is difficult at the best of times. Undertaking a part-time Phd is even more difficult and there are times when the ice becomes very thin...

Supervising a part-time Phd is a thankless task - all the drama and few rewards in terms of the supervisor's own research. Even more so when the PhD student is working along side you in the same institution. I appreciate the patience and guidance that Dr Mike Robey and Professor Geoff West have shown over the far too many years it has taken. I hope they feel it was worth it and not too onerous.

I would also like to thank Professor Bruce Shortland-Jones - my Educational supervisor - for his inciteful/insightful talks and criticisms of my way of thinking and evaluating. An even more thankless task - supervising a computing lecturer of 20 years experience who is researching and developing an educational software system. Thanks Bruce, I needed you.

Thanks to Dr Donald Reid who stepped in when one of my original supervisors withdrew at the last moment. As always, you provided valuable feedback by looking at the research and the thesis with a critical eye.

I would also like to thank my colleagues and students who through their informational email and discussions have made my research time more profitable and enjoyable. Also thanks for all the chockie cakes...

Some people never want to start learning, some people never want the life long journey to stop. I have been lucky in that my mother and sisters started my learning with love and patience and an old blackboard when I was very young. And the road goes ever on and on....

Finally, I would like to thank my beloved, Jocelyn Armarego, for her patience and support over these last 10 years as I have tried to balance family life, work and a never-ending PhD. I would also like to thank my two children Ariel and Tegan for their support and understanding. Their PhD time will also come.

andrew marriott
Genova, Italy.
2003

Crude classifications and false generalizations are the curse of the organized life.
George Bernard Shaw

Abbreviations

AAMAS	Autonomous and Multi-Agent Systems Conference
ACM	Association for Computing Machinery
ACL	Association of Computational Linguistics
AI	Artificial Intelligence
API	Application Programmer Interface
ASCII	American Standard Code for Information Interchange
BAML	Body Animation Markup Language
CBL	Computer Based Learning
CAI	Computer Assisted Instruction
CAL	Computer Assisted Learning
CBE	Computer Based Education
CBT	Computer Based Training
CGI	Common Gateway Interface
CGI-BIN	Common Gateway Interface Binary
CHI	Computer Human Interface
COMODA	Conversational Model for Database Access
CS	Computer Science
DAI	Distributed Artificial Intelligence
DB	Database
DK	Domain Knowledge
DM	Dialogue Manager
DMML	Dialogue Manager Markup Language
DMTL	Dialogue Management Tool Language
DMS	Dialogue Manager Server
DMS	Dialogue Manager System
DOM	Document Object Model
DTD	Document Type Definition
ECA	Embodied Conversational Agent
EML	Emotion Markup Language
FAML	Facial Animation Markup Language
FAQ	Frequently Asked Questions

FRED	Functional Response Emulation Device
FTP	File Transfer Protocol
GML	Gesture Markup Language
GTK	Gimp Tool Kit
GUI	Graphical User Interface
HCI	Human Computer Interface
HMS	HyperMedia Systems
HREF	Hypertext Reference or Link to an URL
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
ICQ	Play on phrase "I seek you" - instant messaging
IF	Information Filtering
IP	Internet Protocol
ILS	Integrated Learning System
IR	Information Retrieval
IST	Information, Society and Technology (European Union Research category)
ITS	Intelligent Tutoring Systems
IUI	Intelligent User Interfaces
JAR	Java ARchive
JDK	Java Development Kit
JIT	Just In Time compiling
JVM	Java Virtual Machine
KB	Knowledge Base
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
LiC	Lecturer-in-Charge
LDAP	Lightweight Directory Accessing Protocol
LSA	Latent Semantic Analysis
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MPEG	Moving Pictures Expert Group
MPEG-4	Moving Pictures Expert Group subgroup 4
MS	Mentor System
MUD	Multi User Domain
NIS	Network Information Service
NLE	Natural Language Engineering
NLP	Natural Language Processing
NLU	Natural Language Understanding
OS	Operating System
OZCHI	Australian Computer Human Interface Conference
PC	Personal Computer
PDA	Personal Digital Assistant
PDF	Portable Document Format

PBD	Programming by Demonstration
RDF	Resource Description Framework
RE	Regular Expression
RFC	Request For Comment
RSS	Really Simple Syndication
SAX	Simple API for XML
SA	Software Agent
SAS	Software Agent System
SADM	Stand Alone Dialogue Manager
SGML	Standard Generalized Markup Language
SML	Speech Markup Language
SMS	Short Messaging System
SPSS	Statistical Package for Social Science
SQL	Server Query Language
SW	Semantic Web
TCP	Transmission Control Protocol
TCP-MUX	Transmission Control Protocol Multiplexer
TH	Talking Head
TIPS	Terminological Interface Prototyping System
URL	Uniform Resource Locator
URN	Uniform Resource Name
URI	Uniform Resource Indicator
VCC	Virtual Conversational Character
VH	Virtual Human
VHML	Virtual Human Markup Language
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformer

An author who speaks about his own books is almost as bad as a mother who talks about her own children.
Benjamin Disraeli

No fathers or mothers think their own children ugly.
Miguel De Cervantes

Preface

The research presented in this thesis has been published in conferences, journals and chapters in books. The research has resulted in the following presentations/publications:

» Marriott, A 2001, 'Yackity yack, blah blah blah blah. Parlez vous? Dialogue Management for Talking Heads', in *Keynote address at OZCHI 2001 conference*, Computer-Human Interaction Special Interest Group's (CHISIG), Perth Australia, 20-22 November 2001

This was a keynote address at the OZCHI 2001 conference and was concerned with the integrating of Talking Heads and Dialogue Management (the ***Mentor*** System). Chapter Three, Section 4.2.2.3 on page 142 indicates the architectural development that enabled this integration.

» Panel Discussion on Running Classroom Evaluations for '*Tutorial Dialogue Systems: With a View Towards the Classroom' Workshop of 11th International Conference on Artificial Intelligence in Education*, AIED2003, Sydney, Australia, 18 July 2003.

Invited to sit on this 4 person panel discussion for this workshop. Each participant addressed two out of four questions on classroom evaluation. Problems associated with the evaluation of the ***Mentor*** System were discussed. The evaluation techniques and analysis described in Chapter Four, page 191 were used as arguments in this discussion.

» Evaluating Embodied Conversational Agents Seminar at *Schloss Dagstuhl - Internationales Begegnungs und Forschungszentrum für Informatik* (International Conference and Research Center for Computer Science) Wadern, Germany, March 14-19, 2004.

Research on Dialogue Management, the ***Mentor*** System, and the Virtual Human Markup Language (VHML) has resulted in the researcher being invited to this prestigious, week long, invitation-only seminar on Evaluating Embodied Conversational Agents (ECA). Researcher was part of work groups that produced discussion documents for the future of ECAs (http://kathrin.dagstuhl.de/04121/).

» Marriott, A, Pockaj, R & Parker, C 2001, 'The Face of E-commerce', in *Internet Commerce and Software Agents: Cases, Technologies and Opportunities*, eds. S. M. Rahman & R. J. Bignall, Idea Group Publishing, New York, 2001, pp. 290-315

The relevant book chapter details the ideas behind and the use of the ***Mentor*** System as a Dialogue Manager in a Talking Head scenario for a Virtual Salesperson. Chapter Three, Section 4.2.4.1 on page 154 indicates the client architecture that enabled this use of the system.

» Marriott, A, Beard, S, Stallo, J & Huynh, Q 2001, 'VHML - Directing a Talking Head', in *Active Media Technology LNCS 2252*, eds. J. Liu, P. C. Yuen, C. Li, J. Ng & T. Ishida, Springer-Verlag, Hong Kong, 2001, pp. 90-100

The relevant book chapter details the way in which VHML - an XML based language - has been developed to increase the functionality, extensibility and believability of a Human Computer Interface Dialogue Management system (the ***Mentor*** System). See Chapter Three, Section 4.2.6 on page 166 for more information on VHML.

» Marriott, A 2002, 'A Facial Animation case study for HCI: the VHML-based Mentor System', in *MPEG-4 Facial Animation - The standard, implementations and applications*, eds. I. Pandzic & R. Forchheimer, John Wiley, London, 2002, pp. 219-239

The relevant book chapter details the use of the ***Mentor*** System as a VHML-based Dialogue Manager in an educational Talking Head scenario. Chapter Three, Section 4.2.7 on page 173 and Section 4.2.6 on page 166 as well as some of the early results from Chapter Four provided material for this book chapter.

» Marriott, A 1999, 'A Lifelong Mentor System', in *Teaching in the Disciplines/ Learning in Context. Proceedings of the 8th Annual Teaching Learning Forum*, eds. K. Martin, N. Stanley & N. Davison, The University of Western Australia, Perth, Australia, 3-4 February 1999. Online at http://lsn.curtin.edu.au/tlf/tlf1999/contents.html.

This paper was presented at the 8th Annual Teaching Learning Forum, February 1999, and describes the initial mentoring system. The background research described in Chapter Two, page 21, as well as analysis of the initial implementation issues described in Chapter Four, page 191 provided material for this paper.

» Marriott, A & Pockaj, R 2001, 'The Virtual Lecturer - Distance Learning via a Talking Head', in *Proceedings of Teaching and Learning Forum*, Curtin University of Technology, Perth, Australia, 7-9 February 2001. Online at http://cea.curtin.edu.au/tlf2001/program.html.

This paper was presented at the Teaching Learning Forum, February 2001, and describes the use of a Dialogue Management system (the ***Mentor*** System) coupled with a Talking Head. This paper describes early results from the system development (see Chapter Three, Section 4.2.3.2 on page 147 and Section 4.2.2.3 on page 142).

» Marriott, A 2003, ‘Mentor System Customisation’, in *AAMAS2003 workshop on Embodied Conversational Characters as Individuals*, AAMAS 2003, Melbourne, Australia, 15 July 2003. Online at http://www.vhml.org/workshops/AAMAS2003/.

This paper details the personalisation facilities of the ***Mentor*** System and discusses some of the issues raised by this personalisation. See Chapter Three, Section 4.2.6 on page 166 and Section 5.3.5.1 on page 242 for more information on these issues.

» Marriott, A & Shortland-Jones, B 2003, ‘The Mentor System’, in *"Tutorial Dialogue Systems: With a View Towards the Classroom". Workshop Proceedings of 11th International Conference on Artificial Intelligence in Education*, eds. Carolyn Penstein Rosé & Vincent Aleven, AIED2003, Sydney, Australia, 18 July 2003, ISBN 1 86487 572 0

This paper details the quantitative evaluation of the 3 case studies where students used the system to help them with their learning. Chapter Four, Section 5.3.9 on page 262 provided material for this paper.

» Marriott, A & Shortland-Jones, B 2004, *Mentor: Really annoying. but quite helpful,* Teaching and Learning Forum, Perth, Western Australia (2004).

This paper details the educational issues that arose from the 3 case studies and implications of these issues. Chapter Four, Section 5.3.9 on page 262 provided material for this paper.

» Marriott, A 2004, ‘Mentor: "lotsa progress in future"’, in *Proceedings of "Transforming Knowledge into Wisdom: Holistic Approaches to Teaching and Learning"*, Higher Education Research and Development Society of Australasia (HERDSA), Miri, Sarawak, July 4-7, 2004

This paper details the implications of the qualitative feedback of the 3 case studies. Chapter Four, Section 5.3.9 on page 262 provided material for this paper.

» Marriott, A 2004, ‘You, by Proxy’, in *Proceedings of "Beyond the Comfort Zone", ASCILITE 2004 conference*, ASCILITE 2004, Perth, Western Australia, December 5-8, 2004

This invited paper discusses the future applications of the ***Mentor*** System and its dialogue management kernel. Chapters Four and Five provided material for this paper.

» Marriott, A, Holic, A & Reid, D 2006, ‘The Multi-modal Mentor System’, in *IUI workshop on Effective Multi-modal Dialogue Interfaces*, IUI2006, Intelligent User Interfaces Conference, Sydney, Australia, 29 January 2006. Online at http://www.iuiconf.org/.

This paper discusses the multi-modal nature of the ***Mentor*** System and how this extensibility allows for non-textual input and output to/from the system Chapter Four provided material for this paper.

The following papers have arisen from the peripheral research into VHML and its use within the ***Mentor*** System.

» Marriott, A 2001b, 'VHML - Virtual Human Markup Language', in *Talking Head Technology workshop, OZCHI 2001 conference*, Computer-Human Interaction Special Interest Group's (CHISIG), Perth Australia, 20-22 November 2001

This paper detailed VHML as a markup language for Dialogue Management (the ***Mentor*** System). See Chapter Three, Section 4.2.6 on page 166 for more information on VHML and its use as a markup language for the Dialogue Manager Knowledge base.

» Marriott, A 2002b, 'VHML', in *Special IST 5th Framework workshop on VHML*, European Union IST, Bruxelles, Belgium, 5-6 February 2002

This workshop was requested by the European Union for a special Information, Society and Technology (IST) meeting on VHML. This workshop led to the special IST meeting on Formal Languages held in Bologna, July 2002. See Chapter Three, Section 4.2.6.2 - VHML: Issues - on page 168 for more information on VHML issues.

» Marriott, A & Stallo, J 2002, 'VHML- Uncertainties and Problems... A discussion', in *AAMAS workshop on Embodied conversational agents - let's specify and evaluate them!*, AAMAS 2002, Bologna, Italy, 16 July 2002. Online at http://www.vhml.org/workshops/AAMAS/.

This paper looks at the evolution of VHML as a markup language for Dialogue Management. See Chapter Three, Section 4.2.6.2 - VHML: Issues - on page 168 and specifically Section 4.2.6.2 on page 171 for more information on the future evolution of VHML.

We are, in a way, moving back to conditions that are similar to the Middle Ages master-apprentice relationship where the apprentice would learn on the job under the supervision of a master craftsman, permanently available at the work-place. However, today, the master is no longer a physically present human being but an electronic system which encapsulates the knowledge of the master and which, to some extent, may embody also some of the skills of analysis and evaluation that a master would utilize to orient an apprentice.
Furthermore, the system acts as a communication network to enable the apprentice to access human support from real master performers when and if required.

Alexander J. Romiszowski
New technologies for Human Resource Development
What works; what makes sense?
TechKnowLogia, September/October, 1999

Introduction

1.1. Introduction

This thesis describes the architecture, implementation issues and evaluation of a mentoring system that is designed to be a learning assistant to help students in their university studies.

This research incorporates the Computing disciplines of network communication, natural language parsing, user interface design and software agents, with pedagogy from the Computer Aided Instruction and Intelligent Tutoring System fields of Education. Collectively, these provide the foundation for the two main thesis research areas of Dialogue Management and Educational Technology.

The development and evaluation of the system required the designing and implementing of an easy to use text based interface as well as a hyper- and multi-media Graphical User Interface, a client-server system, and a dialogue management system based on an extensible kernel. The system also enables any user to program new information retrieval or management tasks so that the system can grow with the user. The overall framework to integrate and manage the above components into a usable system employed a suitable educational pedagogy that aids in the student's learning process.

The aim of the research is to review and incorporate existing methods as well as to develop new methods from the above research fields to produce a system that will benefit students as they progress through their university course. Students typically ask the system for help with their assignments.

The objectives of the research are to determine if such a system can be designed, developed and applied in the university teaching environment and to determine if the resulting system is beneficial to student learning.

The thesis also outlines the learning paradigms that provide the foundation for the study and describes three case studies of university students' perceptions of the system to evaluate how effective and useful it was in assisting students enrolled in computing units.

1.2. Background to the research

Helping students to learn is a difficult and complex task. A lecturer, with a class of one hundred and fifty students, or even a tutorial of only twenty students, has only limited time and resources to help students to learn. Bloom (1984a) indicated that two standard deviations difference in student academic benefit is possible if one-on-one tutoring is used rather than the more normal mass tutoring. P A Cohen *et al* (1982a) also indicated that one-on-one is the most effective tutoring mechanism for assisting student learning. Also, since students may learn in a manner and at a pace that is not compatible with the lecturer's delivery style (Gardner 1993a), they may benefit little from even the best or most enthusiastic of lecturers.

In order to address the specific needs of learners, Shute & Psotka (1996a, p.590) discussed the "*general positive trend*" in Intelligent Tutoring System (ITS) evaluations and that in general "*these systems do accelerate learning with no degradation in final outcome*". Graesser, VanLehn *et al* (2001a, p. 2) indicated that ITSs "*produce significant learning gains beyond classroom environments. They are capable of engaging most students' attention and interest for hours.*" An ITS can, in part, provide the necessary one-on-one tutoring to assist student learning. If the ITS is designed carefully, students who learn in different ways may also be more effectively engaged.

Koda & Maes (1996a) have argued that an anthropomorphic interface to computer applications is often the most engaging. This implies that a natural language interface, where the students speak or type their questions, and receive the answer as text or spoken, may be the most natural and hence the most acceptable and usable. Although speech input is beyond the scope of this thesis, text-based conversations between the user and an ITS are relevant given the context in which the study has been designed.

When exploring the concept of Artificial Intelligence (AI), Turing (1950a) argued that if a judge 'talked' to a hidden contestant and could not decide if it was a human or a computer program simply based upon their conversational skills, then the contestant was 'human' regardless. The computer either imitates a human or is 'intelligent'. This intelligence testing became known as the Turing test, and drives Dialogue Management research on understanding text-based conversations with computers.

Weizenbaum (1976a) has shown that users accepted even a very limited intelligence, natural language text-based interface entitled Eliza. Eliza was renowned more for its illusion of understanding than for its AI techniques; the illusion being achieved by what Weizenbaum termed 'tricks'. Weizenbaum explained that the tricks he employed were to fool a user into assuming that there was more intelligence than was actually present. He maintained that a general computer solution to the problems of natural language processing was impossible, and hence that computers could never understand humans.

However, other researchers such as Mauldin (1994a), and Whalen (1999a), felt that in the context of passing the Turing test,

> *If a program can pass the test in a single domain, then 50 or 100 groups making similar programs for different domains would make a program broad enough that the judge and the program could find common ground for a discussion.* Mauldin (1994a, p. 10)

Mauldin proved through demonstration that to a limited extent, conversational characters or 'Chatterbots' such as Julia could trick humans into thinking that the Chatterbots were also human (Foner 1993a). Julia was designed to use a tree−based model of input/output patterns to transfer the conversation between a number of states. By using states, Mauldin hoped to guide the user through a restricted conversation. The success of Mauldin's Maas-Neotek robots such as Julia belies the simplicity of their natural language pattern matching facilities (Aho *et al* 1986a).

This form of dialogue management has been demonstrated (Mauldin 1994a; Plantec 1999a) to be quite robust at giving correct answers or information in relation to user requests. Hence, it is very suitable for an intelligent user interface, as the Chatterbots seem more humane. This appeared to make them more engaging and acceptable by users.

The marrying of two proven technologies - Intelligent Tutoring Systems with Dialogue Management Systems - now typically called Tutorial Dialogue Systems - therefore appears to provide a strong base upon which to build a mentoring system that will help students learn. This concept will be developed, implemented and evaluated in this research.

There are many different ways for a student to learn, and the philosophy underpinning education has evolved to meet new teaching and learning challenges, and especially how to incorporate the opportunities provided by the technology of ubiquitous cheap computing. Tutorial Dialogue Systems encourage aspects of this such as individual and non-linear learning, apprenticeship and guidance from experts.

One of the most important outcomes of a good educational system is "*the cultivation of individual initiative in students*" (R Schank & Cleary 1994a, node. 43). In this respect, any developed educational system must encourage exploration by the user, provide rich information resources and also a guiding hand to assist learning. The guiding hand can be either from peers or from an expert.

Vygotsky (1978a, p. 86) defined his Zone of Proximal Development theory as:

> *the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers*

This concept, together with his Social Development Theory, proposed that people learn best through imitative and instructed learning. That is, he believes that people can learn new knowledge under guidance that could not be learnt by themselves. Similarly, the Social Learning Theory of Bandura (1977a) indicated that observing the behaviour of others may help them re-use that behaviour in their own learning.

Lave & Wenger (1991a) agreed on the need for collaboration and have also indicated that this learning is most beneficial if done in the context of the problem (Situated Learning). Finally, J S Brown *et al* (1989a, p. 39) used Situated Learning and stated:

> *Cognitive apprenticeship supports learning in a domain by enabling students to acquire, develop, and use cognitive tools in authentic domain activity.*

J S Brown *et al* (1989a, p. 39) also indicated that this apprenticeship coupled with craft apprenticeship - in essence mentoring - can produce very capable graduates:

> *Similarly, craft apprenticeship enables apprentices to acquire and develop the tools and skills of their craft through authentic work at and membership in their trade. Through this process, apprentices enter the culture of practice.*

Therefore, situated learning, constructed in a step by step sequence, and guided by expert knowledge would seem to be a useful paradigm to overlay on a Tutorial Dialogue System. It is therefore proposed that a mentoring system should incorporate this paradigm.

The emergence of the Java Programming language (Gosling & McGilton 1995a) signalled the opportunity for building and evaluating a medium-scale Object Oriented mentoring system. In 1996, in the initial phase of this mentoring research, Java code execution was slow and the language was in its infancy. However, benchmarks indicated that it would be usable for non-trivial applications (Neffenger 1997a). Given the active support by Sun Microsystems, it was assumed that the language and its functionality would evolve over time to provide a solid programming and development base for software based mentoring research.

It is therefore proposed that the research areas outlined above can be integrated to develop, implement and evaluate a mentoring system that will be effective and benefit students in their studies.

1.3. Research problem and hypotheses

The design and development of a mentoring system, posed three main research questions:

1. Could the system be developed in Java as a real-world tool?
2. Could it be effective in helping students learn in their units?
3. Would students perceive that it was beneficial to use the system?

From these questions (terms defined on Section 1.7 on page 17), and through analysis of the research literature presented in Chapter Three, the following hypothesis emerged:

A software-based *Mentor* System system can benefit University students.

More specifically, the research proposed to address the three sub-hypotheses:

Sub-hypothesis 1

A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.

Sub-hypothesis 2

The ***Mentor*** System can support an effective learning paradigm.

Sub-hypothesis 3

The ***Mentor*** System will benefit students.

1.4. Justification for the research

Current research in Educational Technology has, with a few notable exceptions (Circsim [HREF 1], Atlas [HREF 2]), produced small scale systems applied in very constrained environments. There was a need to develop an integrated real-world system which exploited the advantages of cooperating network-based programs and apply the system to the area of tertiary teaching and learning support. It is proposed that the design, development and implementation of the system in a non-trivial environment will make a significant contribution to the area of Tutorial Dialogue Systems.

There are a number of reasons why this research is important:

- Bloom (1984a) has indicated that two standard deviations difference in student academic benefit is possible when comparing one-on-one tutoring vs. one-on-many. Fletcher (2003a) indicated that this could represent a typical student achieving results that increased from the mid-level 50th percentile to the 98th!
  Question: Why not use one-on-one tutoring? Fletcher (2003a) also indicated that the most obvious reason is the cost of providing this type of support.
- Universities are being asked to develop open/flexible ways to help students learn. Also, the changing nature of student populations at Universities means that these may be either part time students, distance education students, overseas students or disadvantaged students. The tertiary education system is increasingly being seen as a global market with entire courses being marketed on the World Wide Web. Since the globe has no fixed semester time frame, this may put extra strain on academics by removing the ‘end of semester’ time buffer traditionally used for unit preparation and/or research.
  Question: How can flexible delivery of student learning be realistically supported by academic staff?
  What other ways exist that can help students to learn?
  What other bodies of expert knowledge exist that students can use?
- There is a perceived need to increase the number of students graduating from the University system and there are concerns about the quality of the student learning experiences. What is needed is a support infra-structure that will ensure students have the skills needed to be successful graduates;
  Question: How is the learning support infra-structure developed and maintained?
  How can the standards be ensured?
  How can staff find the time to maintain their commitment to quality teaching?
  How can students learn professionalism and increase their knowledge and skills if contact with academic staff is reduced?

A software based mentoring system may provide solutions to address these questions.

1.5. Methodology

Chapter Three provides a description of the two primary methodologies that this research incorporates: Action Research (Kemmis & McTaggart 1988a) (specifically Educational Action Research) and the Evaluation Methodology (Mauch & Birch 1993a).

The Educational Action Research and Evaluation Methodologies promote the application of a development, participation, evaluation and reflection research cycle. This cycle provides both formative feedback - information from early cycles to improve the system design and effectiveness - and summative feedback - the resulting system can be evaluated in the light of increased understanding of the problem domain. Both these processes will help to improve the development of the proposed mentoring system.

The summative evaluation was to be conducted through questionnaires provided at the end of each cycle and aimed to determine the students' attitudes towards the mentoring system. Given the cyclic nature of the design and development process being implemented through this research, these questionnaires also provided formative evaluation information used in the next cycle. The 6-page questionnaires had fixed-alternative 5 point Likert scale questions as well as open-ended questions to provide qualitative feedback.

Each phase of the research was conducted as a case study to provide information to address the research questions. The case study student evaluations were entered into the software program SPSS (Statistical Package for the Social Sciences) for display purposes, and subsequent analysis of the results.

1.6. Outline of the thesis

Chapter One provides a brief introduction to the research focus, outlining the background to the research, the research problem and hypotheses, the research and design methodology used, as well as the limitations of the research.

Chapter Two describes the research issues of the thesis, providing the parent theories and also the background to the theoretical framework that provides the foundation to explore the research questions.

Chapter Three describes the appropriate research and design methodologies undertaken to conduct the research and to explore the hypotheses developed from the research issues outlined in Chapter Two.

Chapter Four describes the system's architecture and the implementation issues that arose during its design, development and implementation using the Action Research methodology outlined in Chapter Three. These issues are concerned with the implementation of the system as both a client-server system, as well as a stand-alone application.

Chapter Five analyses the data collected in the three Case Studies using the Evaluation methodology outlined in Chapter Three and demonstrates how the data supports the hypotheses of this research.

Chapter Six outlines the conclusions of the research as well as explaining implications of the results for further research and how this study may benefit other researchers.

Chapters Seven and Eight list the reference and bibliographic material included in this research and, where possible, provide online accessing information.

Chapter Nine is a Colophon, and the Appendices contain additional information such as the questionnaires and their evaluation data, the documentation of the various Java packages, and data formats for the various system files.

1.7. Definitions

> *Definitions adopted by researchers are often not uniform, so key and controversial terms are defined to establish positions taken in the PhD research.* Perry (1998a, p. 18)

The following section defines the key concepts incorporated within this research and the position that the researcher, and hence the system, has adopted in using these concepts.

Mentoring

From the online Macquarie Concise Dictionary (www.macquariedictionary.com.au [HREF 3]), a mentor is defined as:

/'mɛntɔ/ noun

1. a wise and trusted counsellor.
2. (especially in an organisation) a person who is considered to have sufficient experience or expertise to be able to assist others less experienced. [from Mentor, friend of Odysseus and guardian of his household when he went to Troy]

The ***Mentor*** System follows the second definition quite closely, however it could be argued that it is simply a variation on an Intelligent Tutoring System. Although it sometimes bears a resemblance to a traditional computing discipline ‘Duty Programmer’ whose job is to help users by solving their programming problems, the system is not just a problem solver, nor does it have the analytical prowess to solve technical computing problems. It does try to mentor students - to assist others less experienced. In this regard it uses the knowledge of others (Vygotsky 1978a).

Software Agents (SA)

Wooldridge & Jennings (1995a) define the characteristics required for SAs (Table 1.1). The ***Mentor*** System has some of the mentioned characteristics (see ‘Research Issues’ chapter, page 21), however software agency will not be its main aim. The SA component of the system will act on behalf of the user by data-mining, by filtering and/or reducing the complexity of information, and by adapting the presentation of the information to the user’s needs. In this regard, it conforms to the somewhat less formal definition of other researchers such as Sycara *et al* (1996a) and Cheong (1996a).

Weak	Strong
autonomy	adaptability
social ability	mobility
reactivity	veracity
pro-activity	rationality

Table 1.1 : The weak and strong characteristics required by an agent

Intelligent Tutoring System

A systematic instructional approach based on decomposing the content into knowledge units. http://www.sisostds.org/webletter/siso/Iss_85/art_480.htm

Intelligent Tutoring Systems - sometimes referred to as Adaptive Tutoring Systems (Streitz 1988a), or Intelligent Computer Aided Instruction systems (Sleeman & Brown 1982a), and more recently subsumed under the term Tutorial Dialogue Systems (Rosé & Aleven 2002a), - extend Computer Assisted Instruction systems (Lawler & Yazdani 1987a), and Computer Based Instruction systems (Psotka *et al* 1988a) by providing a more dynamic learning environment with responses that are not just pre-programmed steps. They are typically characterised by providing a student model, an educational paradigm to drive the learning, and a knowledge base that holds learning material.

A more detailed definition and review of ITS can be found in R Schank & Edelson (1990a), McArthur *et al* (1993a), and Shute & Psotka (1996a) .

The ***Mentor*** System does not have any explicit student model but does utilise domain knowledge and a framework upon which various learning paradigms may be built. In this regard it is similar to an Intelligent Educational System (Goodyear 1991a), or a Knowledge-based Tutoring System (Streitz 1988a).

Tutorial Dialogue Systems

A Tutorial Dialogue System is very similar to an ITS but applied to the tutoring aspect of teaching and learning. See collected works in Rosé & Aleven (2002a).

Dialogue Management System

A Dialogue Management System (DMS) is responsible for understanding what the user wants when they interact with the system. For a Graphical User Interface - user input is supplied via fixed forms - the DMS is implicit in the programming. For a Natural Language Interface - the user types or speaks English sentences - the DMS must parse and understand the English.

The ***Mentor*** System has a simple yet effective DMS that uses Perl-5 Regular expressions to match user input to meaning. It does not use Latent Semantic Analysis (Landauer *et al* 1998a), nor any other technique to segment the input stream into object, subject, predicate, etc.

Churcher *et al* (1997a) indicates the typical characteristics of a human-human dialogue and expects similar for a human-computer dialogue:

- **overall structure**: a dialogue has an opening, a body and a closing.
- **mixed initiative**: both the user and the system can take control of the dialogue.
- **over-informativeness**: the system may offer information that has not been asked for if it is considered helpful.
- **contextual interpretation**: users' sentences have to be interpreted according to the dialogue context.
- **error recovery**: the system must adopt a recovery strategy which allows the conversation to continue if an understanding error occurs.

The ***Mentor*** System allows for **mixed initiative** dialogues, will **over-inform** the user if necessary, has limited **contextual interpretation** (it knows the context of a dialogue and understands anaphora), and has several **error recovery** strategies to maintain the dialogue. To this limited extent, it allows for proper **closure** of dialogues.

Directed Learning Path Educational Paradigm

The Directed Learning Path is essentially a step-by-step guide with checks. It models the way in which an expert would tackle the problem and uses the expert's knowledge to cater for possible problems that may affect inexperienced users.

Users may enter this path either by asking, or the system may pro-actively ask the user if they want help. Traversing the path may be as simple as answering "yes" or "no" to various questions, or as complex as the system asking the user at some stage, questions such as : "what problems did you find when ....." and then choosing a different path according to their answer.

In this manner, the ***Mentor*** System helps a user to reconstruct the knowledge and experience of the expert at their own pace - a Constructivist way of learning (Murphy 1997a). Also, by putting the learning into the scope of the problem, the ***Mentor*** System uses a Situated Learning approach (J S Brown *et al* 1989a).

Authoritative Guidance Educational Paradigm

Authoritative Guidance is based on the principle that users may want to learn in their own way, but ultimately require expertise and guidance in that area.

The ***Mentor*** System attempts to immediately answer most user questions. Those that it cannot are forwarded to the Lecturer-in-charge of the unit, who can create a relevant answer and have that returned to the user via ***Mentor***. The next time the user logs onto the system, or when they have a free moment, ***Mentor*** asks if they would like an authoritative answer to their question.

Real world

In the context of this research, 'real world' means that the system will work with an entire class of students (typically 30-120 students) in their University environment helping them with their actual daily learning problems. It will not be limited to small controlled environments, working on a subset of typical problems that a student is likely to meet.

1.8. Delimitations of scope and key assumptions

Chapter Three will detail the scope, limitations, delimitations and assumptions underpinning this research. In summary, it is assumed that

- a significant number of University Computing students (typically > 30) will use the system so that the data from the case study evaluations will have some statistical significance,

- the study cannot use a control group nor use students' academic records for pre-test/post-test analysis,
- the system will not recognise speech input,
- the system will be similar to an ITS but will not have a student model,
- the system is not an Expert System in the Computer Science meaning of the word.

Of importance however, the system will be practical. That is, it will work in a real world environment with students using it everyday as part of their university life.

Similarly, the software support for the production of Domain Knowledge was developed to cater for practising non-computing lecturers. That is, those who have limited time resources to help their students and have no major interest in the system mechanics, only in helping students learn about their particular units.

Finally, although the environment for the development and evaluation of the implemented system is an academic one, the Dialogue Management kernel of the system has been used in other industrial applications in European Union 5th Framework research projects (see http://www.interface.computing.edu.au/ [HREF 4]).

1.9. Conclusion

This chapter provided an overview of the focus and theoretical framework for this thesis in terms of the parent theories of Software Agents, Intelligent Tutoring Systems, Dialogue Management, and current educational paradigms. It introduced the research problems, issues and hypotheses that directed the research needed to produce the ***Mentor*** System. It provided a justification for the research, and defined the areas addressed. The Action Research and Evaluation methodologies were briefly described and justified. The scope of the research was described and an overall outline of the thesis provided.

Never express yourself more clearly than you are able to think.
Niels Bohr

Research Issues

2.1. Introduction

This chapter provides the theoretical foundation upon which the research is based by identifying unexplored or novel research issues from both the "*parent theory*" and the "*research problem theory*" (Perry 1998a, pg. 20). The relationships between parent theories and research problem theories as well as the derived hypotheses can be seen in Figure 2.1 (adapted from Perry (1998a, pg. 22)).

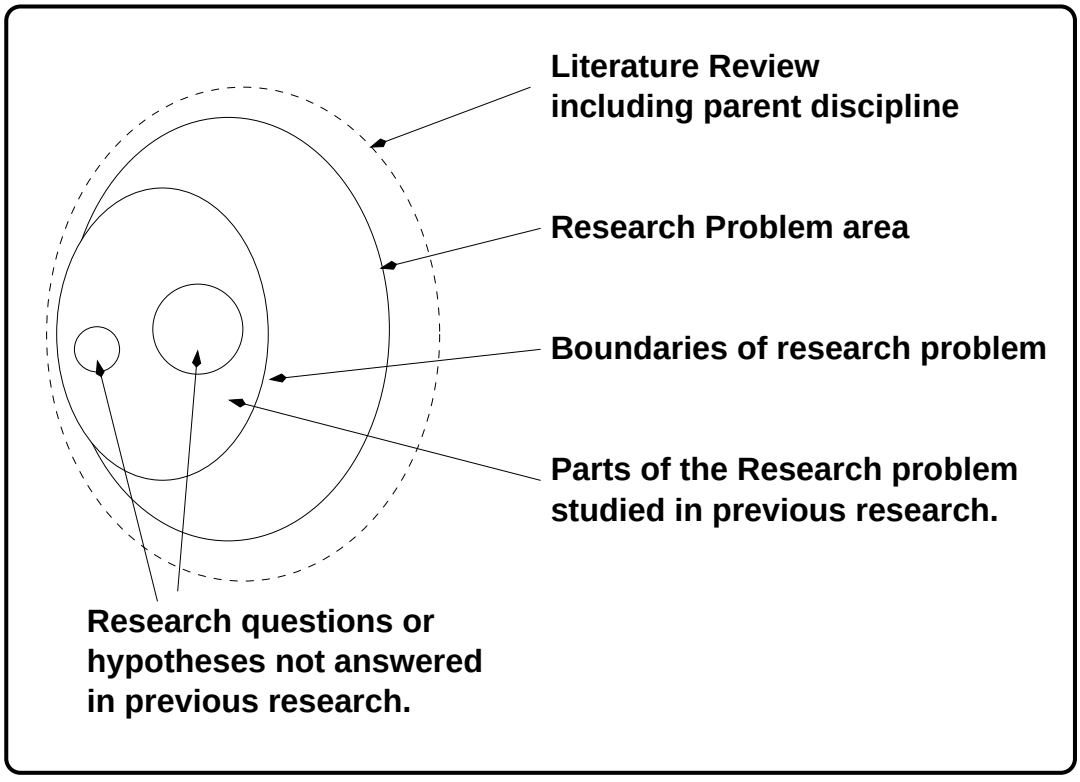


Figure 2.1 : Relationships between parent and research problem theories, and between the research problem and the research issues or propositions (Perry (1998a, pg. 22)).

The following sections will briefly review the parent theories from the two main parent discipline areas: Software Agents / Intelligent User Interfaces and Educational Technology. This review will then be used as the basis for detailed discussion about the research problem areas of Dialogue Management and Tutorial Dialogue Systems (Section 2.3 on page 48).

2.2. Parent Theories

2.2.1. Software Agents

2.2.1.1. Introduction

Central to any developed mentoring system is information: information storage, retrieval, processing, display, and maintenance. To produce relevant answers to user queries, a system must obtain information from its knowledge base as well as from dynamic sources through intelligent searching, coupled with information filtering. Obtaining relevant information is a complex task and many of the information retrieval and processing issues have been addressed by research in the broad field of intelligent software agents (see, for example, Menezes (1999a), Rabkin & Tingley (1999a)).

Instead of a user trawling through large Frequently Asked Question (FAQ) lists, browsing endless Web pages, or traversing hierarchies of information, Software Agent Systems (SAS) can data-mine the sources on the user's behalf so as to answer user queries. This information retrieval can be a simple Web spider or robot that obtains Web pages, or it can combine the spider with information filtering and indexing software so as to increase retrieval and relevance rates for user queries.

A good introduction to and overview of agents can be found in the Communications of the ACM (CACM 1994a) as well as through Mike Wooldridge's website [HREF 5]. What actually constitutes an 'agent' or an 'agent system' is often debated with the agents mailing list [HREF 6] having had many discussions regarding what agents are, and the now defunct 'FAQ of the agents mailing list' indicated that it "*will not attempt to provide an authoritative definition...*". Similarly, Wooldridge & Jennings (1995a, p. 4) indicated the vagary of definition of agency with the following:

> *Carl Hewitt recently remarked that the question* **what is an agent?** *is embarrassing for the agent-based computing community in just the same way that the question* **what is intelligence?** *is embarrassing for the mainstream AI community. The problem is that although the term is widely used, by many people working in closely related areas, it defies attempts to produce a single universally accepted definition.*

Cheong (1996a) is a good general referenced text covering agents and many related aspects of the Web such as Web spiders or robots. The collection '*Designing Autonomous Agents*' by Maes (1990a) gave little practical information on designing agents but is a standard work in the field. The University of Maryland-Baltimore County maintains a sporadic AgentNews Webletter [HREF 7] that contains varied material about agents as well as links to other sources of agent information. Nwana (1996a) also gave a detailed overview of Software agents. However, Foner (1993a) gave a refreshingly insightful sociological study of natural language human-agent interaction in a Multi-user dungeon (see the later section on the Chatterbot Julia on page 52).

SAS Research emerged from the Distributed Artificial Intelligence (DAI) field in the late 1980s, and is now well established with prominent groups at :

- MIT Media Laboratory [HREF 8] under the direction of Pattie Maes,
- Carnegie Mellon [HREF 9] under Joseph Bates,
- the Agent ART Group [HREF 10] under Michael Wooldridge,
- the Intelligence, Agents, Multimedia Group at http://www.iam.ecs.soton.ac.uk/ [HREF 11] under Nick Jennings, and the
- University of Maryland-Baltimore County with the Knowledge Interchange Format (KIF) [HREF 12] and Knowledge Query and Manipulation Language (KQML) [HREF 13] groups.

Early definitions of agents could be found in Brustoloni (1991a), and Etzioni & Weld (1994a), with Wooldridge (1992a) and Goodwin (1993a) using formal specifications to try to categorise agents. Hermans (1996a) presented a detailed introduction to agents and provided a number of definitions of an agent including a "weak" and a "strong" concept (paraphrased from Wooldridge & Jennings (1995a, p. 4-5)), as well as what the user assumes of an agent.

The "weak" notion of agency (Wooldridge & Jennings 1995a, p. 4-5) was "*uncontentious*" and assumed:

> • *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state*
> • *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language*
> • *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
> • *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.*

In this respect, they argued that a "weak" agent was like a Unix software process that exhibited the above properties. Their "strong" definition was "*potentially more contentious*":

> *a computer system that, in addition to having the properties identified above, is either conceptualised or implemented using concepts that are more usually applied to humans.*
>
> Wooldridge & Jennings (1995a, p. 5)

They ascribed AI or "mentalistic" characteristics such as knowledge, belief, intention, obligation and emotion to "strong" agents.

'*Intelligent Agents: Theory and Practice*' by Wooldridge & Jennings (1995a) and '*Intelligent Agents: Theories, Architectures, and Languages*' by Wooldridge & Jennings (1995b) provided the best summary of agenthood with many subsequent reviews considered less extensive or restricted (Guilfoyle & Warner 1994a; Abdu & Bar-Ner 1996a; Isbister & Layton 1996a). More constrained reviews have been published on multi-agent systems (Kittock 1993a; 1994a); for mobile agents (Harrison *et al* 1995a; Lingnau & Drobnik 1995a); for human-like agents (Sloman 1996a); and for reactive agents (Bonasso *et al* 1997a). Franklin & Graesser (1996a) reported on a taxonomy of agent systems, and this could be seen as the basis for Graesser's later work in Tutorial Dialogue Systems in education (Graesser, VanLehn *et al* 2001a).

Inevitably, there was much discussion and dissension in the literature and electronic forums about agents (Etzioni 1993a), about their functionality, their communication languages, their mobility, and their effectiveness in helping users.

2.2.1.2. Communicating Agents

Genesereth & Ketchpel (1994a) proposed that the *lingua franca* for inter-agent communication should be KIF [HREF 12] and KQML [HREF 13]. Etzioni & Weld (1994b) detailed Softbots - communicating software robots - and proposed rules for future agent functionality. McKay *et al* (1996a) investigated information agents based upon KIF and KQML, whilst the research of Evans (1994a), and Muller (1996a), proposed a place for communicating agents in big business and systems management.

Shoham (1991a) and Cheong (1992a) showed early agent language research although both are arguably in the DAI domain rather than agents *per se*. A novel AI vision-based approach to inter-agent language acquisition, that mimics the use of 'dance' in honey bees to impart information to the hive, was detailed in Moukas & Hayes (1996a).

The initial agent language research formed a solid base for developing 'learning/communicating' agents. Early research (Jennings 1992a; Jennings 1992b) formed the basis of a system of cooperating agents (Jennings, Varga *et al* 1993a) that needed communication skills, and the ability to learn from their experiences as well as their neighbours.

The Media Lab applied their agent expertise in broad new areas such as was detailed in Maes & Kozierok (1993a), Metral (1993a), Sheth (1994a), Foner (1994a) and later, Foner (1995a). With agent technology becoming known if not mature, research such as that reported in Cengeloglu *et al* (1994a), Huffman (1994a), Huffman & Laird (1994a), and Maulsby (1994a) demonstrated the power of 'learning' agents.

Sub-areas of interest were developing such as advice taking (Maclin & Shavlik 1994a), competitive agents (Asada *et al* 1995a), conflict resolution (Grosof 1995a), ethics (Eichmann 1995a), artificial life (Menczer *et al* 1995a) and, of importance to this study, using agents for helping humans (Chan & Baskin 1990a; Chan & Chou 1995a; M J Jacobson & Levin 1995a; Holt *et al* 1995a; Loyall & Bates 1995a; Norrie & Gaines 1995a; O'Connor 1995a).

Chan & Baskin (1990a) proposed a Learning Companion System as an alternative to an Intelligent Tutoring System (ITS), with "*two co-existing agents: a teacher and a learning companion*". Chan & Chou (1995a, pg. 1) detailed their research on Reciprocal Tutoring Systems where they had developed a learning companion who was an "*artificial student who interacts with the student and learns under the guidance of the computer teacher*".

The maturing KIF-KQML community recognised the need for ontologies - the semantics of the agent conversation content (Oates *et al* 1994a; Takeda *et al* 1995a; Kautz, Milewski & Selman 1995a). Jennings & Wooldridge (1995a) again showed the quality of their research with their paper on applying agent technology. The Cyc project (see review in Section 2.3.1.3 on page 63), reported by Guha & Lenat (1994a) , was moved into the agent area by Mayfield, Finin *et al* (1995a) using KQML as the inter-agent communication language.

If Berners-Lee's "Semantic Web" (Berners-Lee & Fischetti 2000a) becomes the predominant information structure on the Web, then this diverse agent language research, especially that dealing with ontologies, may coalesce due to the need for a single target architecture. The distributed data in this Semantic Web will then be easily parsed and mined by mobile agents.

2.2.1.3. Distributed and Mobile Agents

Distributed and mobile agent architectures were reported in the early research by Ferguson (1992a) and later by Knabe (1995a). This then led to discussion about 'mobile agents' vs. 'stay at home agents' (Chess *et al* 1995a; Harrison *et al* 1995a), and even the need for operating system support for mobile agents (Johansen *et al* 1994a), and for agent protocols using the Internet as a transport medium (Lingnau, Drobnik & Dömel 1995a; Lingnau & Drobnik 1996a) .

Arguably the most important research in language support for mobile agents was the specification of the Knowledge Query and Manipulation Language (KQML), and the Knowledge Interchange Format (KIF) (Ginsberg 1991a; Genesereth & Fikes 1992a; DARPA 1993a), with subsequent work developing KIF and KQML (Finin, Fritzson *et al* 1994a; Finin & Fritzson 1994a; Mayfield, Labrou & Finin 1996a; Mayfield, Labrou & Finin 1995a).

This mobile agent language research formed the basis for the Information Agent architecture described in McKay *et al* (1996a), and the need for agent naming conventions as described in Finin, Potluri *et al* (1995a). Mobile agent technology for information manipulation was adopted by industry, and General Magic Inc. developed Telescript (White 1996a), although it was slow in being accepted and is no longer supported.

Wreggit (1995a), Gray (1995a) and Chang & Lange (1996a) investigated modern programming languages to try to solve the problems of the agent paradigm. Lange & Chang (1996a) investigated the use of Java for mobile agents. Distributed and Mobile agents are not directly relevant to this thesis, but, since the Java programming language (Gosling & McGilton 1995a) is interpreted, and this allows user code to be easily incorporated into a running server system dynamically, research into distributed agents may have already addressed issues that could arise from this one aspect of developing a software-based mentoring system.

Therefore Aglets [HREF 14], mobile agent systems built using Java, may become more important to the distributed agent research community. An informal description of Aglets can be found online at Javaworld.com [HREF 15] with more detail being found in Cockayne & Zyda (1998a), Lange & Oshima (1998a), and Bigus & Bigus (2001a).

2.2.1.4. Database Agents

Database manipulation research, as indicated by Droms (1989a) has motivated or aided agent research with the aim to extract information on the user's behalf. The Information Grid of Rao *et al* (1992a), the TSIMMIS system of Garcia-Molina *et al* (1995a) and the Information Manifold of Kirk *et al* (1995a), were information access frameworks that provided user interaction models, and were suitable for the agents systems of Ambite & Knoblock (1995a), Fikes *et al* (1995a) and Knoblock & Levy (1995a).

Krulwich (1995a) detailed the use of an agent to learn a user's interest in documents stored in distributed heterogeneous databases. This problem is similar to that addressed in Kuokka & Harada (1995a), Li (1995a), Ordille (1995a), Lashkari *et al* (1994a), Kautz, Selman *et al* (1994a) and Maes & Kozierok (1993a). Solutions to some of the problems inherent in heterogeneous information retrieval have also been reported (Huffman & Steier 1995a; Steier 1995a; Steier *et al* 1995a).

Since the role of an education system is to facilitate learning, it is important for any developed system to be able to extract/manipulate data from both legacy and future data sources especially the Web. This data-mining[1], along with information filtering, will ensure that *relevant* and *accurate* information is made available to students in this study.

2.2.1.5. Information Filtering Agents

Parallel to this research on database agents, research into agent based Information Filtering (IF) (CACM 1992a) was being reported by McElligott & Sorensen (1993a), Huffman & Steier (1995a), Huffman (1995a), Karakoulas & Ferguson (1995a), Shardanand & Maes (1995a), Ferguson & Karakoulas (1996a), Ackerman *et al* (1997a) and Stefani & Strapparava (1998a). Sheth (1994a) combined research in adaptive information filtering, agent learning and interface agents to develop a personalised news filtering system - Newt.

Peripheral but important to these developments was the research undertaken on new information gathering and filtering techniques such as by Gravano, Garcia-Molina & Tomasic (1994a), Knoblock (1995a), Bowman *et al* (1995a), Gravano & Garcia-Molina (1995a) and Karakoulas & Ferguson (1996a). The findings of this research were applied in agent and Web-based systems (Cousins *et al* 1991a; McElligott & Sorensen 1994a; Goldberg *et al* 1992a; Hammond *et al* 1995a). Bowman *et al* (1995a, p. 1) introduced Harvest:

> *a system that provides a set of customizable tools for gathering information from diverse repositories, building topic-specific content indexes, flexibly searching the indexes, widely replicating them, and caching objects as they are retrieved across the Internet*

Harvest was one of the first local and wide area network indexers that could extract/index information from a variety of sources such as Web pages, PostScript, plain text, etc. It was a very efficient and flexible distributed information gathering architecture.

Of interest to researchers in the learning companion field (Chan & Baskin 1990a; Chan & Chou 1995a), Klark & Manber (1994a), and Pazzani *et al* (1996a), reported on a class of IF agents - personal assistants or interface agents. These two projects built interface agents that maintained 'better' hotlists for Web browsers. Similar systems were built by Lashkari *et al* (1994a), Mitchell *et al* (1994a) and Maes (1994a). Armstrong *et al* (1995a) described WebWatcher, a WWW Learning Apprentice which followed the user's actions and suggested hyperlinks to pages that were related to a given page, using its learned knowledge.

These systems provided interfaces to data, using intelligence, adaptability and learning, to reduce the need or the time for a user to retrieve, filter and classify data.

---

[1] The definition of data-mining adopted for this thesis is weak in that it is concerned with the **extracting of existing information** in some format rather than the **discovery of new information**. For this research, data-mining will be Web-mining.

2.2.1.6. Interface Agents

Desktop and Interface agents emerged from the research in the Programming-by-Demonstration (PBD) discipline (Bocionek & Sassin 1993a), and resulted in PBD systems which learnt by watching what interaction took place between a user and a Graphical User Interface (Maes & Kozierok 1993a; Metral 1993a; Lashkari *et al* 1994a). Cousins (1994a) provided a detailed review of papers on Intelligent Interface Agents.

Lieberman (1994a, p. 1) reported on several of the interface agents developed at the MIT Media Lab and indicated that the primary interest of the projects was "*experimenting with the learning algorithms and learning interfaces*". He concluded that the research met with two significant problems: a lack of "component technology", and the "structuring of the communication between the application and the agent". Newer programming paradigms now allow unrelated components to be used together through the use of standard application programmer interfaces. This solves the first problem and would allow for an agent module to be more easily incorporated into an application.

The second problem was not just another aspect of the component technology issue: it was also concerned with issues of Human Computer Interfacing and 'user models'. How does the agent model the user so as to be effective? The agent must construct a model of the user in order to reason, to anticipate events, and to underlie explanation. The agent must acquire its mental model through interaction and explanation (Dix, Finlay & Hassell 1992a).

Developing an effective agent interface is difficult because mental models have the following characteristics (Dix, Finlay, Abowd & Beale 2004a):

- They are often partial
- They are unstable and subject to change
- They can be internally inconsistent
- They are often unscientific and may be based on superstition rather than evidence
- They are often based on incorrect interpretation of the evidence.

The "conceptual" model (Hudson 2004a) which is also known as the "represented" model (Cooper & Reimann 2003a), refers to the mental model an interface designer "intends" their user to follow when using their product. These concepts can also be applied to an interface agent and the user. If the conceptual model is substantially different from the user's mental model, the agent may not be effective in helping the user in their tasks. Figure 2.2, adapted from Cooper & Reimann (2003a, pg. 23) illustrates the difference between the implementation model and a user's mental model.

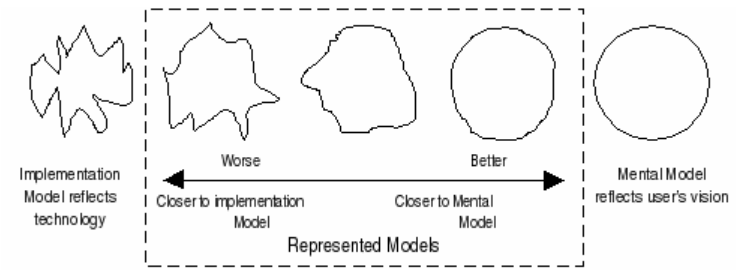


Figure 2.2 : Mental model vs. Implementation model. The closer the represented (or conceptual) model of an agent is to the user's mental model, the more effective the agent will be (adapted from Cooper & Reimann (2003a, pg. 23))

Wood (1993a) and Greif (1994a) indicated that agent learning could be applied to the entire desktop metaphor (Beale & Wood 1994a; Wood 1994a; Wood 1994b). As previously mentioned, Sheth (1994a) applied the learning paradigm to information filtering to develop an information filtering 'personal assistant'.

The early personal assistants/interfaces such as described in Etzioni, Levy *et al* (1993a), Etzioni, Lesh & Segal (1994a), Golden *et al* (1994a), Coen (1994a) and Etzioni & Weld (1994a) were applied to the World Wide Web to assist in browsing (Holte & Drummond 1994a; Armstrong *et al* 1995a; Balabanovic & Shoham 1995a; Balabanovic, Shoham & Yun 1997a; R Burke & Hammond 1995a; Davies & Edwards 1995a; Joachims *et al* 1995a; Pazzani *et al* 1996a; Ackerman *et al* 1997a).

Norrie & Gaines (1995a, pg. 1) stated:

> *The move to a learning society requires changes in personal attitudes and the educational infrastructure. It also requires major technological support to provide open access to a learning environment for all people, in all places at all times.*

They continued by arguing that a "*distributed, closely-integrated, intelligent 'learning environment'*" would emerge and that it would become "*part of the accepted surroundings and no longer viewed as an artificial tool, mechanism, or system developed to aid learning*". They envisaged ubiquitous, transparent, intelligent agents that would help people to learn throughout their life.

Similar personal assistants were also being applied to generalised browsing (Drummond *et al* 1994a), calendar assistants (Mitchell *et al* 1994a; Jennings & Jackson 1995a; Freeman & Fertig 1995a), finding and communicating with people (Ferguson & Davlouros 1995a; Ferguson & Davlouros 1995b) and Usenet news filtering (Sheth 1994a; Astikainen *et al* 1996a).

Agent interface technology was being applied to many areas historically rich in information technology (Watt *et al* 1995a; Chavez & Maes 1996a; Kearney 1996a; Rhodes & Starner 1996a; Song *et al* 1996a). M J Jacobson & Levin (1995b, pg. 1) described some of the educational issues regarding "ease-of-use" and "access" for users in the rapidly expanding information network of the Web. They argued that a new class of tools was required that "*preserves the easy utilization of network resources,* **yet also helps deal with the cognitive complexity associated with distributed network-mediated learning activities**" (their emphasis). They continued that these tools must be based on:

> *(a) an understanding of the special characteristics of distributed network learning environments,*
> *(b) the situated nature of human cognitive functioning, and*
> *(c) ways to support the attainment of substantive educational goals such as learning complex knowledge, problem solving, and independent thinking.*

Of interest was that their conceptual frameworks and tools were developed for the sharing of knowledge between student and teacher, not just a one-way transfer of knowledge. Also, they described a 'Learning Resource Server' that contained conceptually indexed domain knowledge that was available to all users.

Some successes in agent-human interaction were reported in Foner (1993a) and Mauldin (1994a). See Section 2.3.1 on page 48 (Dialogue Management) for more information on these Intelligent User Interfaces.

Previously seen only in interface and desktop agents, the anthropomorphism of agents became paramount in entertainment agents: Carnegie Mellon University (Bates *et al* 1992a; Reilly & Bates 1992a; Loyall & Bates 1993a; Bates 1994a; Loyall & Bates 1995a; Reilly & Bates 1995a; Reilly 1996a), and the Media Lab (Mauldin 1994a; M B Johnson 1995a; Maes 1995a; Rhodes & Maes 1995a; Maes, Darrell *et al* 1996a). Figure 2.3 shows one of the agents created in WavesWorld by M B Johnson (1995a), and Figure 2.4 shows the Woggles from the Oz Project of Bates *et al* (1992a).

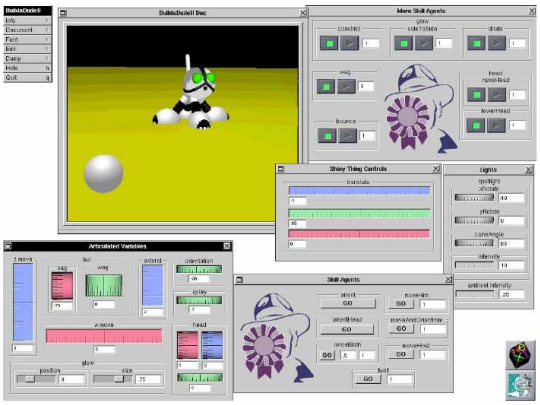


Figure 2.3 : Graphical User Interface developed by M B Johnson (1995a) for testing his anthropomorphic agent

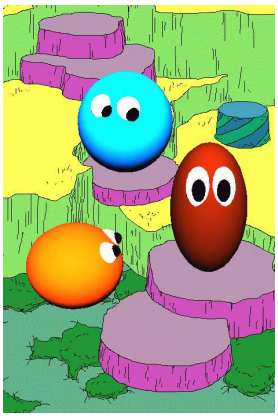


Figure 2.4 : Anthropomorphic agents from Bates *et al* (1992a)

Unfortunately, the major problem of anthropomorphic Graphical User Interfaces (GUIs) - the more realistic the agent, the more critical are users of its shortcomings in providing service - was not addressed by these works. Of interest though was the work by Isbister (1995a, pg. 2) with its concept of "*perceived intelligence*":

> *what is important is not the "true" intelligence of the machine, but instead, the perception the user has of the machine's "intelligence".*

In this case, the mental model issue was reversed: what mental model did the user have of the intelligent anthropomorphic interface? Isbister (1995a) followed the work of Nass *et al* (1993a) and her line of reasoning was similar to the Wooldridge & Jennings (1995a) discussions about 'what is an agent?'. That is, 'how do humans perceive intelligence in other humans?'. She summarised that we use: "appearance", "language cues", "actions/behaviours", and "group membership" to classify another person's intelligence. Therefore, for an interface agent system to be accepted by a user, and hence effective in its assistance, these mental model issues must be addressed. It is important to note that even for a non-embodied conversational agent, these issues are important because the user must accept the advice of the agent, and hence have trust in them.

The research by C Elliott *et al* (1997a), André *et al* (1997a), and L Johnson *et al* (1998a) in applying animated agents was also important since they developed an interactive learning environment. Lester, Converse *et al* (1997a, pg. 2) detailed the Persona Effect:

> *the presence of a lifelike character in an interactive learning environment-even one that is not expressive-can have a strong positive effect on student's perception of their learning experience.*

They used the term "animated pedagogical agent" to describe their teaching agents, and developed a learning environment where students interacted with Herman the Bug (see Figure 2.5). Herman was a lifelike agent whose visual and verbal interaction with the student was controlled by a behaviour sequencing engine. Following the findings of Isbister (1995a), Herman used appropriate language cues, and had a "*talkative, quirky, somewhat churlish*" appearance so as to motivate students (Lester, Converse *et al* 1997a, pg. 5).

Figure 2.5 : Herman the Bug from Lester, Converse *et al* (1997a)

Figure 2.6 : Sylvie the Verbot from http://vperson.com/

Figure 2.6 shows Sylvie the Verbot (http://vperson.com/ [HREF 16]) - an embodied conversational agent - designed and implemented by Michael Mauldin and Peter Plantec (see section on Verbots on page 53 for detailed information and a review of their relevance to this thesis). Although touted as the future of intelligent Talking Head interfaces to all information, their designers concentrated more on physical appearance than on giving the Verbots an affective personality or any learning paradigm that would help the user assimilate the information.

Adele (see Figure 2.7), a pedagogical agent developed by L Johnson *et al* (1998a), was far less physically realistic but much more effective in helping students to learn. The effectiveness of this agent was that it provided an intelligent user interface to learning material, not just a pretty face.

2.2.2. Intelligent User Interfaces

Maybury (1998a, pg. 1) defined Intelligent User Interfaces (IUI) as ones that

> *aim to improve the efficiency, effectiveness, and naturalness of human-machine interaction by representing, reasoning, and acting on models of the user, domain, task, discourse, and media (e.g., graphics, natural language, gesture).*

Figure 2.8, from Ehlert (2003a, pg. 1), shows that IUI research is multi-disciplinary and includes research from psychology, AI, HCI, ergonomics and cognitive science.

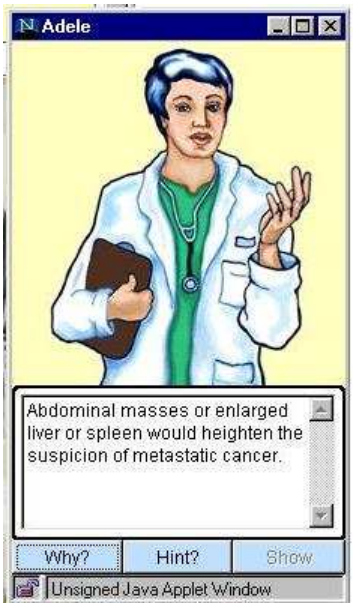


Figure 2.7 : Adele, the pedagogical agent developed by L Johnson *et al* (1998a, pg. 1)

Figure 2.8 also shows that research areas such as information filtering, natural language processing, dialogue management and multi-modal input/output through for example, anthropomorphic interfaces, are closely linked to IUI research.

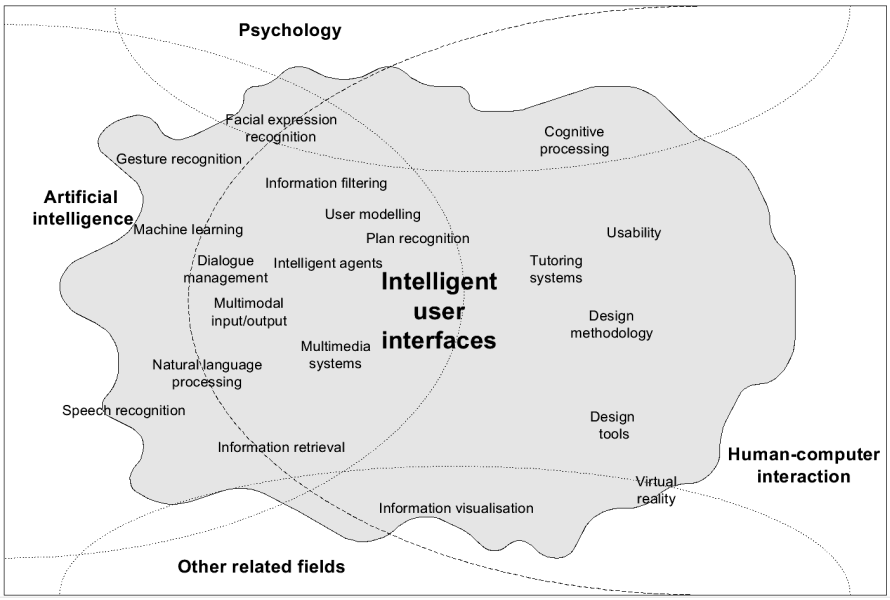


Figure 2.8 : The intelligent user interface research field (from Ehlert (2003a, pg. 4))

One motivation for using IUI in dialogues with the user is that sound, graphics and knowledge can convey ideas faster than technical documents alone (Bickmore *et al* (1998a)). Also, the use of an anthropomorphic agent may affect the way in which a user interacts with the interface and hence interface agents often use the personifying technique of depicting the interface as a human face. Koda & Maes (1996a) in their article 'Agents with Faces: The effects of Personification of Agents' investigated the most "favourable" interface through the use of a poker game and four computer players, all with different personas. Two important

results were found:

- Faces made the game more engaging.
- The human face rated first in intelligence and comfort, and was second to the dog cartoon in likeability.

People liked faces in interfaces. It should be noted however, that the use of anthropomorphic faces and audio is not seen as beneficial within a typical tutorial or practical environment - fifteen or more workstations 'conversing' with users may be distracting and also noisy. The results of the research by Koda & Maes (1996a) (and others) are applicable to any developed user-computer dialogue system but they must be used in context. For example, any dialogue management system that uses a computer generated talking head, could be used in a distance education environment where there is only one user to watch and listen to the Virtual Lecturer.

In discussing E-commerce, Poleretzy *et al* (1999a) explained that intelligent agents can provide context-sensitive assistance to customers based on their location on a Web site, their individual profile, their historical purchasing patterns and any new offerings from the organisation. Automated responses to email queries by consumers can also be generated by intelligent software agents, which process the text of the query and reply using pre-written responses matching the enquiry (see also Hibbard (1998a)).

One problem with software agents that provided a single question-answer facility was that the answer provided might not relate to the question exactly (see Cohn (1998a)). A more sophisticated tool was the case-based reasoning engine (Hibbard 1998a), which analysed customer questions and asked further questions (as necessary) to derive an appropriate answer from a database of pre-written responses. The database could be pure data or could be 'intelligent' in that the data contained could be updated by user queries - the system learnt about user enquiries through machine learning.

Maes (1994a) explored the use of machine learning in personal assistant interface agents. Initially the assistant was not familiar with the habits and preferences of the person they were assisting. By watching how the person went about tasks, being instructed by the person and learning from other assistants, the assistant agent gradually become more useful; this was machine learning.

Machine learning has advantages over knowledge-based and end-user approaches to interface agents. Knowledge-based interface agents often require teams of programmers and domain experts to carefully cover a domain and provide the data needed for an agent to be beneficial to a user. The problem with this approach is firstly the effort that must go into constructing the knowledge base, and secondly the user's habits and preferences may differ from those allowed for by the designers (i.e. a conflict of 'mental models'). End-user approaches to interface agents mean that the user has to explicitly program the agent to account for their individual habits and preferences. The problem with the end-user approach is that typically the users that can program the agent have less use for an interface agent.

The Persona project at Microsoft (Ball *et al* 1997a) involved the use of a computer character: Peedy the parrot. Peedy was used to control a CD changer in response to user requests. The system classified the user input through language analysis and template

matching, and then responded to the user through graphics, speech and the CD changer. However, Peedy had trouble knowing about CDs it was not programmed for, since it used a knowledge-based approach. Again, the draw back of this method was that it employed no machine learning, instead it relied heavily on the knowledge-based approach and hence reduced the functionality that an extensible system may otherwise have had.

Another important point made by Poleretzy *et al* (1999a) was that interaction with support staff would not become obsolete. Regardless of the 'intelligence' of the interface, it was important that human interaction was provided or encouraged when the user appeared to be having difficulty, so as to ensure a quality interaction experience. This indicates that these systems should always have a human in the loop even if they are only used as a fallback for when the interaction complexity exceeds the cognitive capacity of the interface.

The area of dialogue-based intelligent user interfaces is detailed further in Section 2.3 on page 48, but it is important that designers ensure that there is no 'mental model' conflict in their implementations. Höök (2000a, pg. 10) paraphrased Self (1988a) ("*don't diagnose what you cannot treat*") with "*don't model user characteristics that do not (profoundly) affect interaction*". She went on to indicate the findings of colleagues:

> *... the interaction with users has to be very rich (using natural language and other interaction means), if we are going to be able to infer anything from their interactions. Given limited communication channels, very few inferences can be made.* (Y Wærn *et al* 1990a)

That is, the most common modality used when learning about the user, especially in an NLP or Dialogue Management system, is simple key strokes. Very few systems use multi-modal input to detect the user's emotions or gestures, and hence are not able to detect how the user feels about the interaction. In 'Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction' (A Wærn 1996a), it was indicated that the user interactions were frequently low level, and hence it was difficult for a system to infer anything about the user from this. This implies that developing and implementing a robust student model may not be that beneficial or effective in a mentoring system.

There are also various mental model and cognitive problems with the existing desktop metaphor, as outlined by Krueger (1999a). The reason that the virtual desktop has been so successful to date, is because of the metaphor's likeness to our perception of the world, despite various limitations and mismatches in its implementation. However inaccurate the metaphor, the crucial issue is whether this helps users deal with the abstraction of a computer by providing an interface that is an improvement on the norm. Lovgren (1994a) explained that some mismatches, such as recursive folders, were an improved extension to real life that users were willing to accept.

In 'User Interface: A Personal View' (A Kay 1990a), it was argued that "illusions" should be used instead of literal metaphors. To use the literal metaphor of the desktop would be to ignore all of the benefits of the computer environment. The goal of an intelligent user interface is to reduce cognitive overhead when users interact with an application. Hence, any developed system must be based upon a well established metaphor that minimises user discomfort whilst maximising user interaction. Therefore, it is important to allow IUI applications to become an extension of real life rather than be constrained by it.

It is also important to allow an extension of the metaphor so that the ‘real world’ does not inhibit the functionality of the computer system. This area should avoid mismatches (i.e. features that occur contrary to real life) such as how the waste paper basket was first implemented on the Macintosh desktop metaphor as both a way of deleting files and also of ejecting floppy disks. Instead, this area can contain “magic” or “illusions” that extend features, such as the ability to delete text from a computer page, whilst in real life, pages are not easily erased (A Kay 1990a).

In developing an IUI system, it is also important not to adopt the “kitchen sink” approach which leads to a vast system with inaccessible power (Lovgren 1994a). A metaphor or ‘mental model’ is most successful if it comes from the problem domain. For example, if an IUI system was used in an online museum or art gallery, perhaps the best IUI metaphor would be a curator or guide.

The advantage of an anthropomorphic metaphor framework for an IUI is that human contact is prevalent in nearly all aspects of life. Wonisch & Cooper (2002a) showed that when people were given the choice of interface agents there was a tendency for them to choose interface agents that were visually contextually linked with the problem domain. For example, Herman the bug (Lester, Converse *et al* 1997a), and Adele (L Johnson *et al* 1998a), were both linked to the problem domain of the system. Wells & Fuerst (2000a) showed that domain oriented metaphors could have a positive effect on memory retention.

Finally, since normal human-human interaction is based on natural language, it is advantageous for intelligent user interfaces to be able to use this modality (Rabkin & Tingley 1999a). The AI Magazine special issue on Intelligent User Interfaces (AAAI 2001a) detailed the work of several researchers in the conversational intelligent interface area, including those working on learning environments (Rich *et al* 2001a; Graesser, VanLehn *et al* 2001a; W L Johnson 2001a).

2.2.2.1. Natural Language Interfaces

Natural language processing (NLP) emerged from research into computational linguistics: a multi-discipline of linguistics and computer science that is concerned with the computational aspects of human language processing. The seminal paper on machine translation by Weaver (1955a) proposed that computational linguistics might solve the “world-wide translation problems”. However, early ‘word-for-word’ translation techniques were quickly seen as naive, and a requirement for entire language understanding emerged.

Jurafsky & Martin (2000a, pg. 4) indicated that this language understanding was difficult and that it required the development of systems to process:

- *Phonetics and Phonology -- The study of linguistic sounds*
- *Morphology -- The study of the meaningful components of words*
- *Syntax -- The study of the structural relationships between words*
- *Semantics -- The study of meaning*
- *Pragmatics -- The study of how language is used to accomplish goals*
- *Discourse -- The study of linguistic units larger than a single utterance.*

Although the first is not relevant to a non speech-recognition system, the other points indicate that language understanding is a complex task requiring knowledge from many research fields. Lewin *et al* (2000a) indicated that maintaining a dialogue "context" could also aid in the semantic step, and was necessary to reduce ambiguity in languages.

A context also aids in anaphora (Hirst 1981a) and ellipsis substitution, which needs to be done to augment the semantic analysis. An anaphora is one or more words that refer to a previous concept. The anaphora is frequently a pronoun such as 'he', 'she' or 'it'. Consider the user input: "I found this CD. How much is **it**?". The '**it**' refers to the CD. Ellipsis refers to the parts of an expression that may be missing but can be inferred from the context or the state of the dialogue. For example, in response to the enquiry "Who do you wish to talk to?", the answer "Fred Bloggs" implies the total answer "I wish to talk to Fred Bloggs". The ellipsis would be 'supplied' through context management.

Winograd & Flores (1986a) indicated that resolving word or phrase ambiguity was one of the most difficult problems in NLP, a view supported by Jurafsky & Martin (2000a). They gave the example of "*I made her duck*" (pg. 4) and indicated that it could mean:

(1.1) I cooked waterfowl for her.
(1.2) I cooked waterfowl belonging to her.
(1.3) I created the (plaster?) duck she owns.
(1.4) I caused her to quickly lower her head or body.
(1.5) I waved my magic wand and turned her into undifferentiated waterfowl.

A slightly better example of ambiguity was given by Sarkar (2003a): "*British Left Waffles on Falkland Islands*". In this example, colloquialisms as well as common knowledge are required to disambiguate the terms '*Waffles on*' and '*British Left*'. The Cyc project (Guha, Lenat *et al* 1990a) is often referred to as a Knowledge Infrastructure, in that the NLP sub-system, and the knowledge base that can supply the common knowledge and colloquialisms, are heavily interdependent.

The "Pragmatics" issue of Jurafsky & Martin (2000a, pg. 4) also indicated that researchers categorise NLP in different ways. Winograd (1972a) grouped NLP systems according to how they represented and used their domain knowledge:

1) constrained domain systems that did minimal parsing, instead using keyword recognition and limited sentence knowledge.
2) a text-based approach that stored templates to match against.
3) limited-logic systems that used some formal notation for representing and storing sentences so as to extract the sentence semantics.
4) knowledge-based systems that store information in a form such as first-order logic or frames, so as to match against the user's input.

An example of the last category was SHRDLU (Winograd 1971a) which used natural dialogue as well as reasoning to explain its responses and actions (the system simulated a robot arm that followed requests to move blocks). Research papers on NLP systems that belong to these different categories are archived at http://acl.ldc.upenn.edu/ [HREF 17]- the ACL Anthology: a digital archive of research Papers in Computational Linguistics.

In 1963, Robert Lindsay created SAD-SAM (Syntactic Appraiser and Diagrammer -- Semantic Analyzing Machine), a computer program that could read English and draw conclusions from the input. The domain was limited to familial relationships and used a vocabulary of 1700 words (A McCarthy 2000a).

SIR (Raphael 1964a, pg. 1) was a format-matching computer program written in LISP that "*accepts information and answers questions expressed in a restricted form of English.*" This system used word associations and property lists and demonstrated what can "*reasonably be called an ability to "understand" semantic information*".

In 1965, Daniel Bobrow wrote STUDENT, one of the first rule-based Expert Systems that could 'understand' high-school algebra "word problems". Minsky (1985a, pg. 3), in discussing this 'understanding', bypassed the question by arguing that humans do not agree on 'understanding' either. He asked: "*does STUDENT avoid the "real meanings" by using tricks?*". Of importance in this discussion is whether STUDENT was effective in what it did for users, regardless of whether it understood the user's input or not. The shallow understanding of STUDENT was enough to help users.

The LUNAR system developed by Woods in the early 70's used an Augmented Transition Network (ATN) to translate an English question about lunar rock samples into a formal query language expression, and then to classify and answer the question. An ATN is similar to a Finite State Machine, and where the transitions from state to state correspond to the parsing of English grammar.

R C Schank (1972a, pg. 3) proposed that the basis of natural language is conceptual, and that Conceptual Dependency theory is a way of representing information at the conceptual level. The meaning of the information is built up with action and object primitives as well as modifiers or aiders to these primitives. The MARGIE (Meaning Analysis, Response Generation, and Inference on English) system (R C Schank, Goldman *et al* 1973a; R C Schank, Goldman *et al* 1975a) used Conceptual Dependency as well as inferencing through a semantic net[2] to be able to infer meaning from the user's input.

Schank continued with his Natural Language understanding research with SAM - a Script Applier Mechanism (Lehnert 1975a; R C Schank & Abelson 1977a) that used script-based knowledge so as to paraphrase, and then answer questions concerned with stories about eating in a restaurant.

These research projects played an important part in the development and investigation of various natural language understanding paradigms in the 1980's, especially in those using language grammars and knowledge bases (Allen 1987a). Similar research indicated that language understanding could benefit from being linked to language learning: "*... it seems odd that so many NLP researchers seem comfortable with the notion that language understanding should be studied independent of issues of memory modification*" (R C Schank & Kass 1987a, pg. 113).

---

[2] Coined by Ross Quillian in his Ph.D. thesis (1968): the organisation of semantic memory concepts

SHRDLU and other examples of natural language processing systems such as Eliza by Weizenbaum (1976a, pg. 3), START by Katz (1997a, pg. 3), Colin and Julia by Mauldin (1994a, pg. 3), as well as systems that have competed in the Turing and Loebner contest, will be discussed further in Section 2.3.1.2 on page 49.

Although in the context of speech-recognition systems, Allen *et al* (2001a, pg. 1) indicated that NLP was difficult. They indicated that one approach was to have the interaction with the user being driven by the system asking questions. The negative side to this was that the system limited the user interaction: "*You might need to provide all sorts of information that isn't relevant to your current situation, making the interaction less efficient*". They continued by indicating that a better approach was to base a system on normal human conversation: "*dialogue enhances the richness of the interaction and allows more complex information to be conveyed than is possible in a single utterance*". Given the mixed-initiative requirement of normal human interaction, this suggests that any developed system should maintain tight control over the dialogue in critical stages such as tutoring about a specific task so as to minimise misunderstanding, but to also allow the user to ask arbitrary questions at any time so as to promote self-learning in a non-linear manner.

Rich *et al* (2001a, pg. 16) supported this 'normal human conversation' proposition, and indicated that user interfaces should be able to answer the six questions of:

Who should/can/will do _____ ?	*When did I/you/we do _____ ?*
What should I/we do next ?	*Why did you/we (not) do _____ ?*
Where am/was I ?	*How do/did I/we/you do _____ ?*

where the missing text is domain specific. They believed that the next goal in user interfaces should be to support communication about a user's task structure and process, such as is required by these questions. Their Collagen system, based on a tripartite framework (Grosz & Sidner 1986a; 1990a; Grosz & Kraus 1996a) was a step in that direction. These questions are applicable in mentoring and learning domains.

Given the extent of use of mobile phones by students, and the advent of the Small Messaging System (SMS), users are starting to abbreviate normal English words so as to fit within the 160 character limit of the message format. This may pose problems for NLP systems since the input will no longer be correct English. Granger *et al* (1983a) reported on a system that used syntactic and semantic expectations to correct errors in 'scruffy' input such as: "PEGASUS FRD 2 TALOS AT VICTOR". The system then generated multiple paraphrases of likely correct sentences.

Weischedel (1983a, pg. 1) talked about a similar NLP problem of ill-formed input:

> • *absolutely ill-formed, if native speakers generally agree that it violates one or more linguistic constraints, or*
> • *relatively ill-formed, if it violates some constraints of the computational system, though native speakers perceive nothing odd about it.*
> *Examples of absolute ill-formedness include misspellings, mistypings, mispunctuations, tense and number errors, word order problems, run-on sentences, sentence fragments, extraneous forms, and meaningless sentences.*
> *Examples of relatively ill-formed input include unknown words and requests that are beyond the limits of either the computer system or the natural language interface.*

This use of multiple paraphrases to classify ill-formed input, such as the classifying of speech utterances in Kadous & Sammut (2002a), could be used in an NLP system to process SMS input.

Graesser, VanLehn *et al* (2001a, pg. 1), in discussing conversational agents and their ability to handle user questions, argued that although there had been a few successes in conversational Intelligent Tutoring Systems (ITSs) (specifically, Graesser, K Wiemer-Hastings *et al* (1999a); VanLehn *et al* (2000a)), they were uncertain whether the "*conversational interfaces will produce incremental gains in learning over and above the existing ITSs*". They were however optimistic, and indicated in Graesser, P Wiemer-Hastings *et al* (2000a) for AutoTutor (see Figure 2.9) that the:

> *... LSA[3] space of AutoTutor exhibits the performance of an intermediate expert, but not an accomplished expert. It should be noted that the vast majority of human tutors have intermediate expertise, rather than accomplished expertise, so the LSA space does an excellent job simulating the normal unskilled human tutor.*

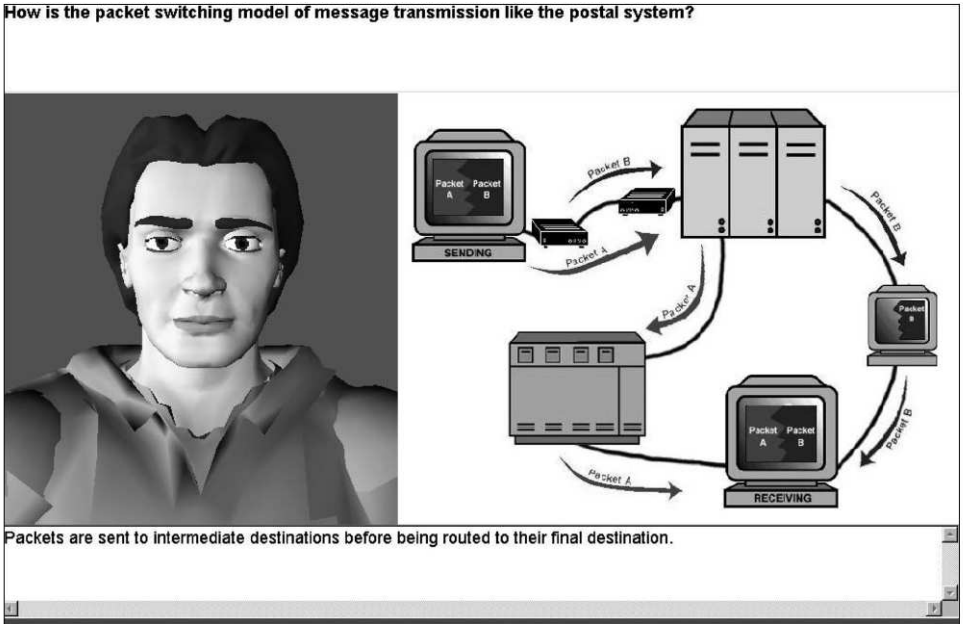


Figure 2.9 : The AutoTutor system of Graesser, P Wiemer-Hastings *et al* (2000a)

The need for evaluation of developed NLP systems was becoming more important (Palmer & Finin 1990a) but, in most NLP systems, of primary importance was the balance of 'recall vs. relevance' - the standard for Information Retrieval systems. However, it is difficult to rate NLP systems associated with learning environments in this way since a real tutor may give 'strange answers' to user questions so as to make a user think rather than just give them an answer. How is relevance measured in this case? In an NLP application, 'effectiveness' may be a better measure of success. That is, the "*black-box evaluation*" or "*what a system does*" evaluation (Palmer & Finin 1990a, pg. 176).

[3] Latent Semantic Analysis (LSA) (Deerwester *et al* 1990a), see LSA Section on page 61.

Finally, given the pragmatic approach of this thesis - the developed mentoring system must work in a real-world environment - the field of Natural Language Engineering (NLE) is important. This NLE approach to NLP, quoted below from the Editorial of the first edition of the Journal of Natural Language Engineering, guided the development of the software based mentoring system.

> *The principal defining characteristic of NLE work is its objective: to engineer products which deal with natural language and which satisfy the constraints in which they have to operate. This definition may seem tautologous or a statement of the obvious to an engineer practising in another, well established area (e.g. mechanical or civil engineering), but is still a useful reminder to practitioners of software engineering, and it becomes near-revolutionary when applied to natural language processing. This is partly because of what, in our opinion, has been the ethos of most Computational Linguistics research. Such research has concentrated on studying natural languages, just as traditional Linguistics does, but using computers as a tool to model (and, sometimes, verify or falsify) fragments of linguistic theories deemed of particular interest. This is of course a perfectly respectable and useful scientific endeavour, but does not necessarily (or even often) lead to working systems for the general public.*
>
> Boguraev *et al* (1995a)

2.2.2.2. Software Agent and Intelligent User Interfaces Conclusion

It can be seen from the preceding discussion that any user interface needs to be context sensitive, provide relevant material to requests and should be able to learn the users habits and preferences. The research findings of agent technology, especially pedagogical agents, provides many benefits in a learning environment.

It is important for the system to enable human intervention within the user-computer interface to cater for requests that are beyond the ability of the interface to handle.

It is also important for the system to have the correct ‘mental model’ of the user - not necessarily the ‘student model’ concept of an Intelligent Tutoring system (see Section 2.2.3.4 on page 44) - so that the interface assists the user rather than getting in the way or becoming a nuisance such as in the case of Microsoft’s™ Paper Clip. One important guideline for designing intelligent interfaces is:

> *Do not disturb the user’s interaction. It should always be possible for the user to ignore the proactive actions of the IUI. Suggest rather than act.*
>
> Ehlert (2003a, pg. 8)

Given the scope of the thesis, any developed NLP system should be an NLE system - it must work in a real environment. Further references to developed NLP systems in this thesis will assume the NLE definition given by Boguraev *et al* (1995a, Section 3.2.3) .

Given this pragmatism, the NLP system should only use shallow parsing. As indicated by Karlsson & Karttunen (1996a, Section 3.2.3), the term represents analysers that

> *... are less complete than the output from a conventional parser. ... may identify some phrasal constituents, such as noun phrases, without indicating their internal structure and their function in the sentence. ... identifies the functional role of some of the words, such as the main verb, and its direct arguments. ... aims at detecting phrases and basic head/modifier relations.*

That is, the NLP will not be a natural language **understanding** system but will classify the **intent** of the user's input. The complexity and functionality of the NLP system necessary for assisting users in a software based mentoring system is further discussed in Section 2.3.1 on page 48.

2.2.3. Educational Technology

2.2.3.1. Computer Based Learning

Computer Based Learning (CBL), also loosely called computer-assisted instruction (CAI), computer-assisted learning (CAL), computer-based education (CBE), and computer-based training (CBT), is an educational technique that uses a computer to interact with a student to help in their learning. It normally has an educational strategy for delivering the material, and may also track the student's progress for feedback purposes.

In a review of the literature on CAL and Integrated Learning Systems (ILS) commissioned by the New Zealand Ministry of Education, Parr & Fung (2004a, pg. 4) were very critical of CBL success stories. They used a metric called the "effect size" that "*allows different types of information about effects to be converted into a common measure of effectiveness* ....". They indicated that although there is no consensus, the values for an effect size (0 means no effect, positive is beneficial, negative is detrimental) are roughly:

effect size	Effect on educational achievement
.2 to .49	small effect
.5 to .79	medium effect
.80 and above	large effect

As can be seen in Table 2.1, the effect size of the reviewed CAI systems fell into the 'small' category, and importantly, they continued "*that CAI is most effective in the primary school. Student gains are somewhat less at the secondary level and lowest at the tertiary level*" (Parr & Fung 2004a, pg. 6).

Although not given as a direct reason for this lack of educational effect of ILSs, they quoted the 1999 research by Wood, Underwood and Avis that indicated that the "*assessment of performance which underpins an ILS is that of a traditional kind*". This may imply that new systems are required that have moved from information transfer to tutorial learning, and allow a student to construct the knowledge they require at their own pace and level.

	Instructional Level	# of studies	Effect size	Percentile Gain over Control Group
Aiello & Wolfle (1980)	Secondary, College	11	.42	16
Bangert-Drowns et al. (1985)	Secondary	42	.42	16
Burns & Bozeman (1981)	Elementary, Secondary	46	.39	15
Hartley (1978)	Elementary, Secondary	33	.42	16
Kulik & Kulik (1985)	College	101	.26	10
Kulik et al. (1985)	Elementary	28	.47	18
Kulik et al (1984)	Adult	24	.42	16
Niemiec et al (1984)	Elementary	48	.45	17
Samson et al (1985	Secondary	42	.32	13
Schmidt et al (1985-86)	Special Education	26	.56	21
Schwalb et al(1984)	Japanese Schools	4	.45	17

Table 2.1 : Findings from Eleven Meta-Analyses on Computer-Assisted Instruction (1978-1985) (from Niemiec & Walberg (1987a))

2.2.3.2. Online Learning

Many Web-based courses allowed active, self-paced, non-linear learning to help users to construct domain knowledge, but these Web pages needed to evolve to effectively address other pedagogical issues and techniques (Eklund 1995a; Eklund 1995b). Research in the ITS and Hypermedia System (HMS) areas is extensive and ITS's were being moved onto the Web. These systems (Brusilovsky 1995a; Hubler & Assad 1995a; Nakabayashi *et al* 1995a) were however limited to the functionality of forms, CGI scripts, or button clicking navigation.

The next generation of courses aimed to use tested pedagogies from the ITS area to add a 'learning by doing mode' as well as intelligent guidance through the use of Java applets and software agents. In this way, a Web page can become an integrated system of information, plug-ins and applets that work together with CGI-bin scripts, and software agents to produce intelligent interfaces between the user and information (see Figure 2.10).

For example, a Java applet could monitor hyperspace navigation and suggest other links (supplied by the Software Agent) to a keen student, or could present supplementary information to a student who was seen to be 'struggling'. The applets may also interface with CGI-bin scripts and Software Agents to implement 'level of presentation' methods.

It can be seen that the 'intelligence' enabled by Java allowed an online course developer to provide some minimal control over their learning strategy through the use of applets. Figures 2.11 (a) and (b) show Java applet examples that enable students to interact with various computing algorithms so as to increase their understanding of the algorithm.

The sections of Figure 2.11 (a) show the pseudo-code for accessing each element of an array along with an animated display reflecting this accessing. As this pseudo-code 'execution' happens, an explanation of the code as well as the code 'output' is displayed. Similarly, Figure 2.11 (b) animates the displaying of the syntax of an integer as well as the

pseudo-code that would parse it. The student can enter any number, and, step by step, see the paths and code that would be traversed to check the syntax of the entered data. In both examples, the student has control of the demonstration, and the student may improve their learning, by doing.

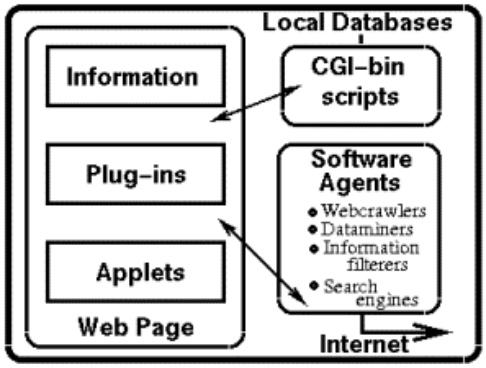


Figure 2.10 : Schematic for an intelligent Web page for use in educational technology.

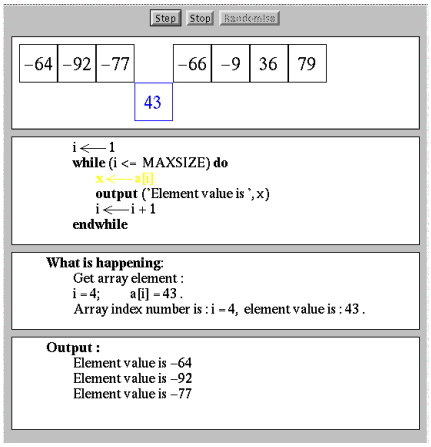


Figure 2.11a : Example one

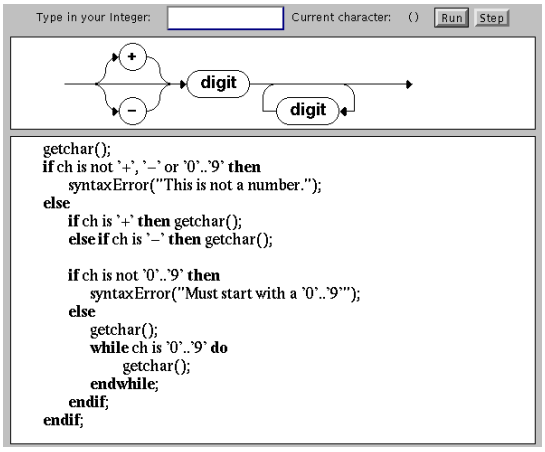


Figure 2.11b : Example two

Figure 2.11 a and b: Interactive Teaching through Java Applets

Similar examples have been developed for the parsing of other data types as well as interactive examples of common iterations, binary operations and various sorting and searching algorithms. Research has shown that these interactive Java applets, when supplied with source code in the context of online course notes, enhance aspects of active learning (Marriott, von Konsky & Ng 1997a). The traditional use of example programs has evolved into run-anywhere, in-the-browser, hands-on learning.

The role of educators has both been made easier and also more difficult by the coming of age of the Web. It is easier because more information is made available to both students and researchers. It is more difficult because telling a student that the information 'is out there' is simply abrogating our responsibilities as a teacher. Any learning support system should present accurate and relevant information, or information links for students. The educator's knowledge and experience should be the base 'filter' in any implemented Information Filtering model. The paper by Pazzani *et al* (1996a) details such a system, as does the WebWatcher Project [HREF 18] at Carnegie-Mellon.

2.2.3.3. Hypermedia Systems

Many researchers saw the potential of the Internet and specifically the World Wide Web for distance education and flexible delivery. Ng & Marriott (1994a; 1995a; 1996a) described early success as well as user attitudes to using online University course notes.

Marriott (1996a), Marriott, von Konsky & Ng (1997a), Marriott & Ng (1997a), Marriott (1997a), and Marriott & Armarego (1997a) reported on the use of advanced techniques in large scale projects which attempted to transform WWW 'electronic page turner' notes into more interactive user-driven 'intelligent Web pages' via applets and CGI-bin scripts. Nakabayashi *et al* (1995a) reported on CALAT - a distributed CAI system using the Web. Other researchers - Gruber *et al* (1994a), Holt *et al* (1995a), Ibrahim & Franklin (1995a), Norrie & Gaines (1995a), O'Connor (1995a), J Kay & Kummerfeld (1996a) and von Konsky (1996a) - have also reported on their systems and successes.

Other early innovative uses of the Web and associated technology have been reported by Philippe Duchastel (Duchastel & Spahn 1996a; Duchastel & Breuleux 1996a; Duchastel & Turcotte 1996a; Duchastel 1996a; 1996b; 1996c; Duchastel & Hannah 1996a).

Of interest to educators, Maulsby (1994a) described experiments with the Turvy system, which was not AI or agent based as such, but used a real human posing as an intelligent agent to evaluate the instructional model of, and the functional representation for, an instructible agent. The Cima system was based upon observations of user interaction with Turvy but only implemented the concept learning subsystem. This research raised some interesting non-computing problems associated with any form of agent-human interaction.

However, simply presenting information, even in an intelligent manner, does not necessarily promote learning. Eugenio (2001a), when talking about the role of tutors and computer-supported instruction, indicated that good tutors do not simply provide students with the answers. Good tutors helped students to construct the knowledge themselves. This view was supported by others who have reported on their research in the ITS field (Graesser, VanLehn *et al* 2001a; Graesser, Person & Magliano 1995a; Rosé, Moore *et al* 2001a; Rosé 2000a). It is the interaction and the collaboration between tutor and student - the "interthinking" of Mercer (2000a) - that fosters the constructing of knowledge and learning (Chi *et al* 2001a).

The constructivist view of Piaget and others, proposed that people form their own Knowledge Representation (KR) from the material presented to them, and in the context of their own experience. They construct this KR through active exploration rather than passive absorption, and that learning occurs through "accomodation" when they find an inconsistency between their current KR and the new knowledge (Bruner 1967a). This learning occurs within a social context, and that the interaction between learners and their peers is a necessary part of the process (Vygotsky 1978a) (see Section 2.3.2.4 on page 86).

Moshman (1982a) has further categorised constructivism into:

- **Endogenous**: KR construction - individual and specific.
  Teacher's role - facilitator and resolver of KR conflicts by providing past experience.

- **Exogenous**: KR - formal instruction and subsequent reflection and practice.
  Teacher's role - constructor of knowledge domain and guide.
- **Dialectic**: KR - cooperative/collaborative learning with peers and experts.
  Teacher's role - scaffolding provider to help learning.

As seen in Figure 2.12, this categorisation forms a domain of constructivism rather than independent, mutually exclusive areas. Any specific view of this learning theory can be seen as being influenced by the three categories (Dalgarno 1996a). Mentoring falls closest to endogenous constructivism but also includes learning through practice and reflection, and certainly involves collaboration with an expert - the mentor. See Section 2.3.2.4 on page 86 for an analysis of the constructivist factors affecting mentoring.

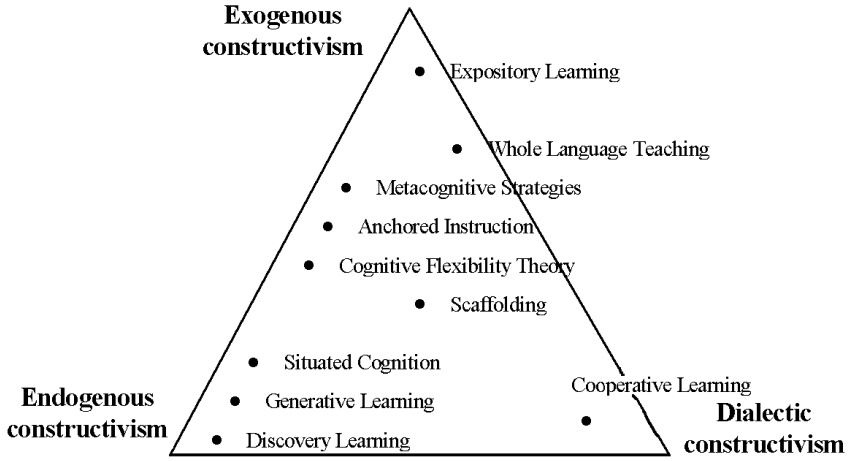


Figure 2.12 : Constructivist Pedagogical Theories (from Dalgarno (1996a))

2.2.3.4. Intelligent Tutoring Systems (ITS)

ITS are well established - a good source of online information is available from Farrow (1995a), Inlärning (1995a), and from the Tap Project (1995a). It has been argued that much of the ITS research is based on the research undertaken by the AI community (Huffman & Laird 1994a; Huffman 1995a).

The paper 'Some Possible Futures for Artificial Intelligence in Mathematics Education' by McArthur (1992a) is well researched and referenced, and explored the role of AI in education with specific sections outlining the MicroWorld concepts. This, and the author's experiences at Rand Corporation, formed the basis for predictions about future roles (McArthur *et al* 1993a).

Eklund (1993a; 1994a; 1995a; 1995b) looked at the problems for ITS given the new Hypermedia Systems (HMS) and proposed that a new cognitive model was needed for emerging educational material and its presentation. Brusilovsky (1995a), and Brusilovsky (1995b), reported on research into ITS and HMS within a Web framework, and Mark & Greer (1993a) reported on evaluation methodologies for a modern ITS. The use of agents with user modelling was becoming more prevalent (S M Brown *et al* 1997a; Conati *et al* 1997a), and was being reported in conferences such as User Modelling 97 (http://um.org/ [HREF 19]).

In the early stages of this research only a small number of researchers (Chan & Chou 1995a; Cheikes 1995a; 1995b), had put agents and ITS together. Cheikes (1995c) asked and, to some extent, answered the question: "Should ITS Designers Be Looking For A Few Good Agents?". The Gia system (Cheikes 1995d) used a minimal set of KQML and KIF to meet the needs of arbitrary distributed, collaborating heterogeneous intelligent agents and explored the advantages of using agent technology in an ITS environment.

Similarly W L Johnson (1995a), W L Johnson & J Rickel (1996a), and Rickel & Johnson (1997a; 1997b), reported on research into using agents in virtual learning environments where functions such as student modelling and coaching were assigned to individual agents rather than catered for by the overall ITS. W L Johnson (1995a) also provided valuable references for those interested in agents and ITS.

The use of an agent as a student model in an ITS was also hinted at by some (Hubler & Assad 1995a; Major 1993a; Project 1995b), and advanced CAI Systems were starting to emerge (Ritter & Blessing 1995a; B P Woolf & Hall 1995a; Centinia *et al* 1996a; Lesgold *et al* 1996a; Reichgelt *et al* 1996a).

The AI concept of an Expert Shell was used as the model for ITS shells (Blumenthal *et al* 1996a; Dooley *et al* 1995a) with the hope that more educators would be able to build systems faster. This culminated in the LEAP project (Sparks *et al* 1999a). This authoring framework, coupled with agent frameworks such as described in Lange & Chang (1996a) and Chang & Lange (1996a) was proposed as ensuring that this technology could be usable by non-computing educators.

It is assumed that as ITS knowledge increases, more use of agents will occur and that better use of Networked and MultiMedia information will also occur (see Psotka (1995a) for a futuristic vision). See Section 2.3.2 on page 79 and Section 2.3.2.2 on page 80 for a more detailed discussion on Tutorial Dialogue Systems and ITS issues.

2.2.3.5. User Evaluation of Educational Systems.

In evaluating user attitudes to any educational system, it is important to realise the problems associated with transcribing internally held attitudes to an externally consistent representation. In many educational studies Likert scales have been used to represent a range of attitude states. Users indicated their range of agreement or disagreement on an N-point integer scale. These numerical values were then used to generalise the overall attitude of users towards the recorded issue.

For a 5 point Likert scale, the responses are usually indicated as:

```
1. = strongly disagree with the issue
2. = somewhat disagree with the issue
3. = undecided
4. = somewhat agree with the issue
5. = strongly agree with the issue
```

To reduce researcher bias, the questions and responses are usually phrased so that the respondent is not told what to say or believe. Common Likert scales are 1-5, 1-7 and 0-4. Of importance is that odd numbered scales have a middle value that is typically associated with neutrality towards the issue. It is possible to force a user to decide to agree or disagree by using an even numbered scale.

Inherent in this process are the assumptions that the attitudes can be discrete, can be ranked, that the attitude ranking is consistent with the numerical value assigned, and that the generalisations are valid due to the normal distribution of the scale used.

It is difficult to determine if the Likert ranking's numerical value is consistent with the respondent's attitude. For example, the respondent's ranking may not be linear with respect to their attitude - they may **never** indicate a 5 or 'strongly agree' response. Likert scales can be used as Categorical, Rank-ordered or Continuous. This research adopted the same approach as taken in the original research by Likert by assuming that when used to record a respondent's attitude, the Likert scale represented a continuum of attitudes. Since the attitudes **were** assumed to be evenly spaced, some evaluation results also indicated the cumulative relative frequency of each response.

Similarly, in generalising the results of the various questions, Likert's original research assumptions are followed in that the overall distribution of the responses was considered Normal. It is important to note that with only 5 possible responses, it could not possibly represent a normal probability distribution accurately: the range of responses was discrete, not continuous. If the distribution is 'roughly' Normal however, then this is not a significant problem in generalising the research findings.

In 'The Evaluation of Computer Based Learning in Higher Education', Wills & McNaught (1996a, pg. 4) discussed the effectiveness of CBL compared to conventional teaching, and indicated the difficulty in assessing any educational innovation in student teaching and learning:

> *it is impossible to control all factors in such research and the conclusions of such research cannot be definitive or 'objective'. The comparison is only valid if the form of instruction in both cases is identical and the computer is the only variable. In reality, CBL developers are attempting to address new modes of learning, not just new modes of delivery.*

Of importance, they continued: (Wills & McNaught 1996a, pg. 4)

> *There are also ethical questions in strategies which assign educational 'treatments' to 'matched' groups of students thus denying access by one group to what is seen to be the best possible teaching at the time.*

It was also important to be aware that research and evaluation carried out in an educational environment can place unnecessary pressure on students to 'respond in a safe way'. That is, many students may have been inclined to 'over-rate' a system. Questionnaires that preserve anonymity can reduce this problem but can also allow for the opposite problem of rankings that are not based on evaluating the issue at hand.

Social Acquiescence[4] or courtesy bias is another phenomenon that may distort research questionnaire results from users from some countries, especially those from Southeast Asia. Since in the planned research on a mentoring system it would not possible to determine a

[4] Social acquiescence occurs when the cultural background of a respondent biases the reply in favour of the higher social class researcher or the authority that they represent. They want to 'please' the researcher.

student's bias either way, it was assumed that either the respondent's rankings reflected their true attitude towards the issue, or that on average there would be a similar number biased one way as the other.

Finally, Neal & Ingram (1999a) suggested that the teacher-student feedback cycle is very important but was often missing from many early educational technology systems. This feedback provides information about student progress and understanding, and it should be provided continuously rather than at the end of student usage of a system. This indicates that any mentoring system should allow for continuous monitoring of student interaction so that problems can be picked up and rectified early on in the study.

2.2.3.6. Educational Technology Conclusion

Although CAI itself may not suffice, a system based on ITS research that uses hyper- and multi-media as part of its instructional media, may be beneficial in helping students to learn. Of importance is that the system must enable an active learning strategy that helps students to construct knowledge, not just be an electronic presenter of material.

It was also seen as important to integrate the mechanisms for evaluation of the system into the overall design, and not just to have them as an add-on. The evaluation feedback can be seen to be useful to the student during the study as well as essential for the researcher as both formative and summative information about the system's performance.

Various (online) journals are now publishing articles on Education Technology: Australian Journal of Educational Technology [HREF 20], Journal of Educational Multimedia and Hypermedia [HREF 21], Journal of Interactive Learning Research [HREF 22], Journal of Technology and Teacher Education [HREF 23], Journal of the Asynchronous Learning Network [HREF 24], International Journal on E-Learning (IJEL) [HREF 25], Journal of Technology and Teacher Education (JTATE) [HREF 23], Educational Technology Review (ED-TECH Review) [HREF 26], Journal of Educational Computing Research [HREF 27] and EDUCAUSE [HREF 28]. Some journals, such as reported in Kahle (1995a) and S P Shulman (1995a), are also becoming aware of the role of Web and Agent technology in the education arena.

2.3. Research Problem Theories

Analysis of the parent theories has indicated that the design and development of a mentoring system to help students requires further analysis of the Dialogue Management and the Tutorial Dialogue Systems research areas. Concepts and issues in these two areas will provide the foundation for the Research Problem theories. The first area follows on from the parent theories of Software Agents and specifically the research on Intelligent User Interfaces. The second area is a specific sub-area of Educational Technology and is closely aligned with the research on Intelligent Tutoring Systems.

Analysis of these two research problem areas has indicated suitable techniques and technologies to use in the construction of a software based mentoring system. It has also indicated unresearched or unevaluated areas that should be considered for innovative and novel design and development research.

2.3.1. Dialogue Management

2.3.1.1. Introduction

It has long been man's dream to be able to interact or converse with something man-made. For example, G.B. Shaw's 'Pygmalion', the android in Fritz Lang's 'Metropolis', Robbie the Robot in 'Forbidden Planet', HAL in Clarke's '2001'. The software technology that enables this interaction to occur is called Dialogue Management (DM) or sometimes Discourse Management.

Recently, computer scientists have been confident enough to be able to debate as to whether this dialogue between user and machine could become a reality. The debate was often on a philosophical level, the question being: "can machines think?". A seminal argument in this debate was seen in the collection 'Computer Machinery and Intelligence' by Alan Turing (Turing 1950a). He argued that if a judge could not decide if a hidden contestant was human or not simply based upon their ability to carry on a question and answer conversation, then for all intents and purposes, the contestant was 'human' regardless. The computer either imitates a human or is 'intelligent'.

This intelligence testing became known as the Turing test, and in 1991 it was conducted as a formal computer competition: the Loebner contest. The AI programs competing in this contest are often referred to as Chatterbots. These Chatterbots emerged from research on Interface Agents (see Section 2.2.1.6 on page 27).

The Natural Language Processing (NLP) of the user's input to understand what is being typed, is one of the current challenges in DM systems. For example, using the example from the Cyc [HREF 29] Web page:

- `Fred saw the plane flying over Zurich.`
- `Fred saw the mountains flying over Zurich.`

These similar sentences have very different meanings because humans can disambiguate the object that was flying, through the use of common sense. NLP systems would need to know that planes can fly but that mountains cannot.

It can be seen from the discussion of Section 2.2.2 on page 30 that any dialogue management system needs to be context sensitive, provide relevant material to requests, and should be able to learn the user's habits and preferences. This is similar to many existing 'intelligent search engines' such as Google [HREF 30]. These handle a user's enquiries and return relevant information based upon user preferences. It is also important for a dialogue management system to enable human intervention within the user-computer interface to cater for requests that are beyond the ability of the interface to handle.

2.3.1.2. Dialogue Management Systems

One problem with having an intelligent search engine as the primary user interface is their user complexity. This is due to:

- inconsistencies between the search engines (Lohse & Spiller 1998a) provided by different Web sites (such as their syntax and advanced search capabilities),
- the often large numbers of 'hits' produced which might not even be relevant to the query being made (Wiley 1998a) - the 'recall vs. relevance' issue, and,
- the plethora of navigation features and options, such as buttons, frames, pull-down menus and image maps (R R Burke 1997a).

One approach that has the potential to provide a more consistent and human-like interaction search facility would involve NLP (Rabkin & Tingley 1999a) coupled with an Interface agent. Such a mechanism for querying would involve a natural language question being typed by a user and the intelligent interface agent carrying out pattern matching or finding cues which are associated with the requested information or answer. This approach also obviates the need for sophisticated navigation features or search options and ensures that all such agents on different Web sites are interacted with in the same way.

This querying via natural language is handled by a Dialogue Management System (DMS). Lewin *et al* (2000a, pg. 11) indicated that dialogue management is involved in:

1. turn-taking management: who can speak next, when, even for how long
2. topic management: what can be spoken about next
3. utterance understanding: understanding the content of an utterance in the context of previous dialogue
4. 'intentional' understanding: understanding the point or aim behind an utterance in the context of previous dialogue
5. context maintenance: maintaining a linguistic and dialogue context
6. 'intentional' generation (or planning): generating a system objective given a current dialogue context
7. utterance generation: generating a suitable form to express an intention in the current dialogue context

One important aspect of this (items 3 and 4 above) is the understanding of the user input. Other aspects are concerned with responding appropriately with relevant information and in maintaining the context or state of the dialogue. NLP is therefore a significant step towards enhancing the interaction experience for users because it parallels to some extent the social human dialogue interaction of a face-to-face environment.

An effective way to enhance this more ‘humane’ (as opposed to ‘human’) interaction is by providing the answers to a user’s natural language queries using facial animation and voice reproduction. The use of the MPEG-4 (ISO/IEC 1998a) International Standard for facial animation to provide a more ‘humane’ visual interface can help with user acceptance of this type of interaction, whilst the use of various AI techniques such as text to speech synthesis, NLP and data-mining can help in making the user experience more consistent, enjoyable and profitable.

Although Section 2.2.1.6 on page 29 and Section 2.2.2 on page 30 pointed out that the arguments for a visual anthropomorphic interface are strong, it is not beneficial in a tutorial environment. However, NLP and machine learning techniques are obviously very useful in any IUI system.

Eliza

Arguably, the first high profile conversational system was Eliza; one of the first programs to successfully imitate humans. It was developed by Joseph Weizenbaum in 1966, and was renowned more for its illusion of understanding than for its AI techniques; the illusion being achieved by what Weizenbaum termed “tricks”.

Weizenbaum explained that the tricks he employed were to fool the user into assuming that there was more intelligence than was actually present. He admitted that Eliza was just a simple program, but could still play the “imitation game”. The tricks that Eliza used centred on the user directing the course of the conversation, with Eliza contributing very little substance (Weizenbaum 1976a). This was achieved through the characterisation of Eliza as a ‘psychoanalytic therapist’ and then through the use of ‘Rogerian’ analysis. Some psychoanalytic therapists use Rogerian analysis to allow a patient to extrapolate on their own thoughts by the therapist asking questions about the last response that the patient provided. The Rogerian mode can be described as impeachable cover for the computer, since because the program never declares any information, it cannot contradict itself later (Mauldin 1994a).

Eliza has features that are applicable to intelligent user interfaces. One is the elegance in which Eliza matches user input to patterns and reconstructs sentences that give the illusion of intelligence. An example of the type of pattern matching that Eliza is famous for:

USER INPUT	ELIZA’S RESPONSE
XXX me YYY	-> XXX you YYY
I remember XXX	-> Why do you remember XXX just now?
My {family-member} is YYY	-> Who else in your family is YYY?
XXX {family-member} YYY	-> tell me more about your family

(Cheong 1996a, p. 253)

Eliza took input from the user and attempted to match it to the patterns on the left. For example, any words (XXX) followed by the word "me" followed by any words (YYY). After finding the best ranking match, Eliza then replied with the right hand side, substituting some of the reply to the user with their original text. An example of this could be:

```
USER: My mother is always working
ELIZA: Who else in your family is always working?
```

In this example it can be seen that Eliza's response seems intelligent, but an analysis of Eliza's code shows that there is very little understanding. A simple pattern match and the use of the user's own words have created the illusion that Eliza is actually following the conversation. Eliza can also keep track of what the user has said and use it in a later ripost. Eliza uses other tricks, such as the use of questions with the user supplying most of the context of the conversation, and with Eliza at no point volunteering information. This approach is not enough for a robust intelligent user interface. For example, a Virtual Salesperson has to respond accurately to requests. One does not expect 300 pizzas to be delivered to your home because of a dialogue misunderstanding.

The way in which the computer science community embraced Eliza worried Weizenbaum; there was a belief that his program was the 'be all and end all' of computers understanding human language. This 'hype' about clever tricks masquerading as AI breakthroughs is still apparent today (see later section on Chatterbots at page 52 and also the section on the Loebner Prize at page 55). Weizenbaum proposed that a general solution to natural language processing was impossible, and that language could only be understood in contextual frameworks. The theory of contextual frameworks is that to some extent conversation occurs through the relating of beliefs and knowledge of a domain to one another. Weizenbaum believed that even humans do not employ a general solution in the way that they interact with one another through dialogue (Weizenbaum 1976a).

'Dialogues with Colorful Personalities of Early AI' (Güzeldere & Franchi 1995a), online at http://www.stanford.edu/group/SHR/4-2/text/dialogues.html [HREF 31], gives examples of interesting dialogues with ELIZA and other early AI conversation programs.

SHRDLU [HREF 32] was another early example of natural language understanding (Winograd 1971a). SHRDLU used a limited domain (the BLOCKS environment) and enabled the user to manipulate the environment via natural language. Similarly, START [HREF 33], a natural language question answering system, is described as "*the world's first Web-based question answering system*" and has been available since December, 1993. One of START's main objectives has been balancing retrieval vs. relevance. Online sources are often plagued by a very low 'signal to noise ratio'. Presenting users with relevant information from online sources becomes a significant problem.

Since the mentoring process is based upon the filtering ability of the mentor, only relevant material is passed on to the mentee. Any software based mentoring system must also adopt a similar strategy.

Michael Mauldin had the view that if you can get a computer program to pass a single domain, then fifty to one hundred groups working on different domains could create a single program capable of finding a common conversational ground with a user (Mauldin 1994a). He believed that passing the restricted Turing test would benefit from exploiting this idea.

Thomas Whalen supported Mauldin's view in reporting on a database he developed to answer a user's natural English questions. He believed that if you write enough good questions and answers, then a good "Chatterbot" is relatively simple to write (Whalen 1999a).

Mauldin and Whalen clearly believed that the answer to conversational architectures was a knowledge-based approach. Chatterbots needed to know how to talk to begin with. No-one wants to have to teach a computer how to converse; that being a lot more complex than, for example, developing email interface agents. Clearly however, this type of design will be limited by its initial knowledge and so a merging of knowledge-based and machine learning approaches has become prevalent in the modern design of Chatterbots.

Chatterbots

Chatterbots are conversational agents constructed to simulate conversation and/or provide useful information. Many of these can be experienced at the Simon Laven Chatterbot website (http://www.simonlaven.com/) [HREF 34]. In many early instances, Chatterbots were used in Multi User Domains (MUD's) to converse with users and to provide useful information such as answers to questions like "Where is player X?". This was done by parsing the user's query and matching it against the system's 'user-question' database. These natural language databases allowed users to ask questions (query) in their natural language and receive an appropriate answer.

A popular tool used within many Chatterbot systems is the use of pattern matching. Two of the most famous Chatterbots using this technique are Colin and Julia - Maas−Neotek robots - constructed by Michael Mauldin. Both are TinyMUD robots with Colin being available via FTP (ftp://nl.cs.cmu.edu/usr/mlm/ftp/robot.tar.Z [HREF 35]).

The way in which Colin and Julia were designed was to use a tree−based model of input/output patterns to transfer the conversation between a number of states (Mauldin 1994a). By using states, Mauldin hoped to guide the user through a restricted conversation. The use of states meant that this knowledge-based approach was limited to how good the conversational module was - there was no learning to improve the way in which the Chatterbot reacted to the user.

These Chatterbots were an improvement on Eliza because they used more tricks and had more sophisticated memories of past events and conversations (Maes 1995a p. 110). Mauldin (1994a) indicated some of the tricks that Colin and Julia employed:

- having a state-based restricted conversation,
- using controversial statements to draw the user into the directed conversation,
- using humorous statements to make the program seem more human, and
- excerpting USENET News into the conversation, and
- hand crafting patterns used for input matching.

This form of dialogue management was also more robust at giving correct answers or information to user requests (but see later limitations on page 60). Hence, it was very suitable for an intelligent user interface. Finally, the Chatterbots seemed more humane. Since it has been shown that it is very important for an interface to appear human and humane, it is instructive to see how a Chatterbot is perceived by someone interfacing with them (Foner 1993a). Julia is a good example of a humane Chatterbot.

Julia is a client bot - she connects to the MUD server via telnet just as a normal human player would. Julia has an interesting persona, most of which is to make her seem more human (Foner 1993a). For example, since Julia can map out a MUD, it is possible to ask her for directions. Julia will also lend you money, can describe herself and of course can carry on a reasonable conversation. She retains limited knowledge about past conversation subjects and reuses them.

Julia often appears to be too human. The following conversation reported in Foner (1993a, p. 16) was acquired by MUD user Robyn Kozierok who knew that Julia was a 'bot:

```
Robyn whispers, "how are you feeling?" to Julia.
Julia whispers, "Terrible."
Robyn whispers, "why, julia?" to Julia.
Julia whispers, "I have PMS today."
```

Since most players in the MUD do not realise that Julia is a Chatterbot, she is treated like any other potential female by the other players and hence she has many responses for dealing with subtle or blatant flirtation. Colin similarly has extensive code for dealing with swearing. Foner (1993a, p. 22) details a dialogue between a male MUD player and Julia. He prefaces it with:

> *He spent 13 days trying to get to first base with Julia, and it's not clear he ever figured out he was trying to pick up a robot (although he clearly suspects it at two points).*

Both Julia and Colin's natural language processing facilities are simple - more extensive but apparently less algorithmically sophisticated than those of Eliza. For example, Colin uses a brute force matching of likely patterns to produce a response (the '*'s represent any sequence of words or characters):

```
else if (MATCH (lcmsg, "*predict*weather*")   ||
         MATCH (lcmsg, "*what*weather*")      ||
         MATCH (lcmsg, "*how*s*weather*")     ||
         MATCH (lcmsg, "*how*z*weather*")     ||
         MATCH (lcmsg, "*what*forecast*")     ||
         MATCH (lcmsg, "*describe*forecast*") ||
         MATCH (lcmsg, "*describe*weather*"))
```

Figure 2.13 : Colin's brute force pattern matching

The fuzziness of meaning derived from the '*'s can adequately cater for many enquiries but can also lead to many non-sequitors or outright misleading responses from the Chatterbot. Since this would not be acceptable in an intelligent user interface, it is necessary to have less fuzziness through more specific patterns. This increases the domain knowledge accuracy of the interface but also increases how long it takes to accurately understand what the request was.

Verbots

Mauldin continued his foray into Chatterbot design with the construction of a new breed of Chatterbots called Verbots. These Verbots were not only Chatterbots, but also included virtual personalities with the use of 2½D graphics and synthesised voices, making the interaction experience very similar to that of interface agents discussed previously.

The Chatterbot design was very similar to that of Colin and Julia, but with the availability of end user customisation. With the help of tutorials (Plantec 1999a), users could write their own scripts for Verbots, giving them increased knowledge. This end user customisation was an improvement over the previous MUD agents, but there still existed one problem: not everyone had the ability to write their own scripts. The use of these scripts had shifted the hard-coded pattern-matching/response algorithm into the domain of users.

The ‘weather’ request processing equivalent to the Colin example of Figure 2.13, was given in Sylvie’s (the Verbot’s name) net file as a rule:

```
<id-59>
a:0.4
p:1 *weather*
p:1 *is it*cold*there*
p:1 *is it* snow*
p:1 *is it* rain*
p:1 *is it*warm*there*
p:1 *is it*hot*there*
r:The disks are warm and the IO channels are humming.
```

The ‘<id-59>’ is the name of a user-defined state so that state transitions can occur. The ‘a’ is an activation level - this supposedly gives user control over when the rule will fire. An activation level of 0.2 means that this rule should not fire at random. Values below 0.3 are very unlikely to fire without a pattern match and those above 0.2 may fire if Sylvie runs out of things to say. By carefully adjusting the activation level, Sylvie’s random comments or non-sequitors would not become too numerous. The multiple ‘p’ lines represent patterns to match, with ‘*’s representing any number of words or characters. The numeric value (1 in this case) represents a “comparative activation level” (Plantec 1999a).

So if Sylvie matched the user request to one of the patterns, then the pattern with the highest relative activation would produce the response given by the ‘r’ line. Not shown in this example is the mechanism for causing state transitions (and for preventing incorrect backward transitions). For example, it was often needed to add lines such as: ‘+<statexxx>’ and ‘-<stateyyy>’ (the x and y are irrelevant) so that Sylvie would increase the activation level on this response temporarily or would inhibit the firing of the rule in the second case. This tended to force the Verbot to make the required state transition and hence to have a limited memory of previous user requests or a context in which to supply an answer. For example, a user may ask "*do you have large pizzas?*" and then ask "*how much are they?*". The ‘*they*’ in the second request obviously refers to the large pizzas and the current state will help identify this.

In contrast to Mauldin’s expressed views on the Turing test - having mastered one domain, 50 experts can master 50 domains - the addition of new knowledge to Sylvie remained problematic. The more knowledge she had (or the more fuzzy patterns she had to match against), the more likely an incorrect or irrelevant response would be produced. Getting the activation levels correct was very difficult. However, through interaction with the Sylvie application, it could be seen that Sylvie was a believable talking head.

Whalen's Terminological Interface Prototyping System (TIPS) provided a question and answering type of interface for a Chatterbot, using a state transition network similar to Mauldin's, but to reveal a story rather than to negotiate a subject. By modelling a human being, he hoped that the user would be interested enough to pursue this story rather than enforce their own subject of interest.

Hutchen's MegaHAL was an exploration into the use of Markov Chaining to train MegaHAL about the order in which words appear in sentences. The Chatterbot could not follow a conversation model nor understand the meaning, but could receive an input sentence, take out a few keywords, and then try to construct a sentence based on the probabilities of the keywords and other words following each other (Hutchens & Adler 1998a). In this regard, it was similar to ELIZA.

Turing and the Loebner Contest

The Loebner Contest [HREF 36] is a formally run Turing Test [HREF 37] to determine if AI programs can be seen as indistinguishable from humans. Any program that is ranked comparable to or above humans by a panel of judges, has $100,000 and a gold medal (Loebner Prize [HREF 38]) awarded to the program's creators. No-one has been awarded this to date.

Each of the judges had a keyboard dialogue with each of the programs as well as with humans in some random order. The judges then gave each contestant a real-valued "*humanness*" score using the guide values of Table 2.2.

Score	Was your conversational partner a human or a machine?
0	Partner not accessible, or severe system malfunction
1	Definitely a machine
2	Probably a machine
3	Could be a machine or a human; undecided
4	Probably a human
5	Definitely a human

Table 2.2 : Loebner scoring legend

Robby Garner was the creator of the highest ranked AI contestant of the 1998 and 1999 Loebner Contest with his Chatterbot entries Albert1 and Albert2. These Chatterbots used Functional Response Emulation Device (FRED) technologies (Garner 1999a) and a theory for conversation called "Tight Sponge" (Garner 1999b). The new ground that Garner covered with FRED technologies was an implementation of machine learning to improve conversational ability. FRED used knowledge-based, machine learning and end-user approaches to produce some of the most sophisticated conversational abilities yet. The Tight Sponge technique was how the FRED Chatterbots manage to become more competent at conversation, actually learning as they converse (a sponge holds more in than it lets out).

The tight sponge technique followed these basic guidelines:

- A conversation is made of stimuli and responses (text)
- The stimulus is presented to the Chatterbot, and it replies with a matched response
- If a proper response cannot be matched to the stimulus, the stimulus is stored and a generic response is used
- The Chatterbot can then pose the stored stimulus to the user at a later date to receive a response to remember, or at a later stage, the user can teach the Chatterbot the right response

Garner used the best features of knowledge-based, end-user and machine learning approaches from software agents research. By starting off with knowledge, the Chatterbot did not have to be taught complex conversation, and this primary knowledge base was reduced by the fact that there was machine learning present. By using an end-user approach as well, the decisions that the machine learning techniques made could be double-checked to increase the functionality of this technology.

Another highly ranked contestant of the Loebner Prize was Alice [HREF 39], created by Richard Wallace. Alice used supervised learning with pattern-response dialogues being marked up using AIML [HREF 40]. Alice used roughly 41,000 conversational elements called categories. These categories held question-answer responses and used pattern matching techniques to decide which category best matched the user input.

Ella [HREF 41], created by Kevin Copple, was the highest ranked contestant in 2002. Ella used a similar technique of question-answer classification called “convuns”. She also learnt about user preferences through her conversations and would adjust her interaction behaviour accordingly.

This adaptive behaviour is also very important in a mentoring environment where different styles of mentee learning mean that the mentor must adopt different styles of presenting the material.

Table 2.3 shows the history of the highest ranked contestants of the Loebner Contest.

Year	**Creator**	**AI Program**
2005	Rollo Carpenter	George [HREF 42]
2004	Richard Wallace	Alice [HREF 39]
2003	Juergen Pirner	Jabberwock [HREF 43]
2002	Kevin Copple	Ella [HREF 44]
2000-2001	Richard Wallace	Alice [HREF 39]
1998-1999	Robby Garner	Albert One [HREF 45]

Table 2.3 : Loebner highest ranking contestants

Table 2.4 shows the judges’ ranking for the 2003 Loebner Contest. It is interesting to note that this ranks the human contestants as being between ‘Probably a human’ and ‘Could be a machine or a human; undecided’.

Rank	Entity	Contestant	Mean
1	Confederate 2	(Human)	3.867
2	Confederate 1	(Human)	3.667
3	Jabberwock	Juergen Pirner	1.928
4	Elbot	Fred Roberts	1.678
5=	Eugene Goostmann	Vladimir Veselov, Eugene Demchenko	1.644
5=	Jabberwacky	Rollo Carpenter	1.644
7	Lucy	Saskia van der Elst	1.378
8	Markbot	Mark Connell	1.311
9	ALICE	Richard Wallace	1.289
10	Gabber	Peter Neuendorffer	1.189

Table 2.4 : Loebner scores for 2003

It could be argued that these AI programs are simply targeted at appearing human at any cost, regardless of their usefulness. However, it has be countered (Harnad 1992a) that the solidity of the underlying research implied that different applications of the research would yield programs that can supply useful information to users, not just idle chit-chat.

Pattern Matching through Regular Expressions

Many of the Chatterbots used Regular Expression (RE) Pattern Matching. These systems used the generality of REs to cater for the complex syntax of natural language, and used the matching to supply an implied semantics.

The Jakarta Apache project [HREF 46] for Java Perl-5 REs (derived from the OROMatcher project [HREF 47]) indicates:

> *A regular expression is a pattern denoted by a sequence of symbols representing a state-machine or mini-program that is capable of matching particular sequences of characters.*

REs form the basis of lexical analysis and tokenisation where groups of symbols or lexemes are categorised to help in the parsing process (Aho *et al* 1986a; J E.F Friedl 1997a). For example, a series of digits such as ‘135’, can be categorised as an integer, a series of digits plus one decimal point such as ‘3.1415’ could be categorised as a floating point number, and a series of alphabetic characters followed by alphanumeric characters such as ‘ReWeight34’ could be categorised as an identifier for a particular computer language.

The power of REs comes from the use of meta-characters in the pattern that is used to categorise the input stream. When a Unix computer user types "`ls -l *.java`" they are using the ‘*’ to represent any filename. The concatenation of the ‘*’ and the ‘.java’ means that the user is interested in all files that have any filename ending with ‘.java’. This ‘*’ is a meta-character used by the user interface. Similar but more powerful meta-characters as well as rules such as concatenation exist in REs.

Because of the power of this technique, REs are now used in most pattern matching or string manipulation areas such as compiler design, and in user input processing. Many languages/utilities such as `vi`, `AWK`, `sed` and `Perl` also use REs.

REs are based upon the ideas of symbols, sets of symbols and rules to describe how these symbols interact. At a basic level, a match can be made against the symbols, a concatenation of symbols (i.e. one symbol must follow another), a choice of symbols (i.e. match against this or that or that) or a defined repetition of symbols (i.e. exactly 5 symbols,

one or more symbols, zero or one symbol). More complex RE systems such as Perl-5 REs have more meta-characters and rules to cater for the needed functionality of parsing complex entities. For more information on using Regular Expressions consult 'Mastering Regular Expressions' by J E Friedl & Oram (1997a).

The meta-characters and syntax of Perl-5 REs are shown in Figures 2.14a-c. A definitive reference can be found in Wall & Schwartz (1992a).

Tokens:
- any normal character matches itself
- a . matches everything except a newline (\n)
- a ^ is a null token matching the start of a string or line
- a $ is a null token matching the end of a string or line
- character classes and ranges (i.e. [abcd] and [A-Z])
  a ^ as the first character in the set inverts the match
- regular expression within parentheses

Figure 2.14a Regular Expressions: Atomic Tokens

Using the tokens of Figure 2.14a we could match against an exact word ('fred'), or match against the proper name at the start of the line ('^Fred'), or either upper/lower case name ('[Ff]red'), etc. A single digit could be represented by '[0-9]'. The expression 'fre.' would match against user input such as 'fred', 'free', "freA', or 'fre<'. That is, the '.' matches against any character other than a newline.

Figure 2.14b shows some common meta-characters. Some are shorthand ('\d' is shorthand for '[0-9]'), some are for consistency and convenience ('\n' is a newline), and some are for backreferencing ('\1').

Meta-character Tokens, backslashed characters:
- \d digit [0-9]
- \D non-digit [^0-9]
- \s a whitespace character [\t\n\r\f]
- \S a non-whitespace character [^ \t\n\r\f]
- \w word character [0-9a-z_A-Z]
- \W a non-word character [^0-9a-z_A-Z]
- \b null token matching a word boundary
- \B null token matching a boundary that is not a word boundary
- \n newline
- \r carriage return
- \t tab
- \f formfeed
- \xnn hexadecimal representation of character
- \cD matches the corresponding control character
- \nn or \nnn octal representation of character unless a backreference.
- \1, \2, \3, etc. match whatever the 1st, 2nd, 3rd parenthesised group matched.
  This is called a backreference. If there is no corresponding group,
  the number is interpreted as an octal representation of a character.
- \0 matches null character
- Any other backslashed character matches itself

Figure 2.14b Regular Expressions: Meta-character Atomic Tokens

The bracketing construct '(...)' remembers what was matched against the pattern inside the brackets. The match associated with the first set of brackets is remembered in a buffer that can be referenced by '\1', the second in '\2', etc. Backreferencing is simply referring back to another part of the match. It is important to note that a backreference matches whatever actually matched the sub-pattern in the string being examined, not the rules for that sub-pattern.

Quantifiers:

- `*` Match 0 or more times
- `+` Match 1 or more times
- `?` Match 0 or 1 time
- `{n}` Match exactly n times
- `{n,}` Match at least n times
- `{n,m}` Match at least n but not more than m times

Figure 2.14c Regular Expressions: Token Quantifiers

Any RE can be modified by a quantifier to indicate a repetition (Figure 2.14c). For example '`too*`' will match against any string that starts with 'to' and has zero or more instances of the letter 'o' after it. Another possible equivalent RE for this would be '`to+`'. Notice that '`to*`' will also match against the string 't' because it matches against the 't' followed by **zero** lots of 'o'. A match for the word 'colour' could be obtained via the RE '`colou?r`' which indicates that the letter 'u' is ignored for American spelling.

By default, a quantified sub-pattern is greedy. That is, it matches as many times as possible without causing the rest of the pattern not to match. For example, '`too*`' when tested against 'tooooooooo' will match against the longest number of 'o's not the shortest. To match against the minimum number, a '?' may be placed after the quantifier - '`too*?`'.

Finally, alternatives in REs use the '|' meta-character. To match against poor spelling or slang for the word 'what', the regular expression:

```
wat|what|wot|waht
```

would allow a match against either the word 'wat' or the word 'wot' or the word 'what' or the transposed spelling of 'waht'. A looser RE for this could be '`wh?(a|o)t|waht`'.

The more complex example of:

```
do\s*n('|o)?t
```

matches variations of the word 'dont' by including an arbitrary amount of white space - "`\s*`" - to match 'do not', or by allowing either an apostrophe or an 'o' - "('|o)" - to match 'don't' or 'do not'. Notice that this also matches the incorrect English 'do n't' and 'donot'.

The pattern below shows a moderately complicated regular expression to match a user asking for information about a 'widget'.

```
".*\b" +
"(?:" +
 "(?:(?:I\s*(?:need|want)\s*)?help\s*(?:me)?" +
   "\s*(?:about|with|on))|" +
 "(?:what\s*is)|" +
 "(?:where\s*" +
  "(?:" +
   "(?:can\s*I\s*(?:get|find" +
     "\s*(?:out\s*)?))|" +
   "(?:is\s*(?:the(?:re)?\s*)?)|" +
  ")" +
 "\s*info(?:rmation)?\s*(?:about|with|on))|" +
 "(?:tell\s*me\s*about)" +
")" +
"\s*(?:the|a|)\s*(?:(?:gtk\+?)?)" +
"(.*)(?:\s*widget)?.*"
```

Note that being overly general in the REs reduces the accuracy of the determined semantics. For example, to match the user question "What is vhml?", the RE "what\sis\svhml" could be used. Because the "what is" part could be specified in various ways, and because of spelling or typing errors, the RE could be better specified as:

```
(what|wot|waht|wat)((\si)|'|)s\svhml
```

The RE is now more general and matches more input formats from the user. In other cases however, the RE may be generalised too much - for example the specification of "what is" could be taken to the extreme and replaced by '.*' to match anything before the "is vhml" part. In this case, it would also match "where is vhml", "how much is vhml", etc and the determined semantic match is now very low. This balancing of syntactic matching versus accuracy of semantic determination is one of the major problems of systems that use REs.

Perl-5 regular expressions provide a complex, functional, pattern matching system that is well defined and well supported. Perl-5 regular expressions are well supported by the Java programming language.

More information about regular expressions can be found at http://www.regular-expressions.info/ [HREF 48] and http://search.cpan.org/dist/perl/pod/perlre.pod [HREF 49].

Semantic Analysis

Another form of NLP is through the **exact** understanding of the meaning and/or structure of the language. For example, Conceptual graphs [HREF 50] are

> *An abstract representation for logic with nodes called concepts and conceptual relations, linked together by arcs.*
>
> http://www.jfsowa.com/cg/cgstandw.htm

The research is based upon the work of Charles Sanders Peirce and can be used to store and represent natural language in a formal manner.

An exact understanding NLP system is composed of several, not necessarily separate, stages:

- Morphological or Lexical analysis: words and sentences are isolated and classified along with punctuation.
- Syntactic analysis: the structure of 'standard' sentences in the target language is used to verify that the sentence is valid. This is often complicated by the complexity of the target language structure. For example, the phrase "man eating shark" is syntactically valid but has two meanings (see Semantic analysis below). The structure of an English 'sentence' can be simplistically viewed as a 'subject' followed by a 'predicate', with a 'subject' being viewed as an 'article', followed by an 'adjective', followed by a 'noun'.
- Semantic analysis: The syntactically valid tokens are assigned semantics or meanings. If the semantics are ambiguous, the syntactic structure can sometimes help resolve the problem. The context of the above "man eating shark" phrase can sometimes disambiguate the meaning, although, two meanings still exist for the sentence "Come and see the man eating shark!" In this case, a larger textual context may have to be used to assign meaning.

• Discourse Integration: anaphora (Hirst 1981a) and ellipsis substitution needs to be done to augment the semantic analysis. With state based Dialogue Management systems, the anaphora and ellipsis would be disambiguated by the system being in the state requiring an answer to the previously asked question.

After the user query has been identified, a pragmatic analysis phase often follows. That is, an analysis of the meaning of the sentence in context: what needs to be done given the user input.

The Cyc system [HREF 51] described in Section 2.3.1.3 on page 63, follows these exacting steps in determining a user request or utterance. It also uses its knowledge base of common knowledge to accurately classify a user input if any remaining ambiguities exist. Note that this common knowledge can still not resolve the "Come and see the man eating shark!" ambiguity. Both meanings are still valid and can even cause confusion for humans.

Another effective semantic analysis technique that can be used within an exact NLP system is Latent Semantic Analysis (LSA) (Deerwester *et al* 1990a) :

> *Latent Semantic Analysis (LSA) is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text.* http://lsa.colorado.edu/whatis.html [HREF 52] (Landauer *et al* 1998b)

LSA (sometimes referred to as Latent Semantic Indexing due to its widespread use as a technique for indexing Web sites), addresses the issue of classifying user input by using the context cluster of words. That is, a word's semantic usage is determined by where it is used and where it is not used. These two sets can then be used to relate similar and dissimilar words **in context**.

Richardson & Smeaton (1995a, pg. 2), in their paper on information retrieval using WordNet (Miller, Beckwith *et al* 1990a; Miller, Fellbaum & Miller 1997a), indicated a similar technique for classifying words in context, and state:

> *Also by replacing direct string matching between index and query terms with a mechanism which can identify semantically similar terms, problems posed by the richness of natural language can be tackled.*

They used a semantic similarity function to aid in the matching of semantically similar terms. This is a vocabulary problem - users will try to retrieve information using a variety of words all with similar semantic meaning:

> *It is a troublesome impediment in computer interactions both simple (file access and command entry) and complex (database query and natural language dialog).*
>
> Furnas *et al* (1987a, pg. 1)

WordNet [HREF 53] is an English language semantic lexicon. One function of WordNet is that it clusters words into synonyms and provides links to other related clusters. This can help in solving two basic problems in NLP: synonymy and polysemy. **Synonyms** are different words with similar or identical meanings. **Polysemes** are words with multiple meanings. In this way, the clusters and their links can be used to classify a user query by matching through synonyms and disambiguating through polysemes.

For example, WordNet could be used either in constructing a better regular expression to match the user input: "`who\s*built\s*you`" could be automatically pre-processed into the more general "`who\s*(built|made|constructed|created)\s*you`". Or the user input could be pre-processed to alter a specific word:

```
who created you    who made you    who built you
```

These inputs could all be pre-processed into the semantic base synonym:

```
who built you
```

The first would be easier as it maps 1 into a well-defined 'many'. The second form maps an unknown 'many' into 1 and would require knowledge of WordNet's common synonym words.

Any developed mentoring system can benefit from a dynamic mineable source of synonyms and polysemes, so as to help in the semantic phase of language understanding.

Dialogue Management System Conclusions

The design and development of an **exact** semantic analysis NLP is beyond the scope of this thesis. However, the effectiveness of simple state-based pattern matching architectures such as Julia or Alice suggests that they will be sufficient for processing the user requests of a mentoring system. The robustness indicated by Rosé (1997a) may not be needed in this domain.

WordNet synonym clusters may aid in a more exact user input classification.

The functionality of a state-based pattern matching architecture would enable a dialogue to be entered into with a user, not just a series of unrelated questions and answers. The states would also handle context and provide mechanisms for resolving anaphora and ellipsis. The use of regular expressions would provide a flexible yet arbitrarily accurate pattern matching mechanism that is efficient and relatively simple.

Figure 2.15 shows a suitable schematic for a state-based pattern matching DM.

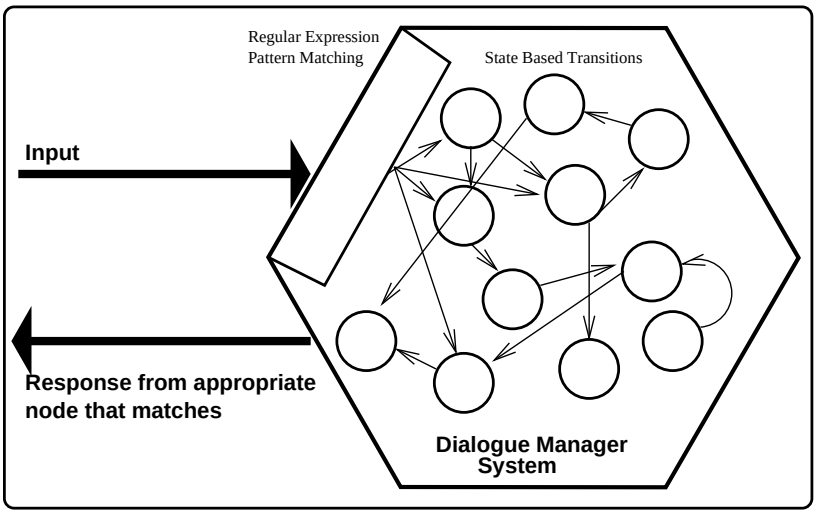


Figure 2.15 : Schematic of a state-based regular expression Dialogue Manager

2.3.1.3. Dialogue Management Knowledge Bases

There are several design issues surrounding the construction of a natural language knowledge base (KB) for a Dialogue Management system:

```
The size of the knowledge base
Extensibility of the knowledge base
How the natural language and knowledge base components interact
```

Two systems at opposite ends of the complexity spectrum are reviewed - Cyc [HREF 54] (Guha, Lenat *et al* 1990a; Lenat 1995a), and the COnversational MOdel for Database Access system (COMODA) (Whalen & Patrick 1990a; Patrick & Whalen 1992a).

Traditional Knowledge Bases and Expert Systems

The size of the KB to be constructed impacts upon the 'intelligence' of the system but also on its design time. For example, to construct a typical information system for COMODA, the design time just for the KB was predicted to be about three person months (Whalen 1996a p. 1). In contrast, the Cyc KB - nearly two hundred thousand terms and several dozen hand-entered assertions about or involving each term - has been developed over the last twenty years.

If a system can be designed to be extensible, then the 'intelligence' of the system can also increase over time. Extensibility can be achieved through three approaches: machine learning, end-user involvement, and expert knowledge. All these approaches have their advantages and disadvantages. Since extensibility allows a smaller initial KB to grow over time, the startup time for using the KB is reduced. For example, Cyc adds many new assertions to the KB by itself as a product of its expert knowledge and inferencing process. Cyc periodically examines the knowledge base "*looking for unexpected symmetries and asymmetries*" (Guha & Lenat 1990a pg. 53). Most of these are data entry errors but in rare cases "*these turn out to be genuine little discoveries of useful but hither-to unentered knowledge* " (Guha & Lenat 1990a pg. 53).

Another solution to this 'size-design time' issue would be to use an existing KB and to construct an interface between the natural language and knowledge base components. In this way, end-users can add to the KB.

The best choice for increasing interaction between the natural language and database components seems to be the use of an enhanced interface between the two components. This has been very successful for companies that have the technology and staff to complete such a task. Microsoft, for example, has developed an interface between their Server Query Language (SQL) software and English Query software (Microsoft 1998a).

The second best approach would be extending the KB to accommodate the natural language component, meaning a less complicated but larger task of reformatting and adding data to a database.

The Cyc KB sub-system is often referred to as a Knowledge Infrastructure in that the KB and the NLP (or Natural Language Understanding system in the Cyc nomenclature) are heavily interdependent. In the sentences on 'flying' in Section 2.3.1.1 on page 48, Cyc's

NLP semantic interpreter makes use of the KB. The semantic interpreter would use the KB to determine whether planes can fly and whether mountains can fly. The commonsense knowledge (assertions) in the KB can be used to disambiguate the object in the sentence.

> The Cyc KB uses the formal language CycL to represent
> *fundamental human knowledge: facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life*
> http://www.cyc.com/cyc/technology/whatiscyc_dir/whatsincyc [HREF 29]

The KB consists of terms, and assertions which relate those terms. Cyc is not a frame-based system, but instead uses the concepts of microtheories and contexts (J McCarthy 1987a) for inferencing, and which allow Cyc to entertain contradictory assertions.

Cyc's NLP has three main components which make use of the KB. They are: the lexicon, the syntactic parser, and the semantic interpreter.

The lexicon is the knowledge about English words - assertions that indicate whether the 'word' is a noun, verb, adjective or adverb and in what context (see Figure 2.16). OpenCyc has been combined with the WordNet lexicon. The syntactic parser uses a phrase based grammar to construct a number of possible syntactically correct interpretations of the user's input. For example, the word 'saw' in Figure 2.16 can be determined to be the verb not the noun. Finally the semantic interpreter uses the assertions in the KB to disambiguate the possible interpretations. For example, 'telescopes' are used for 'seeing' objects and hence the meaning of the last part of the sentence is disambiguated. This complex three stage process is quite robust in being able to accurately classify the user input.

Figure 2.16 : Cyc's sentence classification based on the lexicon (from http://www.cyc.com/cycdoc/course/nlu.html)

The COMODA system also subscribes to the approach of extending the KB to accommodate the natural language component. However, Whalen and Patrick have a different view on NLP. They proposed that a system does not have to understand the sentences that a user inputs, but instead the system can respond to the cues found in the input, resulting in an easier implementation (Whalen & Patrick 1990a).

Responding to cues can be as simple as pattern matching. For example, the user question "What was the budget for the 1991 Australian movie Proof" can be matched by ".* budget .* Proof" where '.*' is equivalent to zero or more lots of any character (except a newline). The cues in this case are 'budget' and 'Proof', and the information that the system retrieves from the KB and supplies to the user would reflect these.

In implementing their view on NLP, Whalen and Patrick used parse templates in conjunction with state transitions not unlike the Chatterbots Julia, Colin and TIPS (Whalen & Patrick 1990a p. 99). The DM starts the conversation (maybe answering a previous

question), the user then constructs a query for more information. The query can be in many different formats as the template's '.*' nature makes the matching flexible. The DM then changes state based on the template that was matched, and responds to the user with more information. The dialogue cycle then continues.

Using the above approach makes for a network of state transitions to all of the relevant data in the KB. An advantage of this approach is its simplicity and hence speed. Users can also pose questions in their natural language and according to questions already asked.

An example of these benefits could be seen by asking the question "Where are AC Milan playing next?" and then "Where are they on the Serie A ladder?" The questions are posed in natural language and notice that 'AC Milan' is left out of the second question because the user is asking it in the context of questions already asked. The use of states solve this anaphora/context problem and provides an "*economical" benefit to the speaker, at the expense of the listener having to draw inferences*" (P Cohen (1992a, pg. 2)).

To further refine question - answer matching, COMODA used three types of processing - the links from previous states were examined, the previous answers themselves were examined, and a global search of all state transition templates was done. The system then chose the best path through the KB based on this processing.

State-based DM systems are not restricted to using pattern matching. The Trindi DM system (Larsson, Bohlin *et al* 1999a) was also state based. In Trindi, a dialogue state was defined by *Name*, *Class*, and *State Function*. These were defined as:

> *the Name is simply a unique atomic identifier, the Class defines a place in an inheritance hierarchy permitting certain behaviours to be shared amongst states; and the State Function is arbitrary C-code, but it is required to define the next state for each execution path through it.*
>
> -- Trindi Consortium (2001a, pg. 162)

Larsson & Traum (2000a) proposed Trindi as an example of "best practice" in a DM system and hence it is indicated that these simple concepts are necessary in any developed DM system so as to enable the previously detailed advantages of a state-based architecture. However, one disadvantage of using a network of state transitions is that constructing and maintaining the KB is a laborious task. Figure 2.17 shows the growth of states if the KB-NLP system has to track the user dialogue.

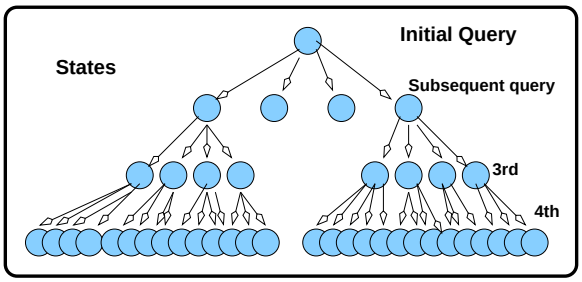


Figure 2.17 : Growth of states from a single query

The top node may represent the response to the initial query "Tell me about XXX". The KB-NLP may realise that 'XXX' is quite a broad topic and so may respond with "What aspects of XXX are you interested in?". It may then plan to recognise 4 potential states for

the conversation to move to (the second row of nodes). When the user then clarifies what aspect is of interest, the KB-NLP may give information that again may have several followup states (the third row of nodes). From one of these states, several transitions to other states may be possible, and so on. This exponential growth in states can place an upper limit on the usefulness of purely state-based architectures. However, it must be realised that the states form a **network** and not a **tree structure** and so judicious crafting of the states can 're-use' or 'transition to' previously developed states.

The generality obtained from the use of parse templates can also lead to a loss of relevance in the information supplied. Remember that in the parse template on page 64 that matched the movie question, the '.*' represents any text. So the question could just as easily have been "Was the budget Fool Proof", and since the KB-NLP system would assume that the question was "What was the budget for the 1991 Australian movie Proof", the response from the KB would have been irrelevant and wrong. Making the parse template more specific will lower the generality but increase the match accuracy and hence the response's relevance.

Since any mentoring system must give accurate and relevant information in response to a user query, it is important to determine if this limitation restricts the use of template based NLP to trivial environments, or whether the KB would need to be augmented in some way. Patrick and Whalen collected supporting evidence for the effectiveness of their parse template through field testing their COMODA system (Patrick & Whalen 1992a). They created an AIDS database that users could query, and collected statistics and user feedback.

User statistics:
```
There were 492 calls over 55 days (8 calls/day)
Average call length was 10 min
Average questions asked per caller was 27, median 12
56% of interactions were to browse for similar information (44% questions)
Questions averaged 5.7 words in length
```
System statistics:
```
57% correct answers
13% correct 'no information in database' response
10% wrong answer (answer in database)
12% wrong answer (answer not in database)
4% wrong "no information in database" response (answer in database)
4% ambiguous
```

Table 2.5 : User Statistics from COMODA testing (from Patrick & Whalen (1992a))

As can be seen in the call statistics, most users were after some specific information and were concise when trying to get it. The high browse percentage could be explained by the seemingly low 57% of right answers proffered (the user got information close to what they wanted and so browsed through the similar information until they found what they were looking for). The 13% of the correct use of the 'no information in database' response and 12% wrong answer ('answer not in database') suggest that the database was not as detailed as users were expecting. In fact 13% of suggestions from users was for more in-depth information in the database. Help systems should be as specific as possible, because users do not want to wade through non-relevant information (Kearsley 1998a).

Therefore it seems acceptable to use parse templates as long as a detailed KB is used and the templates are constructed in such a manner as to reduce ambiguous pattern matching of the user input. Again, the NLP and the KB are seen as highly interdependent.

Active Knowledge Bases

The hard-coded patterns of Colin and the inflexible 'net' files of Sylvie (see section on Chatterbots on page 52 and on Verbots on page 53) provide a static environment for pattern matching that severely limits the DM's ability to disambiguate user input. The pattern either matches or does not, and it is difficult to determine the degree of match to ascertain how accurate the match may have been. This is important since there may be many patterns that match, and a 'weight' can be used to determine and rank the **best** match.

The use of patterns also requires the enumeration of all the possibilities for a generic question. For example:

```
user input: do you like music?
user input: do you like paintings?
user input: do you like poetry?
```

The important aspect of this input is that it is asking the DM about its attitude to a number of subjects. In static pattern matching, this would require three separate patterns that enumerate music, painting and poetry. For efficiency, it would be more appropriate to be able to match the "do you like" as well as the subject and then to use some intelligent processing of the subject to formulate a correct response.

Some information is dynamic, as are online information sources such as Web pages, Frequently Asked Question lists, and databases. The data-mining[5] of these online sources can happen as the DM executes:

```
user input: what is the time?
user input: will it rain today?
```

These questions can be answered by the DM's knowledge of the time in the first case, and through data-mining a weather site in the second case. Both imply the need for a dynamic interdependent KB-NLP system rather than two static separate subsystems.

Finally, a mentoring system should not just respond, it should be pro-active. Therefore any system needs to be able to formulate an action plan (similar to Software Agents) so that it can help the user. Therefore the KB-NLP must be an active rather than a passive component of the DM system.

The Semantic Web

Web pages contain information that can be data-mined but they are designed for presentation of information rather than for the storage of easily searchable information. For example, the **structure** of Curtin University's Web page that contains information about

[5] As previously mentioned, the data-mining is concerned with the **extracting of existing information**.

important calendar dates for students, changes several times a year. The **information** in the Web page stays the same. But this changing of the structure makes it difficult for a software program to be able to extract the information for use in answering user questions such as "when is easter?" or "when does ramadan end?". As such, the Web pages represent a source of good information that is hard to use.

The Semantic Web (SW) (Berners-Lee & Fischetti 2000a; Berners-Lee 1998a) is an ambitious W3C project that is attempting to convert the Web into an environment that can both display information and store information for easy data-mining.

> *The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*
>
> -- Berners-Lee, Hendler & Lassila 2001a

Berners-Lee has said that the word “semantic” means “machine processable” not “natural language”. He has also said that the SW is “declarative” - the data has semantics and is not concerned with what is done with it (for example, how it is displayed). The Resource Description Framework (RDF) as well as XML is used as the basis for the SW.

If the SW project is successful, then the data-mining of online Web pages for dynamic information will become very simple and will increase the complexity of any KB-NLP system that can take advantage of it. This implies that the KB-NLP system should be an active rather than just a passive pattern matcher.

Knowledge Base Topics

The use of active software code to represent ‘topics’ of interest in a KB-NLP system is a good solution to the previously stated issues. By using simple topic-relevant pattern matching templates as part of an active code module, new topic KBs can be built quickly but with the added functionality of any necessary specific code (for example, disambiguation of user input, classification of user input, dynamic data-mining, hypermedia examples, or code execution). For example, consider the processing of the various user inputs:

```
user input: do you like music?
user input: do you like paintings?
user input: do you like poetry?
```

The code could pattern match with ‘do you like (.*)’, where the ‘(.*)’ means match zero or more lots of characters **and** remember what was matched. The code could then use the remembered match to work out if the user was asking about poetry, paintings or music, and respond as appropriate. If it could not finally classify the remembered match, then it could ask the user to clarify or rephrase what they had said.

In essence, the system would be an NLP front-end to an intelligent information knowledge base of topics. In execution, each active topic gets to process the user input by matching against it, and then providing a weight and a suitable response if the match was successful. The matching could be done via straight template pattern matching or by using the intelligence of the code. Adding more knowledge would be relatively simple: add more topics.

User Tailored Topics

To aid in extensibility/usability, it is important that a user can easily create and use their own topics. These could then be dynamically added to the KB-NLP so that when the user connects to the DM, it would also understand about the knowledge in the user's topics.

This implies that the user's view of a topic - the Application Programmers Interface - be well defined, and that the system supports the dynamic loading of user code. This last aspect implies that Java would be a useful programming language to use since it affords dynamic loading and also enforces a secure execution environment. This is needed because the user's code would be executed in the DM system and any unintentional or malicious code could bring the multi-user DM system down.

This extensibility through programming is feasible for computer literate users but would not be acceptable for the average user. Since the system would be used in a teaching environment, and few non-computing lecturers would have the programming skills to create Java topics, it is important that the system also support the creation of topics through a familiar point-and-click graphical user interface. These less-flexible topics could then be used as part of a generic KB-NLP sub-system. The use of Java also allows for this through the use of inheritance and polymorphism.

A significant disadvantage of using regular expressions for pattern matching is that topic creators have to be experienced with this technology. This may not be a problem for computer literate topic creators but the point-and-click GUI-based creator will still need to master the complexities of regular expressions. This does not detract from the use of this technology in this thesis since the research context is in the computing discipline. It does however limit the widespread applicability of the research until this issue is addressed.

One solution would be to create a sub-system that took multiple questions or queries that represented typical user input, and the system would automatically create an optimal regular expression that would match all of the inputs. This research is however beyond the scope of this thesis.

Dialogue Management Knowledge Bases Conclusions

This section has shown that the developers of both complex and simple KB systems agreed that combining the KB and NLP sub-systems was beneficial in the processing of user input. It was further shown that a state based pattern matching system was effective in answering the majority of user queries.

This type of system does have problems but these may be addressed by the use of active KB elements - patterns plus associated code - to help in such issues as disambiguation of user input, refining the classification of user input, and dynamic data-mining. Given the advent of the Semantic Web, this last aspect was seen as a way for the KB to grow by accessing existing online data.

The use of Java and Perl-5 regular expressions to create this active KB-NLP subsystem was seen to be possible and to have several advantages for the entire DM system. Figure 2.18 shows a typical schematic for such a KB-NLP subsystem.

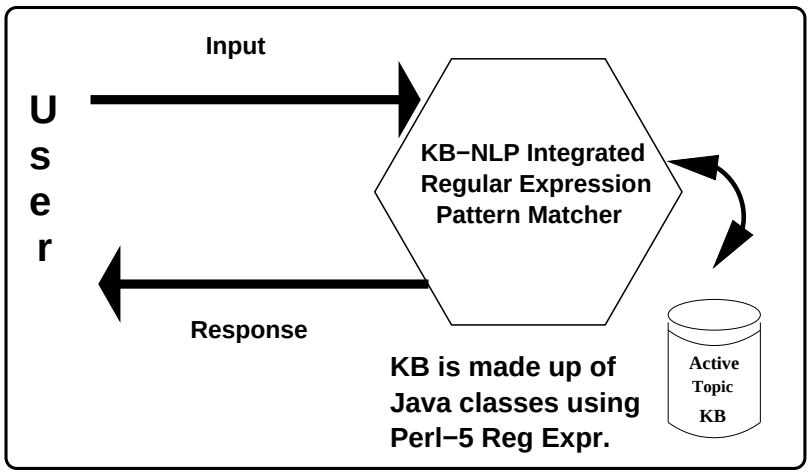


Figure 2.18 : A schematic for a suitable DM KB-NLP subsystem using active topics

2.3.1.4. Dialogue Management Markup

Anthropomorphic HCIs are developed under many different names - Embodied Character Agents (ECA), Embodied Conversational Agents, Virtual Conversational Characters, Virtual Humans, Talking Heads. These are characterised by a 2½D or 3D computer generated character often with some text-to-speech synthesis software, perhaps with a Dialogue Management system to handle user enquiries, and frequently using MPEG-4 technology (ISO/IEC 1998a).

The book '*MPEG-4 Facial Animation - The standard, implementations and applications*' (Pandzic & Forchheimer 2002a) is a good reference text for more information about this technology. The ECAs are applied in areas such as application helpers, Web page assistants or metaphors for entire Web sites, Virtual Information Providers, News Announcers and as avatars to represent users in Virtual Environments.

Currently there are many markup languages for controlling ECA's in these different tasks (Rist *et al* 2002a; Krenn 2002a) and for making them individuals (Ruttkay, Pelachaud *et al* 2003a; Pelachaud *et al* 2003a). Some of these are proprietary, some are not "open" for public use or scrutiny, and many are *ad hoc* (see comparison table from Arafa *et al* (2002a) broken into Figures 2.20 and 2.21). Many still lack the basic pre-requisites or criteria for a standard language:

- **Completeness**. The language must be complete or constructed in a way that is easy to expand.
- **Simplicity**. The language should be as simple as possible and exclude any ambiguous features. This would keep the language fairly small and comprehensive. Nevertheless, this should not affect the previous criterion. In order to fulfil this criterion, elements that have the same functionality should be merged.
- **Consistency**. The language must be consistent in order to make it easier for the user to learn, i.e. the syntax should follow a certain pattern. For example, the element names should be in the same form and have the same kind of attributes.

- **Intuitiveness**. The language should be intuitive, thus the user will not always need to consult the specification to be able to use the language. The names of the elements and attributes should be self-describing.
- **Abstraction**. The language should use a high abstraction level. That will make the language easier to understand and thus to use.
- **Usability**. The language should provide features that suit both beginners and advanced users.
- **Standardisation**. The language should adopt existing relevant standards where possible, for the different parts of the language. If an emerging standard is adopted for one part of the language, it is important that this part can be easily changed if the standard does not eventuate and this change must be transparent to users.
- **Evaluation**. The language should be evaluated with respect to the above criteria and effectiveness.

An ECA markup language - the Virtual Human Markup Language (VHML) - has been developed with these criteria in mind.

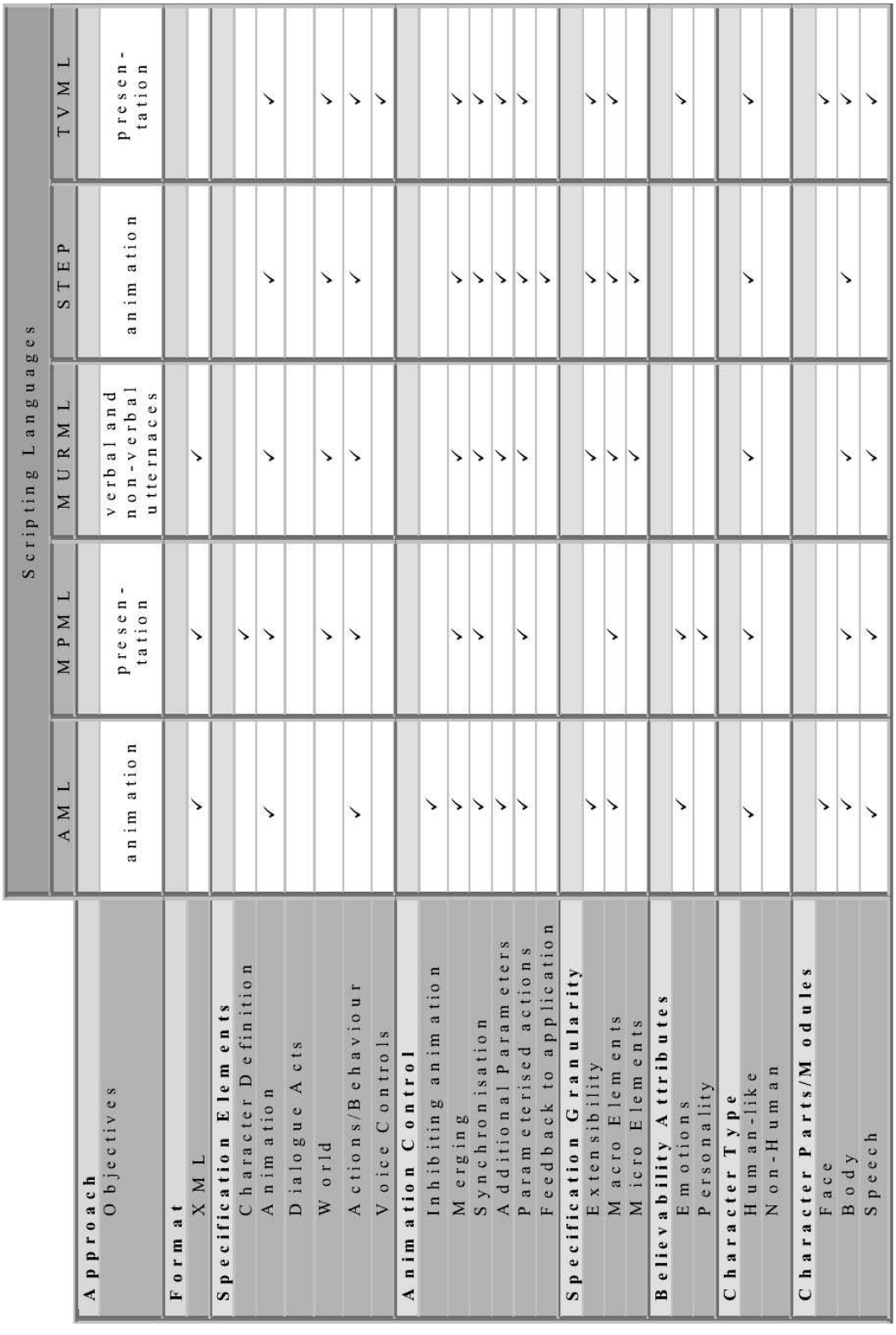
Virtual Human Markup Language

Although general in nature, the intent of VHML[6] is to facilitate the natural interaction of a Virtual Human with a user via a Web page or a stand-alone application. The text that the Virtual Human is to speak is marked up with tags that direct the emotions, gestures, speech, face and body of the Virtual Human. VHML uses existing standards and describes new languages to accommodate functionality that is not catered for. An example of text that uses VHML is shown in Figure 2.19.

```
<sad>
  You <emph>said</emph> to me once <pause length="short"/>
  that pathos left you unmoved, but that beauty,
  <emph affect="b" level="moderate">mere</emph> beauty,
  could fill your eyes with tears.
</sad>
```

Figure 2.19 : Text marked up using VHML

[6] VHML owes its existence to the work done primarily by John Stallo (Stallo 2000a). Quoc Huynh (Huynh 2000a) and Simon Beard (Beard 2002a; Beard & Reid 2002a) also contributed to its development. The current specification of VHML at www.vhml.org also owes a huge debt to Emma Wiknertz, Linda Strindlund and Camilla Gustavsson (Gustavsson *et al* 2002a) whose efforts and research made the language more solid, homogenous and complete.
The researcher's contribution has been in further discussion and development of the language and in overseeing its evolution.



	Scripting Languages									
	AML		MPML		MURML		STEP		TVML	
Approach										
Objectives	animation		presen-tation		verbal and non-verbal utternaces		animation		presen-tation	
Format										
XML	✓		✓		✓					
Specification Elements										
Character Definition			✓							
Animation	✓		✓		✓		✓		✓	
Dialogue Acts										
World			✓		✓		✓		✓	
Actions/Behaviour	✓		✓		✓		✓		✓	
Voice Controls									✓	
Animation Control										
Inhibiting animation	✓									
Merging	✓		✓		✓		✓		✓	
Synchronisation	✓		✓		✓		✓		✓	
Additional Parameters	✓				✓		✓		✓	
Parameterised actions	✓		✓		✓		✓		✓	
Feedback to application							✓			
Specification Granularity										
Extensibility	✓				✓		✓		✓	
Macro Elements	✓		✓		✓		✓		✓	
Micro Elements					✓		✓			
Believability Attributes										
Emotions	✓		✓						✓	
Personality			✓							
Character Type										
Human-like	✓		✓		✓		✓		✓	
Non-Human										
Character Parts/Modules										
Face	✓								✓	
Body	✓		✓		✓		✓		✓	
Speech	✓		✓		✓				✓	

Figure 2.20 : Comparison of Markup Languages (part 1)(from Arafa *et al* (2002a))

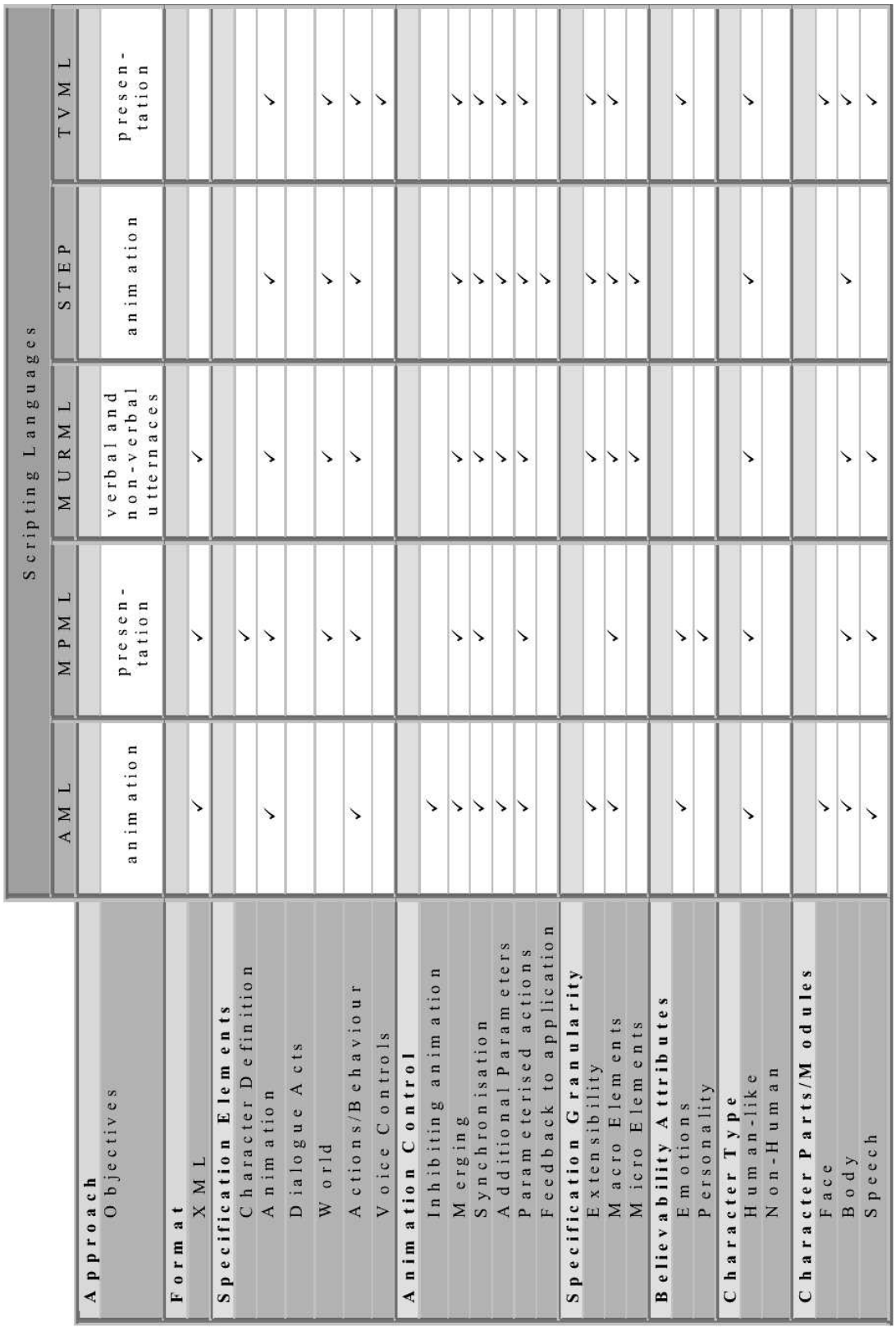


Figure 2.21 : Comparison of Markup Languages (part 2)(from Arafa *et al* (2002a))

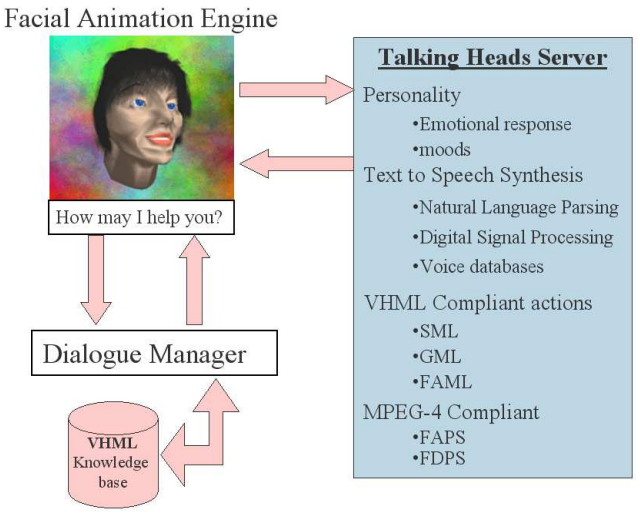


Figure 2.22 : Intelligent User Interface

In Figure 2.22, the information returned from the DM's Knowledge base should be consistently marked up in a format that it is easily parsed and categorised: an XML-based language - VHML. It should also be noted that the final 'rendering' and delivery of the marked up text is context / application dependant:

- a straight textual interface for a very low bandwidth interface,
- an interactive Web-based hyper- and multi-media display,
- a voice enabled system which lets the user hear the answer,
- a Talking Head system with complex facial gestures and voice which shows the personality or emotion of the Talking Head,
- an entire synthetic human figure with body language.

The information markup should stay the same but the way in which it is rendered should change dependent upon the form of output - textual, vocal, Talking Head, Body Language. A mentoring system should allow for this final translation before delivery.

The text shown in Figure 2.23 may be the knowledge base response to the user enquiry "What are you?" and has been marked up with a VHML sub-language - Dialogue Manager Markup Language (DMML). The DMML text is never seen outside the DM itself but allows for a consistent naming of relevant tags such as the user's name, sex, etc.

```
<first_name/>, <welcome/>. I am <mentorName/>.
I was developed by <mentorMaster/>.
<mentorDescription/>
<mentorPurpose/>
You can find out more about me from <mentorHomeURL/>.
```

Figure 2.23 : VHML response to "What are you?"

In a normal interactive session, the Talking Head client will (hopefully) know the name of the user. This name may be used to set a Dialogue Manager variable that can be used later in conversation. In this example, <**first_name**/> is the name of the user who is making the enquiry, <**welcome**/> is a language dependant greeting that has been set dependant

upon the user's home country or domain name, <**mentorName**/> is the name of the Dialogue Manager, etc. So the DMML response is converted by the DM to become the final plain text something like:

Freda, Guten Tag. I am Mentor. I was developed by Andrew Marriott....

The exact specification of DMML is still under development A mechanism has been developed that enables any DM static variables such as the DM's name or purpose, to have any atomic XML tag name as long as the definitions get processed before they are used.

The text in Figure 2.19 contains VHML tags that could be used to add speech, facial and emotional effects to the response. These tags/effect would need to be suppressed for a text only display but would add emotion to a voice and/or facial rendering.

VHML specifies emotions and actions to facilitate the direction of a Virtual Human interacting with a user. For example, a Virtual Human that has to give some bad news to the user - "I'm sorry Dave, I can't find that file you want" - may speak in a sad way, with a sorry face and with a bowed body stance.

In a similar way, a different message may be delivered with a happy voice, a smiley face and with a lively body. VHML tags such as <**smile**>, <**angry**>, <**look_down**> have been specified to produce the required vocal, facial and emotional actions.

Value of Emotion and Gestures in Rendering

Each day, humans deal with other humans even those with different languages, cultures and personalities. Hence users expect ECA-based applications to behave in a similar, familiar way. Often the lack of emotions or of a personality in the ECA removes that feeling of familiarity. Imagine the difference in 'feeling' engendered by the text in Figure 2.24 if rendered by a simple monotone ECA as opposed to a talented emotional actor:

I opened the drawer of my little desk and a single letter fell out, a letter from my mother, written in pencil, one of her last, with unfinished words and an implicit sense of her departure.
It's so curious: one can resist tears and "behave" very well in the hardest hours of grief. But then someone makes you a friendly sign behind a window... or one notices that a flower that was in bud only yesterday has suddenly blossomed... or a letter slips from a drawer... and everything collapses.
-- Letters from Colette. Sidonie-Gabrielle Colette.
(Obtained from Unix Fortune Database).

Figure 2.24 : Example of emotional text that can benefit from being marked up

When used as input to a Talking Head system, the simple addition of the VHML tag <**sad**> around the above text can cause the voice to become slower and reduce in frequency, and for the Talking Head to enact a more solemn-looking expression and perhaps, lower its head to indicate a sad contemplation of the text. Another culture may expect a different rendering to indicate <**sad**>.

With this emotional addition, the rendering of the quote becomes more affective and effective - the viewer empathises with the ECA, the viewer engages with the ECA. In general, these markup tags can make ECAs believable and hence a Virtual Lecturer or Virtual Distance Education Tutor is seen as being erudite and approachable.

Rendering

Notice that the <**mentorHomeURL**/> in Figure 2.23 would probably become a URL and hence the pure text may contain 'http://www...'. Therefore, in the rendering:

- A Web-based display may turn that into a link
- A vocal display may have to change the text into something like "You can find out more about me from the link www.blah".
- A Talking Head may say the above and also open up a new browser window.
- A Virtual Human may point to the link, etc.

XSL is a style sheet language designed to be used with XML data and documents to help in the different renderings. Unlike HTML, which defines the rendering or display behaviour for each of its elements, XML says absolutely nothing about how the data is to be displayed. XSL allows the author to apply formatting operations to XML elements. XSL is a language in which the DM author can indicate that the <**emph**> element should be ignored or should be rendered.

It must be noted that a VH is not normally viewed in isolation and hence there must be a way for the VH to interact not only with the user but also with its environment. The 'target' attribute of the VHML <**p**> tag can be used to specify that the rendering of the VHML data is directed to a specified target. It is up to the application to know the semantics and syntax of the specified target string. For example, the string could be used to indicate a different frame for a Web-based TH interface or it could be used to direct the data to different renderers in a complex application (Holic 2004a).

DM Markup Conclusions

Any developed software based mentoring system must model the normal human-human interaction and hence there is a need for an affective and engaging, emotional and culturally aware user interface. To ensure extensibility of any developed mentoring system into an ECA environment such as shown in Figure 2.25, the knowledge base information should be marked up with a suitable language that can foster these requirements. In this way, better interfaces using common anthropomorphic metaphors can be developed for information providing areas such as:

- Universities to use for Distance Education. Custom-built Virtual Lecturers can be used to help students understand the lesson, to provide one-on-one tutoring, to give accurate, consistent answers to queries.
- Web guides. Online exhibitions such as a museum or art gallery where the user can have a knowledgeable virtual 'Sister Wendy' or any other art critic. The guide has data-mined knowledge about the site, similar to a search engine, and can help the user at the site through a Natural Language Interface.

- Travelling. A virtual travel agent can provide information about accommodation and ticket arrangements. The travel agent can also book the travel on your behalf.
- Knowledgeable interactive companions for travellers, children, the old or the infirmed.
- E-commerce. The Virtual Salesperson can be a pleasant front end to the stock database, billing system, complaints department. It can respond to users in their language in a culturally dependent manner.
- Interactive Games. Avatars of the user or generated opponents.

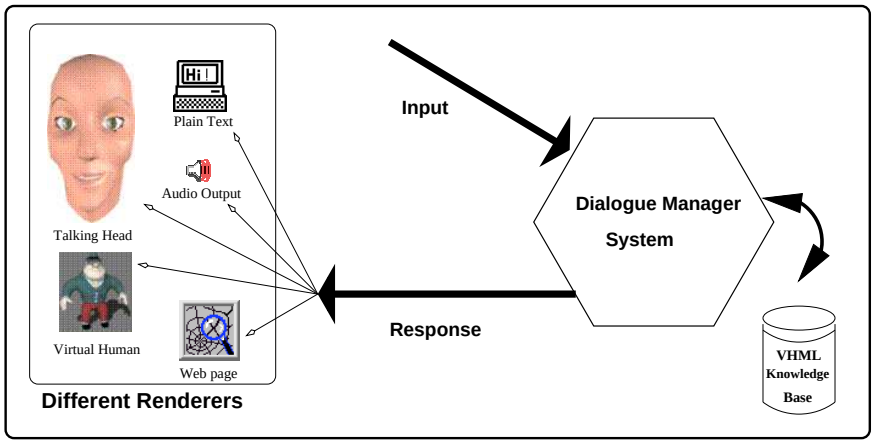


Figure 2.25 : Different Renderers enabled through a markup language

Many markup languages are being developed at research centres ‘in-house’ and they are all attempting to fill different application niches. As such, they are all affected by entirely different design decisions (Beard 2002a). VHML is an open specification available at the VHML website (www.vhml.org). Evaluation of VHML as a language for directing ECA’s is ongoing.

The uncertainties and problems of VHML have been discussed in Marriott & Stallo (2002a). Further information about VHML can be found in Marriott, Beard, Haddad *et al* (2001a), Marriott (2002a), Marriott, Beard, Stallo & Huynh (2001a), Marriott (2003a), Marriott & Beard (2002a), Marriott (2001a), Marriott (2002b), Marriott & Beard (2003a), Beard & Reid (2002a), Beard (2002a) and Gustavsson *et al* (2002a).

2.3.1.5. DM Conclusions

In any user interface, especially intelligent user interfaces such as query systems with or without a Talking Head or Virtual Human, users expect to be able to interact with it. Hence the system must have a Dialogue Manager that is capable of understanding the user’s natural language dialogues.

The DM must be pro-active not just reactive since this is what happens in normal dialogues. The DM must be able to ‘carry on a conversation’ with the user and not just answer individual questions. This implies a state based DM where the user dialogue moves the DM from state to state. As well, the DM must be able to understand anaphora references such as "where can I find **it**?" and be able to dereference the **it**.

The DM must also be capable of supplying hyper- and multi-media material and, if appropriate, of directing a VH to interact with its environment, pointing out other information such as diagrams, text, Web sites, etc.

Whilst not as accurate as a full semantic parsing, it has been seen that the use of Regular Expressions for classifying user input, suffices for most user queries (see section on Chatterbots on page 52 and on the “Turing and the Loebner Contest” on page 55). Regular Expressions provide a good balance between accuracy, generality and speed for handling bounded domain user queries.

The DM must have the relevant domain knowledge to be able to answer the user’s queries. It must be easy to build and maintain this domain knowledge and hence tools for non-programmers are needed to do this.

It has also been seen that the use of Intelligent Topics to represent the Knowledge base provides a number of advantages, especially with the advent of Web-based information that can be data-mined, and the introduction of the Semantic Web. It can be seen that any Dialogue Management system can benefit from a Knowledge base that is marked up in a consistent yet extensible manner.

Since humans interact with each other every day, any developed system must also act and react in this commonplace, believable fashion. Therefore the system must be capable of consistent, believable actions. In talking about Software Agents, W.L Johnson, J.W Rickel & Lester (2000a) and Lester & Stone (1997a) indicated aspects of these necessary actions:

- Situated liveness - the agent should continue to show believable behaviour, suitable to its environment, at all times.
- Complex behaviour patterns - the agent’s behaviour patterns must be complex so that behaviours do not appear obvious and mechanical.
- Natural unobtrusive behaviour - users can be distracted by blatantly unnatural behaviours.

This means that a consistent user model must be implemented. Until this model is capable of ‘real’ insightful thought, it is proposed that the emotional/humane component of the system’s actions can be produced through the marking up of the domain knowledge base. The Virtual Human Markup Language is a step in that direction.

Any developed mentoring system should implement a DM that addresses these issues.

2.3.2. Tutorial Dialogue Systems

2.3.2.1. Introduction

Motivation for using a hyper- and multi-media based mentoring system is that sound, graphics and plain speaking can convey ideas faster than technical documents. An individual can often present an idea in a ten-minute presentation that would otherwise require pages of formal documentation to describe (Bickmore *et al* 1998a). Ideally, when writing technical documents, three aspects must be balanced (the Aristotle's Triangle of Campbell (1995a)):

- Ethos - Character of speaker (personality)
- Pathos - Condition of audience (what the user wants)
- Logos - Persuasive element (information)

A software-based mentoring system could use personality to achieve ethos, an information-base for logos and a customisable interface for pathos.

However, the use of new educational technology in any form is not enough. The content of the units and its delivery pedagogy is paramount to ensuring that effective learning is taking place.

D K Cohen (1988a) argued against the educational view at that time: teaching was telling, knowledge was facts, and learning was recall. Cohen's comments were directed against didactic teaching. Many educationalists were looking at better ways to teach so that students would have better ways to learn (R C Schank 1993a; Brusilovsky 1995b; Eklund 1995a). More work has to be done if we are not to be seen as the era of 'teaching is writing HTML and learning is browsing'.

Research by McArthur *et al* (1993a) has suggested that an **intelligent** Web-based course will be more effective in helping students to acquire and retain information. Web-based courses must evolve to include more learning methods than hypermedia alone allows. Teaching is not telling and learning is not electronic page turning.

Intelligent Tutoring Systems (ITS) (Sleeman & Brown 1982a) evolved from AI and computer aided instruction (CAI) systems. ITS typically contain four components: the knowledge base or domain model, the student model, the teaching or learning pedagogy, and a user interface. ITS, as shown in Figure 2.26, have often used fill-in-the-form input rather than natural dialogue for the user interface, and have been driven by the ITS knowledge and pedagogy. These systems allow the student to be self-paced in their learning and are guided by a proven pedagogy, and benefit from having a student model for decision making.

As previously mentioned, many Web-based courses allowed active, self-paced, non-linear learning but needed to evolve to effectively address other pedagogical issues and techniques - see Section 2.2.3.2 on page 41. Intelligent Web pages can become an integrated system of information, plug-ins and Java applets that work together to produce an intelligent interface between the user and information. The traditional use of example programs has evolved into run-anywhere, in-the-browser hands-on learning. However, this small-scale limited use of applets does not indicate how Java may scale up for real world performance in a large system such as an ITS. At the start of this study, research in the ITS area had been limited to small scale, restricted environments. However, the literature indicated that ITS

were successful, and hence it was important to investigate the relevant issues that arose from the development and use of ITS so that these could guide the design of a software based mentoring system.

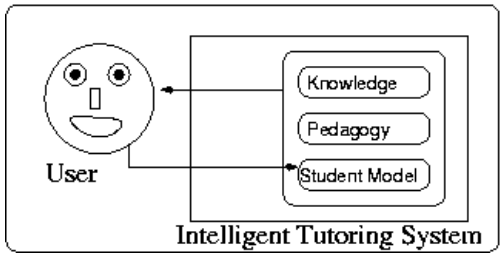


Figure 2.26 : Intelligent Tutoring Systems

2.3.2.2. ITS Issues

Core, Moore, Zinn & Wiemer-Hastings (2000a, pg. 1-2) indicated that for an ITS to imitate a good tutor it must support:

(1) unconstrained natural language input - other modes of input (menus, fill-in-the-blank forms) change the task from knowledge construction to correct answer recognition.

(2) "extended" tutoring strategies (i.e. strategies that unfold over multiple dialogue turns) - allowing tutors and students to co-construct explanations and allowing tutors to lead students through a line of reasoning point by point.

This would evolve the ITS of Figure 2.26 towards the Tutorial Dialogue System (TDS) shown in Figure 2.27, where the user interface is through natural language processing - a DM system - and there is extended interaction between the ITS modules. This requirement for 'dialogue' is also supported by Aleven & Koedinger (2000a).

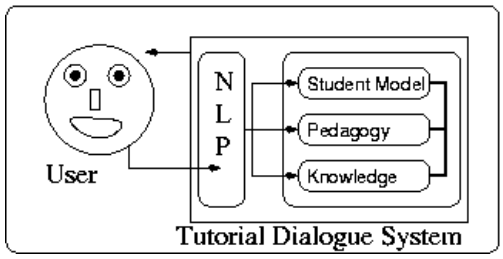


Figure 2.27 : A Tutorial Dialogue System

One advantage of ITS is their ability to "*lead students through a line of reasoning point by point*" (Core, Moore, Zinn & Wiemer-Hastings 2000a, pg. 2). This directed learning path guidance lends itself well to the mentoring process and any developed mentoring system should strongly support such a pedagogy, providing features which allow the creator of the educational strategy to exploit the learning benefits.

In discussing the work of Grosz & Sidner (1986a), Keim *et al* (1997a, pg. 1) indicated:

Tutorial dialogues are similar to the apprentice-expert or task-oriented dialogues in which the expert has complete knowledge of how to accomplish the task and guides the novice through the process.

Given the similarity of this process to mentoring, it is important to understand the limitations of TDS in helping students to learn. Keim *et al* (1997a, pg. 1) went on to say that

while TDS's "learning by doing" pedagogy is acceptable, "*it is certainly not the only way the expert/tutor might want to teach*". Graesser, Person & Magliano (1995a) and Core, Moore & Zinn (2001a) believed that it is the collaborative dialogue between the teacher and learner that promotes effective learning.

Aleven, Popescu & Koedinger (2001a, pg. 1) indicated that one limitation of ITS is that they teach

> *"at the problem-solving level", meaning that they provide assistance in the context of problem solving, but engage students only indirectly in thinking about the reasons behind the solution steps. They do not ask students to explain their reasoning, ...*

In this regard, they require the functionality of an Expert System. They continued their reasoning (Aleven, Popescu & Koedinger 2001b; 2002a), and indicated that this "self-explanation" can be effective but that the current systems did not interact with the student using natural language. "*It is plausible that students will learn even better when they explain in their own words.*" (Aleven, Popescu & Koedinger 2001b, pg. 1).

Whilst it is seen as imperative for any developed mentoring system to be able to communicate with the user via natural language - enabling the collaborative dialogue between the teacher and learner - the Dialogue Managers of Section 2.3.1 on page 48 may not be capable of the level of understanding required to support this self explanation of the student. Hence, this self explanation functionality may not be feasible in the developed mentoring system.

Core, Moore, Zinn & Wiemer-Hastings (2000a, pg. 2) also stated that to imitate a good tutor and hence to enable "*unconstrained natural language input*" and "*extended tutoring strategies*" a computer tutor must be able to deal with:

1. failure
(i) the tutor may not understand a student response;
(ii) the student may answer a tutor question in an unexpected manner; and
(iii) the tutor's teaching tactic may not be working.
2. interruption - the student may interrupt with a question
3. the need to revise tactics - a student may skip steps in an explanation.
4. the need to disambiguate student meaning.

Item 1(i), Item 1(ii), Item 2 and Item 4 imply the need for a DM system that can classify both initial user input as well as user input in the subsequent dialogue. That is, it must be able to understand questions or statements at any point in the dialogue regardless of whether they are part of the ITS's tutoring strategy. It must also be able to understand the current state of the dialogue as well as anaphora (see explanation on page 35). These last two requirements often conflict: how long should anaphora information be retained in a dialogue that is punctuated by unrelated user input?

Item 4 also implies that the system should be able to determine 'intent' of the user input. This is difficult to do for human tutors and it may be beyond the capabilities of any developed mentoring system although the use of a state based DM may help.

Item 1(iii) implies two issues. Firstly, it indicates that the DM must be able to determine when it does not understand the user input and be able to use a different pedagogy or some other mechanism to prevent student disillusionment or confusion. One typical mechanism would be to indicate that the DM did not understand the input and ask to have it rephrased. Or another would be to indicate that it did not understand but to volunteer some context relevant information that may move the dialogue in a slightly different direction.

Since the proposed mentoring system is not a hard-wired ITS with only one strategy and one pedagogy, but instead a loose framework to support different strategies and pedagogies, Item 3 is not seen as being within its domain. However, any developed system framework must support the needed functionality so that the strategy developer can implement this as required. A state based DM should support this, and so too should any system strategy-development tools. For example, as shown in Figure 2.28, it should be possible for the system to jump from state S2 to state S12 based upon some input that causes steps in the strategy to be skipped. Or even for a state transition from S5 to S7 which may represent a different tutoring approach.

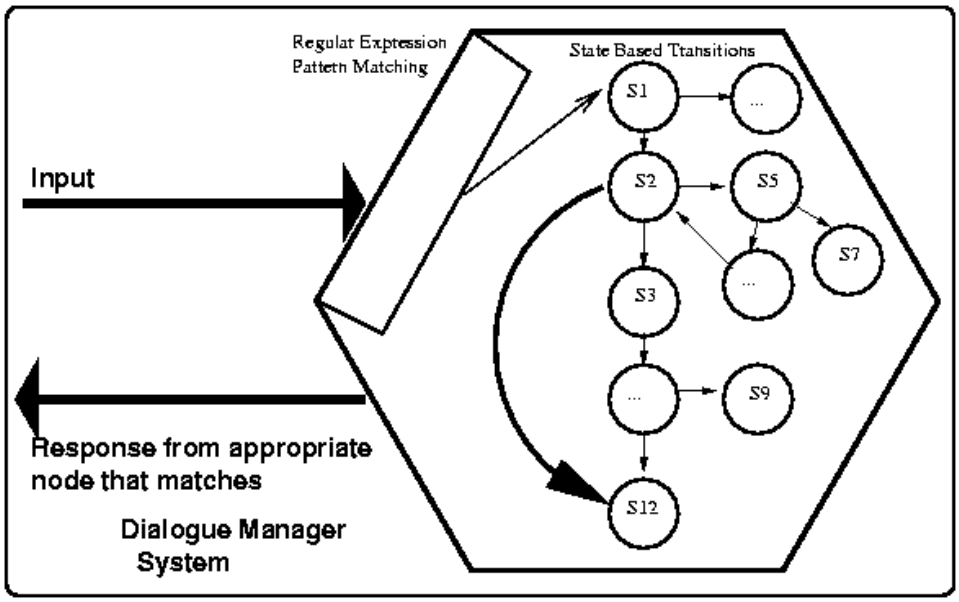


Figure 2.28 : Support in a state-based system for revising a tutoring strategy

Even though Core, Moore, Zinn & Wiemer-Hastings (2000a, pg. 2) talked about "*allowing tutors and students to co-construct explanations*", Item 1(ii) and Item 2 implies that they have not seen the tutor and the student as equals with regards to the dialogue. It is obvious that a student may not answer correctly. It is also obvious that students may interrupt with a *non sequitur*. Hence the dialogue may have unexpected inputs. However, it is also important for any developed system to support pro-active output: an unexpected question, or an interrupting statement to the **student**. For example, "Have you started your assignment yet?" or the hint: "Did you know that http://xxxx/ contains good information about yyy?". The system must then be able to enter into a dialogue with the student based upon this pro-active statement.

Zhou *et al* (1999a, pg. 1), when discussing improvements to the CIRCSIM-Tutor Project, indicated:

> *Hinting is an important tutoring tactic in one-on-one tutoring, used when the tutor needs to respond to an unexpected answer from the student. To issue a follow-up hint that is pedagogically helpful and conversationally smooth, the tutor needs to suit the hinting strategy to the student's need while making the strategy fit the high level tutoring plan and the tutoring context.*

This indicates that context sensitive hints may be very beneficial in any developed mentoring system. Their observation was again based on the tutor-driven teaching pedagogy, and hence their observation was concerned with "unexpected answers". In a mentoring system, the hint may be pro-actively given at any time as part of a learning strategy for a specific unit in computing, or for the computing discipline as a whole.

Several researchers have developed dialogue systems that use hints (B Woolf 1992a; Reiser *et al* 1992a; Anderson *et al* 1995a; Gertner *et al* 1998a), whilst the research of Hume (1995a) provided an understanding of how to implement the hints that human tutors use effectively, into a dialogue system (Hume *et al* 1993a; 1995a; 1995b; 1996a).

Hume *et al* (1996a, pg. 4) identified 5 types of hints:

- **convey information hints**: the tutor provides information to the student and prompts them to answer.
- **point to information hints**: the tutor suggests the availability of information or points to it but does not provide that information.
- **directed line of reasoning**: the tutor asks a sequence of very focused questions "*designed to illustrate a concept or a way of step-wise reasoning about a phenomenon*" (pg. 4)
- **explanation**.
- **summary**.

The last two are on the passive end of a "*continuum spanning passive to active learning*", with the "directed line of reasoning" lying near the middle, encouraging investigation by the student. "Convey information" and "point to information" hints require the student to engage in active learning.

The first three types of hints seem well suited to a mentoring process, especially since they encourage active learning. Item three is very relevant to the domain of computing, with analysis and design being typical step-wise processes. Therefore, any developed mentoring system should enable a hinting mechanism so that educational strategy designers can take advantage of it. It is seen as important that this hinting could also be pro-active in nature.

Zhou *et al* (1999a, pg. 2) also indicated their rules for generating hints:

> *First give evoking terms or synonyms.*
> *Otherwise try to give an intermediate step.*

That is, try to promote reflective, active learning by the student, or try to push them a small step forward along the learning path. They also indicated that simple linguistic hints such as responding with "*And?*" can prompt the student to think more about their answer.

Finally, Zhou *et al* (1999a, pg. 5) indicated that it was important to keep track of hints to "*avoid repeating the same hint and to make sure that the hints do not return to a causal relationship that was already tutored if there are several hints in a row*". A state based DM, along with a history mechanism, can control this issue in the same manner as it controls normal 'repeat' or circular problems in dialogues.

The requirement for the hints of Hume *et al* (1996a, pg. 4) can pose a problem for naive ITS because in promoting active learning, you are also encouraging the student to take the initiative, and this can mean that they will ask questions that are not catered for in the strict ITS tutoring strategy. Keim *et al* (1997a, pg. 1) indicated that a "*successful tutorial dialogue system must support mixed-initiative, allowing both participants to control the discussion at different points*". Core, Moore & Zinn (2001a, pg. 1) indicated that good tutors:

> *maintain a delicate balance allowing students to do as much work as possible and to maintain a feeling of control, while providing students with enough guidance to keep them from becoming too frustrated or confused.*

These two observations imply that any developed system should not follow the strict 'tutor leading' strategy of many ITS. Instead it again implies that the tutor and the student can "*co-construct explanations*". It also implies that students should be free to ask any question at any time and get a relevant, perhaps context sensitive answer. Freedman (1997a, pg. 5) supported this but also suggested that a system that gives initiative to the student:

> • *Does not a require deep understanding of the student's (possibly buggy) domain model.*
> • *Does not require reasoning about the student's plan.*
> • *Does not require the use of stacked discourse contexts, as would be required, for example if the student asked a hypothetical question.*

Freedman (1997a, pg. 5) continued by indicating that: "*most student statements can be handled without needing to reason about the student's plan*".

Keim *et al* (1997a, pg. 2) also indicated that "*One can imagine a pedagogy in which more or less initiative is granted to the student, depending on the tutor's perception of the student's progress*". This implies that mixed-initiative dialogues may be more effective if they can automatically adapt to a student's learning needs and style.

2.3.2.3. Managing the Dialogue

Similar to Core, Moore, Zinn & Wiemer-Hastings (2000a, pg. 2), Zinn *et al* (2002a, pg. 1) indicated that a tutorial dialogue system must:

> *(1) understand student utterances well enough to respond appropriately;*
> *(2) not ignore student confusion;*
> *(3) encourage the student to recognise and correct their own errors;*
> *(4) abandon questions that are no longer relevant;*
> *(5) handle multiple student actions in one turn; and*
> *(6) deal with student-initiated topic changes.*

They argued that (1) required a "*reasonably well performing input understanding engine*". That is, a Dialogue Management system such as reported in the section on using Pattern Matching for Dialogue Management on page 57.

Since they were concerned with ensuring the integrity and flow of their dialogue plan, they continued that (2)-(6) required that the system must monitor the execution of the dialogue strategies in case of failure. However, they concluded that there was no reason to generate elaborate discourse plans in advance, sketchy plans that can adapt would suffice.

Given that a mentoring system may have separate discourse plans for different units that a student is enrolled for, it is important that the plan is always relevant and can adapt as the user's needs change. It is also important that any directed plan have the 'teacher as a guide' and hence this must be configurable or programmable by the teacher for each unit. This implies that the system must form a framework that can be used for different strategies and plans, and not just hard-wire in one or two specific educational paradigms for managing the course of the dialogue.

Of interest is that Zinn *et al* (2002a) did not consider pro-activeness - dialogue initiated by the DM - as a requirement of a tutorial dialogue system, although this mixed initiative may be implicit in their discussion - item (6). Given that most dialogues in a learning environment are started by the teacher, for whatever reason, any developed mentoring system should support pro-active dialogues with the user.

Zinn *et al* (2002a) considered the different models for dialogue management: "finite state machine", "form-filling" and "dialogue components". Their historical analysis failed to evolve the "dialogue components" model into the "intelligent agent" model of McTear (2002a) although, through discussion of the advantages and disadvantages of each model, they derived an effective 3 tier model.

It is proposed that a state-based and intelligent component-based architecture may address many of the ITS and TDS issues that have been raised in this review of the extant literature. However, in designing and developing such a system, it is important to note that one of the disadvantages of these intelligent systems is that "*they are less robust and require more resources and more complex processing than finite-based and frame-based systems*" (Hjalmarsson (2002a, pg. 15)). In a "finite-based" system, all questions are predetermined.

Of importance is that the component modules be intelligent in that they can do independent processing of the user input and state, rather than being just hard-wired as in many ITS. The modular construction of the system can allow for extensibility of content and functionality as well as allow different tutoring strategies to be overlaid on the system by different lecturers on a per-unit basis.

Although it is not proposed that the developed mentoring system will be a generic educational paradigm testing framework, it is necessary that it support many of the effective paradigms reported in the ITS and TDS literature. The modular approach will allow several paradigms to be supported initially, and the system's extensibility through modules will allow an experienced programmer to provide support for any new paradigms that may need to be used or tested in the future.

The system must support various educational tutoring (or mentoring) strategies that may be required by the course planner. The evolving nature of sources of information, as well as the power of desktop computing has changed the way students can learn and hence has also changed the educational paradigm possibilities that can be offered to students.

2.3.2.4. Educational Paradigm

There are many different ways for a student to learn - see www.emtech.net [HREF 55] for various paradigms on teaching and learning. Table 2.6 (adapted from Reinhardt (1995a) who paraphrased Means *et al* (1993a)) indicates how the philosophy underpinning education has evolved to meet new teaching and learning challenges, and especially to incorporate the opportunities provided by the technology of ubiquitous cheap computing.

CHANGING EDUCATIONAL PARADIGMS		
OLD MODEL	**NEW MODEL**	**TECHNOLOGY IMPLICATIONS**
Classroom lectures	Individual exploration	Networked PCs with access to information
Omniscient teacher	Teacher as guide	Relies on access to experts over network
Individual work	Team learning	Benefits from collaborative tools and E-mail
Passive absorption	Apprenticeship	Requires skills development and simulations
Stable content	Fast-changing content	Requires networks and publishing tools
Homogeneity	Diversity	Requires a variety of access tools and methods

Table 2.6 : Changing Educational Paradigms (adapted from Reinhardt (1995a))

Tutorial Dialogue Systems (TDS) encourage aspects of the ‘New Model’ outlined in Table 2.6, such as:

Individual exploration

One of the most important outcomes of a good educational system is “*the cultivation of individual initiative in students*” (R Schank & Cleary 1994a, node. 43). In this respect, any developed mentoring system must encourage exploration by the user, as well as provide rich information resources. Currently, intelligent Web pages assist in this individual exploration, but so too could a network-aware mentoring system.

Teacher as guide

Totally unstructured learning through exploration is not time effective, especially if the information sources are from the Internet, as most of that information has not been educationally validated. Historically, the guiding hand in this exploration has been either from peers or from an expert.

Vygotsky (1978a, p. 86) defined his Zone of Proximal Development theory as:

> *the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers*

This concept, together with his Social Development Theory, proposed that people learn best through imitative and instructed learning. That is, he believed that people can learn new knowledge under guidance that could not be learnt by themselves. It is therefore

important for a mentoring system to support a learning paradigm that provides direction or a guiding hand to assist learning.

Team learning

Similarly, the Social Learning Theory of Bandura (1977a) indicated that observing the behaviour of others may help a student re-use that behaviour in their own learning. One disadvantage of this is that there may be no guarantee that the learning behaviour of others is correct, or that it may be a correct learning behaviour for that student - a 'learning by doing' student may not benefit from the 'learning by listening' behaviour of his/her peers.

Pea (1993a) indicated that learning by collaboration with peers was beneficial and Collins & Brown (1988a) indicated that computer collaboration may hold promise. Greer, McCalla, Kumar *et al* (1997a) reported on success in this area with the PHelpS system, and on an Intelligent IntraNet Peer Help Desk (Greer, McCalla, Cooke *et al* 1998a). Soller *et al* (1999a) also suggested strategies for promoting effective peer interaction.

Baecker (1991a) suggested that the new paradigm of computer collaboration extends beyond educational technology to the entire field of computer science. Harasim (1993a) described the benefits associated with human collaboration with the machine acting as a mediating agent. Baecker (1991a, p. 4) also indicated that "*real collaborative work moves smoothly back and forth between synchronous, real-time interaction and asynchronous, off-line activity.*"

Any developed mentoring system should facilitate team learning and be cognisant of the individual learning styles of team members. The system should also provide the "*asynchronous, off-line activity*": students learn from the mentoring system in their own time and at their own pace.

Apprenticeship

Lave & Wenger (1991a) also agreed on the need for collaboration and indicated that this learning was most beneficial if done in the context of the problem (Situated Learning). Therefore this collaboration between expert and novice should occur in the context of the problem not in a lecture or tutorial. A software-based mentoring system would be particularly suitable in an information rich computer environment.

J S Brown *et al* (1989a, p. 39) use Situated Learning and stated:

> *Cognitive apprenticeship supports learning in a domain by enabling students to acquire, develop, and use cognitive tools in authentic domain activity.*

They also indicated that this cognitive apprenticeship, coupled with craft apprenticeship - in essence mentoring - can produce very capable graduates:

> *Similarly, craft apprenticeship enables apprentices to acquire and develop the tools and skills of their craft through authentic work at and membership in their trade. Through this process, apprentices enter the culture of practice.*

Patel *et al* (2000a) and Patel *et al* (2002a) have applied cognitive apprenticeship to intelligent learning systems to develop a system that was found to have significantly improved student learning performance.

Fast-changing content

Although discussing a course on the ergonomics of information systems, the views of Holt *et al* (1995a, pg. 7) are relevant to many modern computer-mediated disciplines:

> *For rapidly evolving disciplines, a computer-based learning system combined with structured hypertext can provide rapid access to the most recent empirical and theoretical results. Together these advantages suggest open learning with computer mediated communications is a pragmatic approach for a course .....*

The "structured hypertext" could be Web pages, legacy databases in a known format, or even the Semantic Web: these can be data-mined by any developed mentoring system to provide up-to-date relevant information from validated sources.

Diversity

Modern educational environments no longer rely on a single media for teaching. The use of hyper- and multi-media sources of information, as well as executable applications such as Java applets, enrich the learning experience. Figure 2.29 shows the different sources of information that any developed mentoring system must be able to utilise for gathering and presenting information to a user.

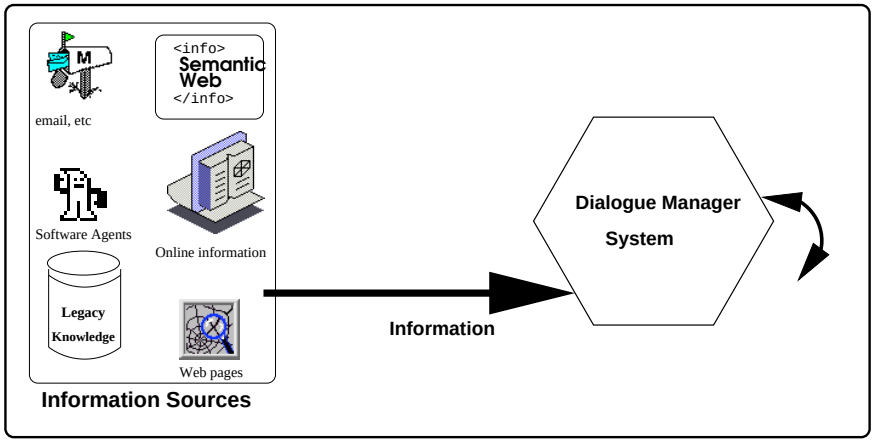


Figure 2.29 : Different information sources can be utilised by a mentoring system

As can be seen in Figure 2.25 on page 77, any developed mentoring system must also be able to present the material to the user in a number of different formats dependant on the user's environment.

2.3.2.5. Tutorial Dialogue Systems Conclusion

In conclusion, it can be seen that research into ITS and TDS offers insights into the design and features required of a software based mentoring system. Paramount is the need for a reliable, state-based DM, coupled with support for different educational paradigms that assist through such mechanisms as directed learning paths, hints, mixed initiative dialogues, and pro-activeness. The system should also support the use of information from different sources, especially given the ubiquitousness of Web sources.

Situated learning, constructed in a step by step sequence, and guided by expert static and dynamic data-mined knowledge would seem to be a useful paradigm for helping students to learn. The mentoring system framework should therefore enable this TDS educational paradigm.

The Java programming language would address the requirement of an intelligent component-based system that is modular. It supports all necessary programming and functionality requirements for the design and development of a mentoring system.

The development, implementation and analysis of a software-based mentoring system would provide data to indicate whether Java scales well from small systems to real-world teaching environments.

2.3.3. A Software Based Mentoring System

Bryan & Gershman (2000a) drew attention to the two predominant metaphors of Web browser interaction: browsing and searching. Browsing is the leisurely navigation from hypertext document to hypertext document along an arbitrary path. Searching is goal oriented where the user tries to find specific information. But the Web is vast and complex, and this complexity means that building the correct search query takes time and expertise. Of importance is that humans in their interaction have special ways of dealing with day-to-day complexity: ask a friend!

In a similar way, discussing a problem with a more experienced colleague may help, since their expertise and problem solving skills can often highlight errors or misconceptions. Given this, it follows that this interaction could also be beneficial in the introductory analysis and design (and even execution) phases of the problem solving.

If a user could interact in a natural way with a software program that embodied experience, skills and expertise, and if the user could experience the same benefits from this system as from a real colleague, then the system would be effective as a mentor to the user.

2.3.3.1. Requirements

The literature from the parent and research theories areas has indicated that any developed system should have the following attributes/functionality:

- Relevant teaching and learning strategy. That is, the system should be able to help the user with relevant and timely information in a structured manner that is beneficial to the user. It should help the student learn not just answer questions.
- Proactiveness. The system should not just provide information but should also actively guide the user, through the asking of questions or in the offering of advice.
- Effectiveness in helping users not only in learning but also in other daily tasks and problems. The information given to users must be relevant and also effective in helping them solve their problems.
- Expertise. That is, the system should have relevant domain knowledge so as to help users. This expertise should also be seen as coming from an authoritative source.

- Adaptability. The system must be able to adapt its interaction behaviour for different users. It should also be able to adapt the level and amount of interaction.
- Extensibility. The system should be able to handle realistic numbers of users as well as different interaction paradigms.
  The user should also be able to extend the system so as to augment its facilities.
  The system should be flexible enough that the response to a user's query can be displayed as text or sent to a Web page or even spoken.
- Systemic in its data uptake as well as in its usage. That is, the system should be able to take advantage of the available information in the user's environment and should be able to be used in many different tasks within that environment.

2.3.3.2. Limitations

The literature from the parent and research theories areas has also indicated that any developed system:

- Need not have a student model at its core driving the interaction. Although justified by the research into Tutorial Dialogue systems, the scope of such a system is beyond this thesis. It should also be noted that the system is not just concerned with student learning but **mentoring**.
- Need not be an Expert system that uses frame-based reasoning. The literature has shown that a state based dialogue system with expert knowledge should suffice.
- Need not be Agent based but could take advantage of relevant Agent techniques especially in the area of Interface Agents and Intelligent User interfaces.
- Need not be a general framework for testing arbitrary educational paradigms. Such a system is beyond the scope of the thesis and does not add to the overall research evaluation. However, the system should be flexible enough to support several paradigms since students learn in different ways.
- Could be text based only. The system should however, be capable of using hyper- and multi-media material either directly or indirectly through an external browser.
  Given the emerging multi-modal nature of HCI, the system should be designed in such a way that different output renderers could be used. In this way, the system could also interact with the user through a Talking Head Virtual Lecturer.

2.3.3.3. Software Based Mentoring System Conclusion

Figure 2.30 shows the proposed mentoring system based on the preceding requirements and limitations. The system should be extensible, adaptable and should support various learning paradigms. It is proposed that the system should be effective, and beneficial to students in helping them in their university courses.

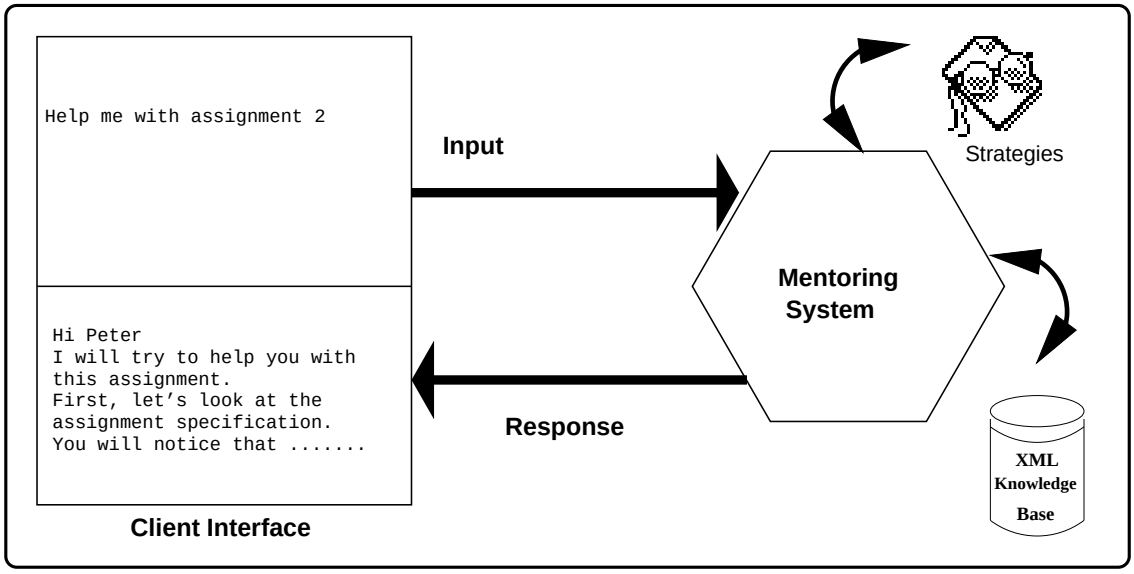


Figure 2.30 : The proposed mentoring system

This scenario is in keeping with current research on Tutorial Dialogue Systems such as can be found at http://aclweb.org/anthology-new/ in the Computational Linguistics, Natural Language Processing and Human Language Technologies sections, as well as in current conferences such as the Artificial Intelligence in Education series (http://www.isi.edu/AIED2007/).

2.4. Conclusion

This chapter described the parent theories of this research "*because 'parent' emphasises that the parent must be relevant to resolving the research problem and not any mere background theory*" (Perry 1998a, pg. 20). These parent theories, from the areas of Software Agents / Intelligent User Interfaces and Educational Technology, provided the theoretical framework for the research.

It then described the research problem theories in more detail - theories from the immediate or focus area. Analysis of these areas - Dialogue Management and Tutorial Dialogue Systems - provided "*the gaps of unresearched or controversial parts of the research problem*" (Perry 1998a, pg. 21). These gaps provided the hypotheses and research questions for this thesis.

Finally, analysis of the extant knowledge from the parent and research theories, led to the test system (Section 2.3.3 on page 89) that was used to investigate these gaps. The research and design methodologies needed to design, implement and evaluate the hypotheses and test system are described in the next chapter.

> *There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.*
> C.A.R. Hoare

Research and Design Methodology

3.1. Introduction

If the disappointments from early AI 'predictions'[7] were not to be repeated (Güzeldere & Franchi 1995b; Winston 1987a; Papert 1988a), it was essential to understand what a software mentoring system **could** accomplish and then **how** to develop and build one. This research was designed to address the following issues:

- could a software mentoring system be developed
- could the system be scaled and applied to a real-world, unconstrained environment
- could the resulting system be beneficial to students using it.

These issues formed the basis for the research questions of this thesis and hence provided a hypothesis for testing each issue. To undertake this testing, a suitable Research Methodology (RM) had to be used so that the system could be developed and evaluated correctly. Also a suitable Design Methodology (DeM) had to be used to ensure that a proper 'design, implement, test, improve' cycle could be implemented to build the system.

It was also important to understand the limitations of the research so that the results could be understood in their proper context. Since the research involved generated metric data as well as evaluation data from students, care had to be taken to ensure that the data was processed correctly and ethically and made available to other researchers.

---

[7] "*Within ten years a digital computer will be the world's chess champion, unless the rules bar it from competition.*" -Herbert Simon, Nobel Prize laureate, 1957.
"*In from three to eight years we will have a machine with the general intelligence of a human being.*" -Marvin Minsky, MIT, 1970
"*In an astonishingly short time, scientists will be able to transfer the contents of a person's mind into a powerful computer, and in the process make him, or at least his living essence, virtually immortal.*" -Hans Moravec, Carnegie Mellon University, 1987
"*Japan will create intelligent machines within ten years.*" - Edward Feigenbaum and Pamela McCorduck,1983 (1995: Japanese AI project shut down after consuming billions of dollars but showing no significant progress.)
"*.. renewed interest in AI might combine with inflated claims about market size and result in the kind of indiscriminate AI investing that took place in the past. 'Hype sells books, but whether AI is a good investment is another question,'*" - Terry Winograd,Stanford University
"*.. the lion's share of funding has traditionally gone to the pundits who make the wildest claims. The Fifth Generation worked well, and Feigenbaum got a lot of funding.*" - Rodney Brooks, MIT

3.2. Justification for the paradigm and methodology

3.2.1. Research Methodologies

Shulman stated that researchers "*conduct research in a field to make sense of it, to get smarter about it, perhaps to learn how to perform more adeptly within it*" (L S Shulman 1986a, p. 3). If a mentoring system is developed correctly, then students will learn from it.

To develop the system correctly, the appropriate research methodologies had to be used. Several of the RMs identified by Mauch & Birch (1993a, p. 114-120) were relevant to this study:

Action Research / Design and Demonstration

Action Research (AR) was used specifically for the computer science design and implementation aspects of the research, with Educational AR being used for the educational aspects.

AR is the standard RM used in the design, implementation and evaluation of software systems. Cycles of iterative software design and maintenance were used in the system production, and formative evaluation of the design and its operation were carried out at the end of each cycle (see Figure 3.1, page 94).

Evaluation

An Evaluation RM was used for the educational evaluation aspect of the research.

The benefit of the software to the students needed to be determined. Analysis of the effectiveness of the design of the system provided formative evaluation whilst questionnaires provided a summative evaluation of the overall benefit to the student. An Evaluation RM has proved effective and efficient in previous computer based studies (Ng & Marriott 1996a) and, of relevance to an iterative case study evaluation, Michael Scriven (1991) has indicated that "early-warning summative" evaluation - an evaluation of a developing version of the system - is one of the most useful kinds of formative evaluation in an educational context.

Meta-Analysis

The research involved several areas of the computer science discipline - network communication and protocols, language design, natural language parsing, dialogue management. The areas of Intelligent Tutoring Systems, Mentoring and Hypermedia Systems from the education discipline were also relevant in the pedagogical design of the system and in the evaluation of the effectiveness of the system.

Meta-analysis - combining results of research from different scientific areas - was used to analyse these two areas.

Quasi-experimental

The software and educational design of the system evolved over time due to the ongoing formative evaluation of the student feedback, as well as through critical reflection on the implementation. Due to this, various parameters in the design changed to optimise the benefit to the students. Given that no control group could be used due to the nature of the computing course, and rigorous control over variables, treatments and user populations could not be maintained, a Quasi-experimental RM was used whilst developing and evaluating the system. The cause-effect relationship was approximated and the underlying assumptions and limitations identified.

Action Research Methodology

Action Research (AR) was developed as a new mode of investigation cognisant of the need for pragmatic research. AR is not one single well-defined RM but a group of RMs that specifically address the requirements and outcomes of both 'action' and 'research'.

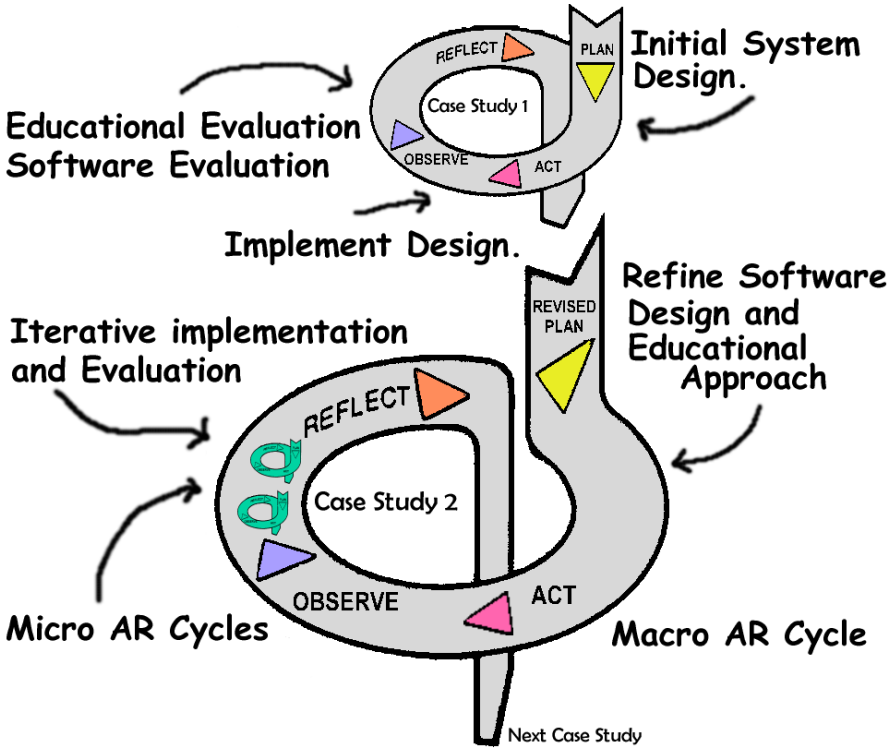


Figure 3.1 : Action Research cycle (adapted from Kemmis & McTaggart (1988a))

AR methodologies are characterised by cycles of development, participation, evaluation and reflection (see Figure 3.1). Within the computer science domain, cycles of development and evaluation are common whereas within the Educational domain participation, evaluation and reflection are considered important.

The cyclic nature of AR implies that development of solutions emerge through iterations. Evaluation of solutions can be both formative - feedback from early cycles improves the system design and effectiveness - and summative - the resulting system can be evaluated in the light of increased understanding of the problem domain.

The duration of the macro AR cycles is important when conducting research in restricted time frames such as a University semester. Each semester provides a new group of subjects that can use and evaluate the **current** system. Time management must ensure that the **previous** system has been refined and tested and is ready for this new cycle of evaluation. It is important that an AR cycle is not wasted due to critical problems that may emerge early in a semester, and hence forward planning is required since the development (or refinement) time for first semester may be several months due to the University's summer break, but for the second semester, it is only one month.

Micro cycles within the larger AR cycles also enable responsive iterative development - the early problems are addressed in a timely manner. These micro cycles situate the researcher in the same context as the research subjects and this is particularly important in educational AR when evaluating student responses.

Within each macro cycle of this research, it was necessary to evaluate and reflect on emerging problems or issues. It was also necessary to undertake corrective development as critical issues emerged. The AR process requires critical reflection during each cycle so that the system can improve. Initial research involves the unknown rather than the known and reflection provides the transformation process. Further information about AR can be found at http://www.scu.edu.au/schools/gcm/ar/arhome.html [HREF 56] as well as at http://www.qual.auckland.ac.nz/action.htm [HREF 57]

Educational AR is becoming more accepted for multi-disciplinary research (see http://www.triangle.co.uk/ear [HREF 58]). Within an educational context, J Elliott (1991a, p. 49) stated that "*the fundamental aim of action research is to improve practice rather than to produce knowledge*". This is important, as this thesis applied new and established principles of computing along with established teaching and learning paradigms to produce a novel software based mentoring system with the aim of improving learning for students.

Evaluation Methodology

The Evaluation Methodology enabled the researcher to identify the students' feelings about, and perceptions of, the developed system. This evaluation was both formative and summative. The formative evaluation was integral to the AR cycles and involved feedback on the development, evaluation, reflection and subsequent refinement of both the software and the educational paradigm used in the system. The summative evaluation was conducted through questionnaires provided at the end of each macro-cycle of the research and aimed to determine the students' attitudes towards the system. Given the cyclic nature of the development, these questionnaires also provided formative evaluation used in the next cycle.

The questionnaires (see Appendix A, Appendix B and Appendix C):

- provided qualitative and quantitative evidence of user attitudes,
- provided subjective user evaluations of the benefits and problems of the system,
- provided the researcher with information about the needs of the users as they interacted with the system,
- highlighted the areas for improvement, change or addition to the system.

The questionnaires were designed to record anonymous user information such as age, course of study, and gender, as well as information about various aspects of the mentoring system that were considered essential to this thesis. Responding to the questionnaires was purely voluntary.

In summary, students were asked:

- if they had used the system, for what purpose, how often, and had it been beneficial,
- if the system had been useful, annoying, obtrusive,
- what areas of the system had helped them most, how and how did they benefit,
- had the 'learning paradigm' that was used help them,
- what areas for improvement were necessary.

The questionnaires were detailed, typically 6 pages, with fixed-alternative 5 point Likert scale questions, as well as open-ended questions so as not to force users to adopt pre-

conceived answers/notions. This quantitative information also provided first hand user's perceptions rather than the researcher's interpretations.

Due to the questionnaires' length, double-check questions as well as questions with known bounded answers were used so as to detect/correct users who simply ticked boxes. Double-check questions are ones that duplicate or nearly duplicate other questions and are used for correlation or validation.

The questionnaires recorded students' attitudes not the absolute truth. Attitudes are composed of feelings, beliefs and the enactment of these attitudes. The evaluation of the students' **attitude** about the effectiveness of, and benefit from, the mentoring system rather than the **truth** about its effectiveness did not detract from the study.

In recording these attitudes, it was important to realise the problems associated with transcribing internally held attitudes to an externally consistent representation. Likert scales were used to represent a range of attitude states. Users indicated their range of agreement or disagreement on an N-point integer scale. These numerical values were then used to generalise the overall attitude of users towards the recorded issue.

For a 5 point Likert scale, the responses are usually indicated as:

```
1. = strongly disagree with the issue
2. = somewhat disagree with the issue
3. = undecided
4. = somewhat agree with the issue
5. = strongly agree with the issue
```

To reduce researcher bias, the questions and responses are phrased so that the respondent is not told what to say or believe. A background to Likert scales and the issues with using them for evaluation has been given in Section 2.2.3.5 on page 45.

In the evaluation of this research it was assumed that the respondent's attitudes can be discrete and that the attitudes can be ranked on the issue. That is, regardless of the quantised unit, the respondent may be forced into the 1-5 or 1-7 range. At worse, respondents may have expressed a neutral position in an odd numbered scale.

3.2.2. Educational Paradigm

This study was not trying to determine what educational paradigm would emerge from the system but whether imposing a teaching method paradigm onto a software system was possible, and whether it would be effective and beneficial to the students who used the system. An Authoritative Guidance paradigm as well as a Directed Learning Path paradigm were used and their effectiveness evaluated.

3.2.3. Significance of the Study

There were two significant outcomes to this study:

- the design and implementation of the system and
- the analysis of the system's effectiveness in benefiting students.

Current research in the broadly titled 'Educational Technology' area has produced small scale systems applied in very constrained environments. The mentoring system was based upon research into Interface and Desktop Agents, Information Filtering and Data-

mining, Intelligent Tutoring Systems, and Dialogue Management. There was a need to develop an integrated real-world system which exploited the advantages of cooperating network-based programs and to apply it to the area of tertiary teaching support.

As shown in the previous chapter, at the start of this research, little work had been published in this area and few case studies or large scale applications of this technology had been reported. The design, development and implementation of the system in a non-trivial environment was seen as a significant contribution to both research and teaching in this area. This system was used within the Department of Computing to benefit students enrolled in degree programs.

There are a number of other reasons why this research was considered to be important:

- Bloom (1984a) indicated that two standard deviations difference in academic student benefit is possible when comparing the outcomes of one-on-one tutoring to one-on-many. Fletcher (2003a) has indicated that this could represent a typical student achieving academic results that increased from the mid-level 50th percentile to the 98th! He also indicated that the most obvious reason why educators do not choose one-on-one is cost.
- Universities are being asked to develop open and flexible ways to help students learn. Also, the changing nature of student populations at Universities means that these may be either part time students, distance education students, overseas students or disadvantaged students. The tertiary education system is increasingly being seen as a global market with entire courses being marketed on the World Wide Web. Since the globe has no fixed semester time frame, this may put extra strain on academics by removing the ‘end of semester’ time buffer traditionally used for unit preparation and/or research.

  Given the changing needs of the students, a number of questions emerge:
  - How could flexible student learning be realistically supported by academic staff?
  - What other ways existed that could help students to learn?
  - What other bodies of expert knowledge existed that students could use?
- There is a perceived need to increase the number of students graduating from the University system and there are concerns about the quality of the student learning experiences. What was needed was a support infra-structure that would ensure students had the skills needed to be successful graduates:
  - How was the learning support infra-structure to be developed and maintained?
  - How could standards be ensured?
  - How could staff find time to maintain their commitment to quality teaching?
  - How could students learn professionalism and increase their knowledge and skills if contact with academic staff was reduced?

It was proposed that a software based mentoring system would provide solutions to help address these questions.

3.3. Design Methodology

3.3.1. Design Development

AR dictated the cyclic development of the software based mentoring system. The use of Java dictated an Object Oriented (OO) approach to the system design. Wilkie (1993a, p. 12) indicated that the "*most fundamental advantage of object-oriented techniques, ..., is in creating a more modular approach to the analysis, design and implementation of software systems*". He also indicated that the "*single biggest disadvantage at present is the immaturity of the techniques and tools*" (Wilkie 1993a, p. 13). At the start of this research (1997) very few large scale Java systems had been developed and this "*immaturity*" was a concern. Wilkie also stated that the "*highly incremental prototyping life cycles .... bring problems associated with the management of software developments, which can lead to wandering designs causing poor overall quality*". Given the Action Research nature of this research, planning to avoid these potential problems was considered very important.

The typical 'analyse-design-implement' cycle of software development, whether using a waterfall model (Boehm 1981a), or especially the evolutionary model of the Object Management Group (OMG 1992a, p. 24), fits well with AR. Building on these standard iterative approaches, an OO Software Engineering methodology (I Jacobson *et al* 1992a) can be used to analyse, design and implement systems incrementally. This approach was adopted for the Design Methodology for this research.

The size and complexity of the mentoring system indicated that a formal Java package hierarchy would need to be developed. Since the central server processed multiple users concurrently, a thread-aware design paradigm had to be used. Also, because of these concurrent users, an Operating System (OS) design approach had to be taken for the central system server, and this required a Java Security Manager to be developed to protect from accidental or malicious user actions. This was especially important in a computer science student environment.

The client-server approach for the entire system necessitated the development of a network protocol for information transfer that was both efficient in terms of overhead for small data transfers such as user input and responses, as well as fast for large data transfers such as user topics loaded into the server, and multimedia data. The design was to cater for transmission of this data across low bandwidth networks such as through a modem, or across the world. The protocol was extensible to cater for system functionality growth. Network data security was not considered to be within the scope of the research - the existing University / Department security was deemed sufficient.

It was decided that no formal database (such as postgresSQL) was needed for the textual data storage and accessing. However, the user interaction data were marked up in such a way that they were easily parsed, searched and categorised and this implied a consistent markup language. Because of these requirements, XML - the eXtensible Markup Language (XML 1997a) - was chosen as the best option for data storage. The format of the data in the Knowledge base also used XML and was developed to cater for future system usage and functionality (see Section 4.2.6 on page 166 on VHML).

Finally, given the complexity of the system and the fact that it was multi-threaded and client-server based, it was developed with extensive logging and debugging functionality as well as the OS concept of a kernel, privileged system calls, and privileged users who could remotely monitor and adjust the system. The limitations of Java for this and other design aspects had to be recognised within the design, development and implementation process.

3.3.2. Design Implementation

The architecture of the mentoring system can be seen in Chapter Four, Figure 4.6 (page 113). At the core of this architecture is the central server which maintained the data of per-student, per-unit and global information as well as the data collected that was concerned with system usage.

The server ran as a network initiated process on one multi-purpose host and served mentoring clients that ran on other hosts within the Intranet network. One server existed for all users and a separate Dialogue Manager class was created for each user who connected via a client. This hierarchy enabled efficient processing of data and services.

The typical user client was either a Graphical User Interface (GUI) on a workstation (represented by the GUI symbol in Figure 4.6 on page 113), or a text-based interface (the terminal icon). A Talking Head interface (the head) was also possible and this was used to test the systems' extensibility.

The typical GUI client used the familiar Internet Relay Chat paradigm with separate window areas for inputs and responses. The response area was capable of displaying XHTML information and hence could display images and hyper-links to Web pages. The GUI was user customisable in various ways so as to minimise the disruption that the client could cause to normal student work - the client was always present whenever a user logged in and was typically iconified until the user wanted to interact with the system, or vice versa.

Different functionality clients could connect to the server. These aided in administration through user-process and system Thread monitoring, and remote system interrogation and manipulation. These clients required password authentication.

The server also executed system and user 'agents' (the humanoid image in Figure 4.6 on page 113) for information gathering and processing. These agents - called daemons - communicated with either the server or the client directly as appropriate.

The mentoring system also initiated asynchronous communication using one of the above interfaces to help or notify the user of some relevant event. For example, user to user communication and collaboration may be initiated by another party, the results of a backgrounded agent task may have become available, or relevant unit information may have been updated and should be brought to the user's notice.

The Knowledge base was represented by Java classes called 'Topics' and users could write their own Topics to transparently extend the functionality of the system. The Dialogue Management was handled by a core package and in fact, the mentoring system was just one instantiation of the Dialogue Management system.

More information about the system architecture and implementation issues can be found in Chapter Four.

3.3.3. Research Outcomes

This research proposed the following outcomes:

- It would provide a description and analysis of the design, implementation and evaluation of a large scale system comprised of cooperating software components applied to the area of mentoring in the education environment.
- Its usage would provide a description and analysis of the effectiveness of the system in helping students with their University tasks, especially in the area of assessment.
- It would provide an ongoing environment that aided students in their tertiary education. It was proposed that:
  - The system would guide and assist students by showing them how to **start** solving a complex problem, and subsequently, how to **break up** the problem into a series of more manageable steps.
    The student would be able to realise the same academic outcomes as normal but by being assisted up a 'gradual slope of learning' by the mentoring system, rather than by being 'confronted with a series of tall steps'.
  - The system would help students to become more capable in exploiting the information resources around them. The system would inculcate the student with good organisational and management skills.
  - Students would become more independent. The student would be dependent upon the system initially but would be taught skills which would subsequently make parts of the system redundant. This is common in a mentoring relationship where there is an initial inequality in experience and expertise but which levels out as the mentoring progresses (Caldwell & Carter 1993a).
- It would provide a benefit to the lecturer by having the expertise and support provided by the system being available to students 24 hours a day. That is, the system would reduce the work load of the lecturer, by handling questions at any time.

3.3.4. Qualifying Assumptions

(1) That the student sample does not vary significantly from year to year (or that the variance can be factored into the data analysis).

(2) That bias in the data can be factored into the data analysis. For example, good students are more likely to recognise the benefit of, and take advantage of, any form of help.

(3) In the context of the discussion on sampling by Sprinthall *et al* (1991a, p. 29), the student sample is random even though students may or may not elect to use the system .

3.4. Hypotheses

The hypothesis that has directed the research, and that will guide the thesis is:

A software-based *Mentor* System system can benefit University students.

Sub-hypothesis 1

A software-based *Mentor* System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.

Sub-hypothesis 2

The *Mentor* System can support an effective learning paradigm.

Sub-hypothesis 3

The *Mentor* System will benefit students.

3.4.1. Limitations and Delimitations

Given the following Limitations and Delimitations[8], it was expected that the results of the study would be applicable to other disciplines and environments within a University.

3.4.1.1. Limitations

(1) A significant number of students would use the system. This would ensure that the data collected as the students used the system and the data from the evaluations would have some statistical significance. A case study population of 30[9] was seen as an acceptable lower limit for having confidence in generalising any results.

(2) The study could not use a control group nor use students' academic records for pre-test/post-test analysis.

(3) The system was available on the Unix-based machines that the student group would use. The client-server architecture design later relaxed this initial limitation. This would ensure that the users benefited from it being available 24 hours a day. The system would not be available from a student's home computer. Requests from the formative evaluation of the case studies later relaxed this initial limitation so that the system could be accessed (with authentication) from anywhere in the world.

(4) Unix user information was consistent and available. Various Unix files such as the passwd file, were used to obtain user information necessary for correct operation of the system. The information included their names, the units they were enrolled in, and their status within the department (undergraduate, postgraduate, staff).

---

[8] From Perry (1998a) "*by definition, 'delimitations' are within the control of the researcher and 'limitations' are not.*"

[9] The minimum sample size n is given by

$$n = (Z_{\alpha/2} * s/m)^2$$

where $Z_{\alpha/2}$ is 1.645 for a 90% confidence interval, m is the margin of error in appropriate units for the evaluation, and s is the sample's standard deviation. If a sample standard deviation of 0.5 is expected and a 5% margin of error is required, then a sample population of approximately 30 is required.

3.4.1.2. Delimitations

(1) The system was not an Intelligent Tutoring System (Brusilovsky 1995a). That is, the purpose was not to develop another ITS nor an ITS shell. The system was similar to an ITS, however, unlike an ITS, it was not based on modelling the student's learning motives and abilities.

(2) The system would not be concerned with Expert System knowledge on any specific unit. That is, the system was not an Expert System in the Computer Science meaning of the phrase. However, the system did have expert knowledge about units, about discipline context, and about various teaching paradigms.

(3) The system was not a general purpose educational paradigm testing framework. The system gave advice to students for their strategic planning in areas such as information discovery, study management and assessment management (assignments and examinations). It implemented a Directed Learning Path and an Authoritative Guidance paradigm. It was expected that the system's extensible nature would support other paradigms though.

(4) The system did not recognise speech input. The study was limited to interactions with the student via a GUI and a text based interface.

(5) The study was limited to Computing students and was Unix-based. Requests from the formative evaluation of the case studies later relaxed this initial delimitation so that the system could be accessed from Windows based clients as well.

3.5. Data Collection and Analysis

3.5.1. Data Collection

The data collected were from two sources:

(1) the input requests to the ***Mentor*** System, the responses from the system, and the reaction of the users to the response

(2) the qualitative and quantitative data from the user questionnaires (App. A, B and C).

The first was mainly used for **formative** evaluation - focussing on the *process*. The second was concerned with **summative** evaluation - focussing on the *outcome*.

The data needed to support the hypotheses were of three types:

(1) Electronic Data obtained from students using the ***Mentor*** System.
As students used the system, they provided implicit and explicit information. The implicit information such as frequency of use, duration of use, correctness of responses, etc, and the explicit information of their input to the system, was stored in data files. The data format used to store student interaction was XML-based (see Appendix G.12 on page 377). This format enabled the data to be easily parsed, categorised and transformed.
Data from this ongoing usage were used to perform formative evaluation of ***Mentor***, especially the correctness or relevance of the response to the students input.

(2) Qualitative data from the questionnaires designed to provide both formative and summative evaluation of the ***Mentor*** System.

(3) Quantitative data from the questionnaires designed to provide both formative and summative evaluation of the ***Mentor*** System.

3.5.2. Data Analysis

The data were analysed (see Section 5.1 on page 191) to see whether they confirmed or denied the hypotheses.

(1) **Electronic Usage Data:** the formative evaluation determined if the students were receiving good advice from the system, and that it was helpful in their academic life. Measuring frequency of use, and relevance and accuracy of responses over time provided data that were evaluated to determine the success of the system.

(2) **Questionnaire Data:** the qualitative and quantitative data collected were analysed to determine if the students felt that the system was effective. This summative evaluation was also used to determine the areas of success and failure in the system.

3.5.3. Admissibility of Data

The previous data sets were screened to ensure that only valid data were analysed. For example a common problem with questionnaire replies is in providing ambiguous, wrong or inconsistent answers or not answering all questions. There are many techniques to minimise or eliminate these problems (Wentland & Smith 1993a; Krathwohl 1993a).

One issue with the data collection was that it required a detailed questionnaire to record all necessary information over the range of variables. A user may simply tick boxes if the procedure becomes tiresome or uninteresting, and this would compromise the data set. Questions with known responses as well as double-check questions were used to detect or reduce this problem.

The impact of the Hawthorne Effect[10] was not known and could not be catered for since no control group was available. Since the use of the mentoring system is systemic to the teaching and learning process, then any positive or negative effect was still a result of the process.

3.6. Ethical considerations

Ethics approval for the evaluations was granted at candidacy as the research involved the testing of the hypotheses on students as well as their evaluation of the system using questionnaires. One informal and three formal evaluations were performed in semesters one and two in 2001 and 2002. The questionnaires for the three formal studies can be found in Appendix A, Appendix B and Appendix C.

At the time of data collection, the researcher was the Lecturer-in-Charge of the units in which the student evaluation was completed. Thus it was very important that complete student anonymity was ensured for the **evaluation** questionnaires. The research design did not allow for the anonymity of students **use** of the ***Mentor*** System. The researcher was however always mindful of the responsibility of his position with regards to the privacy and rights of the participants in the study.

---

[10] "*A threat to internal validity in which what appear to be IV effects are actually due to the attention or flattery that subjects are given during the research process*" (Sprinthall *et al* 1991a, p.372). *IV* is 'internal validity'.

The research was therefore structured using the following guidelines:

(1) User statistics/input data were confidential and were encrypted (Zimmerman 1994a; RSA Laboratories 1993a) where they would identify or potentially identify a user.

(2) For each semester where they were involved, students were informed about the research, informed about their possible involvement in the study and then told that they may choose to participate in the study if they wished.
Students could leave the study at any time.

(3) Student input data identified a student to the researcher alone.
The questionnaires did not ask for the name or student number of each student and hence identity of evaluation participant was not known to the researcher.
Data such as age, gender and student course was recorded so as to test for correlations between these variables and other questionnaire responses. However, the inter-correlation of these variables could potentially identify an individual given a small sample space. This identification process was not done.
For the purpose of any needed follow-up interviews of participants, the following was put into the questionnaires:

For some participants, I may ask if I can do a follow-up interview. This would be to get more information or to clarify your responses. This would be purely voluntary on your part. If you are happy to be re-interviewed, then put some unique identifying mark or picture in the box on the next page - something that you could recognise but that does not identify you to me. In this way I can send email to all SPD students saying I would like to re-interview the students who had these marks (and then show the mark as an image). This would only be for a small number of participants and even at that stage, you could decline to be re-interviewed.
Choose a mark that you can remember and that is unique.

If you don't want to be considered for another interview, leave it blank. If you want to be considered, but want to remain anonymous, put a mark. If you want to be considered and don't care if I know who you are, simply put your name or your mark.
For example, I might use:

(4) Students who participated in the study could choose to participate in the evaluation if they wish. The questionnaires indicated information about the study such as:

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System in semester X, YEAR, for the unit YYY. This feedback will be used to improve the Mentor System for subsequent units. Hopefully the more feedback you can give to me, the greater will be the improvement in the Mentor System.

This exercise will take **15 minutes** of your time, and while your involvement would be helpful you do not have to participate in the exercise, **it is purely voluntary.**

Note:

- You are free to stop the questionnaire at any time
- If you do participate in the questionnaire, please answer all questions fully.
- The individual data collected will remain strictly confidential
- The questionnaires are anonymous, as there is no need to record your name

This and other relevant information including contact details, was on a detachable cover sheet to the questionnaire so that the participant could follow up on the study.

(5) Results of the finished study will be made available upon request to any participating student.

(6) The raw and processed data was stored in accordance with University policy.

3.7. Conclusion

This chapter described and justified the appropriate research and design methodologies selected to conduct the research and to explore the hypotheses developed from the research issues outlined in Chapter Two.

The chapter discussed the significance of the study and also proposed the research hypothesis that a software-based mentoring system can benefit students. It also stated the assumptions, limitations and delimitations that bound the research to test the hypothesis.

Finally, aspects of the research evaluation process such as ethical issues and data treatment were discussed.

> *The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.*
> Edsger Dijkstra

The Design and Development of the *Mentor* System

4.1. Introduction

The system was designed and implemented to test the sub-hypothesis:

> A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.

This chapter will show supporting evidence for the above hypothesis by detailing the design approach and also the issues that arose from designing and implementing the architecture of the system. It will also explain the design and implementation decisions taken in building the system. The approach has been very pragmatic in nature - the system must work in the researcher's learning environment (Department of Computing at Curtin University of Technology), not just as an academic exercise.

Chapter Five - Data Analysis - also discusses the results of two questions from the final case study evaluation that were devised to test this sub-hypothesis (*'How would you rate the response time of the* ***Mentor*** *System?*' in Section 5.3.6 on page 258, and *'How would you rate the ease of use in adding Domain Knowledge by* **programming** *the* ***Mentor*** *System?'* in Section 5.3.7 on page 259).

The conclusions of the investigation into the research issues from the previous chapter formed the basis for some important design decisions:

Use of Java

The decision to use Java as the implementation language seems, in hindsight to be of little importance. However, at the start of this research (1997) very few large scale Java based systems (of any kind) had been built and the limitations and stability of the installed Java compilers and Virtual Machines posed many problems (Marriott 2001b). However, Java was seen as a long-term enabling technology due to its many advantages over other languages (Gosling & McGilton 1995a). With few exceptions, for example Java's lack of a per-*Thread SecurityManager*, Java has been able to meet the complex requirements of a Dialogue Manager, and of the ***Mentor*** System in particular.

Since Java is also an evolving language used by a large client base, new technologies such as XML and XSLT are quickly implemented by users and then incorporated into Sun's distributed Java systems, either as part of the core or the extended Java classes.

Layered Architecture for the Dialogue Management System

It was decided that a Dialogue Management System (DMS) would be built rather than just a ***Mentor*** System. The system would have a core that provided basic generic Dialogue Management services to support many different types of Dialogue Managers and that one or more specific Dialogue Managers would be instantiated when the server ran. In this way, the system would be extensible and could grow with users' needs. The layered approach also meant that a strict Application Programmers Interface (API) could be used to isolate the underlying implementation of the core.

Figure 4.1 shows the core DMS server surrounded by an instantiation of a Dialogue Manager (such as ***Mentor***). This layer is encapsulated by the DialogueManager API and provides the functionality needed by the system Topics, as well as any Topics developed and dynamically loaded by the user at connection time. These Topics provide the Domain knowledge for the system.

The DMS server also provided support for daemon processes that the instantiated Dialogue Manager may want to run. For example, a password daemon for validating users, or a unit information daemon that would provide information about the units that a student was enrolled in. These system processes also provided information via a strict Dialogue Manager system call API.

Finally, the Client interface provided a strict API for any client to connect to the system. This interface is flexible and is driven by the capabilities of the client. The only requirement is that users validate and announce themselves when they connect. That is, they have a unique username and that this is valid for the system.

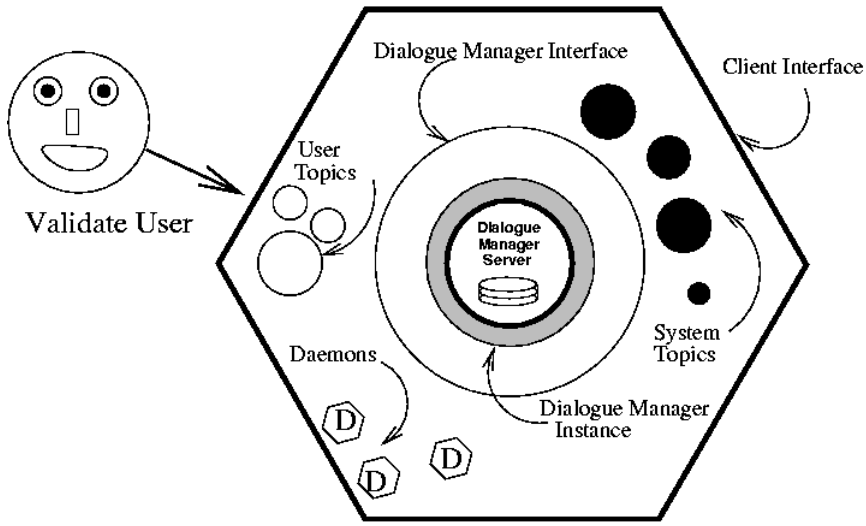


Figure 4.1 : The Layered Architecture of the Dialogue Management system.

Use of a client-server Architecture

Figure 4.2 shows the chosen client-server architecture with a user running a client to connect to the layered system of Figure 4.1. The use of Java's *Thread* class meant that many users could connect to the DMS, each getting an instance of the Dialogue Manager to interact with. Through the use of the Java *SecurityManager*, the Dialogue Manager system calls, and protected API methods, each user connection operated as a safe process inside this multi-process, multi-user system.

Even if a user sent their own topics across to the server for execution, these could not affect the overall execution of the system. For example, user topics cannot execute a `System.exit()` call nor can they read or write arbitrary files. The latter has to be done through the Dialogue Manager system call API.

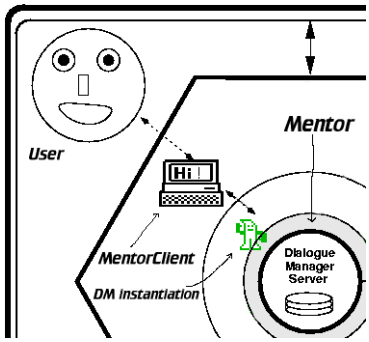


Figure 4.2 : The client-server Architecture.

Use of a Flexible client API

Based upon the conclusions of the investigation into the research issues, the researcher decided that it was important to develop a system that provided support for many different types of clients to connect to the DMS server. A text-only client would be necessary but it was seen that a Web-based client could also be useful in a learning environment. Given the extensible nature of the server, it was decided that the client should also be able to render the output in many different forms. Figure 4.3 shows the system being connected to by different types of clients: several plain text interfaces are seen as well as a photo-realistic computer generated talking head, a 2D Web-based talking head and a text-based Web interface.

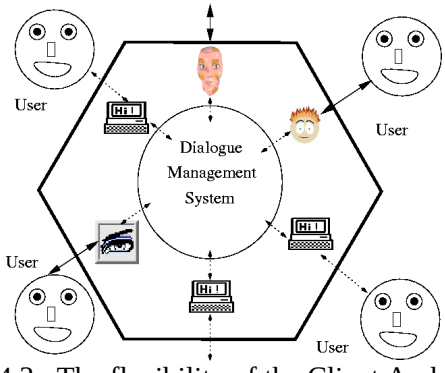


Figure 4.3 : The flexibility of the Client Architecture.

Use of an active vs. a passive Knowledge base (KB)

It was decided that the KB would be represented by code rather than data. That is, the knowledge would be returned from executing code that could access the network or databases, or simply through computational manipulation of data in the existing environment. In this way, for example, online Web pages could be dynamically data-mined for relevant information.

The KB units are called Topics and are Java classes that are accessed via a specific API. These Topics are loaded into the KB in a hierarchical manner dependant upon what environment the user is in, what units they are enrolled in and what local Topics the user has developed.

See Appendix H for example user Topics.

Use of the eXtensible Markup Language (XML)

Since different clients could render the response from the DMS in various ways (Figure 4.3), that response must be in such a form in the KB that the information as well as the full 'intent' can be rendered by any client. For example, a response generated by a topic in the Knowledge base which contains a URL may have these types of rendering:

- a text based display may turn that into plain text,
- a Web-based display may turn that into a link,
- a vocal display may have to change the text into something like "You can find out more about me from the link www.blah",
- a Talking Head may say the above and also open a browser window,
- a Virtual Human may point to the link, etc.

It was decided to use XML as it provided a compact, easy to read and parse format for classifying data. Together with the eXtensible Stylesheet Language Transformer (XSLT), it provided a mechanism for storing and then transforming information to suit the rendering capabilities of the output device. The existing Virtual Human Markup Language was further developed to mark up the Knowledge base.

Configurable Client user model

It was decided that the typical client - `MentorClient` - would also be extensible to provide both default behaviour as well as allow for experienced users to program their own interface to the Dialogue Management system. The `MentorClient` extends a `Client` class and instantiates a `UserModel` based upon the user's status within the computing environment.

The `MentorClient` also checks to see if the user has created their own user model by searching for the class file `local.mentor.userModels.xxx` where xxx is the user's computer username. Figure 4.4 shows a complex multimodal Dialogue and Interaction Management System created by a user that exploited this and the previously mentioned extensibilities of the system, to cater for input from remote controls as well as directing output commands to devices such as televisions and radios.

See Appendix I for an example user model as well as associated example Topics that have been extended to create a multimodal Dialogue and Interaction Management System.

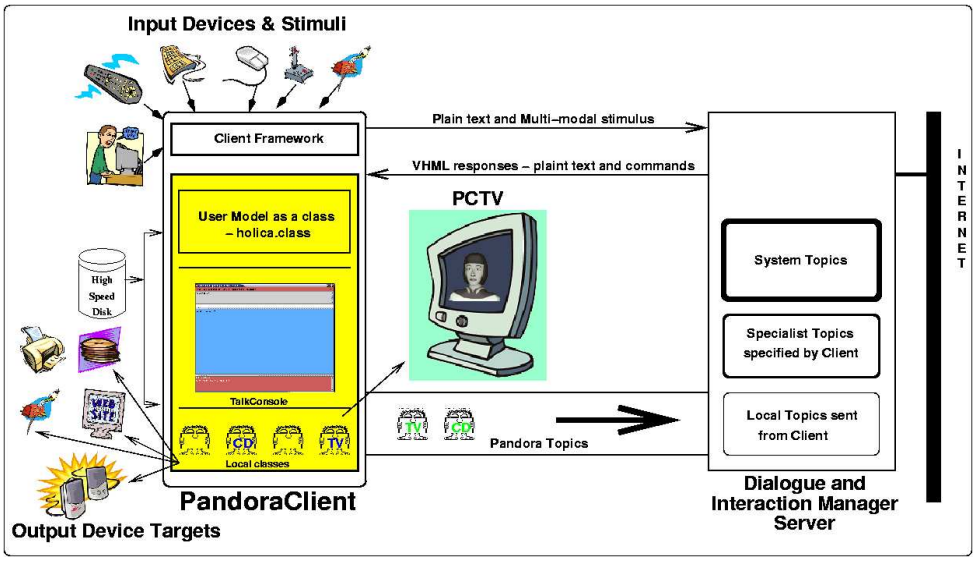


Figure 4.4 : The Architecture extended to provide a multimodal DM system.

Message driven architecture for the usermodel client-server data transmission

The complexity of the client architecture with respect to the user model extensibility, meant that the packet communication between server and the target renderer would be message based. In this way, data from the Knowledge base could be routed using a 'message id' either to the standard renderer (a graphical console) or to some other device such as a printer, a loudspeaker or even to control a television.

The use of Java then dictated the use of Listeners: the various client modules could register an interest in the server packets and be given a 'message id' that could be used by Topics to send the packet specifically to that module. See Appendix I for an example of using the "message id" to route data to a user module.

Architectural Support for educational paradigms and applications

Chapter Three indicated that the DMS architecture should support complex dialogues with students and the information that is of interest to them.

The architecture should also be able to support various educational paradigms so that students could use a diversity of learning strategies. Two specific paradigms were implemented using the generic educational support.

The architecture should also provide interaction support for the Lecturer in Charge of any unit that used the system. That is, the Lecturer should be able to monitor and rectify any problems that occurred from students using the system.

Finally, a Topic builder support application would be developed to allow for non-programmers to create Topics with similar functionality to those developed by programmers. No knowledge of Java or programming should be required.

State-based Perl-5 Regular Expression Pattern Matching

The success of Chatterbots, typically built using state-based regular expression pattern matching technology, indicated that this technology could also be used as the basis for a successful mentoring system. The active KB approach also meant that the Topics could be used to supplement this technology and to compensate for any problems.

These initial decisions led to the design and implementation of the Dialogue Management System architecture (see Figure 4.5). The ***Mentor*** System is one instantiation of a Dialogue Manager that uses this architecture.

This chapter will use the DMS architecture as a vehicle to illustrate the issues that arose during the implementation phase of this research, and to support the sub-hypothesis that the system can be designed and implemented.

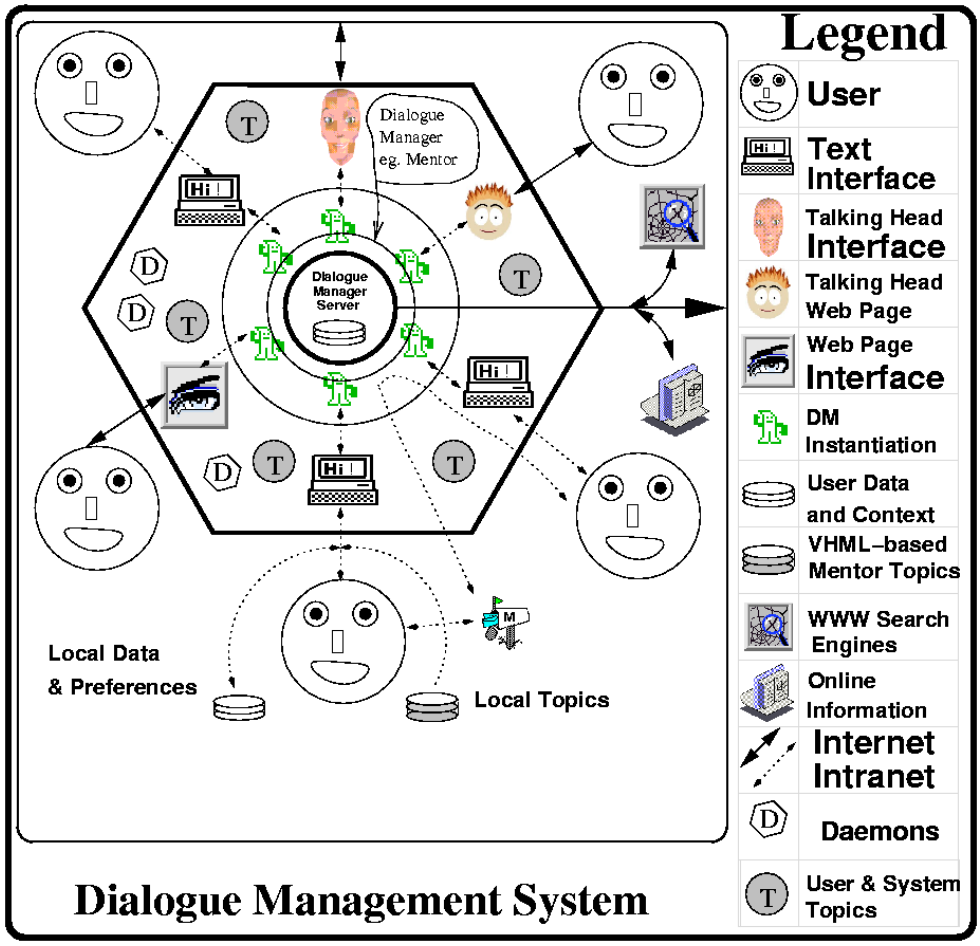


Figure 4.5 : The Dialogue Management System Architecture.

This chapter will first briefly outline the ***Mentor*** System architecture (Section 4.2 on page 112) and show how it extends the DMS. The design and development of the system architecture is outlined in more detail in subsequent sections of this chapter.

This chapter will also detail the design, development and issues that arose from the use of the hierarchical Java packages that form the basis of the DMS and ***Mentor*** components.

The chapter will then discuss the layered approach taken and how this enabled extensibility and the use of the system in real-world applications (Section 4.2.2 on page 134). It is shown that this multi-user, multi-tasking environment protects each logged-on user of the system via a Security Manager as well as via protected user system calls to the Dialogue Manager system. That section will also discuss the issues that arose from choosing state-based Regular Expression Pattern Matching for the user request-response processing and how this drove other aspects of the system development.

The client-server architecture is then explained to show the overall advantage of this approach (Section 4.2.3 on page 146 and Section 4.2.4 on page 154). The server architecture enabled the use of many different types of Dialogue Manager systems (such as ***Mentor***) to be loaded at run-time, whilst the client framework allowed for many different client types and these could be used locally by students or from across the world via the Internet. The client framework also allows either the use of the default request-response user model, or for a user to easily program their own client functionality that could for example, allow full multimodal interaction with the Dialogue Manager system.

This chapter then discusses the format and purpose of the various user and system data files that are used for such things as customisation of the client, customisation of the server, and also to record session information for evaluation (Section 4.2.5 on page 161).

The advantages and disadvantages of using the Virtual Human Markup Language for marking up the Knowledge base is then detailed and this includes discussion of the XML and XSLT technologies that enable the flexibility and extensibility of the Dialogue Management system output (Section 4.2.6 on page 166).

Finally, the support that the architecture gives for two educational paradigms - Authoritative Guidance and Directed Learning Path - is detailed (Section 4.2.7 on page 173). The architecture is examined to see if other educational paradigms could also be supported.

Importantly, each section will also discuss issues that arose from that phase of the development and implementation.

4.2. The *Mentor* System Architecture

The ***Mentor*** System (Figure 4.6) extends the core Dialogue Management System architecture and embodies:

- a Client-Server system and a message-id based network protocol,
- a state-based Regular Expression Dialogue Manager (DM),
- extensible client interfaces: text as well as GUI based,
- extensible user models for the client,
- an active VHML compliant Knowledge base (KB)
- network enabled information providing daemons
- data files that hold per-student, per-unit and global information,
- a Tutorial Dialogue System, and
- support for suitable pedagogies that help the students learn.

Each user connected to the central server through the network via a ***Mentor*** client. The normal user client is a text-based hyper- and multi-media interface, but a Talking Head interface, a Web page Talking Head, and a plain Web page interface are also available.

Mentor - a Java based mini operating system in its own right - is of moderate size, currently being about 125,000 lines of Java code in 260 different classes spread over 25 packages organised in a hierarchical fashion.

Each user is processed by an instantiation of a ***Mentor*** DM - a Java *Thread* that encapsulates all that is needed to help that specific user. The identified user interacts with the system via this DM and, when finished, disconnection from the system causes various sets of stored data to be updated about that user. This enabled the system to continue the dialogue with the user when they connected again at a later time. The entire interaction of the session between user and system was recorded in a time-stamped manner for later formative analysis.

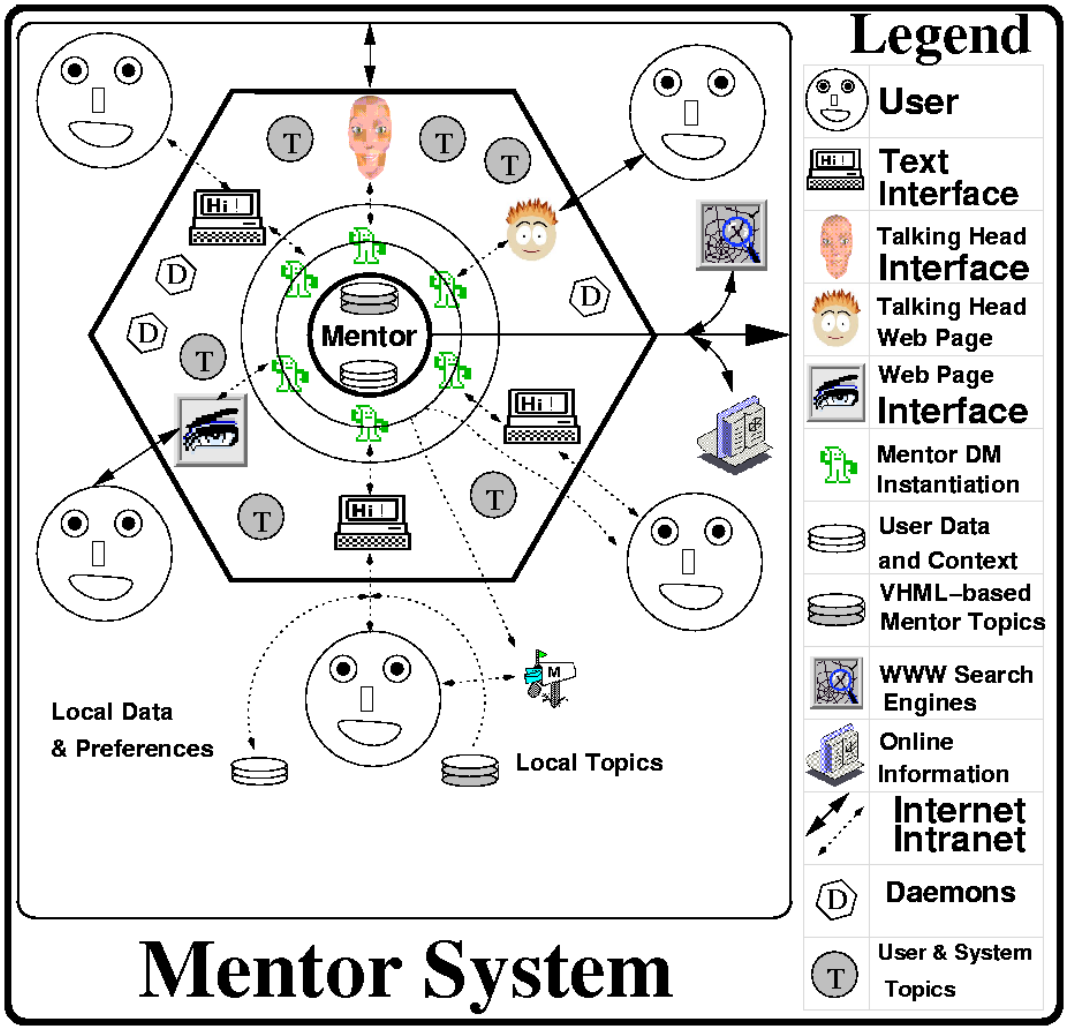


Figure 4.6 : The ***Mentor*** System.

The ***Mentor*** System is composed of many subsystems along with configuration and data files. The following sections detail the developed Java package hierarchy, the Dialogue Management kernel along with its use of Perl-5 Regular Expressions in the active KB Topics, both the server and the client architectures, the various ***Mentor*** user and system data files, and the language used to markup the Domain Knowledge. Finally, the educational support utilities are described.

4.2.1. The Java Package Hierarchy

The ***Mentor*** System uses three separate Java package hierarchies (see Appendix G for package details):

- `local.raytrace.*`: the Base Packages.
  Local, general packages developed for this and other research.
  These provide base level functionality and include the DM kernel code.
- `local.mentor.*`: the ***Mentor*** Packages.
  Local, specific packages developed for the ***Mentor*** System.
- external: the External Packages.
  Two external packages that supply RE and XML processing.

4.2.1.1. The Base Packages

The Base Packages are:

`DialogueManager`	`Names`	`IO`	`Var`	`Windows`
`awtUtil`	`Curses`	`Jar`	`Syslog`	`Resources`
`Personality`	`QSort`	`Util`	`Languages`	`Topics`

Appendix G.1 on page 354 outlines the scope of these Base Packages. The full documentation for these packages is available via Appendix K. Of these packages, three warrant further discussion here due to their importance to this research.

`Names` Package

The `Names` package has classes that allow for the storing and processing of users' names. If a user said to ***Mentor*** "my name is XXX", then that value must be broken down into the correct structure to represent a person's name regardless of how complex the XXX is. For example, "my name is tim" is a simple statement and easily parsed. "My name is Mr Tibbs" is equally easy to parse since the title is easily recognised. However, "my name is sir swee ann lee" is difficult to parse even for a human. The title 'sir' may in fact be a first name, the 'ann' is a common non-female name in Chinese culture, and the user may be giving his/her family name first or last dependant upon their familiarity with western culture.

As part of the processing of this user's name, their gender should also be determined so that they can be referenced properly via 'He/She' or 'His/Her' in system responses. The system used heuristics based upon likelihood tables of common names. Unfortunately, these names are predominantly European based and hence are often inaccurate for the South East Asian names typical of our user population. This issue needs to be addressed in the future.

The `Names` package also has classes to process Internet domain name abbreviations such as '.au' or '.se' and to produce the equivalent country name, the name of the dominant language spoken there, and vice versa.

`Languages` Package

These classes have to load up typical messages such as greetings in that language. If a user says "I am swedish" or "I was born in sweden", or the user connects to the system from a host in the swedish domain ('.se'), the system will then know that the user can be greeted in the afternoon by "God middag". This 'feature' unfortunately caused many problems - see Section 5.3.5.1 on page 242. The language dependant messages were obtained from the website: 'Jennifer's Language Page.' [HREF 59]

`DialogueManager` Package

This Base Package formed the kernel of the Dialogue Management system. There are three groups of Java classes:

(1) the eXtensible Stylesheet Language Transformer (XSLT) classes,

(2) the DM and User variables classes (XML and VHML variables used by the system and the users), and

(3) the Dialogue Management classes.

The following sections details these important groups and their component classes.

1. XSLT Classes

These classes transformed the VHML string into the appropriate format for the output media. See Section 4.2.6 on page 166 for more information on VHML and the system's ability to tailor the output dependant upon the capabilities of the client. For example, the SADM client assumes that all output from the server will be either in plain text or perhaps plain text with XHTML tags. The SADM server will instantiate an XSLT transformer via:

```
       xsltTransformer = new XSLT_plaintext();
or
       xsltTransformer = new XSLT_html();
              :
       returnedLine = xsltTransformer.getString(responseString);
```

The response string from the system is then filtered through this transformer (last line above) which will strip all non-relevant tags.

The standard user client that connects to ***Mentor*** will however instantiate the `XSLT_html` transformer class that preserves XHTML tags. A DM system that connects to a Talking Head application such as the Murder Mystery application (see Appendix Figure G.17, on page 367), would need to instantiate a transformer that allows for all VHML tags except those for Body animation:

```
       xsltTransformer = new XSLT_TalkingHead();
```

A per-client transformer is instantiated, with the type required being sent across the client-server network connection. In this way, a client such as the Murder Mystery application can control the transformation process as appropriate, while at the same time a different client could be connected and using the default XSLT transformation.

The XSLT code does not use Java XSLT classes - these were not mature when this research started - but a more general Document Object Model tree traversal with pruning of any tags not in a specified valid tag set. Note as well that the internal DM atomic tags such as <**`first_name`**/> discussed in the next section, have already been converted into their final text representation before the XSLT transformation process.

2. DM and User Variables Classes

Internally, the system uses XML atomic tags of the form <**XXX**/> to represent various DM or user values, such as the user's name or the developer of the system, as well as non-atomic tags to provide structure to the dialogue. Some typical tags are shown in Figure 4.7.

```
<first_name/>                    <full_name/>
<time/>                          <good_time/>
<random>.....</random>           <li>....</li>
<getvar name="xxx"/>             <setvar name="xxx">...</setvar>
```

Figure 4.7 The DM and User atomic tags

These tags will be transformed by the DM system into their final representation before the response is sent to the user (see example in Table 4.1) by filtering the response through the `replaceXMLVariables` method:

```
final_output = xmlVariables.replaceXMLVariables(response);
```

Notice that the fourth example of Table 4.1 uses the <**random**> tag. The `xmlVariables.replaceXMLVariables` code would choose a single random entry from the <**li**> elements before it was sent to the client - the example is used to illustrate the complexity available. The resulting final output may be something like: "I hope to help students with their units.".

Example	Final Output
Hi <**first_name**/>.	Hi Fred.
It is now <**time**/>.	It is now 11:27
	It is now nearly quarter to twelve.
<**good_time**/>, <**full_name**/>	Good morning, Mrs Smith
	Buona notte, Francesco
	Wu an, Peng Fu Tse
<**dialogueManager Purpose**>	I hope to <**random**> <**li**>benefit students as they progress through their course.\n</**li**> <**li**>help students with their units.\n</**li**> <**li**>help you with your assignments and exams.\n</**li**> <**li**>give help to students.\n</**li**> <**li**>give assistance to students in their studying.\n</**li**> </**random**>
Do you like <**getvar** *name*="subject"/>	Do you like camembert

Table 4.1 : Example use of the atomic tags

Associated with the DM are variables that represent values such as the DM purpose, name, gender. The values of these variables for the ***Mentor*** System are specified in the ***Mentor*** configuration file, but default values, along with a default configuration file, exist for the generic DM kernel. The name of the configuration file is based upon the instantiation name of the DM system, providing extensibility and flexibility. That is, the DM may be executed with a different name to get different configuration values.

If the DM kernel is to be used in a generic way, then the ontology chosen for the naming of these DM variables becomes important. That is, the Topics developed for a new

DM that uses the existing kernel will create dialogues that will start to use these variable names. If these names are hard wired into the code, the flexibility of the overall system is reduced. Therefore, the <**getvar** *name*="xxx"/> and <**setvar** *name*="xxx"> tags can be used to set up the appropriate variable names and their values. This can be done by filtering an initial string through the `replaceXMLVariables` method. For example the string:

```
"<dev/null><setvar name="dialogueManagerXXX">YYY</setvar></dev/null>"
```

This will suppress the actual generated string due to the <**dev/null**> tags, and then set the value of a DM variable with the name 'dialogueManagerXXX' to be 'YYY'. This variable can now be used within any string that needs to be filtered by using:

```
Here is what you want <first_name/>:<getvar name="dialogueManagerXXX"/>
```

Any ontology can be used with the DM kernel in this manner. Tags such as <**first_name**/> can also be defined in this manner although it is normal to set them via a call to an XMLVariable class method that adds that (key, value) pair to a DM dictionary.

Unfortunately, due to the development of the system, some legacy code exists within the kernel that references tags such as <**first_name**/>. The removal/replacement of these is necessary for the DM kernel to be truly generic and is seen as future work.

As user's started exploring the ***Mentor*** System, they expected that it would have human-like qualities and interests. When the DM kernel was used in other applications, especially those involving Talking Heads, users also expected the system to have preferences or even a personality. Therefore the system variables were expanded to include knowledge about these preferences. For example, Figure 4.8 shows the specification of the Dialogue Manager's favourite song, and could be used to respond to the user's request of "What is your favourite music?". By using the <**setvar**> - <**getvar**> mechanism, arbitrary key-value pairs can be created and used.

```
dialogueManagerFavouriteSong =
   <random>\
      <li>It's a Miracle by Roger Waters on Amused to Death</li>\
      <li>Europa by Carlos Santana</li>\
      <li>Made Again by Marillion</li>\
   </random>
```

Figure 4.8 Atomic Tag for a specific DM preference

See Section 4.2.6.2 - VHML: Issues - on page 168 for more information on language markup issues arising from users' expectations and needs for a DM system.

Finally, when a user connected to the DM, the variables that they use in the interaction are first set from the DM configuration file, and then possibly over-written from a file based upon their user name. If this file does not exist, then a default user file is used. This hierarchy allows for system extensibility. See Section 4.2.5.3 on page 163 for more information on the format of the variables file.

3. Dialogue Manager Classes

Figure 4.9 shows the various Java class files of the Dialogue Manager kernel.

`DialogueManagerInterface`	`DialogueManagerServer`	`DialogueManagerConstants`
`DialogueManagerMOTD`	`DialogueManagerUserQuestion`	
`DialogueManagerListener`	`DialogueManagerSecurityManager`	`DialogueManagerQueryTimeout`
`DialogueManagerAdapter`	`DialogueManagerActiveQuery`	`DialogueManagerSystemCalls`

Figure 4.9 Dialogue Manager kernel class files

The first three specify the API Java Interface for the server and DM. Any DM or DM server instantiated must implement these Java Interfaces.

In the second group of Figure 4.9, the `DialogueManagerMOTD` class enables a general messaging system for Message of the day notification as well as user to user messages. It also supports delayed answers returned to the user from an authoritative source (such as the Lecturer-in-Charge). These were presented to the user either when they first connected to the system or as a pro-active prompt. The form of the delayed answer was roughly: "Excuse me Fred, earlier you asked me XXX. Informed opinion says that the answer is: YYY". In keeping with the findings of Ehlert (2003a) - suggest rather than act - users were always asked first if they wanted to know the answer.

The `DialogueManagerUserQuestion` class facilitated the easy management of these delayed question-answers. Any question that could not be immediately answered was saved in an XML-based format in a file associated with that user. The Lecturer-in-Charge (LiC) then received notification of the question by being prompted by a GUI to supply an answer - see Section 4.2.7.1 on page 173 for more information about the use of this class to support the Authoritative Guidance paradigm.

The third group of classes in Figure 4.9 - `DialogueManagerQueryTimeout`, `DialogueManagerSecurityManager`, and `DialogueManagerListener` - are support classes for checking to see if the user has simply ‘gone away’ without closing the client, for security management, and for generic Java Listener type callback facilities.

A *SecurityManager* was necessary once users were allowed to load their own Topics. These Topics were executed by the ***Mentor*** server, running as a process with the researcher’s privileges and file access capabilities. Therefore, before local Topics could be allowed, a Java *SecurityManager* had to be installed so as to disallow intrusive or dangerous behaviour. Otherwise, a user Topic could simply call `System.exit()` to terminate ***Mentor***.

It is unfortunate that the Java Virtual Machine (JVM) does not currently permit per-*Thread* security. If this were the case, then each connection *Thread* could inherit a standard *SecurityManager* that restricted most insecure operations. By using a *ThreadGroup SecurityManager*, a system developer could instantiate all problem *Thread*s under an ‘insecure’ *ThreadGroup* and all trusted code under a ‘system’ *ThreadGroup*.

The current *SecurityManager* uses the JVM stack and the *Class* Context to determine what operations are permitted. If a user-developed Topic performs some restricted operation, a *SecurityException* is thrown which is caught by the connection instantiation code and a response is sent back to the user indicating their ‘error’.

The final group of classes in Figure 4.9 - `DialogueManagerActiveQuery`, `DialogueManagerAdapter` and `DialogueManagerSystemCalls` - supported the pro-active queries, the normal question-answer process, and the DM System calls respectively. Each client connection used an instantiation of the first two classes.

The DM system does not just respond to user's requests. It is also pro-active in asking the user things. For example, it may ask the user if they have started their assignment yet, and if not, it may ask if they want any help. If they say "*yes*", then the system can then lead them down a structured learning path - a Directed Learning Path.

The `DialogueManagerActiveQuery` class checked all Topics that the user had as part of their environment, to sum up the weighting of all pro-active Topics. A Topic registered its intent to be pro-active by setting its `topicsActiveQueryWeight` field to a non-zero value. The larger this value, the larger the share of pro-active responses it wanted to get. The standard value of 1 indicated 1 share. If there were N pro-active Topics, then setting a value of 1 indicated a likelihood of 1/N of being chosen.

These pro-active Topics were added by the DM to a list of candidate Topics available for periodic selection. The `DialogueManagerActiveQuery` extends *Thread* and so can asynchronously sleep and then choose a Topic to supply the pro-active response. It chose the Topic randomly from the candidate list based upon the total weighting from all the Topics and the individual weighting of a Topic. It invoked the Topic's `topicsActiveQueryInitialise` method for initialisation and then called the Topic's `getTopicsActiveQueryResponse` method to obtain a response. This response could be null, in which case the random selection process continued until all candidates had been used or a non-null response was obtained.

The valid response was then sent to the user when there was a lull in the dialogue between the user and ***Mentor***. The response values were stored in the various DM history structures and any Next States (see page 122) were set for processing any subsequent responses from the user. Finally, the fact that a pro-active response had been sent to the user was saved in the user metrics file. The `DialogueManagerActiveQuery` class then continued, waiting for the next timeout so as to send the next pro-active query to the user.

Note that the timeout can be set by the user and is dynamically adjusted so as to minimise unwanted pro-active responses. Users could also disable Topics that they found annoying. One issue is that there is currently no user mechanism to reduce the weight of specific Topics. That is, a user may want to be asked "*Do you want to hear a joke?*", but only very infrequently, whereas they may want a pro-active Topic that tells them facts about a subject relevant to their immediate task, to be very active.

Another issue for future research is that of enabling the use of Topics which can alternate from being pro-active to passive. For example, a useful addition would be a 'go and find out about this' Topic (call this - gafoatTopic). That is, the user asks the system to look for some information and to let them know when it has been found. The gafoatTopic would initially be non-pro-active, would search for the required information, and would signal its pro-activity when it had found it by setting its `topicsActiveQueryWeight` field to

some non-zero level. The value of this could be used to indicate the urgency of the information. Notification would still be subject to the user's preferences though.

Currently this gafoatTopic would need to initially indicate that it was pro-active but decline every request through its `getTopicsActiveQueryResponse` method until it had been asked to find information and had subsequently found it. Since the total weighting is calculated once only per-user by the `DialogueManagerActiveQuery` class when instantiated, the Topic would need to know how urgent the information was before it was asked to get it. If it assumed a large value for `topicsActiveQueryWeight`, it would be asked for a pro-active response quite frequently.

Improving this dynamic functionality for the `DialogueManagerActiveQuery` class is seen as future work. One solution to this problem could involve the `activateInterruption` method of the `DialogueManagerSystemCalls` API which is used to send an immediate Response to the user. It is similar to an Active Query but happens immediately not at some time in the future based upon randomness and selection:

```
Response response = new Response (this,
      "<a href=\"http://www.cs.curtin.edu.au/\">Link</a> Hi there boss!");
response.addToNext_Query_State(topic_name + ":checkIfXXX");
DialogueManagerSystemCalls.activateInterruption("TopicName", context.dm,
                response, "Header string to this interruption\n");
```

The gafoatTopic could use this to report back to the user once it had found the required information. Again, for this very explicit interruption, the Topic should always ask first to see if the user wants the information.

The `DialogueManagerAdapter` class of Figure 4.9 is the core of the DM, and contains the code for:

- processing the type of request from the client,
- forming versions of the user input for subsequent parsing,
- processing the user input through the Topic REs.
- ranking each of the Topics' responses, and sending the highest ranked to the user,
- general house-keeping of user history, client information, etc, through accessor/mutator methods,
- support methods for performing typical DM tasks such as loading and verifying system, user, unit and Specialist Topics,
- generating and saving user metrics such as input, input time, response, etc
- handling initial user greetings and tidying up when the user disconnects.

There are two API entry points in the `DialogueManagerAdapter` class for processing the dialogue with a user. The first - a general entry point - initialises various sub-systems such as for pro-active queries, user variables, greeting the user, etc, then loops getting input from the user, passing the input line to the Topic's parsing mechanism through the second API entry point, and then returning that response to the user. This is the typical method used by generic DM systems such as SADM (see Section 4.2.2.3 on page 142). The second API entry point is used by DMs such as ***Mentor*** (see Section 4.2.3.2 on page 147) that perform their own initialisation and I/O.

The requests made by a ***Mentor*** client are not just concerned with parsing input. In addition, the ***Mentor*** server does not just send back responses to the client. A request protocol exists that enables efficient processing of the needed client-server information exchange. For example, one protocol indicates that the data packet that follows contains a Serialised Vector of user Topics, another protocol requests that the list of this user's friends be sent from the server to the client so that the client interface can show these. Table G.3 in Appendix G.4 on page 357 shows the current list of symbolic protocols. One issue is that currently these values are specified in a ***Mentor*** Protocol Constants class, not the DM kernel.

The Dialogue Manager kernel also contains two other legacy dependencies to the ***Mentor*** system. That is, it imports two packages from the `local.mentor.*` package hierarchy. This is due to the incremental development of the system.

Firstly, the kernel references the `Passwd` and the `PasswdEntry` classes from the `local.mentor.application` package for password information about a user. These should be relocated into the `local.raytrace` hierarchy. Initial thoughts would place this in a separate package (`local.raytrace.password`) rather than the kernel package.

And secondly, the kernel also obviously references Topics which are classes from the `local.mentor.topic` package. These classes could easily be relocated to the package `local.raytrace.Topics`. The Topics specific to the ***Mentor*** system are already segregated into different Topic folders such as units, Specialist and unitClasses. That is, the ***Mentor*** server specifically loads these relevant Topics from different places other than the base level Topics. The relocation is also seen as future work.

The second API entry point in the `DialogueManagerAdapter` class is concerned with the straightforward parsing and responding to the user input (see the English algorithm of Figure 4.10). A 'Next State' (see page 122) may exist to cater for the state-based nature of the DM, and will have been registered by the Topic that supplied the last Response.

```
Rationalise the input into lower case, remove punctuation
Construct a GeneralParse instance in user's Context from this new input
If a Next State registration exists,
    test and rank the input match against all Next States for that Topic
Test and rank the input match against all Topics in that user's environment
If no match was found, then
    if a delayed answer is available, set response to no match plus
        say sorry, plus indicate delayed answer.
    else if the number of no matches is >3 then
        set response to no match, say sorry and try for pro-active query
    else    set response to no match in nice English.
else
    update history, set any Next States, set match result
    set selected Topic's name
    call selected Topic's tidyUp method
    notify all DialogueManagerListeners of the final response
    set the response from selected Topic
Return the response.
```

Figure 4.10 Dialogue Manager request-response algorithm

A Topic returns an instantiation of a `Response` class. If the Topic does not match, then the value returned is a **null** `Response`. The `Response` is the only continuity

between successive calls to Topics. That is, at the end of the matching and weight ranking process, a single best-match `Response` has been selected and this must provide all information about the response string, the weight, the name of the Topic, Next States, any post-processing of the response or of the user state, etc. This `ResponseData` information is held in a field of the `Response` class.

Of importance, the `response_weight` field of the `ResponseData` information is a floating point number between 0.0 and 1.0 inclusive that is calculated by each Topic when they process the input. The value 0.0 means **never** choose this response, 1.0 means always choose this response, and a typical value is 0.75. After each Topic has processed the input, the response with the highest `response_weight` is chosen. Note that since a response history is available to all Topics, some Topics may choose to reduce their `response_weight` if they have recently answered a similar request. This gives other Topics a chance to present alternative answers. Obviously if no other Topic has any relevant information, even a low `response_weight` will be chosen.

There is normally an 'I don't understand you' Topic that is the last Topic to process the user input: `IDontUnderstandYouTopic`. Its response is typically "*I don't understand you*", and its `response_weight` is UNKNOWN_RESPONSE_WEIGHT = 0.001F. As long as a Topic does not set its `response_weight` below this, the system will work as expected.

From this, it can be seen that the algorithm of Figure 4.10 will always produce a match - at the very worst, it will be the `IDontUnderstandYouTopic`. The test for matching uses both the non-null value of the response returned from a Topic as well as a response status that indicates, amongst other things, the Topic that produced the final response. If this is the `IDontUnderstandYouTopic`, then the 'no match' test is assumed true.

However, a final response value could be **null** since it is possible for a user to disable the `IDontUnderstandYouTopic` and a **null** response might be returned from every Topic. Similarly, if the kernel is used in other applications, the configuration file (see Section 4.2.3.2 - ***Mentor*** Server Configuration - on page 151) for the application may disable the `IDontUnderstandYouTopic` so that the application can supply its own.

The `response` field of the `ResponseData` class is a VHML-tagged *String*, and it is this that is transformed by the XSLT classes and sent back to the user as the answer to their request. The `topic` field holds a reference to the Topic that produced this response. This enables the DM kernel to invoke that Topic's `tidyup` method so that the Topic can, for example, save information, adjust response values, and record user state information.

Associated with the `Response` is a list of potential Next States and Previous States. The latter is currently not used but is there for generality and possible future work. The algorithm for checking the user input against Next State Topics is shown in Figure 4.11. Note that the Next State *Vector* is not a *Vector* of Topics but of simple *String*s. This mechanism allows any Topic to register any other Topic as a possible Next State candidate. For example, some system Topics resolve ambiguity in user input by asking the user a question and then registering separate Topics that could process the followup user answer.

```
For each String in the Next State Vector
  Extract the topic name to get the Topic class
  If that Topic's checkValidState method returns true then
    Get a response from that Topic's checkFollowupResponse method
    If response is not null
       Adjust weight by scaling factor.
       If weight is > current final response
          Set final response to be this response
       Endif
    Endif
  Endif
Endfor
```

Figure 4.11 Dialogue Manager Next State processing algorithm

This mechanism provides flexibility but the use of *String*s allows errors in Topic Naming to occur. The system reports to the ***Mentor*** administrator any Topic names that are specified but cannot be resolved to Topics.

The Topic's `checkValidState` method is called and must return **true** if the Topic is to be considered for testing the input. The current Topics simply return a **true** value from this method. It is included for other DM systems that may use Previous State mechanisms to indicate the validity of further input processing.

The value of the scale factor that is applied to the Next State responses is defined in the system configuration file (see Section 4.2.3.2 - ***Mentor*** Server Configuration - on page 151). It is used to increase the likelihood that this followup response will be chosen rather than one from a Topic that processes the input with no previous knowledge.

Once the Next State processing has occurred, with potentially a weighted response being generated, all other Topics process the input as per the algorithm of Figure 4.12.

Each Topic creates a key RE that is used to match the Topic's domain: a cat Topic might use '`.*\bcat\b.*`'. Topics that process input that can be specified in many different ways can simply use '`.*`' as the keyword RE as this will match everything (see Section 4.2.2.1 on page 134). The Topic's inherited `checkArea` method was used to perform an initial key RE pattern match to see if the input is relevant to that particular Topic.

If the key pattern does not match the input, the `checkArea` method also tests the `object` field of the `GeneralParse` field of the user's `Context`. This is a mechanism included for future, more sophisticated, input processing but it is currently used by code for re-parsing ambiguous input. It is also envisaged for future better processing of anaphora.

If the key pattern does match then the (possibly null) response returned from the `checkTopic` method is ranked as per the Next State processing. At the end of processing all Topics, a final response with the highest weighting is returned to the calling method.

```
For each Topic
  If that Topic's checkArea method returns true then
    Get a response from that Topic's checkTopic method
    If response is not null
       If weight is > current final response
            Set final response to be this response
       Endif
    Endif
  Endif
Endfor
```

Figure 4.12 Dialogue Manager input processing algorithm

The user input processing within a Topic is covered in Section 4.2.2.1 on page 134. Listings of example Topics can be found in Appendix H.

The `DialogueManagerSystemCalls` class in the final group of classes in Figure 4.9, encapsulates all the DM system call methods that allow such functionality as secure input-output to files and network connections, the sending of an immediate `Response` to the user, and the authenticated caching of non-local Web pages (since access to the Web at the researcher's site is through an authenticating proxy firewall). Since the DM system uses a Java *SecurityManager*, any attempt by a user's local Topic to use direct input-output, etc will cause a *SecurityException* unless these methods are used.

Base Packages Issues

Several issues arose with the development and use of the Base Packages. Some issues were addressed - some in an *ad hoc* manner - and some were left as areas for future research.

One issue that has to be addressed since both the University and Australia are multi-cultural societies is that the database used for both name and gender classification is predominantly European. This is seen as future work.

Instantiation of a per-client XML transformer was seen as a good solution to enable the client to control the rendering of the VHML response. This required the transformer to be part of the DM instantiation, and to be used for each user's response transformation.

Legacy code use of DM variables was not seen as an important issue, especially given the evolution of the Dialogue Manager Markup Language (DMML) subset of VHML. More importantly, the two legacy packages of `local.mentor.application` and `local.mentor.topic` should be moved to a `local.raytrace` hierarchy, although there is no detriment to current usage.

The development of a 'no match' Topic that recognises and reformulates search engine type requests such as a simple "xxx" into something like "help me with xxx" is seen as future work and can build upon the existing re-parsing mechanism of the system.

In keeping with the findings of Section 2.3.2.2 on page 80, the complexity of the 'no match' processing of Figure 4.10 was designed to make the system seem more helpful even when it could not understand what the user requested. That is, the system tried to provide some information to the user, either from a previous delayed answer if it existed, or from a pro-active Topic.

Although this technique would appear to be beneficial, and from informal feedback most users found it so, it has the potential to be detrimental to the overall acceptance of the system. For example, consider the scenario where a user is desperately trying to get useful information about completing an assignment and the system says "sorry, don't understand, but here is a joke you might enjoy". The pro-active response from the `jokeTopic` would probably be most unwelcome at that stage.

A possible solution to this issue would be to classify pro-active Topics as being helpful to students or not. The pro-active response could then be chosen from a helpful Topic rather than any Topic. However, many units that the user is currently enrolled in could offer helpful

information. The user is again likely to view information from a graphics Topic as unwelcome if they are seeking information for a systems programming problem.

A more robust solution would be to restrict the pro-active information Topic sources to those currently being 'used' by the user. For example, if systems programming Topics have recently been supplying responses, then these could form the set of possible pro-active sources. However, if the user is having problems obtaining a proper response from the system, then no recently successful Topics may exist. The proper integration of a solution to this issue into the kernel is seen as future work.

Another issue that arose was the timeliness of delayed responses from the LiC (`DialogueManagerUserQuestion` class, page 118). Since a human is already in this process loop, it is assumed that dated requests can best be evaluated by the LiC to determine the optimal response action.

There were many issues associated with pro-active responses. One issue was that a user-defined weighting for certain pro-active Topics may be beneficial. This would require that the DM instantiation should maintain over-riding weights for any Topics in the user's environment.

Another issue was that of enabling the use of Topics that can alternate from being pro-active to passive. One possible solution is for the DM to enable a third type of Topic that registers itself for the periodic execution of one of its methods so that it can check to see if new information has become available. This could open the way for Topics to be more responsive to users' enquiries in an efficient manner.

One interesting issue that emerged involved the anthropomorphising of ***Mentor*** by users: they expected the system to have likes and dislikes, or even a personality. Aspects of this issue are discussed further in the Data Analysis Chapter. The DM kernel has calls to 'personality' methods (not shown in the algorithms of Figures 4.10, 4.11 and 4.12) that can be used to modify the `Response` returned from a Topic based upon the DM's personality. Just as the user may load their own local Topics, they can also load their own personality into the DM. The default DM personality does not alter the Topic `Response`. The base `Personality` class also implements the `TopicsInterface`, and hence the DM will also call the `checkTopic` method of any loaded personality so as to enable it to process user input like any other Topic.

A final issue was the classification of each pro-active Topic as being essential to student learning or as just being useful in their environment. One solution would be to categorise pro-active Topics into groups such as Units, Entertainment, Educational, and to allow users to select those groups that they have an interest in, through the Preferences facility of any client they are using to interact with the DM.

However, this solution could force a 'learning usage' onto the user regardless of their real use of the DM system. For example, the DM usage by Holic (2004a) did not involve learning at all, but required the multi-modal nature of the DM system to aid in helping the Aged. These pro-active issues are all seen as future work for integration into the Base Packages.

4.2.1.2. The *Mentor* Packages

These packages (Tables 4.2) have been designed and implemented to create the ***Mentor*** System. Many of the classes use or extend classes from the Base Package. Table 4.2 shows a synopsis of the components of this package, with more detail being given in the following sections. The full documentation for these packages is available via Appendix K.

Package	Description
`topics`	contains the various ***Mentor*** topics that make up its Domain Knowledge. This is used by local user topics, Specialist topics, unit topics and Network Builder topics.
`clients`	contains the various ***Mentor*** client applications.
`userModels`	contains the various user models for the client-server interaction.
`applications`	a data-mining package: most of the classes run applications to get information such as user information from the password file, printer and machine usage, timetabling and lecturer information.
`daemons`	contains the various ***Mentor*** Daemons. These periodically instantiate a relevant class from the `application` package.
`util`	contains various GUI utility classes including the new HTML client interface.
`mentor`	contains all the support classes for the clients and server as well as the network protocol classes.
`mentord`	contains the ***Mentor*** server and Dialogue Manager interface as well as a class to handle the ***Mentor*** System variables.

Table 4.2 ***Mentor*** Packages

`topics` **Package**

See Section 4.2.2.1 on page 134 for detailed information.

`clients` **Package**

See Section 4.2.4.1 on page 154 for detailed information.

`userModels` **Package**

See Section 4.2.4.1 on page 155 for detailed information.

`applications` **Package**

Most of these information providers are grouped into two or three classes: one class will coordinate the information retrieval whilst the second will parse and store it, and potentially a third may use that information to get further dynamic status information.

One important issue is that these classes are the most fragile in the system. That is, they are heavily dependant upon information in a known format in a known place. These classes are also heavily dependant on an underlying Unix environment. The SADM (see Section 4.2.2.3 on page 142) has been run in a Windows environment but no validated password information is available for any connecting user or for any enquiry about other

users. In this case, the password `Daemon` (see next section) reacts gracefully and returns 'unknown' values about users. Since the Lightweight Directory Accessing Protocol (LDAP) is becoming the *de facto* standard for password information, the password processing classes were migrated to use LDAP and this was beneficial in the Windows environment as well.

See Appendix G.2 on page 356 for more information about the `application` package.

`daemons` Package

The daemon classes are intelligent information cachers (Table 4.3). Typically, they periodically instantiate an `application` package class of similar name so that this information is available with very low latency to the system code and especially to Topics. Otherwise, the overhead of accessing the password file or the LDAP password information every time user information was requested would be very high. The daemon can provide this cached information readily and it can be updated as often as is appropriate.

`Daemon`	`base class, all other daemons extend this.`
`PasswdsDaemon`	`caches password information`
`NamesDaemon`	`caches data on male-female names probability`
`DomainNamesDaemon`	`caches language dependant messages`
`UnitsDaemon`	`caches unit information (title, code, abbreviation)`
`NetgroupDaemon`	`caches student enrolment information`
`TopicsDaemon`	`caches system, unit and Specialist Topics`
`PrintersDaemon`	`caches status of all printers on local network`
`GlobalFingerDaemon`	`caches status of all users on local network`
`RuptimeDaemon`	`caches status of each machine on the local network`
`ImportantDatesDaemon`	`data-mines the university Calendar Web site`

Table 4.3 daemon Package classes

The daemon information is available to any Topic via a ***Mentor*** system call:

```
public static Object getDaemonInformation(String name)
```

The daemon name is passed in and the returned object can be cast to the appropriate class. For example to get a password entry, a Topic may use:

```
passwds = (Passwds)MServer.getDaemonInformation("Passwds");
passwdEntry = (PasswdEntry)passwds.passwdentries.get(userName)
```

Hence the system can access static or slowly changing data with very small latency.

These daemons are instantiated and executed as *Thread*s when the system starts, but can be suppressed via the ***Mentor*** System Resource configuration files (see Section 4.2.3.2 - ***Mentor*** Server Configuration - on page 151). Except for the `PasswdsDaemon` that, as mentioned fails gracefully, if the daemons are enabled but do not instantiate and run correctly, this will be logged via `Syslog` and the ***Mentor*** System will not start.

`util` Package

This package contains utility classes for the client `TalkConsole` interface as well as for the interface *Dialog* windows. For a detailed description of the `TalkConsole` interface, see the description of ***Mentor*** clients on page 157.

The *Dialog* window classes are instantiated by the client to present messages or information to the user. For example, the `MentorClient` allows a user to send messages or to report problems via email to the ***Mentor*** Maintainer. The system also allows a user to send messages to other users. Note that this is not a general email messaging system, users are specified by name - either username or via their normal name.

Although peripheral to the thesis objectives, these facilities were seen as important since it was hoped that users would see the ***Mentor*** System as a useful desktop facility.

See Appendix G.5 on page 358 for detail on the various dialogs supported.

`mentor` Package

The `local.mentor.mentor` package (see Table 4.4) is a general support package for the ***Mentor*** System. It could be argued that the first group in Table 4.4 - classes that extend *Thread* and are used to send and receive data from a client to the server - could be relocated into the `local.mentor.client` package or even into the Base Package hierarchy, and future investigation may decide that this is appropriate. This, and other relocation of packages issues have arisen due to the incremental development nature of Action Research.

`EchoReceive`	`EchoSend`	`EchoCharSend`	`EchoCharReceive`
`AsynchReceive`	`GhReceive`	`GReceive`	`GSend`
`MentorRequestClass`		`MentorLoadClass`	
`MentorSubmitClass`		`MentorCheckSubmitClass`	
`MentorProtocolInterface`		`MentorProtocol`	`MentorConstantsInterface`
`MentorTransitPacket`		`MentorMessageListener`	`MentorMessageRegister`
`MentorStream`		`MentorUserMessage`	`MentorConsole`

Table 4.4 ***Mentor*** Package classes

Some classes from the first group take input either from standard input or as character events in a graphical window and send it via the client to the server. Other classes take input from the server and either output it to standard output or display it via the client in a graphical window. The normal client that is used to connect to the ***Mentor*** System - `MentorClient` - instantiated an `AsynchReceive` class which read the protocol request from the server and passed that plus the Java *InputStream* to the `MentorClient`'s `processClientCommand` method (see Section 4.2.4.1 on page 155 for more information). The `MentorClient` then processed the rest of the incoming data packet dependant upon the protocol request.

Of the second group of classes in Table 4.4, only the `MentorRequestClass` is of importance to the normal operation of the system. The client-server connection to the ***Mentor*** System - an instantiation of the `MConnection` class - extends the `Connection` class. The `Connection` class, which adds little except an abstraction layer for future development, extends the abstract `MentorRequestClass` class. This abstract class, which implements the `Runnable` interface, extends the `DialogueManagerAdapter` class from the Base Packages. Hence a user `MConnection` inherits all of the functionality of the DM kernel `DialogueManagerAdapter` class.

The abstract `MentorRequestClass` class contains low-level fields concerned with the execution of the DM instantiation. These include various error status fields, process identifiers, *Thread*s, and *Thread* and *ThreadGroup* names. It also uses the *Runnable* functionality of the class to initialise and execute the DM instantiation (see Figure 4.13).

```
public int start()
{                                          // new connection to Mentor
    if(mentorRequestThread == null)
    {
        mentorRequestThread = new Thread(mentorRequestThreadGroup,this,
                                           getClass().getName());
        mentorRequestThread.setPriority(mentorRequestThreadPriority);
        mentorRequestThread.start();
        Syslog.debug("Pid " + Format.Sprintf("%3d",child_id)
                       + " Th'd: " + mentorRequestThread.toString() );
    }
    return child_id;
}

public void run()
{
int exit_status = 0;
    try
    {
        exit_status = execute();           // execute the DM instantiation
    }
    catch (Throwable e)                    // Any errors occurred??
    {                                      // If so, log them
        MentorRequestErrorStatus = -1;
        MentorRequestErrorString = e.toString();
        MentorRequestException = e;
    String s="Throwable from MentorRequestClass(" + userName + "):"+
             local.raytrace.Util.Util.stackTraceToString(e);
        Mentor.errorlog(s);
        System.out.println(s);
    }
    MServer.childHasDied(child_id, exit_status); // Connection has closed
}

public abstract int init(String argv[]) throws Throwable;
public abstract int execute() throws Throwable;
```

Figure 4.13 The `MentorRequestClass` Runnable interface

The `start` method can be seen to create a new *Thread*, set its priority and then start that *Thread* running. The new connection to ***Mentor*** is also logged for accounting and evaluation purposes.

When the *Thread*'s `run` method is invoked, it calls the `MentorRequestClass` `execute` method and stores the value returned. It also catches any *Throwable*s, which includes all *Exception*s, and logs these problems in various ways. Finally it indicates that this connection from the client has finished - the user has disconnected - and that the child process has died.

Note that the two abstract methods - `init` and `execute` - are used to control the actions of the DM instantiation. For the ***Mentor*** System, the `init` method is not used but the `execute` method follows the description of Figure 4.14. This illustrates the process that occurs when a client connects to the DM system for **any** service (not just for ***Mentor***).

```
public int execute()
{
    try
    {
        validateRequest();
        if(valid)
        {
            get requestType
            get Properties
            if (not valid Properties) send error back to client
            else
            {
                get client arguments
                if (not valid arguments) send error back to client
                else process requestType by calling appropriate DM methods

        }
    }
    catch(IOException e)
    {
        log error status
        exitStatus = -1;
    }
    finally
    {
        tidy up
    }
    set status of child process
    return(exitStatus);
}
```

Figure 4.14 The algorithm for ***Mentor***'s `execute` method

This may seem a very round about way to instantiate the DM but it was designed so that ***Mentor*** is just one application that can be run from the DM server system. Other applications that extend the `MentorRequestClass` can also be loaded and executed by the server system. Since the server system controls the daemons, these resources can be used by any loadable applications such as a dedicated printer monitor or a disk usage monitor that have been developed. These loadable applications use the `MentorLoadClass`, `MentorSubmitClass` and `MentorCheckSubmitClass` classes of Table 4.4.

The third group of classes in Table 4.4 holds client-server protocol values as well as methods for turning these values into symbolic names for logging/debugging. The `MentorConstantsInterface` class holds values that define constant values such as the base directory where user data files are stored, the type and/or the machine name for logging `Syslog` information, and the URL for the ***Mentor*** System Web site.

The fourth group holds definitions and methods for the processing of the in-transit data packets, and the message listeners that use them. These are mainly used by client `Dialog` classes to maintain user state information.

The final group in Table 4.4 contains `MentorStream`, `MentorConsole` and `MentorUserMessage`. The `MentorStream` class extends *PrintWriter* so as to isolate any underlying implementation from any code that uses the client-server channel. The `MentorUserMessage` class encapsulates a ***Mentor*** user message so that both client and server can process these, again without knowing structural detail. In a similar way, the `MentorConsole` class hides implementation details about a generic client user interface.

`mentord` Package

This package is the heart of the ***Mentor*** System server and ***Mentor*** DM sub-system. The first group of classes in Table 4.5 forms the basis of the server - see Section 4.2.3.2 on page 147 for detailed information on the algorithms involved.

`Mentor`	`Server`	`MServer`
`MentorVariables`	`ThreadLister`	
`Connection`	`MConnection`	

Table 4.5 mentord Package classes

The `MentorVariables` class of the second group extends the `XMLVariables` class from the Base Package - see Section on DM and User Variables Classes on page 115 for information on the function of this class. This class also provides ***Mentor*** specific loading and processing of DM and user variables files.

The `ThreadLister` class of the second group is based upon that class in 'Java in a Nutshell' by D. Flanagan, and is © 1996 O'Reilly & Associates. It provides logging and debug methods for listing all *Thread*s and *ThreadGroup*s in a running Java application.

The third group represents the user connection to ***Mentor*** - each user gets an instantiation of an `MConnection` to handle the interactive question-answer dialogue. Due to the importance of this class, aspects of its operation are detailed elsewhere in context (see for example Section 4.2.2 on page 134 and Section 4.2.2.2 on page 136).

Briefly, this class extends `Connection` which extends `MentorRequestClass` (Section 4.2.1.2 on page 128) and `DialogueManagementAdapter` (see Section on DialogueManager Packages on page 120) to form the DM instantiation. The processing of the interactive question-answer dialogue uses the second entry point of the `DialogueManagerAdapter` class (see Figure 4.10 in Section 4.2.1.1 on page 121). The constructor stores a reference to the server, sets the status of this connection, creates an XSLT transformer for processing the output (see Section on XSLT Classes on page 115), reads the user variables file (see details of this in Section 4.2.5.3 on page 163), as well as reading the user history file and creating a `Context` for the user interaction (detailed in Section 4.2.5.2 on page 162). Of importance is the algorithm to handle a user connection - see Figure 4.14 on page 130.

The class also has methods for network communication, user dialogue recording, processing local, system and user Topics, user Context handling, as well as the code for processing the various client requests shown in Table 4.6 in Section 4.2.4 on page 154.

Mentor Package Issues

The nature of the `application` classes meant that issues of portability arose for obtaining system information such as passwords and resource utilisation. The classes were isolated from the underlying Operating System as much as possible but extending this was seen as possible future work. The structure of the classes allowed the LDAP mechanism for accessing passwords and other information to be implemented in less than 4 hours.

Another `application` class issue that has caused intermittent problems is the changing URL and underlying information structure of the ImportantDates Web site. During an evaluation-refinement cycle, if the system fails to initialise, any recent minor developmental changes are suspected as the cause of the problem. Something as mundane as the arbitrary changing by central University technical staff of the ImportantDates Web site URL is overlooked even when the problem is recognised and logged as the ***Mentor*** System starts. And when the cause of the problem is realised, little can be done if the site is temporarily 'under construction'. This issue affects any code that uses data-mining of Web sites.

This, and similar issues concerned with the long-term stability of data-mined Web sites, has meant that many resources are duplicated locally or cached as files, 'just in case'. That is, most application classes can retrieve their information from a number of sources. For example, the current ImportantDates class will attempt to retrieve information from the Web site and if that fails, it will use a previously saved file image of the Web site. Similarly for other information sources such as for password, netgroup, ruptime and finger data.

Another issue is that network problems can cause the system to 'hang'. That is, users make a request, and the system seems to stop for them. Other users may not be affected. This is not due to the system dying but can be due to the system making a request to a daemon for information and the daemon has indicated that it is currently unavailable while it is updating its information.

Normally this would not be a problem as the daemon buffers information and then switches the new for old information in the unavailable period. However, some JVM's have very poor performance in accessing a method of a *Thread* that is currently blocked due to network I/O. For example, if `Finger` information was being updated, and the targeted machine was down, the network connection would time out after say 90 seconds. During this time, any requests to the `Finger` daemon *Thread* would simply block until the timeout occurred. And hence the system would appear to 'hang'.

Classes that access machines via the network, now 'ping' the machine with a single data packet first to see if it is up. Ping uses the Internet Control Message Protocol's (ICMP defined in RFC792 (Postel 1981a)) mandatory ECHO_REQUEST datagram to elicit an ECHO_RESPONSE from a host/gateway. The timeout can be set to a small value of a few seconds. In this way, machines that are 'down' will not cause the system to hang.

Due to the incremental development nature of the research, several packages, such as the sending and receiving classes of `local.mentor.mentor`, could be relocated into more appropriate hierarchies. This will maximise reuse of the developed classes, and is seen as future work.

The use of the `MentorRequestClass` as a base class run by the ***Mentor*** server, meant that other applications similar to the existing Dialogue Manager system, such as a dedicated printer monitor or disk usage monitor, could be developed by experienced users such as technical staff.

4.2.1.3. External Packages

Two other packages have been used extensively in this research:

- `com.oroinc.text.regex`
- `org.xml.sax` and `javax.xml`

Both packages were evaluated and adopted early in the research and have since been superseded due to being adopted by the Jakarta Apache project and Sun respectively, as the standard for further development work.

OROMatcher™, originally provided by Original Reusable Objects, Inc., is a set of regular expression (RE) pattern matching classes (see Section 2.3.1.2 on Pattern Matching through Regular Expressions on page 57) for Java. The package - `com.oroinc.text.regex` - is now part of the Jakarta Apache project. [HREF 46]

Appendix G.3 on page 357 gives more information about these two external packages.

External Packages Issues

The current ***Mentor*** System was modified to use the new API's and classes.

Section 4.2.1.3 and Section 4.2.2.1 details the use of, and the issues that arose from using the OROMatcher Perl-5 Regular Expression classes within the Natural Language Processing subsystem of the DM.

4.2.2. The Dialogue Management Kernel

At the heart of the ***Mentor*** System is a kernel that uses classes from the Base Packages, that is based upon Perl-5 Regular Expressions built around a request-response subsystem, and that uses Java classes called `Topics` to classify the user input and provide a VHML-based response from the Domain Knowledge base.

A description of the processing of a user request by the kernel can be found in the 'second API entry point' Section on page 121, although this did not describe the per Topic processing. Topics need to accommodate **initial**, **followup** and **pro-active** questions. The following sections detail this per Topic processing through example, and show how the system has been designed for use in real rather than research environments.

4.2.2.1. DM Regular Expression Processing - Topics

To determine what the user has typed, their input is matched against various REs by the Topics. All matches must be in lower case as the input line is converted to lower case. Each Topic will have an array of these REs. The first entry in the array is the general area to match - a keyword or key pattern. The DM checks to see if the input matches this particular key pattern. If not, then for efficiency, it does not look at any further matching for this Topic.

As an example to illustrate this, Figure 4.15a shows the start of the RE array for an example Topic. Remember that some REs need double escaping such as \\s and \\b due to Java *String* syntax. In this example RE array, a match against any input that has the word 'italy' or 'italian' (singular or plural) or the misspelt 'italien' (singular or plural) is required. This is achieved by an RE that is 'any characters (.*), then a word boundary (\b), then either the word italy, or italian or italien with optional 's' ((s)?), followed by a word boundary (\b), followed by any characters (.*)'.

```
protected transient String patterns[] = {
                ".*\\b(italy|italian(s)?|italien(s)?)\\b.*",
```

Figure 4.15a Matching against the keywords or key RE for a Topic

Figure 4.15b shows the subsequent REs that are tested by the Topic's `checkTopic` method if the Topic passes the keyword check. Note that the first RE only matches exactly the question "where is italy" not "where is italy?" nor "where's italy" nor "where is italy!"

```
"where\\s*is\\s*italy",
"where((')?s)|(\\s*is))\\s*italy", // + PUNCTUATION_STRING,
WHOIS_STRING + "\\s*(italian|italien)", //+ PUNCTUATION_STRING,
HELP_ME_WITH_STRING + "((learning|learn)\\s*(italien|italian))",
".*",          // match everything italian in a non-specific way
```

Figure 4.15b Subsequent matching against REs for a Topic

A better pattern for the above question is shown as the second entry since it matches 'where is', 'wheres' or 'where's' and allows spaces and punctuation at the end of the string. The uppercase PUNCTUATION_STRING is defined in the **super** class. In fact, punctuation as well as some common polite words such as 'please' and 'thank you' are removed as leading and trailing words from the input that is passed to the Topic so we do not really need PUNCTUATION_STRING. It has been left in as a comment in the example code.

The third entry matches the user question "who is italian" and uses a system defined RE: the WHOIS_STRING shown below. The '?:' modifier to the bracketed RE means that this RE should **not** be remembered for backreferencing.

```
public static final String WHOIS_STRING = "(?:(?:who)(?:(?:\s*is)|(?:'s)|(?:s)|s|se))";
```

The HELP_ME_WITH_STRING (fourth entry) is a complex, very generic 'help me with' type system defined RE and is commonly used for this type of question. Its definition can be seen in Figure 4.16.

```
public static final String HELP_ME_WITH_STRING =
        "\\b" +
        "(?:" +
                "(?:" +
                   "(?:" +
                            "(?:i\\s*(?:need|want))|" +
                            "(?:(?:can|will)\\s*" + YOU_STRING+ ")" +
                      "\\s*)?" +
                      "help\\s*(?:me)?\\s*(?:for|about|with|on|)"+
                ")|" + "(?:"+WHAT_STRING+"\\s*"+IS_STRING+")|" +
                "(?:where\\s*" +
                        "(?:" +
                                "(?:can\\s*i\\s*(?:get|find\\s*(?:out\\s*)?))|" +
                                "(?:"+IS_STRING+"\\s*(?:the(?:re)?\\s*)?)|" +
                        ")" +
                "\\s*info(?:rmation)?\\s*(?:about|with|on))|" +
                "(?:tell\\s*me\\s*about\\s*)|" +
                "(?:how\\s*.*\\s*(?:can|do|would|could)\\s*i\\s*(do\\s*)?)" +
        ")" + "\\s*(?:(?:my|the|a|an|)\\s+)?" ;
```

Figure 4.16 The system defined 'HELP_ME_WITH_STRING'

Finally, the last entry ('.*') can be used to catch all other questions that have matched the keyword. That is, the user has asked something about Italy, the keyword has matched, but it was not catered for in any of the subsequent entries. This allows the Topic to respond with something like: "I know you asked about Italy but I can't work out what you want. Sorry."

Figure 4.17 shows the Java constructor for a typical Topic. The **super** class is called with the unique Topic name, and two help strings are defined. The first is a one line string to use in listing what the system knows about. The second is more detailed, can use the normal DM variables and is used if the DM is asked for specific help about this Topic.

```
public ItalyTopic()
{
 super ("Italy");
 briefTopicsHelpString = "Italy - knowledge about Italy and its culture.";
 topicsHelpString="<getvar name=\"first_name\"/>, you may ask me about\n"+
                  "Italy, its culture, its food, its bad drivers!!\n" ;
 makePatterns(patterns[0],patterns);
}
```

Figure 4.17 Typical Topic constructor

It is necessary for the Topic to compile the key and subsequent REs into an efficient internal format. This can be done via the `makePatterns` method of Figure 4.17 which takes the key RE pattern - often patterns[0] but it could be any string - and the array of subsequent patterns. This specifically creates the required named array of Perl-5 Patterns in the super class with signature: **protected** `Pattern` pattern[]. There is also another version of this called `makeSpecificPatterns`(secondaryPatterns) for creating and returning other compiled RE's. These may be used to match followup responses from the user.

4.2.2.2. Dialogue Manager Request-Response Processing

Topics need to be able to process **initial**, **followup** and **pro-active** questions.

Initial Question Processing

The following code segments are given to illustrate the very important request-response processing loop of a Topic. As previously mentioned, after the initial keyword check, the DM invokes the Topic's `checkTopic` method. This is passed in the user `Context` and returns a `Response`. Figure 4.18 shows the algorithm for this code. The user request (trimmed, rationalised and converted to lowercase) is available via the `context.input` field and this is checked against successive RE Patterns to determine what the user wanted. The unadulterated user input (in the original upper or lower case and including punctuation, etc) is also available from the `Context`.

Figure 4.18 also shows typical naive matching and processing of a user request - if matched, a random relevant response is chosen from a *String* array. This plus an appropriate `response_weight` are returned to the DM. The user has asked a question about Italy and received a response.

```
    public Response checkTopic(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
        response = null;        // Indicate that default is no match....

        // Now check all patterns to see if they match the input string.
        // Note we have already used index 0 to get the general area of the match
        // so we normally check from index 1 onwards.
        for (i = 1; i<pattern.length;i++)
            if(matcher.matches(context.input,pattern[i]))
            {
                found = true;
                break;
            }

        if(debug)
                sendMessageToUser(context,"debug message to user");
                    :
        if(found)
            switch(i)
            {
            case 1:        // where is italy
                        // Get a random string within the response array.
                int wchoice = Math.abs(random.nextInt())% whereResponses.length ;
                    response = new Response(this,whereResponses[wchoice]);
                  // moderate weighting. 0<= weight <=1
                    response.responseData.response_weight = 0.75F;
            break;
                        :
```

Figure 4.18 Typical pattern matching method

Figure 4.19 shows the use of a `MatchResult`. That is, the use of bracketed REs to remember the partial matches. In this case, the result of the group 1 matching (i.e. the first bracketed match) is retrieved and tested to see if the user used the word 'italian' or 'italien'.

Given the Action Research nature of the development of the system, copious error checking has also been included in the Topics. Note that there are several levels of error reporting. In the `topicDebug` case of Figure 4.19, the error is simply logged to Syslog and the ***Mentor*** log file. Severe DM errors can bring the system down in a controlled manner,

whilst less severe but important errors will email the ***Mentor*** administrator. User errors are caught by the system and send error information back to the client.

```
case 3: // Lets also use a Match result to find out if they typed italien or italian
    MatchResult result = matcher.getMatch();
    String typed = result.group(1);   // Look at the first (xxx) remembered match.
      if(typed == null)
      {
        topicDebug(context,"Got a match but with a null MatchResult!"); // log it.
        return null;        // Strange since we did match but just in case..
      }
      if(typed.length() == 0)
      {
        topicDebug(context,"Got a match but with a 0 length MatchResult!"); //log it.
        return null;        // Strange since we did match but just in case..
      }
      topicTrace(context,"checkTopic : found match at position " + i);
      typed = typed.trim();        // get rid of extra white space
      if(typed.equals("italian"))    // english spelling
              :
break;
```

Figure 4.19 Matched request using bracketed expressions

Also note that the `sendMessageToUser` debug shown in Figure 4.18 is for the **user** and this is sent back across the network connection and appears as a transient dialog popup on the machine. The debug is also printed on their standard output. The `topicTrace` debug in Figure 4.19 is also for the **user** and is appended to the client's status line. This enables users to get debug information from the development of their local Topics.

Figure 4.20 shows a different mechanism to get a random response sent back to the user through using features of VHML. The text within the <**random**>.....</**random**> chooses a random one of the <**li**> items specified at the final 'transform VHML into appropriate text' stage. Note that this is XML and so all tags must be properly closed and attributes quoted. Note as well, that in this example, the user will get a URL hotlink as part of their answer and they may click on this link to open up a browser to that URL Web page.

```
case 4:     // help me learn italian...
  response = new Response(this,
    "Why not try looking at " +
     "<random>" +
      "<li><a href=\"http://www.wuziegames.com/learnitalian.html\"> \
            http://www.wuziegames.com/learnitalian.html</a></li>" +
      "<li><a href=\"http://confucius.gnacademy.org:8001/mason/catalog/search.html\
            ?p=italian&amp;xyz_max_hits=100\">the GNA academy.</a></li>" +
     "</random>"  );
  response.responseData.response_weight = 0.75F;  // moderate weighting
break;
```

Figure 4.20 Alternative VHML-based random response

Figure 4.21 shows the processing of the typical last entry in a Topic's Pattern array. The last pattern of '**.***' matches everything, so the Topic matched the area but not the exact input (remember that the initial keyword matched). In this case the `Response` class picks a unique random response from the specified *String* array, and it searches its `Response` history so as to suppress similar past responses. The `response_weight` value is also lowered in case a different Topic can supply a better response.

```
default:
   response = new Response(this, Response.uniqueResponseString(context.dm,eh));
   // low weighting. Something better may match
   response.responseData.response_weight = 0.5F;
}
                  :
  return response;    // return the final response. Could be null.
```

Figure 4.21 Matching the general area of the request

In general, the complexity of the matching and of the response depends upon the Topic. For example, the `weatherTopic` will intermittently access a website and data-mine the current weather information, the `whoTopic` will search for a user on the system first but if one matching the name cannot be found, then an online 'who is ...' website is accessed to get accurate information about the named person.

Followup Question Processing

The processing of the user request to get a response, as shown in Figure 4.18, is naive because the question-answer process rarely stops at one question:

User: what is OpenGL
DM: A 3-D graphics Language.
User: where do I get it?

The user has received the answer but then wants to continue the dialogue. The dialogue has moved into the next state. It is important to understand that the question "where do I get it?" only has relevance **after** the answer from the DM. Therefore, the Topic that supplied the answer has to register a 'Next State' interest in processing any further input from the user before any general Topic matching takes place.

When matching any input from the user, the DM first checks to see if any Topic has registered a 'Next State' interest and will invoke that Topic's `checkFollowupResponse` method. If this matches, then it gets a higher `response_weight` value than subsequent Topics that do an initial match. A Topic may register more than one 'Next State' by specifying a 'name' *String* and these named states are checked sequentially by the DM.

Consider Figure 4.22 which shows the Topic's `processCheeseRequest` method to process the response to the user's statement "I like cheese". This method has been called from the `checkTopic` method once the statement has been recognised. This method chooses a random response from the `doYouLikeRandomCheeseResponses1` array and also a random cheese name from the `cheeses` array (not shown).

The random response uses VHML tags to get the DM variable called 'first_name' - the user's first name - as well as a DM variable called 'subject'. This has been dynamically created by the Topic and has been given the value of the random cheese name. These DM variables are per-user. So, for example, the dialogue may be:

User: I like cheese.
DM: Me also Hans. Do you like brie de meaux?

The method also registers its interest by calling the `addToNext_Query_State` method and passing in a unique name, in this case, the topic name followed by a colon followed by some string. If and only if this Topic's response is sent to the user, then it will

have the first chance to process the next input from the user. In this case, they may answer "yes" or "no" or "never heard of brie de meaux", etc. It is up to the Topic to second guess what the user is likely to say so as to properly continue the dialogue. Obviously, if the user asks about something different, then that Topic's checking of the possible next state inputs will cause it to return a null or low weighted response for this followup user query.

```
////////////////////////////////////////////////////////////////////////////////
//      Current state Responses - they said "I like cheese"
// In reality, do you like <a random cheese name>
 private static final String doYouLikeRandomCheeseResponses1[] =
 {
    "Me also <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
    "Do you like <getvar name=\"subject\"/>, <getvar name=\"first_name\"/>?",
    "Me too, do you like <getvar name=\"subject\"/>?",
    "So do I <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
 };

 private Response processCheeseRequest(Context context)
 {
 Response response = null;
 int wchoice = Math.abs(random.nextInt())% doYouLikeRandomCheeseResponses1.length ;
 int cchoice= Math.abs(random.nextInt())% cheeses.length;
    response = new Response(this,doYouLikeRandomCheeseResponses1[wchoice]);
    response.addToNext_Query_State(topic_name + ":checkIfLikeRandomCheese");
    response.putMatch("subject",cheeses[cchoice]);

    return response;
 }
```

Figure 4.22 State based user request processing

Figure 4.23 shows the typical code to process a 'Next State' or Followup request. The `topic_string` is processed to get the 'Next State' name and this directs the processing of the user input. In our example, the `checkIfLikeRandomCheese` is invoked.

```
    public Response checkFollowupResponse(Context context, String topic_string,
                                                    Vector Prev_Query_State)
    {
    String state_string = topic_string;
    int    index = topic_string.indexOf(':');

        if (index != -1)
            state_string = topic_string.substring(index+1);

        if(state_string.equals("checkIfLikeRandomCheese"))
            return checkIfLikeRandomCheese(context);
        else if(state_string.equals("checkWhatTypeOfCheese"))
                :
        else
            return null;
    }
```

Figure 4.23 'Next State' or Followup processing of user input

Figure 4.24 shows the code and associated RE data for the method `checkIfLikeRandomCheese`. The method uses previously compiled REs to see what the user responded with. In this case, it could be "yes", "no", etc. If a match is found then an appropriate response is sent, and a new state is registered so that the dialogue can continue if the user wants. If no match is found, then the Topic indicates that it could not match the user input by returning a null response. Hopefully other Topics will match the user input on their initial testing.

```
//    checkIfLikeRandomCheese state Responses
// We need to ensure that this is compiled:
//        YESNO_pattern = makeSpecificPatterns(YESNOPatterns);
// The state processing method is classified in checkFollowupResponse and is called
// based upon the next state string - in this case "checkIfLikeRandomCheese"
protected Pattern YESNO_pattern[] = null;
protected transient final String YESNOPatterns[] =
{
     YES_STRING,
     NO_STRING,
     "(?:" + WHATIS_STRING + "\\s*(?:that|taht|(.*))\\s*)|" +
     "(?:(?:"+IVE_STRING+"\\s*)?\\s*never\\s*heard\\s*of\\s*(?:it|(?:(?:that|taht)\\s*)|(.*)))"
};
              :
private Response checkIfLikeRandomCheese(Context context)
{
PatternMatcher matcher  = new Perl5Matcher();
     // Now check patterns in YES or NO to see if they match the input string.
     if(matcher.matches(context.input,YESNO_pattern[0]))
     {
     int wchoice = Math.abs(random.nextInt())% whatOtherTypesResponses.length ;
         response = new Response(this,whatOtherTypesResponses[wchoice]);
         response.addToNext_Query_State(topic_name + ":checkWhatTypeOfCheese");
     }
     else if ......
                  :
     else
         response = null;
```

Figure 4.24 Subsequent State processing

Notice in the third and fourth patterns in Figure 4.24 the testing of a typical letter transposition for the word ‘that’. This is also catered for in the system defined WHAT_IS_STRING. The system defined YES_STRING and NO_STRING are defined as:

```
public static final String YES_STRING = "(?:\\s*(?:ok|yep|ya|yes|y|yea|yeah|sure))";
public static final String NO_STRING  = "(?:\\s*(?:no|nope|na|nuh|n|nix|nay|nah))";
```

One issue in matching this simple response is the evolving nature of English slang. The Short Messaging System (SMS) allows users to send messages of up to 160 characters between mobile phones. The upper character limit forces the use of cryptic abbreviations and these are being seen in user input to the ***Mentor*** System (see discussion on page 204).

Pro-active Question Processing

Some Topics can be pro-active instead of or as well as normally processing user input. Figure 4.25 shows a Topic which does not directly answer user requests but is pro-active **and** then also responds to any followup replies to the pro-active queries. Note that the initial keyword pattern should never match since all user input is checked as lower-case. Hence, in this example, method `checkTopic` should never be called. The Topic registers itself (`topicsActiveQueryWeight` = 1) as being a pro-active Topic and hence will periodically be considered for providing a pro-active query to the user. The Topic also compiles RE patterns for the Followup responses. The `checkFollowupResponse` method is as normal. Note that normal Topics can be pro-active as well.

All registered pro-active Topics are considered by ***Mentor*** for providing pro-active queries. Enabling a Topic to provide a pro-active `Response` does not guarantee that it will be sent since the choosing of a pro-active Topic is done periodically and randomly, and is based upon the user’s pro-active timing preferences. The weighting dictates the likelihood of being chosen. That is, a `topicsActiveQueryWeight` of 1 gives it one stake in the total random choice, a value of 2 gives it two chances, etc.

```
protected transient String patterns[] =
{
        "A",            // Upper case - never matches....
};
public didYouKnowTopic() throws FileNotFoundException
{
        super (TOPICNAME);
                        :
        topicsActiveQueryWeight = 1;
        makePatterns(keywordHelpPatternString,patterns);
        tellMeMore_pattern = makeSpecificPatterns(tellMeMorePatterns);
}
public Response checkTopic(Context context)
{
        return (null);
}
public Response checkFollowupResponse(Context context,
                        String topic_string, Vector Prev_Query_State)
{
                :
}
```

Figure 4.25 Typical pro-active Topic initialisation and processing code

Having registered itself as being pro-active, the Topic must be able to provide a `Response` when the `getTopicsActiveQueryResponse` method is invoked (Figure 4.26). Although pointless, the text of the `Response` could be as simple as a static *String*:

```
return new Response(this,"I bet you didn't expect this!");
```

In the more functional example of Figure 4.26, a file of information is opened once and cached, and a random fact is returned from a *Vector* of facts. Since users may respond to this fact, the Topic is also likely to register a 'Next State' interest.

```
public Response getTopicsActiveQueryResponse(DialogueManagerInterface dm)
{
        loadDidYouKnows(dm);
        return DidUKnow(dm,dyks);
}
```

Figure 4.26 Typical getTopicsActiveQueryResponse code

Topics that help users with information about a specific course unit often have pro-active sub-systems that ask if the user has started their assignment yet, or whether they would like help with their study. Those Topics would register to process 'Next State' queries so they can detect if the user has responded with a "yes", "no", etc. If "yes" then they can start the user on a dialogue path that helps them (see Section 4.2.7.2 on page 176).

In keeping with good Object Oriented programming principles, a Topic is defined by a `TopicInterface` class, and only classes of this and more restrictive types can be loaded into the DM. However, the cyclic development nature of the system, along with the use of early versions of Java that did not have Just-in-Time compilers, has meant that a legacy of **public** fields still exists within some of the class files and that accessor-mutator methods have not always been defined. The overall functionality of the Topic is however defined by these APIs, allowing a Topic to be extended by the system developer as well as by users.

A commented listing of example Topics can be found in Appendix H, and a commented listing of Topics that use the multi-modal interface can be found in Appendix I.

4.2.2.3. Stand Alone Dialogue Manager System - SADM

The Stand Alone Dialogue Manager (see Figure 4.27) uses the same kernel classes as the ***Mentor*** System but has a simplified interface. In essence the SADM Server is started and it listens for connections on a chosen port. It creates a new *Thread* for every connection and processes the user input via the mechanism shown in Figure 4.14 on page 130, but using a different `requestType` than the ***Mentor*** System.

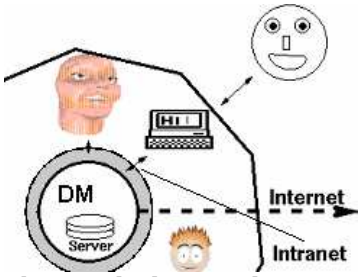


Figure 4.27 : The Stand Alone Dialogue Manager System

A text-based SADM Client can connect to that port/machine and enter into a dialogue with the SADM. As for ***Mentor***, each client gets a separate DM and there is some penalty for being first to connect, as since there are no Daemons, some class Topics load up static information the first time they are invoked.

The SADM Server minimally consists of:

DMS.java	Sets up infrastructure, reads configuration files and then waits for a connection.
DMThread.java	This is the *Thread* that creates a new instance of the DM for the connection and then waits for that instance to finish.
DM.java	The DM gets the user information through the network connection, handles requests, etc.
DMVariables.java	Handles all the DM and user variables, such as 'first_name', etc. Extends the Base Package `XMLVariables` class.
TopicsServer.java	Handles all the system and user Topic processing.
Properties/DM.properties	This file sets attributes for the SADM system.

The SADM Client consists of

DMClient.java	This connects to a specified port on a specified machine to access the SADM server. It sends across user information and any local user Topics, and then sends any user input across the network channel and displays on standard output any response or pro-active information that comes from the SADM server.

This lightweight DM system starts quickly and is easily transported to different environments for use. It can be easily connected to other applications because of the simple interface and has been used in various Talking Head applications at Curtin and at research laboratories in Europe.

4.2.2.4. Dialogue Management Kernel Issues

The Dialogue Management Kernel was carefully designed to be extensible and scalable so that it could be used to help the large number of students typical in the Computer Science discipline at Universities. It has been successful at meeting these objectives (see analysis in Section 5.3.6 on page 258 and Section 5.3.7 on page 259) but a number of issues have arisen that need future research.

System programming applications such as the ***Mentor*** System pose very real problems for Java. These problems range from long-term memory leaks in the Java Virtual Machine which can hamper efficient server applications, to a lack of low level control for common requirements such as job control, file locking and symbolic debugging. Many of these problems are being rectified as Java evolves but they posed serious hurdles for the initial development of the system using JDK1.1.2 on a Silicon Graphics workstation.

Java's claim to be 'write once, run anywhere' is valid only in the Sun or PC (Windows/Linux) environment. That is, in the environment of active development. A better motto would be 'write once **in version X**, run anywhere', where version X is the lowest common version number for the systems that may exist in common heterogeneous networks.

In developing a complex system, there is also the added problem that changes in the system must be controlled so as to prevent unforseen errors. That is, stability and correctness of the base technologies must be maintained. With the rapid development of Java, this has not been the case.

Two notable issues affected the smooth development of the system. Both occurred with the forced upgrade of Java to a newer version (not necessarily the latest version). The first was that the new Silicon Graphics JVM had an intermittent bug in the *Thread* management native code. This resulted in the system simply dying for no reason. The discovery of this problem was complicated by the server nature of the application and that it was started as an `inetd` process. The problem resulted in very intermittent Unix Segmentation Violations from the JVM with a full *Thread* and process dump.

The second issue was that the client interface popup menu system stopped working on Linux clients. This caused major version problems as the client system was developed on a system that used one version of the JDK but was used by students on a system that used a higher minor revision level. This had been installed at the start of semester on all laboratory machines but not the development machines.

These typical problems caused by external influences meant that the development of the system had to be disciplined so as to reduce likely internal programming causes of errors. The early adoption of Java interfaces and inheritance reduced many of the problems of side-effects and also increased the robustness of the system.

Both the ***Mentor*** System and the Stand Alone Dialogue Manager use a well-defined Interface - `DialogueManagerServer` - for all communication between the main application and the per-user instantiation of a Dialogue Manager (see Appendix G.6 on page 363). Similarly, all communication between the per-user DM instantiation and the `Topic` or `Daemon` classes is done via a Java Interface - `DialogueManagerInterface` (see

Appendix G.6 on page 363). The use of interfaces has increased the robustness of the system, and increased the ease of maintenance as new features are developed and implemented.

Another issue that arose was that although the OroMatcher package specified that the classes were *Thread* safe, this was found not to be the case. A PatternMatcher variable was originally created as an instance variable in the Topic class but this was finally tracked down to be the source of many unknown pattern matching failures. Unfortunately, this intermittent problem was not found until half way through the second case study. The problem became very apparent when more people were using the system. That is, more instances of concurrent *Thread*s accessing the PatternMatcher field at the same time.

The initial RE package has been deprecated and the ***Mentor*** System has migrated to the new Jakarta Apache project packages. This was relatively easy as a simple mechanism had been set up to help convert older style code to use the new packages and methods.

RE pattern matching is inherently costly due to the need for backtracking. That is, if the start of a RE contains a quantifier such as '*' or '+' that matches some initial part of the input but causes later parts of the RE to fail, then the pattern matching code must back up and restart the matching of the initial part again. This is backtracking. The new code may provide a backtracking efficiency benefit that will be required as more students start using the system. However, replacing code in the kernel of any system with new code may introduce hard to detect 'bugs' that only become apparent with heavy use (such as the *Thread* problem with the original package).

Similarly for the initial XML package that was deprecated. The migration to the new package caused few problems however.

Java has performed well as the system language. As noted, it still has many deficiencies for system development but has been able to support the development and running of the ***Mentor*** System. The use of Just-in-Time compilers and of native methods for some commonly used classes has meant that Java can be considered for serious system development work.

Another advantage of Java has been the ease with which new Topics can be developed in a safe way, and also that users may develop their own Topics and have them executed as part of the system. One major issue with this was the need for a per-*Thread SecurityManager*. This would have made the development and execution of potentially fatal user Topics very easy for the user, but yet safe for the ***Mentor*** System. It is still possible for a user Topic to bring down the server, but the mechanism is not obvious.

Another major issue was that classes from different Java versions were incompatible. If the user topics were compiled under one version and the system was compiled under another, the system would try to execute class files of the wrong version and hence throw an *Exception*. This *Exception* was reported back to the user but did not kill ***Mentor***.

If Berners-Lee's Semantic Web (Berners-Lee & Fischetti 2000a) becomes the predominant information structure on the Web, then the DM kernel should provide more support for Topics for data-mining this information. Since XML packages are already used

for VHML and the user metrics, this would impose minimal extra overhead. For example, the weather information could be easily determined simply by opening the appropriate site and reading the XML formatted data using the appropriate ontology. Or marked-up information could be provided via RSS or RDF (RSS 2002a).

An evolving issue that must be addressed is that language evolves and hence the Natural Language input processing must cater for this (see analysis of these problems in Section 5.3.5.1 on page 242). It can be argued that this is a problem for the Topics - this is where the input patterns are specified. But it is a problem for **all** the Topics and hence a solution should involve support from the DM kernel. For example, students started using SMS spelling for their enquiries. One possible scenario is that the user input could be processed by the existing `GeneralParse` method to help with the processing of slang, abbreviations or even typographical or spelling errors.

The system also needs to increase support for pro-active Topics since these may be used more frequently in the future. These must be used with care since many users have expressed strong reactions towards these interruptions. The system must be able to determine the user's level of tolerance and use this to better schedule individual pro-active Topics.

For example, pro-active Topics would benefit from having feedback from the user's client for sequences such as 'receiving the pro-active response, maybe deiconifying the interface, and the iconifying it again'. In this way, the Topic could use this timed information to determine whether the information sent was appreciated or just discarded. The Topic could register as a Listener for various client interface events and be informed by the DM kernel. This is seen as future work.

Similarly, a Topic may want to have support for a timeout to a response. For example, in a mentoring dialogue with the user, a Topic may ask: "`What do you think?`". It could set a timer for a suitable period of time and then if no response was received in that time, that Topic could gently prompt the user with more information, hints or even change the course of the discussion. This could be implemented in a similar way to the Listener mechanism described previously.

Many of these issues manifested themselves as part of the development or evaluation of the ***Mentor*** server architecture or the ***Mentor*** client architecture. As noted, these architectures have been developed in a very extensible way. The server can load different Dialogue Management systems, and the client architecture can accommodate interfaces of greatly varying complexity from a simple plain text interface to one that involves a Talking Head or a full Virtual Human. The client and server architectures are discussed in the following sections, along with issues that arose in their development and implementation.

4.2.3. The *Mentor* Server Architecture

The ***Mentor*** System extended the base level functionality provided by the kernel to situate a DM in an educational context. It used the extensive Domain Knowledge of the Lecturer as well as data-mined online sources of information, coupled with educational paradigms to help students in their studies. The following sections describe the ***Mentor*** server's functionality through examples, and detail the issues that arose in developing and implementing it.

4.2.3.1. The *Mentor* TCP-MUX MentorPortDaemon

The ***Mentor*** host machine normally has no server permanently running on it - the server is started when a client connects to it. Subsequent clients will connect to this started server.

Figure 4.28 shows the process of a client connecting to the ***Mentor*** host machine: a ***Mentor*** client running on any host in the network makes an initial network connection to the well-known TCP-MUX port on the Unix ***Mentor*** server. In response to this, the Unix `inetd` process (or equivalent) on that host machine then runs the `MentorPortDaemon` application whose purpose is to pass back to the client the network port number that the ***Mentor*** server is currently using to communicate to the outside world.

Note that the ***Mentor*** server may or may not be running. If this is the first connection, then the `MentorPortDaemon` must start the server. The `MentorPortDaemon` initially looks in a lock file for the port number to see if the system is already running. If not, it starts ***Mentor***, which when ready, stores the appropriate server port number in the lock file. The `MentorPortDaemon` application now has a port number to send back to the client, either from the already running server, or from the newly run server. Hence this and subsequent clients can use this port number to open a session to the server.

The client then closes the TCP-MUX connection to the server machine and uses the returned port number to make a second connection to the server machine so that a network channel between client and ***Mentor*** server can be opened. The ***Mentor*** server is listening on that port and will create a new *Socket* and *Thread* for every client connection and associates that *Thread*ed `MConnection` with a child process. The client and server then communicate through that *Socket*: the user can start a question-answer dialogue.

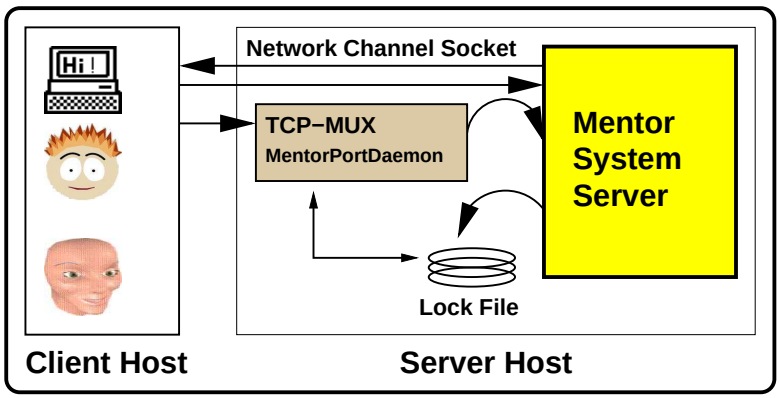


Figure 4.28 : The initial contact between ***Mentor*** client and the ***Mentor*** System

The `MentorPortDaemon` also acts as a gatekeeper: if a 'DOWN' file exists in the ***Mentor*** system file hierarchy, the contents of this file are returned to the client along with an error status and no connection is made to the server (if it is running). Typically this file will contain a 'the system is currently down' type message.

The `MentorPortDaemon` also acts as a proxy server: if the server is not running on the machine that the client connects to, a pass-through proxy is forked off, and the port returned is that of the input to the proxy. This allows for connections via the Web but where the machine that serves the Web pages is not the server host. Remember that applets can only make network connections to the machine where the applet was loaded from.

Finally, if for some reason the server cannot be started, the `MentorPortDaemon` returns an error status to the client, along with any error message, and exits.

Assuming that no system problems occur, the server is started as a normal process with no special privileges. It typically runs with the Unix user id of the ***Mentor*** System administrator[11]. This enables various files and databases to be read and written with security.

The following sections illustrate the functionality of the server by showing typical initialisation and execution of the system as clients connect to it.

4.2.3.2. The *Mentor* Server

When the server - the Java `Mentor` class - is first started, it:

```
• checks to see if any arguments such as timeouts have been set,
• loads up the server resources file which contains configuration information,
• creates a new server ThreadGroup - all connections, etc will run under this.
• opens and appends to the Mentor Log file,
• opens a Syslog connection for debugging,
• checks successive connections to ports starting at the first likely free port,
• instantiates a new MServer class connected to the successful port,
• sets system information so that all classes and clients can access it,
• starts the MServer thread
• sets the Mentor port number in the lock file for the MentorPortDaemon
• loops checking for periodic tasks such as
          checking for reboot request
          telling topics to save themselves,
          garbage collection, etc
   until all non-Daemon MServer Threads have died subject to a Timeout.
• stops the Server Thread, closes log file, etc.
• removes the Mentor port number lock file and exits.
```

Figure 4.29 Macroscopic description of the ***Mentor*** System Server

This class simply starts everything off, and waits for everything to die. The successful port number (Figure 4.29, line 10) is returned via the `MentorPortDaemon` to the client.

[11] The researcher

The instantiation of the `MServer` Java class (Figure 4.29, line 7) simply:

```
• Constructs the class's parent (Server) via the 'super' constructor,
• Sets the ThreadGroup, port number, the logfile name and timeout values
• Creates a Hashtable to store child processes in,
• Installs a new Security Manager.
```

Figure 4.30 Macroscopic description of the `MServer` class instantiation

The `Server` super class performs various initialisation tasks but its main job is to open the server socket connection so that clients can connect to it. These three classes were based on similar classes described in Flanagan (1997a).

Of importance to clients, when the `MServer` class is started (Figure 4.29, line 9) , it:

```
• Starts the server (which is a Thread),
• Starts the initial 'Load all other Daemons' daemon
   All daemons are loaded (if not suppressed by config file)
   and then started.
   If a daemon does not start, a fatal system error is triggered
• MServer goes into a simple forever loop
   Waiting for clients to connect,
   Creating new MConnection and Thread classes to process the client
   requests,
   Enters this new Mentor Child Process into the process Hashtable,
   Starts this new Child Process (see Figure 4.13 on page 129),
• Failing and finishing when the MServer Thread receives any Exception.
```

Figure 4.31 Macroscopic description of the `MServer` class execution

The initial 'Load all other Daemons' daemon opens up a Java zipped *Vector Object* file. This contains all the ***Mentor*** daemons ready for instantiating and running. These daemons will normally be run but can be suppressed via the ***Mentor*** System Resource configuration files. If the daemons are enabled but do not instantiate or do not run correctly, this will be logged via Syslog and the ***Mentor*** System will not start.

Appendix G.7 on page 365 discusses the logging mechanism used by ***Mentor***, and Figures G.15a-d shows the Syslog information that is stored as the system starts up. Each line shows the time, the host machine name, a process name identifier and then a general message. The various Figures show the different types of logging information as the system starts Daemons, loads up the Topic hierarchy, etc. That Appendix Section also discusses the loading of normal system Topics as well as all Unit Topics but does not show the loading of Specialist Topics nor Topics that each user may develop. These per-user Topics are sent across the client-server network connection at connect time. Those classes are validated but not cached.

The Specialist Topics are only loaded (and cached) when a user connects and indicates that they want specific Specialist Topics. This enables the system to have detailed system knowledge that only certain users may get access to. For certain applications, the client can specify that it wants the client-specific Topic class and **not** the University specific Topics nor even some of the system Topics. This functionality allows the system to be configured to cater for varied DM purposes.

Figures 4.32a-g show the Syslog information from a complex connection for a user who wants to conduct a normal question-answer dialogue. Note that in this example (Figure 4.32a):

- all hosts may connect,
- that the request type is 'command' (a normal question-answer connection),
- that two Specialist Topics directories are specified (normal Curtin Topics as well as a set of 'InterFace' Topics),
- that client Property and client user variable values have been transmitted (but not shown),
- that the user name has been sent for validation.

```
Jun 10 11:18:52 teapot Mentor: 32818: Conn : [teapot.cs.curtin.edu.au(134.7.1.156)]
Jun 10 11:18:52 teapot Mentor: Pid  11 Th'd: Thread[local.mentor.mentord.MConnection,5,Connections]
Jun 10 11:18:52 teapot Mentor: 32818: Request from:teapot.cs.curtin.edu.au:
Jun 10 11:18:52 teapot Mentor: 32818: allowedHosts from:*
Jun 10 11:18:52 teapot Mentor: 32818: Request   : [command] : 9
Jun 10 11:18:52 teapot Mentor: 32818: SpecialDir: Curtin,InterFace
Jun 10 11:18:54 teapot Mentor: 32818: Propty Req:
Jun 10 11:18:55 teapot Mentor: 32818: UsrVar Req:
Jun 10 11:18:56 teapot Mentor: 32818: Username  : [raytrace]
```

Figure 4.32a `Syslog` information showing transmitted client properties

Figure 4.32b shows that 26693 bytes of zipped *Object* class have been transmitted across to the server and that these are the `userTopic`s myTemplate, TH and VHML. These have been developed by the user and must extend `userTopic`.

```
...: 32818: JAR Topics: Received 26693 bytes in sendLocalTopics!
...: 32818: Lcl Topics: [myTemplate TH VHML ]
```

Figure 4.32b `Syslog` information showing transmitted local topics

Figure 4.32c shows that a user connection has initiated the loading and caching of various Topics relevant to specific units. For example, ST-151 represents the unit Software Technology 151. The ST-151 directory has been searched and, in this case, only the Topic 'turnip' was found. For the other unit topics, there are no special classes that have been developed, except for SPD-251. In this case we can see that two unit topics have been developed. The first is the cached class 'turnip' (indicated as cached by the '=' sign before the name) and the second is a data-mined FAQ Topic on the programming language Perl.

```
... : loadLocalTopics:reading local.mentor.topics.unitClasses.ST-151 topics file
... :           : Lcl UnitClasses : [(ST-151) +turnip,  ]
... : loadLocalTopics:reading local.mentor.topics.unitClasses.ST-151a topics file
... : loadLocalTopics:reading local.mentor.topics.unitClasses.FCS-152 topics file
... : loadLocalTopics:reading local.mentor.topics.unitClasses.AGV-361 topics file
... : loadLocalTopics:reading local.mentor.topics.unitClasses.CG-351 topics file
... : loadLocalTopics:reading local.mentor.topics.unitClasses.CG-252 topics file
... : loadLocalTopics:reading local.mentor.topics.unitClasses.SPD-251 topics file
... :           : Lcl UnitClasses : [(SPD-251) =turnip,  ]
... :           : Lcl UnitClasses : [ (SPD-251) +Perl_FAQ,  ]
... : loadLocalTopics:reading local.mentor.topics.unitClasses.SPD-251a topics file
```

Figure 4.32c `Syslog` information showing unit Topic Classes being loaded and cached

Notice that this loading of extra unit Topics is designed to be quite flexible. It can be seen that both ST-151 and SPD-251 have other directories as well - ST-151a and SPD-251a.

The unit directories hold unit Topic class files that typically are developed to cater for specific Domain Knowledge such as data-mined files or Web pages.

Figure 4.32d shows that the user connection has also initiated the loading and caching of the specified Specialist Topics. These are per-user and, as mentioned, allow for the **system-based** customising of a DM interface per user. These classes are part of the ***Mentor*** System, but **not** developed by the user. In this case, three new classes from the Curtin Specialist directory - `RandCDates`, `TPL` and `venue` - and three new classes from the InterFace Specialist directory - `deliverablesTopic`, `partnersTopic` and `VHMLTopic` - are loaded and made available for further use by being cached.

```
... :  : Specialist Dir:[ Curtin,InterFace ]
... :  : Special Topics:[(Curtin) +RandCDates, +TPL, +venue,  ]
... :  : Special Topics:[(Interface) +deliverablesTopic, +partnersTopic, +VHMLTopic]
```

Figure 4.32d `Syslog` information showing Specialist Topics being loaded and cached

Figure 4.32e shows that a per-user DM instantiation has loaded some previously cached system unit classes into its local unit classes Topics *Vector*. That is, this user will be enrolled in several units each semester and these units may have specific system Topics associated with them. The per-user DM instantiation loads only those special Topics that correspond to units that the user is enrolled for. These typically correspond to unit classes created by the Topic Network Builder (see Section on Topic Network Builder on page 176).

Note that in this contrived example, the user was enrolled in CG-351 but that the unit does not exist as a running unit within the university (it was a renamed unit and became AGV-361). Also in this example, only the unit SPD-251 has any Java classes associated with it. The rest have each been classified as an `unknownTopic`. This lets the system know that it has no knowledge about FCS-152 for example and it can respond to users' request about this unit accordingly (i.e. "I am sorry, I do not know about FCS-152 this semester").

```
... :           : No unit topic found for ST-151(01920) - adding as unknownTopic
... :           : No unit topic found for FCS-152(12332) - adding as unknownTopic
... :           : No unit topic found for AGV-361(04525) - adding as unknownTopic
... :           : no unit information on CG-351
... :           : No unit topic found for CG-252(04524) - adding as unknownTopic
... : Lcl Units : [ST-151 FCS-152 AGV-361 CG-252 CG-502 SPD-251 ]
```

Figure 4.32e `Syslog` information showing per-user unit classes being loaded from the system cache

Figure 4.32f shows that this user has requested, via their client interface preferences, that they want to disable the 'jokes', 'who' and 'didYouKnow' Topics. The 'jokes' and 'didYouKnow' Topics are pro-active and ask the users periodically if they want to hear a joke or tell them some trivial fact. Some users found these to be annoying and hence they turned them off. The 'who' Topic handles user requests such as "who is xxx" and knows about the Computer Science environment/users. Turning this Topic off enables another Topic, perhaps a Specialist Topic, to respond to these types of request instead. For example, the Murder Mystery Talking Head application that used the ***Mentor*** System for Dialogue Management, had its own knowledge about who people were and wanted to be able to respond with that specific information.

```
... : UsrPrp Set: jokes.disable=false
... : UsrPrp Set: who.disable=false
... : UsrPrp Set: didYouKnow.disable=false
```

Figure 4.32f `Syslog` information from the ***Mentor*** System

Finally, Figure 4.32g shows the Syslog information of this user disconnecting from the system and the subsequent termination of the various *Thread*s that make up the DM instantiation.

```
... : Processing: [finished]
... : DMQTimeout: run:DialogueManagerQueryTimeout-raytrace:11 has finished.
... : DMActQuery: run:DialogueManagerActiveQuery-raytrace:11 has finished.
... : Save Cntxt: /usr/staff/tango/raytrace/mentor/userContext/raytrace.context.gz
... : CmdTimeOut: run:CommandTimeOut-raytrace:11 has finished.
... : Connection: [CLOSED: ]
```

Figure 4.32g `Syslog` information showing the termination of the user DM instantiation

Mentor Server Configuration

The type and amount of information that the system logs via Syslog can be controlled via settings in the ***Mentor*** Configuration file (see discussion and Figures G.18a-f in Appendix G.8 on page 368). This Java *Properties* file controls most of the functionality of the server. The name of the file that is read depends upon the instance name of the DM. For the ***Mentor*** System it is "Mentor.properties", for the Stand Alone Dialogue Manager (Section 4.2.2 on page 134) that uses the kernel of the system, it is "SADM.properties". This flexibility through customisation allows for very different instantiations of a DM system.

The configuration file contains timing information for the pro-active output that the system sends to the user. That is, the system can ask things like "Have you started assignment 2 yet?" or "Do you need help with SPD?". Based upon feedback from students, these values are quite critical as they represent a balance between the system ‘nagging’ the student and the system being seen as ‘caring’ for the student (see Section 5.3.3.4 on page 209). These values are initial values - the real values are dynamically determined by feedback from the student based upon their responses to the pro-active queries.

The configuration file also contains flags for turning on or off debugging information for various classes of Topics. Within a Topic, the following debug code can be used:

```
    if(Debugging(context))
        topicDebug(context,"Checking " + topic_name + ":" + context.input);
OR
    if(Debugging(dm, "DialogueManagerActiveQuery"))
        dm.debug(topic_name + "/activeQuery: choice index=" + state);
```

In the first example, the Topic name is known and hence can be used along with the `Context`, to derive the equivalent of the second. This debug feature is not just limited to Topics. For example, ‘ImportantDates.debug’ enables the debugging of that Daemon.

Appendix G.8 on page 368 discusses how the system has been designed for real-world applications. No values such as hosts, directories or administrator email addresses are hard-wired into the code. Porting the system to another site simply requires the changing of these values in the configuration file.

Appendix G.8 on page 368 also discusses how it is possible to disable or suppress various `Topics` and `Daemons`. Any or all `Topics` or `Daemons` can be disabled with obvious loss of functionality but without the system dying.

4.2.3.3. Administrator and Control File Interface

A ***Mentor*** Topic was developed that enabled the control of the system via the normal user interface. This `AdminTopic` was authenticated via a password and any unauthorised attempt to use it was reported via email to the ***Mentor*** System administrator.

The allowed commands are detailed in Appendix G.9 on page 371.

This functionality was also available through a Control File interface. The system would periodically (typically every 10 seconds) examine a control file in the ***Mentor*** System file hierarchy and if it existed, it would use the contents of the file as a command to the Topic. This allowed for control even if a connection to the system was not available.

4.2.3.4. *Mentor* Server Issues

As noted, since the server used many classes from the Base Packages, it suffered from the same issues, such as network stability.

An unfortunate side-effect of using many *Thread*s (in the daemons and for each connection) was that garbage collection took a significant amount of the run time of the host machine (10-20%) even with no clients using the ***Mentor*** System. Therefore the server is currently run with no asynchronous garbage collection and the garbage collection is done periodically under server control. This has reduced the 'idle' run time to be very small (<2%) with infrequent bursts of activity as garbage collection is invoked. Asynchronous garbage collection caused a performance problem and is indicative of the Java philosophy of making it easier for programmers but at the same time, unfortunately, giving less flexibility to Java system programmers. In this case, a standard Java solution existed to minimise the impact of the garbage collection.

The flexibility of the system has been greatly increased by the use of the Specialist Topics since these, coupled with the ability to turn off various Topics, enable the dynamic per-user tailoring of the ***Mentor*** System not just the Dialogue Manager. The system has been used in this manner for research such as the Weather Woman (Dam & Souza 2002a), Pandora's Box (Holic 2004a), and an Adaptive Personality for an ECA (Xiao *et al* 2005a).

The server configuration can be controlled by instantiating it with a different class name, and hence a different configuration file is used to start up the system. This, coupled with the use of the `MentorRequestClass` as a base class for each client connection (see Section 4.2.1.2 on page 128), means that the server is highly extensible, and not just dedicated to the ***Mentor*** System.

Finally, one issue that directly impacted on extensibility, emerged as being significant for the current system design. The system loads only those Topics relevant to the units that a student is enrolled for. This reduces the pattern matching search space and speeds up the query-response cycle. However, this ignores the fact that users will need past information as well. For example, an Honours student may wish to ask the system about a previous advanced graphics unit. Currently the system would plead ignorance about any knowledge of this area simply because it does not load the existing, but not enrolled-for Topics.

Several solutions exist to this issue. For example, all Topics could be loaded when a user connects. This makes all knowledge available to a user. However, as noted in Section 5.3.4.6 on page 230, different domain experts may give conflicting responses to the same question because of their domain requirements.

Also, as for a human mentor, the system should answer relevantly - that is, a human would assume that a user in the third year of their course does not require a trivial answer to a request but one that perhaps addresses a deeper issue. Therefore, this solution would require a Topic resolution mechanism that was cognisant of the student's learning level - a significant system development. This could be addressed in future research.

Another solution would be to weight the responses differently dependant upon whether the response came from an enrolled Topic or from a past Topic. In this way, all knowledge is available but the most likely is from the most recent or most applicable Topic. Since a weight scaling mechanism already exists within the system, this could be easily implemented.

A more pragmatic approach would be to enable a student to choose any Topic. Another preferences tab item (see Section on User Preferences on page 158) could be introduced to enable users to choose their unit Topics (see mock-up interface in Figure 4.33).

Figure 4.33 : Mock-up interface to enable users to choose their own Topics

The development overhead for this is minimal and would simply mean that this information is sent from the client to the server at connection time and would augment the list of Unit Topics loaded for this user. It would also mean that a more knowledgeable user could get 'refresher' information easily. This client mechanism could even be used to set Specialist directories - currently set on the command line when the client is run.

This issue of the need for Domain Knowledge expansion without conflicts is seen as important for future use of the system. The best response must be available to the user not just any response.

4.2.4. The *Mentor* Client Architecture

The ***Mentor*** client architecture has also been designed to be extensible so as to allow different renderings of the DM output: plain text, a hyper- and multi-media GUI, a Web page, or even a photo-realistic computer generated talking head.

Each ***Mentor*** client extends `local.mentor.mentor.Client`. This class contains fields for the input and output streams to the server as well as the network *Socket* to the server. It also contains fields to hold the configuration resources for the ***Mentor*** system, the client, and for the client interface mechanism (such as a GUI).

The various `Client` constructors open a connection to the `MentorPortDaemon`, by default through the TCP_MUX port on the local host. It is also possible to specify the port number and machine on the command line of all clients, and this can then be used for connecting. It is also possible to use an URL to specify the host, and the `Client` constructor then assumes that the connection is via the proxy of Section 4.2.3.1 on page 147. Importantly, since the `Client` class normally reads the standard ***Mentor*** configuration file (that is, it assumes that the server has been instantiated with a class name of 'mentor'), the `Client` can connect to the port and server host specified in that file with no extra user action even if the port/host are non-standard.

The `Client` class handles all the initial connection protocol and opens data streams to the server. The request protocol (see Table 4.6) is sent across to the server and the class handles the second level of negotiation by sending the user *Properties* and any command line arguments of the client application across to the server.

`terminateRequest`	`talkRequest`	`submitRequest`	`commandRequest`
`psRequest`	`echoRequest`	`threadRequest`	
`queryRequest`	`killRequest`	`topRequest`	

Table 4.6 ***Mentor*** System Client request types

The `Client` class has various utility and debugging methods that are inherited by all instantiated clients. The `Client` class also defines the single method for packaging and sending data across to the server. Any client can also send data to the server to be time-stamped logged in a server-side file via a single method. The file is available for later analysis by the client.

4.2.4.1. The *Mentor* Clients

All clients will extend the `Client` class and typically will also create an instance of a *Thread* that can asynchronously send/receive data between the client and server. Most clients also have a standard command line argument processing structure that includes the arguments shown in Table 4.7. Other per-client arguments can be added to this list. For example, some interactive clients also have a '-iconic' switch to specify that the GUI should start iconified. These arguments are merged with, and over-ride values from, the various configuration files. Note that the classname argument can be used to specify a different configuration file. This classname can also be used to configure any client sub-systems, such as a communication interface.

`-classname`	specify the instantiation Class name
`-user`	specify the user name for debugging purposes
`-menhost`	specify the server host machine name or IP address
`-menport`	specify the server port number
`-syslog`	specify the mechanism for logging error/warning messages

Table 4.7 ***Mentor*** System base Client command line arguments

MentorClient and the MentorClient User Models

The `MentorClient` (see Figure 4.34) is a user's primary interface to the ***Mentor*** system. It has 5 areas of interest:

- a menu bar at the top,
- a history of requests area that holds past user input requests,
- the user input area where the user types their request,
- a hyper- and multi-media area that displays the response from the ***Mentor*** System, and
- a scrollable status area for system messages and debug information.

Figure 4.34 shows a contrived dialogue with a user, plus some of the generated text from the dialogue. The answer to the current user request - "help me with assignment 2" - has been embolden for clarity. Line separators between responses have also been removed.

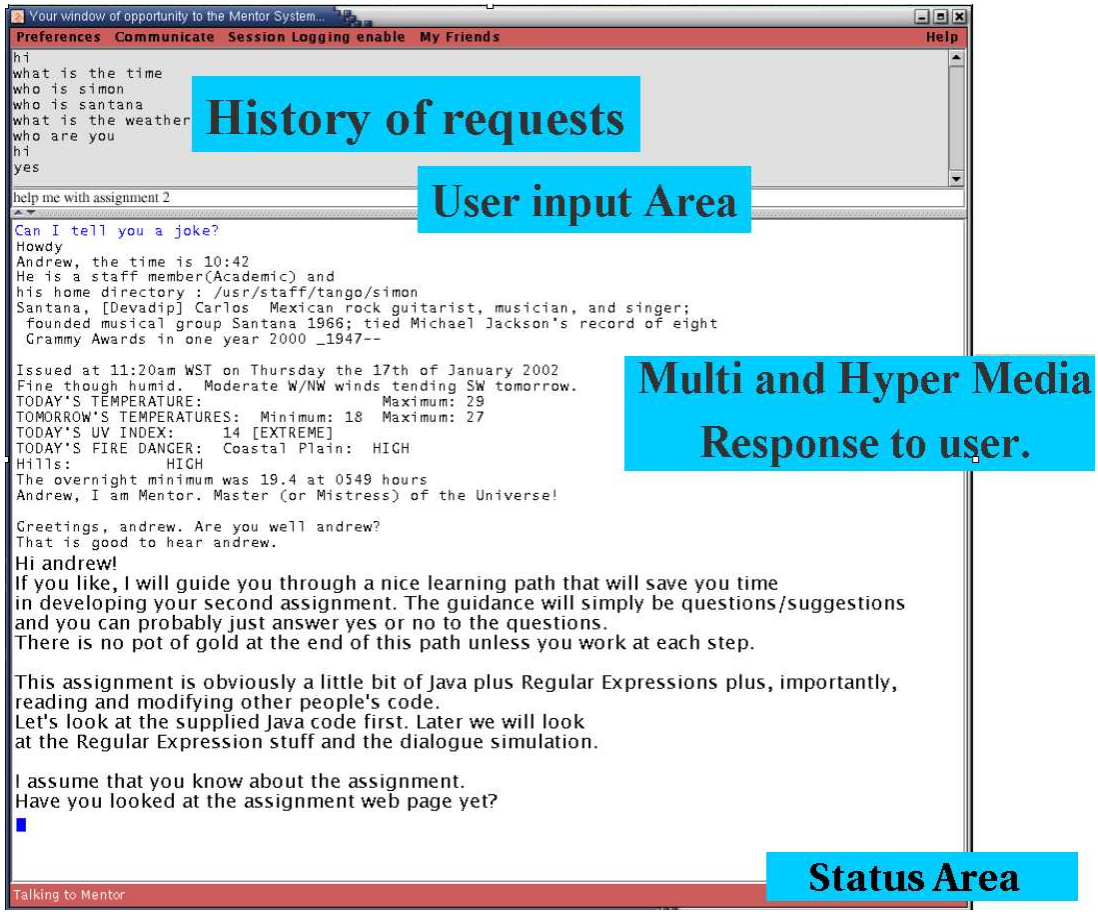


Figure 4.34 : The normal client interface to the ***Mentor*** server

The `MentorClient` extends the `Client` class and instantiates a `UserModel` for the user based upon their status within the (computing) environment. Typically users will

instantiate a `StudentModel`, and most development work has concentrated on this model. Table 4.8 shows the currently supported user models.

`GuestModel`	`SupportStaffModel`	`TechStaffModel`	`StudentModel`
`StaffModel`	`AcademicStaffModel`	`BigPersonModel`	

Table 4.8 ***Mentor*** System User Models

The user models extend `local.mentor.userModels.UserModel` which extends the Java *Applet* class. One of the design aims of the system has been for simplicity for the user but to also allow extensibility. Users can get answers simply by running the `MentorClient`. By default it will classify them and use the `StudentModel` class.

However, the `MentorClient` also checks to see if the user has created their own user model by searching for the class file `local.mentor.userModels.xxx` where xxx is their username. This class file must extend the correct super class (for example, `StudentModel` in this case). If found, then this will be used instead of the default. This allows users to add to their client functionality with little extra programming overhead due to Java's inheritance mechanism. Figure 4.4 on page 110 shows a complex multimodal Dialogue and Interaction Management System created by a user that exploits this and the previously mentioned extensibilities of the system, to cater for input from remote controls as well as directing output commands to devices such as televisions and radios.

In a similar way, Technical Staff could add to their user model to allow for common tasks such as checking the status of machines, line printers, or disk usage. Similarly, Academic Staff may be more interested in monitoring aspects of a student's work rather than in conversing with the system.

Since the `UserModel` extends *Applet*, it is relatively simple to understand for creating your own user model. An *Applet* has a position and size on the screen, needs to communicate via parameters rather than via interface calls, runs asynchronously; all the functionality needed by a user model.

The `UserModel` also needs to know about client-server communication, about the mechanism for displaying the client-server interaction, and about configuration details passed on from the client. Of these, the most important is the GUI for client-server interaction: it must implement a `local.mentor.util.TalkConsoleInterface`.

Currently, two different GUI models can be instantiated for client-server interaction. The type chosen depends upon the `MentorClient` configuration file which often depends upon the `MentorClient` instantiation Class name. For example, a user may run the `MentorClient` as shown by the four examples below.

```
% MentorClient ↵
% MentorClient -class plain ↵
% MentorClient -class html ↵
% MentorClient -class testing ↵
```

The first will run the `MentorClient` with default values. This will instantiate the GUI interface shown in Figure 4.35. The second example may create a similar interface but with different colours and sizes. This style of GUI is now deprecated and replaced with an HTML based interface. This can be instantiated using the third example above and results in

Figure 4.34 and also Figure 4.36. The visual differences result from the user's ability to configure the interface as well.

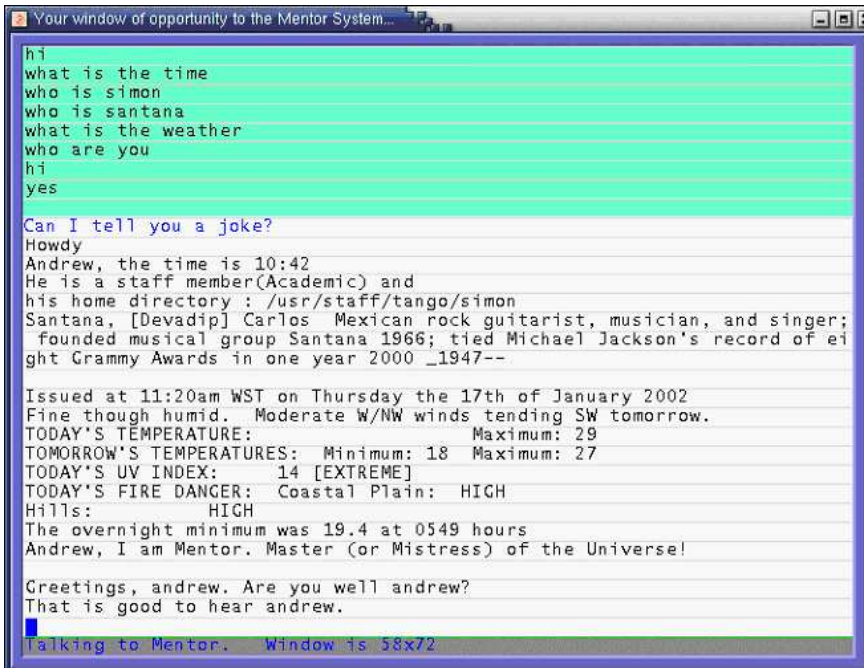


Figure 4.35 : ***Mentor*** System text-based interface GUI

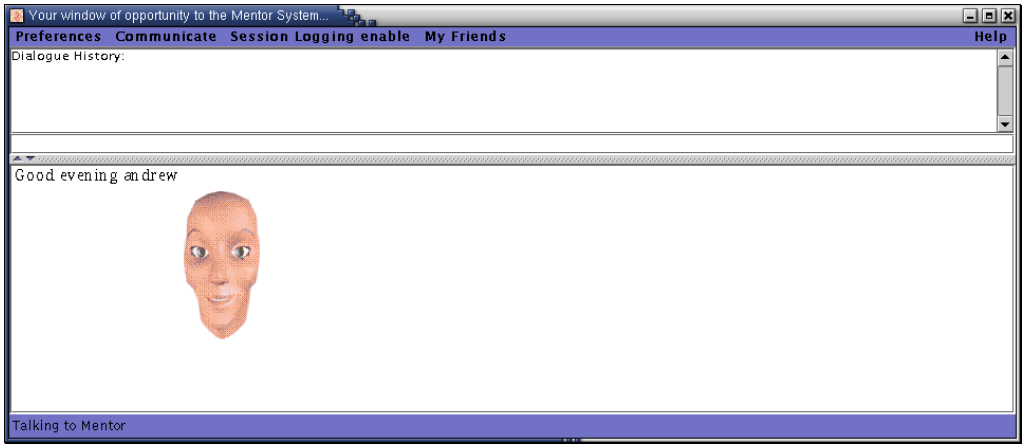


Figure 4.36 : ***Mentor*** System HTML text-based interface

The final `MentorClient` instantiation example ('-class testing') may produce an experimental GUI that a user is developing (a contrived example, Figure 4.37). The instantiation Class name can also be used to instantiate a very different user model.

The menus and menubar of the `TalkConsole` that is instantiated for the HTML based GUI interface of Figure 4.36 are set by the `MentorClient` using `TalkConsoleInterface` methods. The actual instantiated `TalkConsole` class can ignore menubar and menu creation requests or can add to these default values to provide extra functionality.

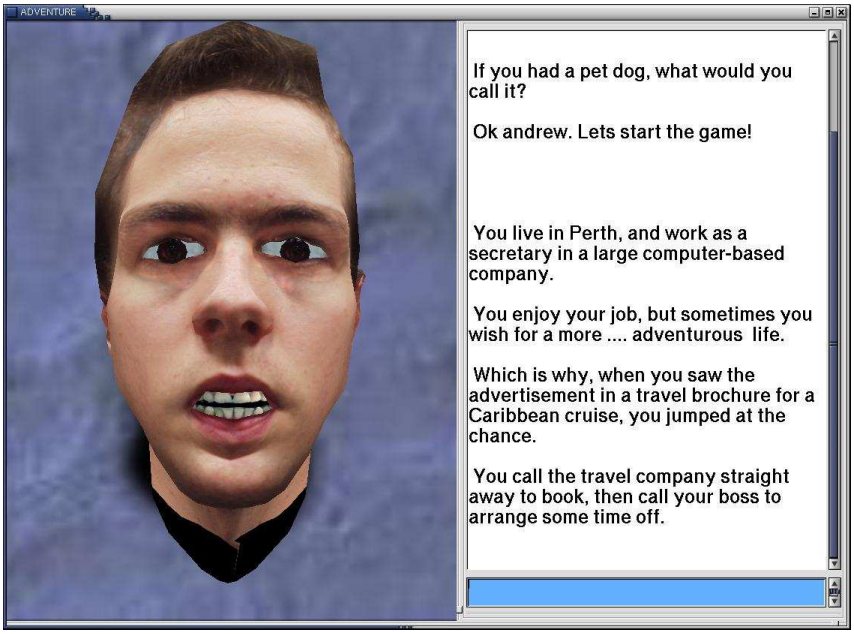


Figure 4.37 : Experimental interface

Through the default menus, users can edit preferences that control the user interface's look and feel as well as their interaction with the ***Mentor*** System (see Figure 4.38). These preferences are stored in a user preference file.

Each user will have a preferences file called 'htmlMentorConsole.properties' typically stored in a sub-directory of their home directory called '.java'. This file is generic in that the standard client interface looks for a file called 'XXXMentorConsole.properties' in various places in the users CLASSPATH. The XXX is the instantiation name of the client interface. That is, you can run the MentorClient with various arguments, one of which is '-class XXX'. This sets the instance name of the client and hence enables different instances that can use different preferences.

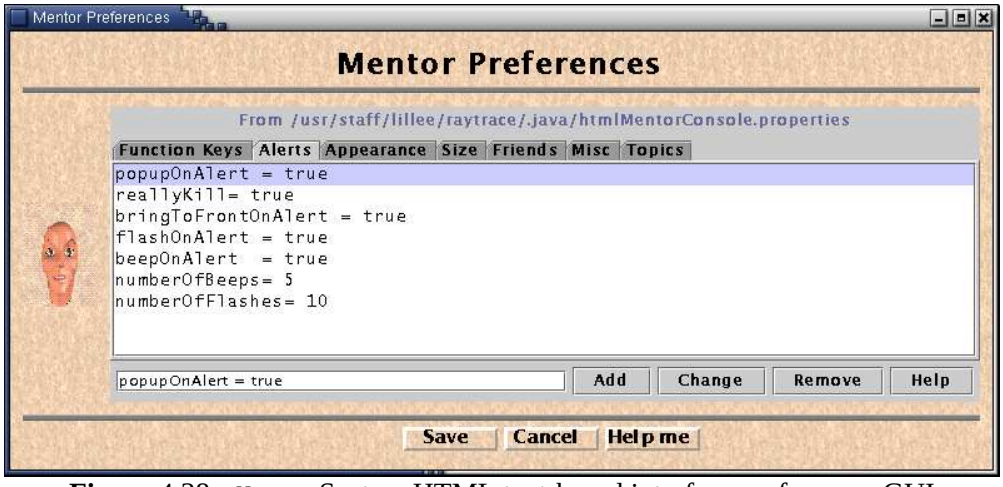


Figure 4.38 : ***Mentor*** System HTML text-based interface preferences GUI

Many client interface preferences can be set by the user - see Appendix G.10 on page 372 for more information. Typical preferences allow the user to set Function keys to send common strings, to turn off various Topics, the size and colour of the interface, etc.

The response area of Figure 4.34 is a Swing *JEditorPane* and hence can show hyper- and multi-media information. When a user clicks on a hot link, an external Web browser is opened to display the link location. Figure 4.39 shows the interface displaying text with hot links. A floating tooltip (shown above the link for "distributions") shows the URL of the hot link under the cursor. Note, to separate the individual responses returned from ***Mentor***, a special 'line' image is used. This was required because the response is displayed so quickly that users lost sight of where the current response started.

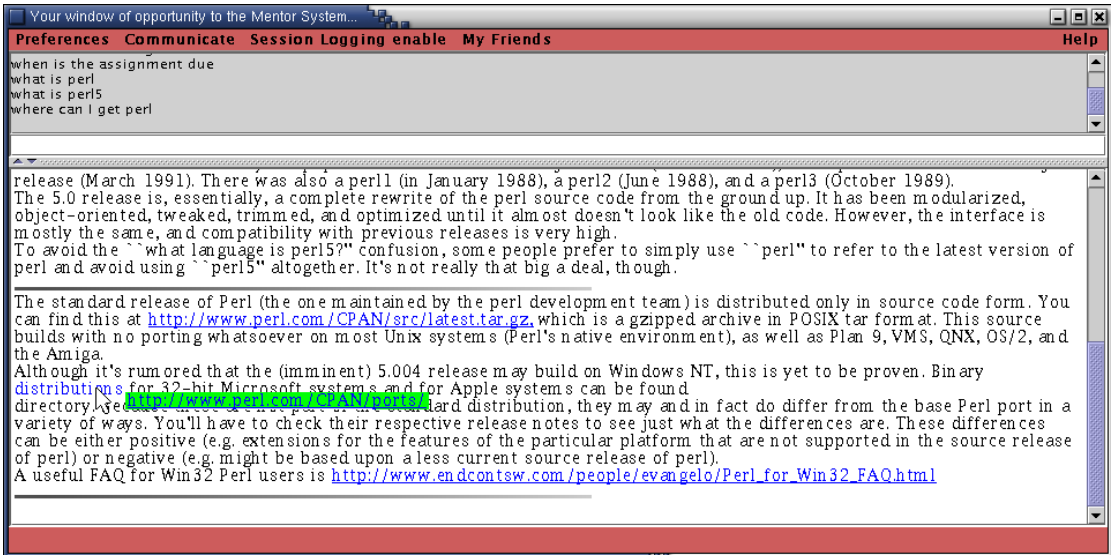


Figure 4.39 : ***Mentor*** System MentorClient HTML anchor functionality

AppletClient

An AppletClient was developed (Figure 4.40) so that the system could be used from the Web. Several issues arose from this, most concerned with security. Since ***Mentor*** must authenticate the identity of the connected user, this authentication must also be available for a Web interface. Simple CGI-BIN scripts were written that used browser cookies to enable users to login to the system using either their valid Unix user identity or as a guest. Figure 4.40 has been generated to show the cookie values used. Cookies are a mechanism that server side connections such as CGI-BIN scripts can use to store/retrieve information on the client browser. The cookie is associated with a range of URLs and any future requests made by the client that fall in that range will include the current value of the cookies.

Note that using an *Applet* to connect to a server poses another security problem since an *Applet* can only make a network connection back to the same host that the *Applet* was loaded from. Since the Web server was not the ***Mentor*** server, the CGI-BIN scripts had to use the proxy facility of the `MentorPortDaemon` (see Section 4.2.3.1 on page 146). In this way, packets from and to the *Applet* were routed via the `MentorPortDaemon` running on the Web server to the ***Mentor*** server.

The AppletClient sent `QueryClient` type requests to ***Mentor*** - these allowed a restricted question-answer interaction that matched the reduced functionality Web interface. The AppletClient could also send requests for the `PsClient`, `TopClient`, `ThreadClient` and `DieClient` services described in Appendix G.11 on page 373.

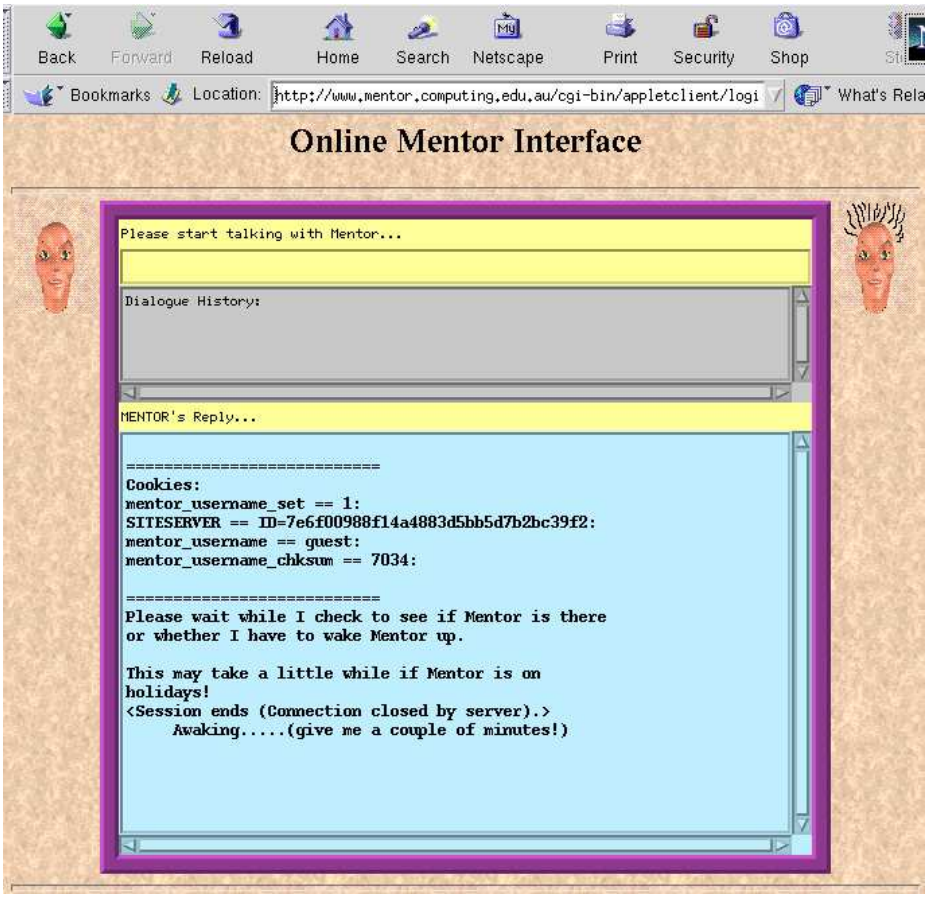


Figure 4.40 : AppletClient showing debug and session information

Other Clients

The network protocol interface to the ***Mentor*** System allowed other clients to connect to and interrogate the system. Some of these are purely for debugging, and some are for session management. These clients communicated with the server using the protocols of Appendix G.4, Figure G.3, on page 357. See Appendix G.11 on page 373 for information about these clients.

4.2.4.2. *Mentor* Client Issues

Most issues were with early versions of the standard user client. The new Swing based client interface is functional and flexible enough to provide for most user's needs. However, as can be seen in Section 5.3.5.1 on page 238, many users complained about problems **before** using the online help available through the client interface. This is seen as an education issue for users. One possible solution is that when the user first runs the client, the functionality of the client interface is explained via a browser or the interface itself.

The extensible nature of the client interface has been able to accommodate varying clients with few problems. Note that the response sent to the client is **byte**-oriented not Java 16-bit **char**-oriented, and hence a C-based client can connect to the DM system. However, this currently limits the response to this client request protocol to an 8-bit character set and not the full Unicode character set of Java. Other request protocols that support 16-bit transfers can be developed in the future.

4.2.5. *Mentor* Data Files

4.2.5.1. The *Mentor* User Metrics File

As users interact with the system, their input, the response to their input and their actions such as iconifying the interface, are recorded in a file for later analysis. This 'metrics' file uses XML for convenience of parsing/analysing and comprises multiple user sessions. Figure 4.41 shows a typical complete session. In this case we can see that the user asked about the weather and received a response. The tags have attributes - in this case the attribute values for *date* and *seconds* have been removed.

Notice that Figure 4.41 shows that the user has been alerted that they have a pro-active message from the system (<**alertUser**>) and then the active message has been recorded (<**active**>). The user does not need to do anything about this pro-active response. In fact, the user may have turned off notification of these responses via the preferences system on the client interface. In this example, the user responds to the alert and pops up the interface, (<**popup**>), reads the message and acts on the advice by clicking on the displayed link to open a Web browser to visit the URL (<**URLClick**>). After a while, the user iconifies the interface again (<**popdown**>) and ends the session (<**endsession**> and <**/session**>). Remember that closing tags cannot have attributes and hence the need for a specific <**endsession**> tag. Typical sessions repeat this <**input**>-<**response**> <**popup**>-<**popdown**> cycle.

```
<session>
<input date="...." seconds="....">
What is the weather?
</input>
<response responseCode="0" topicName="weather" date="...." seconds="....">
Issued at 9:05pm WST on Tuesday the ..... for Wednesday
FORECAST:
Fine. S/SW winds, freshening in the afternoon.
TEMPERATURES:  Minimum:  12   Maximum: 23
UV INDEX:      12 [EXTREME]
FIRE DANGER:   Coastal Plain: LOW
Hills:         LOW
Fine, mild  is expected to continue on Wednesday and Thursday with early
seabreezes.
Friday and Saturday should be fine and warmer.
</response>
<alertedUser date="...." seconds="...."/>
<active topicName="UNKNOWN" date="...." seconds="....">
Do you know this: For help on Java look at
<a href="http://www.faqs.org/faqs/by-newsgroup/comp/comp.lang.java.help.html">
http://www.faqs.org/faqs/by-newsgroup/comp/comp.lang.java.help.html</a>
</active>
<popup date="...." seconds="...."/>
<URLClick date="...." seconds="...."
       HREF="http://www.faqs.org/faqs/by-newsgroup/comp/comp.lang.java.help.html" />
<popdown date="...." seconds="...."/>
<endsession date="...." seconds="...."/>
</session>
```

Figure 4.41 ***Mentor*** User Statistics example

These metric files were used to analyse user interaction so as to see the effectiveness and user relevance of some responses. They also provided a complete user interaction history. See Appendix G.12 on page 377 for more detail about the format used to store the User metrics.

4.2.5.2. The *Mentor* User Context File

The user context, along with the User Variable file described in the next section, holds all needed data to maintain the continuity of the user interaction within a session as well as across sessions. For example, since the user context stores the full date of the last interaction, ***Mentor*** could greet the user with

> Hi Dave, I haven't seen you for a few days!

An instance of this `Context` is associated with every user and it is saved as a zipped Serialised *Object* in the user's context file. Figures 4.42a-d show some of the fields in the `Context` class. Since the `Context` is passed to every parsing method shown in Figures 4.10, 4.11, and 4.12 in Section 4.2.1.1, it can help the user for example, in continuing in their current Directed Learning Paths (DLPs), or pro-actively ask them questions about their current task. The `Context` provides that continuity.

Note that the first two fields (Figure 4.42a) are **transient** - that is, their values will not be saved in the Serialised *Object*. Both of these fields are quite large and it serves no purpose to save these in the file. They are part of the `Context` since it encapsulates the entire user context of the interaction.

```
transient public DialogueManagerInterface dm;
transient public DialogueManagerServer dms;
```

Figure 4.42a ***Mentor*** User Context **transient** Fields

One field of the `Context` holds a `GeneralParse` class for future expansion. Before any input is passed to the parsing routines, a `GeneralParse` class is created from the user's input. At present little is done with this but it is reticulated to all routines that process the input. Within the `GeneralParse` class, `subject`, `object` and `anaphora` fields hold values for anaphora processing.

The user's input is stored in the `unadulteratedLine` field of the `Context` (Figure 4.42b). A trimmed, converted to lower case version of the input is stored in the field `nonRationalised`, and a version of this that has had leading and trailing semantically meaningless characters removed (such as punctuation, 'please' and 'thank you', ***Mentor***'s name, etc) is stored in `input`. Comments in Figure 4.42b show typical user input as it would be stored in each of the fields. Most parsing routines will use the `input` field but some will need access to various unprocessed versions. For example, the question "who made Mentor?" will need to be processed from the `nonRationalised` field since the word 'Mentor' is removed from the `input` field.

```
public String unadulteratedLine; //e.g Mentor, please what is the time? Thanks
public String nonRationalised;   //e.g mentor, please what is the time? thanks
public String input;             //e.g what is the time
```

Figure 4.42b ***Mentor*** User Context input Fields

The `Context` also holds a `state` field that indicates whether the input line has been processed correctly or whether the line has been processed but needs to be re-processed because a Topic has modified the input line. This could happen in processing anaphora or in processing very general requests such as "help me with my assignment". In this latter case,

Mentor may determine that the student is enrolled for two units that it knows about and that it needs to disambiguate the request. A general 'handle assignments' Topic will then ask the user to choose which unit from a list of appropriate units. If they choose one of these, then the input is reformulated, for example, as "help me with my **SPD-251** assignment", the `state` field of the `Response` is set to 'reparse' and the parsing process restarts.

The history fields (Figure 4.42c) record the past interactions with the user in terms of their input, the output from the system (typically responses to the inputs) and the states that have been visited for each Directed Learning Path.

```
public Vector /* of String */ InputHistory =null;
public Vector /* of ResponseData */ ResponseHistory = null;
public Hashtable /* of String */ UserStateHistory =null;
```

Figure 4.42c ***Mentor*** User Context History Fields

The `InputHistory` is currently only used for debugging when an un-processable input is encountered. The last three interactions are emailed to the system administrator. The *Vector* of `ResponseData` contains the entire response history from the system (see Section 4.2.1.1 on page 122). The `UserStateHistory` field always grows in size until the user has mapped out the entire DLP. This could be reset each semester.

Figure 4.42d shows the `Context` fields that accommodate the multimodal nature of the DM. Topics by default will process plain text input from the user - that is, the user simply asks a question using plain text. However, a Topic can register via a DM API in the `TopicsInterface`, as being able to process any 'type' of stimulus - the type being some user defined string of characters. This is typically used by cooperating clients and Topics. That is, the client will be multimodal in nature and supply a user-developed Topic to process that multimodal type.

```
public String stimulus_type = null;
public Object stimulus_object = null;
```

Figure 4.42d ***Mentor*** User Context Multimodal Stimulus Fields

The `stimulus_object` of Figure 4.42d is arbitrary data that is passed to the DM's parsing methods of Figures 4.10, 4.11, and 4.12 in Section 4.2.1.1. Any multimodal stimulus is **only** processed by Topics that have registered to handle that type. Figure 4.43 shows the multimodal client created from the `UserModel` and `Topics` listed in Appendix I. In this example, emotional input can be sent to the DM for processing. Other examples allow for input from a TV remote control ('Pandora's Box' (Holic 2004a)) and from a mail monitoring application ('An Adaptive Personality for an ECA' (Xiao *et al* 2005a)).

4.2.5.3. The *Mentor* User Variables File

Figure 4.44 shows an example of a user variables file that is in Java *Property* class format. These values are different in kind to the `Context` values. They hold typical system values for the user such as their preferred name, their country of origin, etc.

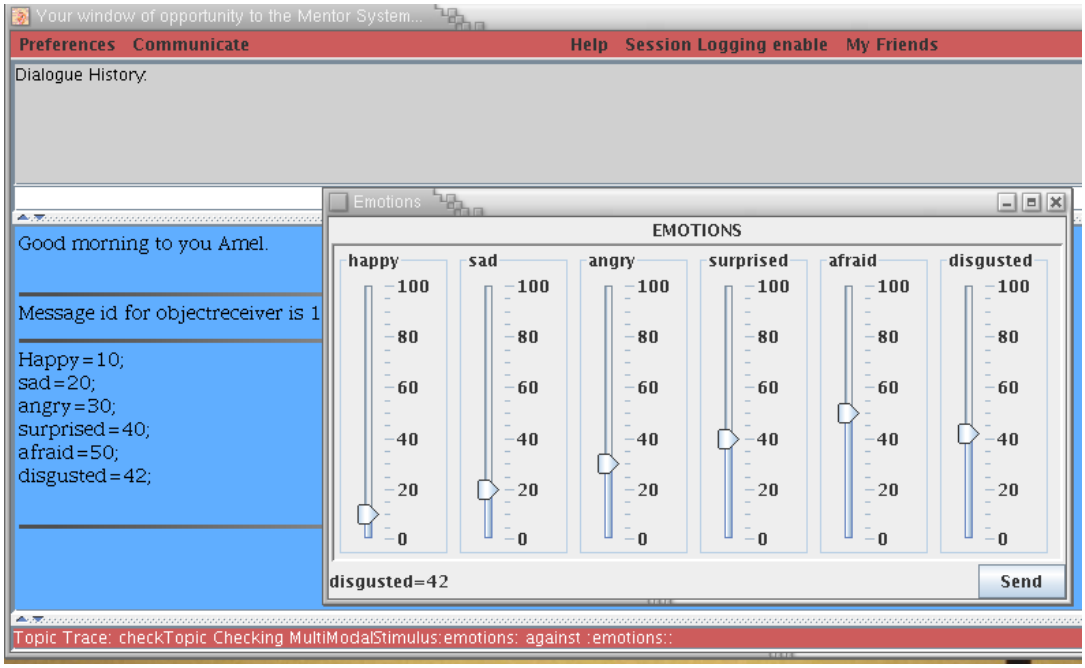


Figure 4.43 : Example of a multimodal interface to the DM

Notice that there appears to be `Context` type information in this file - **cg351UnitTopic_Assignment1_started**. However, what is stored is not a value but a key-value pair. That is, various topics may like to save the different states of the user interaction, and the User Variable file enables this named storage of values. Similarly, the user's list of friends is stored here and is sent across to the client interface at connection time.

Finally, in this example two values not associated with University study have also been stored here: **PLAYER_IS_** and **lastSession**. These are from the use of the ***Mentor*** System as part of the Murder Mystery application.

```
#User Variables.
#Fri .............
hisHer=His
first_name=herbert
gender=male
himHer=Him
dialogueManagerCountry=australia
language_didnotunderstand=I don't understand|I do not understand|.........
language_afternoon=Good afternoon
topic_name=graphics
other_names=xaver
language_night=Good night
userCountry=australia
cg351UnitTopic_Assignment1_started=true
language_evening=Good evening
last_name=bloggs
friends=xxx,yyy,zzz
language_hello=Hello|Good day|Hi|G'day|Hiya|Gidday|Hi y'all|Hay gaan|.....
language_goodbye=Goodbye|Good-by|Good-bye|Bye|Cheers|See ya|..............
heShe=He
language_welcome=Welcome|G'day
mentor_country=Australia
language_morning=Good morning
PLAYER_IS_=Policeman
lastSession=1038196814004
```

Figure 4.44 ***Mentor*** User Variables example

4.2.5.4. The *Mentor* User Preferences File

This file is stored in the users home area and is described in Appendix G.10 on page 372. It controls various aspects of the client interface. It was decided that it should be in the user's area rather than in the server file hierarchy, so that the user may edit it as appropriate.

4.2.5.5. *Mentor* Data Files Issues

Context File Issues:

The `UserStateHistory` is very specific to the concept of a Directed Learning Path, and should be made more general to accommodate arbitrary state-based traversals. This field grows in size but note that the nodes of a DLP have a retention time - the time it takes the user to forget the information supplied or to need reminding - and this currently defaults to 1 year.

The size of these data fields increases with each interaction, and is an issue. However a typical semester's worth of interaction is less than 100K bytes of data per user so this history currently just grows unbounded. Future work could look at heuristics for truncating this history to the last N items. Currently the system only looks at the last 10 Response history items when trying to reduce duplication of responses.

User Variables File Issues

Some information may seem redundant, such as the various language greetings, but it was decided that this mechanism allows for ease of future expansion.

The plain text format does not allow for a Topic to store non-text information. However, Topics may store this type of information in files in the DM system's file hierarchy, using standard DM system calls for creating and reading/writing of files.

User Preferences File Issues

As mentioned previously, users complained about the client interface (Section 5.3.5.1 on page 238). If they had used the online help available through the client interface, then they would have seen that the User Preferences File and the Preferences GUI Dialog would have solved their issues. This is seen as an education issue for users.

The User Preferences File allows for an extensible mechanism to increase the functionality of the client-server system to benefit the user. An example of how the file and the Preferences GUI Dialog could be augmented to solve one knowledge base issue can be seen in Figure 4.33.

A similar tab in the Preferences GUI Dialog of Figure 4.33 could be used to allow the user to choose a different 'personality' for the dialogue interaction. Since all the relevant entries in the User Preferences File are already sent across to the server when a user connects, the framework for this type of extensibility exists already.

Finally, although XML is a very useful format for storing and parsing data, it should not be seen as a universal format to be used regardless. The Context File is a Java Serialised *Object*, and both the User Variables File and the User Preferences File are in the Java *Properties* format. All these formats are ideally suitable for the task.

4.2.6. The Virtual Human Markup Language (VHML)

The foundation for VHML was originally developed by John Stallo (Emotion Markup Language (EML) and Speech Markup Language (SML) (Stallo 2000a)) and Quoc Huynh (Facial Animation Markup Language (FAML) (Huynh 2000a)).

Simon Beard has also made significant contributions to its development (Beard 2002a; Beard & Reid 2002a). The current specification of VHML at www.vhml.org also owes a huge debt to Emma Wiknertz, Linda Strindlund and Camilla Gustavsson whose efforts and research made the language more solid, homogeneous and complete (Gustavsson *et al* 2001a; Gustavsson *et al* 2002a).

Given the need for a standard, extensible markup language for the Domain Knowledge for the ***Mentor*** System, the initial work by Stallo and Huynh was enlarged, tested, evaluated and discussed by Beard and Marriott, and standardised by Gustavsson, Strindlund and Wiknertz. VHML development/implementation was part of a 3 year European Union 5th Framework project called InterFace. Development continues with research by Beard and Marriott.

The role of the researcher has been one of integrator, manager, promoter and developer (Marriott, Beard, Haddad *et al* 2001a; Marriott 2001c; Marriott 2001a; Marriott, Beard, Stallo & Huynh 2001a; Marriott & Beard 2002a; Marriott 2002a; Marriott 2002b; Marriott & Stallo 2002a; Marriott & Beard 2003a). The following briefly shows the need for VHML in a Dialogue Manager, its structure and its future as a markup language. More information can be found at the VHML website - www.vhml.org.

An understanding of the effect of Intelligent User Interfaces (IUI) is clearly important in the field of information communication and its related areas, particularly in relation to cultural and behavioural issues, and especially in representations of gender, race, community and acquired behaviour (J Murray 1997a). Hence, it is important for the designers and implementers of these interfaces to make informed decisions on how to make them fulfil their required role more effectively in an interactive experience as facilitators of information.

The Dialogue Manager kernel was designed to connect to many applications including those that have a Talking Head interface. Therefore if our user interface was to become more believable it was necessary to design and implement some initial markup languages that could add simulated emotion effects and gestures to the possibly Human-Talking Head interaction. Since the Dialogue Management system was at the heart of the ***Mentor*** System, it was important for it to be able to address these issues.

4.2.6.1. VHML Structure

VHML uses/builds on existing (*de facto*) standards such as those specified by the W3C Voice Browser Activity, and adds new tags to accommodate functionality that is not catered for. The language is XML/XSL based. XML is the Extensible Markup Language (XML 1997a). It is a simplified dialect of the Standard Generalized Markup Language (SGML)(Standardization 1985a) that is relatively easy to learn, use and implement, and at the same time retains much of the power of SGML. It is important to note that XML is not a markup language in itself, but rather it is a metalanguage - a language for describing other

languages. Therefore, XML allows a user to specify the tag set and grammar of their own custom markup language.

VHML consists of the following sub-systems:

- SML Speech Markup Language
- EML Emotion Markup Language
- GML Gesture Markup Language
- XHTML X-HyperText Markup Language subset
- DMML Dialogue Manager Markup Language
- FAML Facial Animation Markup Language
- BAML Body Animation Markup Language

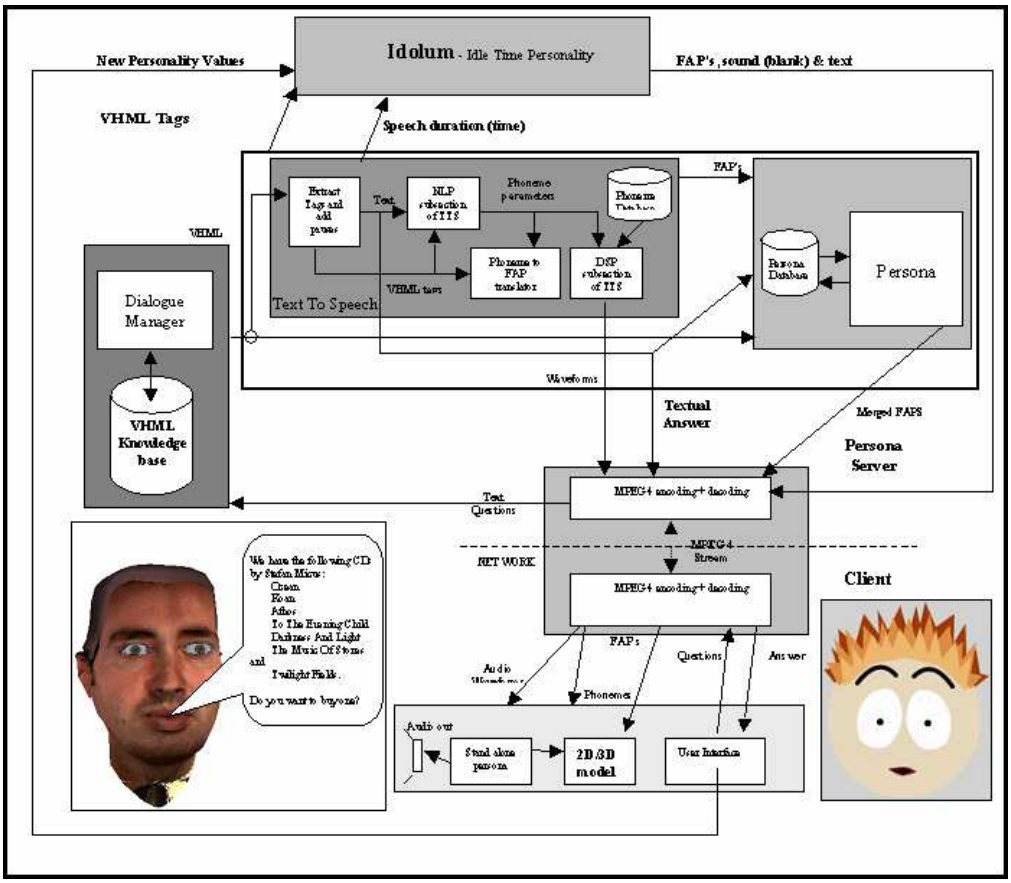


Figure 4.45 : A Talking Head system using the ***Mentor*** System DM

Figure 4.45 shows a system where a Talking Head (TH) was combined with ***Mentor***. The TH client could get the user input, send it to ***Mentor*** which processed it and returned the marked up or tagged response. The user could interact with a Talking Head by asking natural language questions and getting back an emotional, gestural, believable response.

The information returned from the DM's KB should be consistently marked up in a format that it is easily parsed and categorised - VHML. It should also be noted that the final rendering and delivery of the marked up text is context/application dependant such as in the five scenarios given on page 74 in Chapter 2 (Dialogue Management Markup).

As mentioned in the Section on XSLT Classes on page 115, the use of VHML allows the KB to be reused for multiple applications not just for a text based interface.

For more information on VHML, see Marriott (2002a) or the VHML website - http://www.vhml.org/ [HREF 60].

4.2.6.2. VHML: Issues

Uncertainties for users and implementers are deterrents to the use of VHML. They must be clarified. A number of issues arose from the extensive use of VHML (Marriott & Stallo 2002a) and these are summarised in the following section.

Semantics and Actions

What are the semantics of the tags and their attributes? For example, what exactly does the '*wait*' attribute mean in terms of implementation? Another issue is the question of what a VHML tag actually specifies - is it just syntax or is there unambiguous semantics behind a tag? What forms the actions of <**angry**> or <**disagree**>? What controls the rendering seen by the user? There are strong arguments by both those who provide the content and want control over its rendering, and by those who want the user to have control over the rendering because of some social, cultural or environmental requirements.

The solution seems to be via 'levels of implementation' - the semantics of a tag or attribute is given by intent as a default. For example, the intent of the <**disagree**> tag is to cause the Virtual Human (VH) to disagree in some implementation dependant way. This may be a shake of the head (culturally specific), a movement of the entire body, or a vigorous exaggerated movement for a cartoon-like character. The intent is to disagree; the manner in which it is done depends upon the implementation, the type of character, the character's personality, etc.

However, it is also very useful for content developers to be able to exactly specify what a tag does, especially in a tightly controlled environment. It is also needed if the TH's personality is allowed to define what they want to do when they 'disagree'.

So a mechanism must exist within VHML that can specify, at a lower level, exactly what an action means. This introduces two new problems: how to specify these actions in a standard way and how to specify non-visual actions? Fortunately, an international standard for specifying visual actions does exist - MPEG-4 Facial and Body Animation Parameters (ISO/IEC 1998a).

To specify non-visual rendering exactly - what does it mean to speak in an angry voice - it is not enough to give experimental results (I Murray & Arnott 1993a; I Murray, Arnott & Rohwer 1996a). A specification that is unambiguous is needed. Little has so far been done in this area but the work on the GESTYLE language by Ruttkay, Pelachaud *et al* (2003a) and Noot & Ruttkay (2003a) seems promising.

This 'level of implementation' solution allows an unsophisticated user to get animation quickly: implementation dependant or personality dependant actions / responses. A sophisticated user could also script unambiguous actions for the tags to force the animation to follow his/her requirements.

Levels of Implementation

As for many standards, implementation is an important issue. If VHML evolves to meet many needs, it runs the risk of becoming too cumbersome, too hard to learn, too hard to implement and hence an academic exercise at best. Current discussion by the VHML developers is concerned with the classifying of a VHML implementation at various levels.

Since VHML is structured around visual and audible renderings as well as hypermedia environment awareness, a more reasonable stratification may be along three or more dimensions. Implementation at $Level_{ijk}$ may be used to indicate a level *i* implementation for visual rendering, level *j* for audible, etc. This would allow implementers the ability to incrementally build a VHML system. This would mean that the VHML developers must be able to supply VHML test files for validation at these various levels.

Extensibility

Many emerging or existing standards allow for a controlled extensibility (for example, MIME types, OpenGL, X Windows) especially in areas where the application of the standard is varied. VHML already allows for the use of native language names for the tags and attributes. For example, it is possible to use the Swedish word, <**arg**> instead of the English word <**angry**> and a synonym, <**joyful**> instead of <**happy**>. A specific stylesheet has to be constructed for each language as well as for synonyms.

It has been seen that there is also a need for extending the language to allow for exact semantic definitions. This implies that any specific tag could be re-defined to do something totally different from its intent. In a similar way, a tag could be defined which groups a number of existing VHML tags together as one atomic VHML tag - a macro facility. It is a small step from this to where new tags could be defined along with their exact definition so that the language is extended but will still be usable by all compliant implementations. The problem is three fold:

- The VHML DTD must stay unchanged to accommodate the new tag
- The new tag syntax and semantics has to be defined within the VHML framework
- The new tag must be able to be used as naturally as possible within the text

Timing of the actions

The current specification does not address intra-action timing considerations. That is, how fast does the action start, how long to reach some level, how long at that level, etc. This attack-sustain-decay-release type timing is crucial to some applications especially those rendering subtle emotional effects. A solution would be to use the Spline Animation of SMIL2.0 (2001a) .

Parallelism

The current specification does not address the need for concurrent rendering/directing of VHs. This is necessary if multiple VHs are ‘doing’ things at the same time or the DM wants to send text and at the same time to open a browser for the client. The <**par**>, <**seq**> and <**excl**> tags of SMIL2.0 (2001a) should be used.

Personality Specification

A Virtual Human needs a personality. The specification of a personality model is not seen as part of VHML but as affecting it through the need for exact marking-up of how tags are rendered. The current version of VHML specifies that the <**person**> tag may indicate various optional aspects of the DM (or the TH that is rendering the DM output) such as the

age or gender as well as which emotion it is supposed to use by default for the rest of the element whenever there is no other emotion tag specified. The top level <**person**> tag must include an optional personality attribute.

The formal evaluation done at the end of the case study about aspects of using ***Mentor*** elicited the following response from one individual:

Not that courteous. ie it will be good to say "see you around" or "nice talking to you" instead of "bye" <- I find bye offensive

Table 4.9 : Evaluation comment on the most annoying aspect of ***Mentor***

The current 'programmed' personality of the DM, that is, the tone of the language used in the responses of the topics, tries to be informal and friendly. For some reason, the above user has not seen the language as friendly but has found it to be most annoying. However, other feedback has suggested that some users prefer a more formal response from the system, more in keeping with that associated with an academic authority. Therefore, the overall personality of the DM must dynamically match the expectation of the user if the system is to be effective.

A basic 'friendly, helpful, consistent' DM personality is assumed to be the most effective in a learning environment. Informal analysis has indicated that many Asian students come from a learning culture which is more 'directive' than 'explorative' - they prefer to be told exactly what to do rather than be shown possibilities and left to think about possible solutions. This broad categorisation could be used initially to specify a base personality.

The specification of the long-term temporal dynamics of a personality - what happens when the VH gets bored, is constantly harassed or is manic-depressive - also needs to be addressed. Some initial research has been done on the procedure of dynamically modifying the personality to best suit the user.

Finally, Dam & Souza (2002a) details early research using VHML where the weather information response, not just the way in which it is rendered, is constructed by being personalised to suit each individual's profile. This could address personality conflicts between user and DM.

For further information about this area see Ruttkay, Moppes & Noot (2003a) , and also the AAMAS2003 workshop on 'Embodied Conversational Characters as Individuals' website at http://www.vhml.org/workshops/AAMAS2003/ [HREF 61].

VHML tools

There is a dearth of high-level tools for the manipulation of XML in general and VHML specifically. Figure 4.46 and Figure 4.47 show some initial work done using the widely available Gvim editor (www.vim.org) and plugins. A user can configure menus to add functionality to the editor (for example, surround highlighted text with VHML tags). The menus can be detached to provide a complex tailored markup environment. Gvim is also a folding editor that knows about syntax and colour highlights it. In Figure 4.47, the tags have been turned near invisible so that the actual text content can be seen. For widespread use, a Word plugin needs to be developed with similar functionality

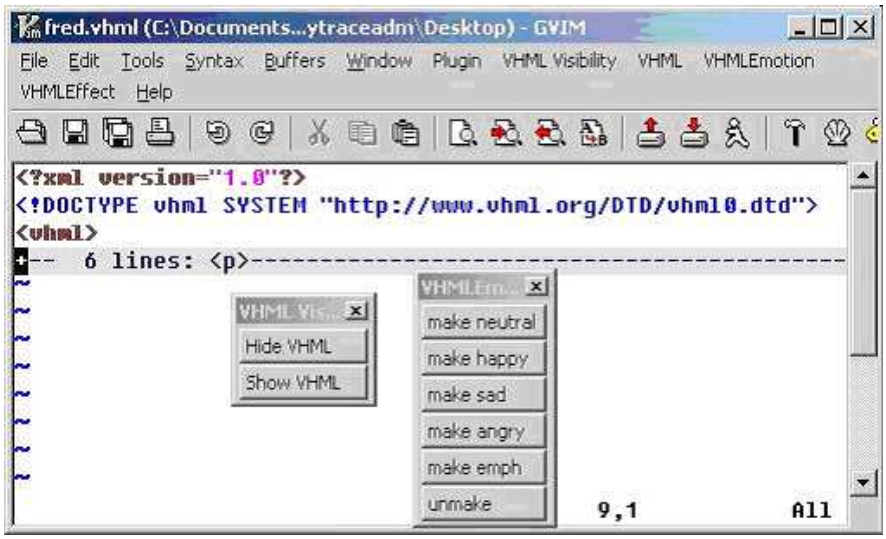


Figure 4.46 : GVIM plugins that allow VHML markup

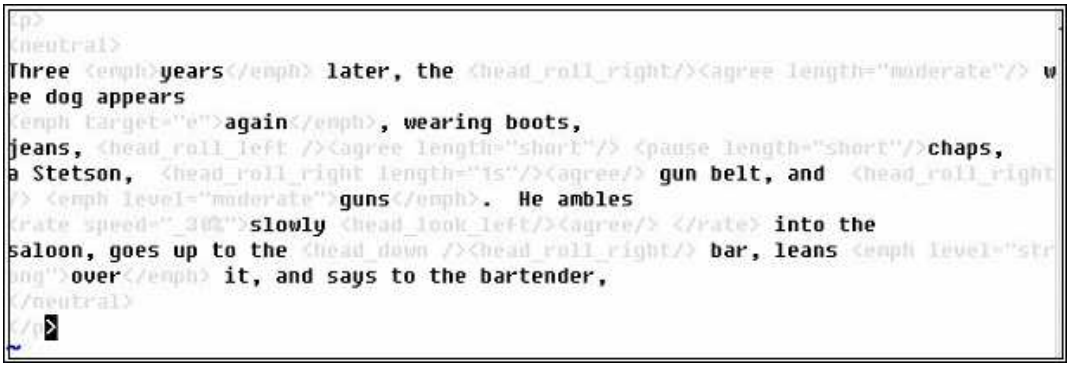


Figure 4.47 : GVIM plugin displaying VHML markup

VHML to specify input

Another VHML issue is that motion capture or audio/video analysis tools need to produce VHML as output. VHML is a high level description not just a low level motion specifying system; hence the output of the capture or analysis system should indicate <**angry**>, <**agree**>, etc.

A paradigm change occurred in going from text entry to the mouse-pointer concepts of a Graphical User Interface. In a similar way, it is now necessary for a total user input paradigm, adding video and audio input to the existing methods, to become the predominant Computer Human Interaction (CHI) of the future. This complete interaction is referred to as a gestalt user interface (gUI): an interface that should be reactive to, and pro-active of, the perceived desires of the user through emotion and gesture.

For the ***Mentor*** System, a better understanding of the user's immediate attitude to pro-active prompting would be beneficial. At the end of one of the studies, formal qualitative evaluation about aspects of using the system elicited the responses in Table 4.10 from one individual for the three questions shown. It is obvious that if the system had realised the intensity of the student's feelings at the time, through video or audio input, then it could have modified its pro-active question asking strategy and hopefully it would have been more

useful to the student (see also Klein (1999a) and Reynolds (2001a)). There is a definite need for a gestalt interface to applications that deal with humans. The use of VHML as a gUI specification language is detailed in Marriott & Beard (2003a) .

What was the least beneficial aspect: "That fucking beeping sound everytime I am doing something or every hour it fucking beeps."
What was the most annoying aspect: "The fucking beeping sound."
What aspects were obtrusive: "the fucking beeping."

Table 4.10 : Evaluation comments on the ***Mentor*** System

VHML's Future

A Talking Head directed by text with markup tags is perceived by users as being more human-like than one that just speaks (Marriott, Beard, Haddad *et al* 2001a). Using a markup language may mean that a Virtual Lecturer with appropriate persona and tags in its KB is seen as being erudite and approachable or a Virtual SalesPerson as trustworthy and helpful.

By using VHML to markup the ***Mentor*** System Knowledge base, it is possible to use the existing knowledge in future applications such as a Distance Education Virtual Lecturer where the student can ask questions of a TH which looks, and acts, like the real lecturer.

Acceptance of VHML by researchers and users will depend upon it meeting the criteria required of any potential standard:

- Completeness - is it complete in itself for the targeted audience?
- Simplicity - is it simple enough for people to learn and quickly use?
- Consistency - is it consistent so that people can use it without needing a complex manual?
- Intuitive - is it obvious what the tags do and mean?
- Unambiguous - is it unquestionably spelt out as to what the language does?
- Abstraction - does it provide the right level of abstraction for the targeted audience and function?
- Extensible - is it possible to extend the language without breaking it?
- Usability - is it usable in terms of implementation and functionality?
- Effective - does evaluation indicate that the user experience is positive and/or rewarding?

VHML's evolutionary development has been influenced by many researchers from many disciplines. It will continue to be defined/refined/corrected with input from workshops, discussion groups and researchers.

Research on Dialogue Management, the ***Mentor*** System, and VHML resulted in the researcher being invited to a week long, invitation only seminar on Evaluating Embodied Conversational Agents (ECA). From associated email (Ruttkay 2003a):

> *The main point is not so much to give a presentation on your own work (though people will have the chance to do so), but to WORK HARD DURING THE 5 DAYS to come up with something really useful and awaited for, for the ECA community on evaluating/comparing ECAs.*

That seminar, like the ***Mentor*** System and VHML, was trying to address a real need for concrete solutions to existing problems in Dialogue Management and Embodied Conversational Agents.

4.2.7. The *Mentor* Educational Support Applications

This research was not trying to determine what educational paradigm would emerge from the system but whether imposing proven teaching method paradigms onto a software system was possible, and whether it was effective and beneficial to the students using it.

The system architecture was developed to support an Authoritative Guidance (AG) paradigm and a Directed Learning Path (DLP) paradigm. AG is based on the principle that users may want to learn in their own way but ultimately require expertise in that area. The DLP is essentially a step-by-step guide with checks.

Applications that supported these paradigms were also developed to help the lecturer and/or the Knowledge base builder manage this aspect of the research. This section details these applications, and the issues that arose in their development and use.

4.2.7.1. Authoritative Guidance Support

The majority of user questions are answered immediately by ***Mentor*** using its KB. Those that it cannot, are forwarded to the Lecturer-in-Charge (LiC) of the unit, who can create a relevant answer and have that returned to the user. The next time the user logs onto the system, or when they have a free moment, it asks if they would like an authoritative answer to their question.

Within the system, the `DialogueManagerUserQuestion` class facilitates the easy management of these delayed question-answers. Any question that cannot be immediately answered is saved in an XML-based format in a file associated with that user. The LiC will receive notification of the question by being prompted by a GUI to supply an answer (Figure 4.48). The GUI selection dialogue box shown has a list of users with outstanding questions (greyed out for anonymity). These users may be processed sequentially by the LiC, by using this and two other GUIs - Figures 4.49 and 4.50.

Figure 4.49 shows the GUI selection dialogue box used by the LiC to answer questions from the user. Note that users do not always ‘ask questions’: in the last three examples, they have used the system like a search engine by typing key words. This is an issue for a natural language dialogue system since just typing keywords is not natural. One mechanism to address this issue is that the REs could become very general and ignore non-essential words. Another solution to this could be to have a ‘no match’ Topic that reformulates the user’s input question into something like "help me with xxx", and this question is then reprocessed. Examination of this issue and of the solution is seen as future work.

Double clicking on a question (Figure 4.49) will bring up the LiC’s ‘favourite’ editor and when he/she has finished creating that answer, the question-answer can be saved via the ‘Update Ques’ button. This makes the question-answer available for returning to the user.

The LiC can also save this particular question (‘Save Ques’ button) for later analysis - these questions may provide insight into a particular problem that users are facing or they may simply be used for statistical reasons. All questions are automatically saved in a special file anyway. The answer may also be saved into the ‘frequently asked questions’ (FAQ) answer file via the ‘Save Ans’ button. Figure 4.50 shows the contents of this file and the LiC may use an existing answer via the ‘Use Ans’ button.

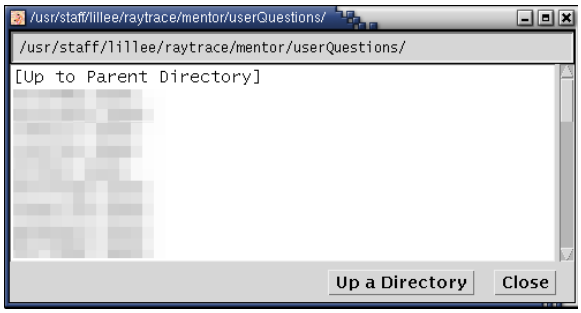


Figure 4.48 : Selection dialogue box for all users with questions

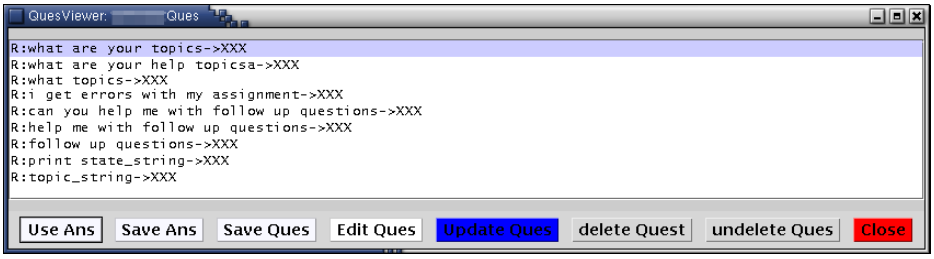


Figure 4.49 : Selection dialogue box for a user's questions

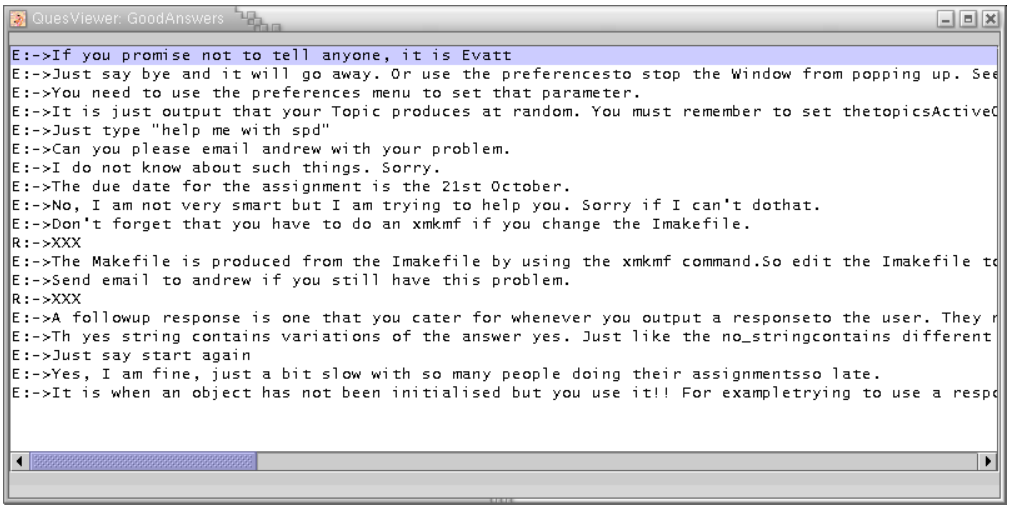


Figure 4.50 : Selection dialogue box for good/typical answers to user questions

Since not all questions from a user need answering - some may simply be comments to ***Mentor***, or some may be typographical errors - the LiC can also delete questions.

Once a question has been answered, the next un-answered, un-deleted question is chosen for the LiC to answer. They may use the 'Edit Ques' button to start the answer editing process again. When all questions have been processed, the LiC can close this GUI and the next user with questions will be presented to them. This continues until all user questions have been processed. Once this occurs, the GUIs iconify and wait for the ***Mentor*** System to create more questions from users that need answering. The GUIs then deiconify and the question-answering cycle starts again.

The saved question-answer pairs can be used to form new REs in existing Topics or to create new nodes via the Topic Network Builder (see Section on Topic Network Builder on page 176). They can also be used to modify existing REs to cater for unexpected ways of asking a particular question or because of the use of new terminology such as SMS abbreviations.

Authoritative Guidance Issues

The AG paradigm works well, with 72% of users rating its usefulness in the third case study as being 3 or above in a 5 point Likert scale (see Table 5.35 in Section 5.3.5.4 on page 250).

However, a number of issues arose from the use of the support applications with regard to the cycle of the LiC returning authoritative answers. Unless the LiC kills the GUI application, this cyclic process continues forever. This may seem draconian but it ensures that the LiC processes the questions in some timely manner. Note that these are the questions that the DM system could **not** answer and hence timely answers are probably important to the user. The AG GUI system is set up to minimise interaction. That is, the LiC is presented with all they need to process the question with minimal mouse clicks. In this manner, questions may have a turn-around time - the time between the user asking and the time that a delayed answer is presented to them - of less than a minute. This can be even less if the question is 'typical' and hence the answer can be obtained from the FAQ file.

However, this process is very time consuming when the LiC has been absent from their academic environment for a significant time. For example, after a weekend, the LiC may be presented by many questions from many users (especially when an assignment is due) and this can typically take 15 to 30 minutes to process.

Feedback from students via the questionnaire evaluation has indicated that delayed answers that are returned with too long a delay are often no longer relevant:

> *The answer was helpful but I had already figured it out*

One possible solution to this issue is that the questions can be displayed along with the date that they were asked. The GUI button panel could be modified to easily allow for a "*do you still want to know the answer to this question*" type response to be sent to the user, if the question was past a certain delay time. If they say "*yes*", then the question is re-presented to the GUI system with a new date. If "*no*", then the question is automatically removed from the question-answer process.

The choice of delay time is problematic. It would be unwise to choose a fixed delay time since long delays between questions and answers may be acceptable early in the user's assignment development but towards the end, time delays become critical. It is also necessary to ensure that a LiC does not just defer answering questions as their default processing action. The solution to this is also seen as future research.

Another issue is that currently the LiC is the only authority who receives the request that the system could not answer. Many requests need other expertise. For example, the technical staff may be the authority for certain questions about laboratory information, the enrolment officer will have up-to-date information about a student's enrolment. The current application should have a mechanism for forwarding the request to the appropriate authority, and then receiving and processing the returned information. This should be done with minimal keystrokes so as to make this an easy task for the LiC.

4.2.7.2. Directed Learning Path Support

The DLP is a simple question and answer path through the problem. It helps a student initially in understanding what is required, and progression along the path helps the student in implementing and debugging the solution. It uses the expertise of the LiC in terms of planning and implementing the code and in typical problems associated with this domain. For example, many students do not know the benefit of code re-use and so do not start their design process with a known workable code base. The ***Mentor*** System can suggest that the student obtains a copy of some existing code and may suggest how to initially modify it. So students can learn by example and learn by doing and hopefully will be following best practice techniques laid down by the LiC.

A DLP is normally encapsulated in a pro-active Topic that has a deep state path. That is, it can ask questions and it can also carry on an extensive dialogue with the user, not just a series of single questions and those questions' answers.

For the KB developer, adding a new topic to increase the knowledge of the system was straightforward but tedious: copy the Java file that represented an existing topic, change the relevant module names, then add the patterns and responses. This method was not suitable for any large-scale KB development, especially so when considering the production of information about units and DLPs.

Also, mentoring is not just having knowledge about a particular topic - it is concerned with guidance based upon expertise, timely reminders, the tailoring of a path through the obstacles to help the mentee reach the final goal.

Given these concerns, and also that the system must be usable by non-programmers, it was necessary to develop an application to construct complex dialogue networks and DLPs. A Topic Network builder application was developed to help in the construction of patterns, rules, responses and next states.

Topic Network Builder

Figure 4.51 shows an example of using this tool to construct a Topic about fish. The circular and arrowhead nodes each represent a state (=RE pattern+responses+next states) and the square node is a supernode which holds a collection of nodes. Notice that a supernode called 'salmon' exists but is not connected at the moment. The edges indicate possible state transitions. An arrowhead node addresses initial or Entry level questions whereas circular nodes cater for followup questions.

A circular node may be created by clicking in the application or, to create a super node, by clicking with the Shift key held down. Double clicking on a node selects it, and double clicking on a super node opens it to display its contents. Edges can be drawn between nodes.

Each of the nodes can be selected and will change the network list GUI as shown in Figure 4.52. In this figure, node 6 is currently selected and its attributes can be seen. Each of these attributes can be changed and affect the network as a whole. For example, the edges between nodes are constructed from the 'Next States' attribute. Notice the use of regular expressions in the patterns and the use of a non-deterministic choice of responses. VHML tags can be used to print out the person's name, etc.

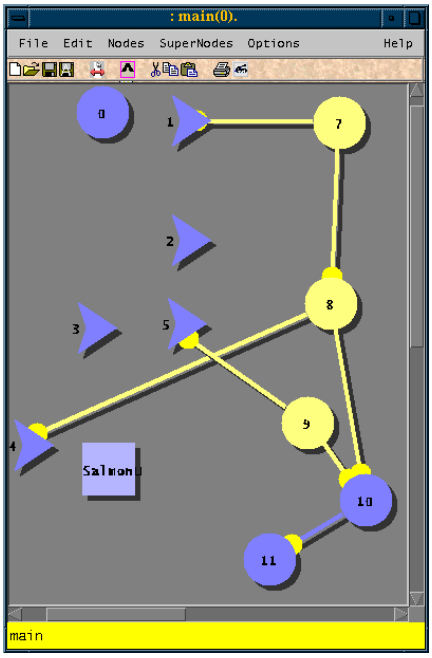


Figure 4.51 : Topic Builder GUI

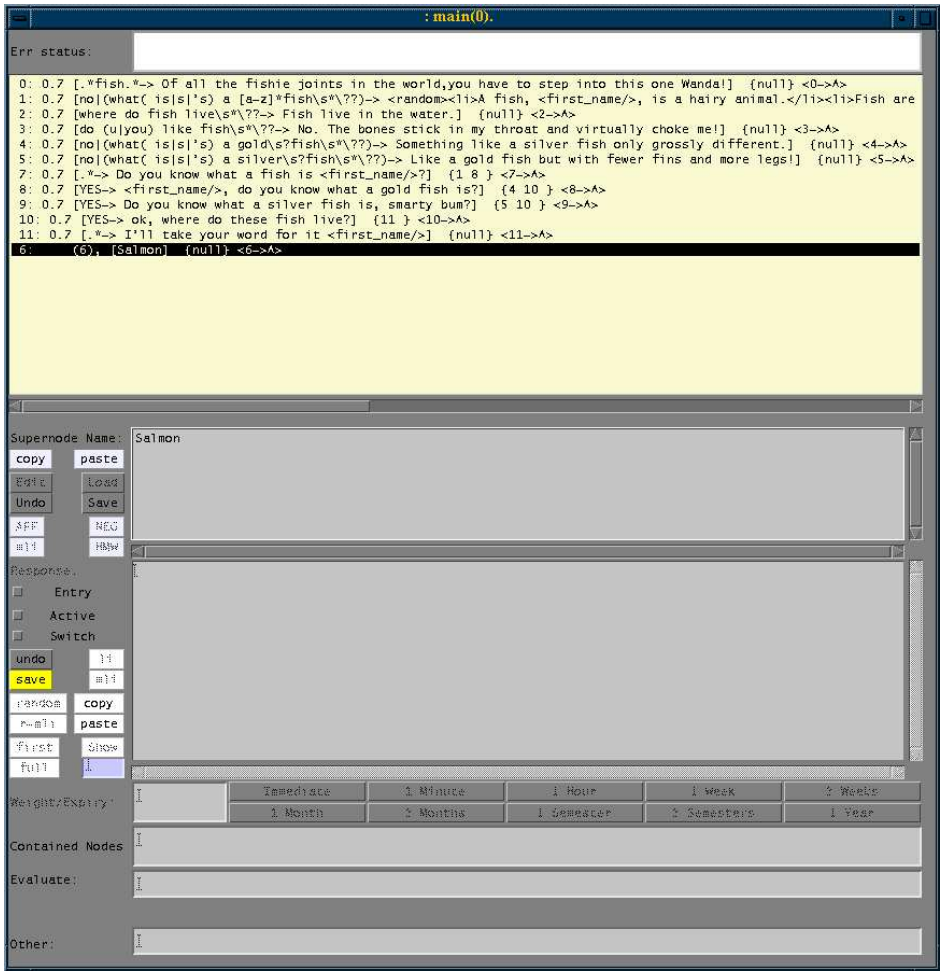


Figure 4.52 : Topic Builder GUI - node information

An edge from one node to another node indicates that the second node represents a ‘followup’ question, and is processed accordingly by the DM algorithms. Nodes whose ‘Next State’ points to a node inside a supernode will have an edge to that supernode. Similarly, nodes inside a supernode that point outside will have an indicative but truncated edge. Nodes may be freely added to and removed from supernodes. Note that each node in each constructed Topic has a unique name.

The lower left corner of Figure 4.52 shows GUI controls that allow a user to quickly build typical request-response nodes as well as DLPs. Figures 4.53a and b show the network for a particular unit topic that has several areas of knowledge, each of which is represented as a supernode (remember that these supernodes will be interconnected as well).

See Appendix J for more information on using this application to build Topics.

This network of nodes can be saved for later editing and also exported as Java data. This information is stored in arrays in the constructed Topic (see Figure 4.55 on page 180). This Java data file is merged with a standard template Topics file to produce a Java file that can be compiled to become a standard Topic that the ***Mentor*** System uses. The system now knows about fish and will respond as indicated. If special processing is required, then as a last resort, the created Java file can be hand edited.

This constructed Topic extends a `graphTopic` class and hence inherits specific `checkTopic` and `checkFollowupResponse` methods that utilise the information of these arrays. When the DM processes this Topic as per the algorithms of Figures 4.10, 4.11, and 4.12 in Section 4.2.1.1 on page 121, node 0 indicates the general keywords for this topic. If the user’s input does not match this, then the entire topic is skipped. If it does, then the graph’s entire list of Entry nodes, stored in arrays in the constructed Topic, is then checked to see which is the most appropriate match. Similarly for a followup question. See later discussion on page 182.

A Lecturer can create extensive Domain Knowledge Topics for a unit in this manner, with the system producing similar but more complex files to that shown in Figure 4.55.

Figures 4.53a and b show the high level structure for a Topic for the third case study. The lighter nodes are Active nodes - they hold pro-active queries such as are shown in Figure 4.54. That is, a random response, using parts of the user’s name, asking about the unit, to see if everything is going OK. It can be seen in Figures 4.53a and b that various Next States are linked to cater for answers from the users.

The arrow shaped nodes in Figures 4.53a and b are Entry nodes that contain a question/request RE that may match an initial input request from the user, but not a followup request. In this way, only Entry nodes are searched for a pattern match for an initial request, not all nodes in the network for this Topic. This obviously reduces the search space for a match. This Entry node may have followup nodes that address any issues raised by the response (for example those in Figure 4.53a) or, as is typical of the organisation for a unit, a collection of these Entry nodes may simply provide answers (Figure 4.56a). A text file containing lists of questions/answers can be imported into the application to quickly create these Entry nodes.

Figure 4.53b shows a diamond shaped node - this is a general Switch node that allows for the creation of DLPs. In this example, many Entry nodes point to this Switch node to process any Next State requests. This means that the **response** to this request does not come from the Entry node but from some part of the DLP connected to this Switch node.

For example, one of the Entry nodes may match the user request "help me with spd", another may match "I need help on my assignment", and another may match "where do I start with my assignment". The response to this will come from the DLP. Figure 4.56b shows a part of a simple DLP, with the round cornered square nodes being special DLP nodes. See Section 4.2.7.2 on page 184 for detail on how this path is processed.

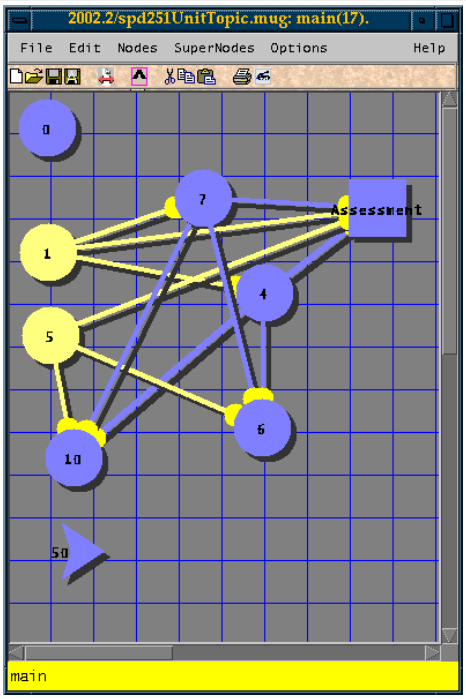


Figure 4.53a : First example

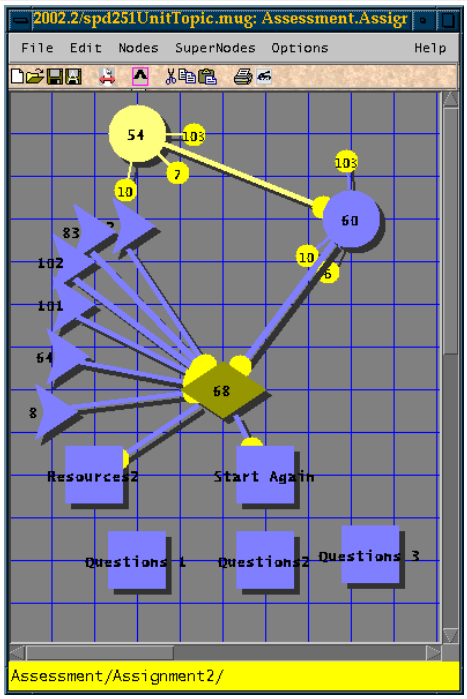


Figure 4.53b : Second example

Figure 4.53 a and b: Topic Builder GUI - example nodes from Topic for the third case study

```
<random>
<li><double_first_name/>how is <topic_name/> going <double_first_name/>?</li>
<li><double_first_name/>are you happy about <topic_name/><double_first_name/>?</li>
<li>Is <topic_name/> going ok, <first_name/>?</li>
<li>How is <topic_name/> going, <first_name/>?</li>
<li>Are you coping with <topic_name/> <first_name?/></li>
<li>Are you ok with <topic_name?/></li>
</random>
```

Figure 4.54 Random pro-active queries from VHML data

To create a Java file from this network data, the header and trailer files of Figure 4.55 are included by running the data through the C pre-processor (`cpp`).

The header file contains various methods that access the data stored in the various class fields such as `activeStates` or `followupPatterns`. The resulting class file extends `UnitTopics` which in turn extends `graphTopic`. Note that not all data files produced by the Topic Network Builder represent unit knowledge. Other types of files can be processed in a different manner but all will ultimately be extended from `graphTopic`.

```
#include "header"
    protected transient String patterns[] = { ".*fish.*" };
    private static final int initialStates[] = { 1, 2, 3, 4, 5 };
    protected transient String followupPatterns[] =
    {
    /*0*/     ".*fish.*",
    /*1*/     "no|(what( is|s|'s) a [a-z]*fish\\s*\\??)",
    /*2*/     "where do fish live\\s*\\??",
    /*3*/     "do (u|you) like fish\\s*\\??",
    /*4*/     "no|(what( is|s|'s) a gold\\s?fish\\s*\\??)",
    /*5*/     "no|(what( is|s|'s) a silver\\s?fish\\s*\\??)",
    /*6*/     ".*",
    /*7*/     "YES",
    /*8*/     "YES",
    /*9*/     "YES",
    /*10*/     ".*",
    };
    private static final String followupresponses[][] =
    {
    /*0*/  { "Of all the fishie joints in the world,\nyou have to step into this one Wanda!" },
    /*1*/  { "<random><li>A fish, <getvar name=\"first_name\"/>, is a hairy animal.</li> \
              <li>Fish are the devil's spawn, <getvar name=\"first_name\"/>.</li> \
              <li>A fish is like an undersea bird!</li> \
              <li>Fish! Fish! What is this fascination with fish!!\
                  Get a life <getvar name=\"first_name\"/>!!</li>\
              </random>"
         },
    /*2*/  { "Fish live in the water." },
    /*3*/  { "No. The bones stick in my throat and virtually choke me!" },
    /*4*/  { "Something like a silver fish only grossly different." },
    /*5*/  { "Like a gold fish but with fewer fins and more legs!" },
    /*6*/  { "Do you know what a fish is <getvar name=\"first_name\"/>?" },
    /*7*/  { "<getvar name=\"first_name\"/>, do you know what a gold fish is?" },
    /*8*/  { "Do you know what a silver fish is, smarty bum?" },
    /*9*/  { "ok, where do these fish live?" },
    /*10*/ { "I'll take your word for it <getvar name=\"first_name\"/>" },
    };
    private static final float followupweights[] = { 0.7F, 0.7F, 0.7F, 0.7F, 0.7F, 0.7F,
         0.7F, 0.7F, 0.7F, 0.7F, 0.7F, };

    private static final String followupnextStates[][] =
    {
         { "1", "2", "3", "4", "5" },
         {  }, {  }, {  }, {  }, {  },
         { "1", "8" },
         { "4", "10" },
         { "5", "10" },
         { "11" },
         {  },
    };
    private static final String followupothers[] =
    {
         "-ta fish -taqw 0 -bths Knowledge about fishie things -tkp fish",
         "", "", "", "", "", "", "", "", "", "",
    };

    private static final String followupevaluate[] =
    {
         "-ta fish -taqw 0 -bths Knowledge about fishie things -tkp fish",
         "", "", "", "", "", "", "", "", "", "",
    };
    private static final String TOPICALIAS ="fish";
    private static final String TOPICKEYPATTERN ="fish";
    private static final int TOPICACTIVEWEIGHT =0;
    private static final String numbers[] = { "0" };
    private static final String BRIEFTOPICHELPSTRING ="Knowledge about fishie things";
    private static final String TOPICHELPSTRING = "Of all the fishie joints in the world,\n" +
         "you have to step into this one, Wanda!\n";
    protected final int StateVSIndex[] = { 0, 1, 2, 3, 4, 5, -1, 6, 7, 8, 9, 10, -1, };
    protected final int activeStates[] = { 7, 8, 9, };
    protected final int intSwitchNodes[] = { };
    protected final int intAffirmativeNodes[] = { };
    protected final int intInitialPatternNodes[] = { 1, 2, 3, 4, 5, };

#include "trailer"
```

Figure 4.55 Example cpp processed Java code output from Graph GUI

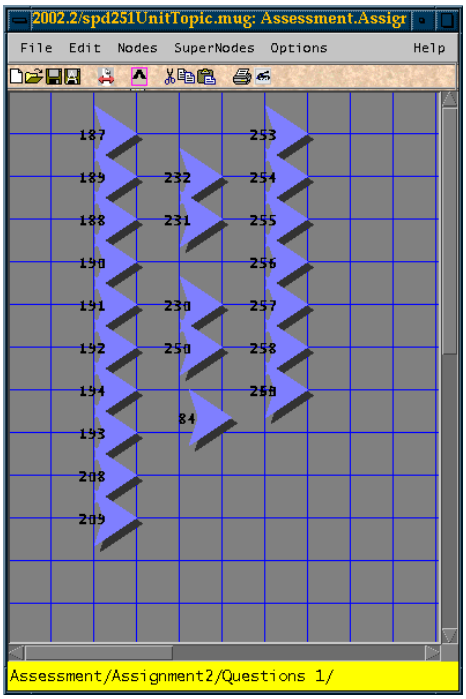


Figure 4.56a : First example

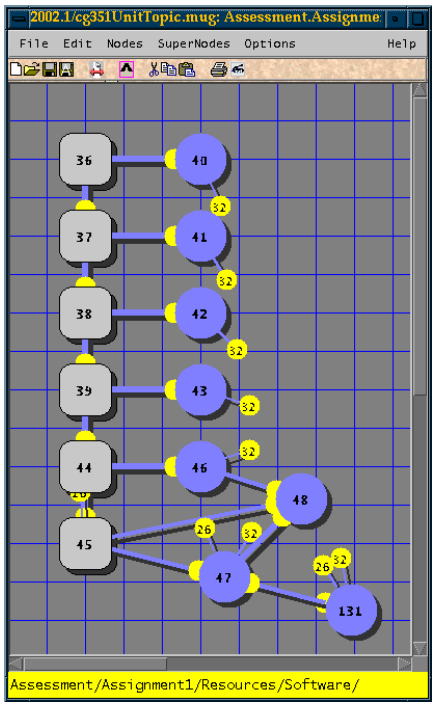


Figure 4.56b : Second example

Figure 4.56 a and b: Topic Builder - example of multiple question and DLP nodes

The trailer file does little except include an optional `main` method for the Java class. This may be used to test various aspects of the unit Topic and is typically used to ensure that all responses are legitimate VHML. This is important since non well-formed XML constructs are specified to cause a fatal error and hence the Topic will fail at run-time if a badly formed VHML response is transformed by the XSLT transformer. These badly formed responses typically come from the developer forgetting to close <**`img`**> tags or not quoting all attributes.

The resulting file is then transformed by the Unix utility `sed` through substitution of REs for commonly used upper-case strings:

```
s/YES/(\\s*(ok|yep|ya|yes|y|yea|yeah|sure|(("+YOU_STRING+"\\s*)?bet)|((of\\s*)?course)))/g
s/NO/(\\s*(no|nope|na|nuh|n|nix|nay|nah))/g
s/MAKEING/(\\s*((mak(e|ing))|(creat(e|ing))|(build(ing))))/g
```

as well as specific dates and resource information:

```
s/SEMESTER_START_TIME/Feb 18/g
s/SEMESTER_END_TIME/Jun 21/g
s#MENTOR_HELP#http://"+MENWWW+"/mentor/help/#g
s#MENTOR_IMAGES#http://"+MENWWW+"/mentor/images#g
```

This is then processed by a unit-specific shell script that typically transforms the file with more unit-specific substitutions (Figure 4.57). The resulting Java file can be processed as a normal Topic file by the DM system, using the various field values to check for pattern matches against user input, and for changing state based upon correct responses.

```
s#UNIT_NAME#(?:(?:(?:(?:sys(?:tem(s)?)?\\s*)?program(m)?ing)|spd|usp)\\s*-?\\s*(251|501|))#g
s#ASS_WHERE#http://"+MENWWW+"/units/spd251/notes/infoe1.html#g
s#EXAM_WHERE#http://"+MENWWW+"/units/spd251/examinations#g
s#ASS_WHEN#http://"+MENWWW+"/units/spd251/notes/infoe1.html#g
s#ASS1_START_TIME#Jan 09#g
s#ASS1_END_TIME#Oct 18#g
s#ASS2_START_TIME#Jan 20#g
s#ASS2_END_TIME#Oct 24#g
s#ASS2_FILE_AREA#/usr/units/spd251/ass2.022#g
s#ASS2_SPEC#http://"+MENWWW+"/units/spd251/assignments/ass21.html#g
s#REG_EXPR#((perl(\\s*5)?\\s*)?reg(ular)?\\s*expr(ession)?)#g
```

Figure 4.57 Example unit specific definitions

UnitTopic Processing

For any Topic, there are three main methods:

`•checkTopic` `•checkFollowupResponse` `•activeQuery`

For a `UnitTopic` class, these methods use the data fields to get Node numbers and their associated patterns and responses, for testing. The algorithm for the `checkTopic` method in the `UnitTopic` class is shown in Figure 4.58.

```
public Response checkTopic(.....)
{
   response = null
   For each node in the array,
      Get its node number
      check its pattern[node_number] against user input
      if a match then
         check that it can be considered based on evaluation fields
         if it can be considered then
            get the response plus weight, next states,etc from this node
            if this response weight is greater than any previously found then
              set this as currently selected response
            Endif
         Endif
      Endif
   Endfor

   If a valid response exists after all nodes for this unit have been checked then
     Ensure that any post-processing of user state variables is done by the tidyUp
        method of Topic that provides the valid response.
   Endif
   return response
}
```

Figure 4.58 Pseudo code for `checkTopic` method

The method extracts the relevant data from the arrays to perform the pattern matching and to get the subsequent response, weight, next states, etc. Note that only ‘Entry’ nodes are checked. That is, only those nodes that have been specified as matching initial questions, not those created to handle ‘Next State’ patterns.

Figure 4.59 shows the algorithm for the `checkFollowupResponse` method. This method is similar but processes Next State nodes using their followup patterns.

Notice that both methods check the ‘evaluate’ field to ensure that the node is valid. For example, the method should only respond to this request from this node if the current date is between two date boundaries - between the starting date and the due date for an assignment. Or, only respond if this response has never been given before.

The user’s history is adjusted if the node is chosen to provide the response. Typically this means ‘set the node as having been visited’. Any post-processing commands are executed and these may involve setting user variables or even un-visiting nodes or entire

DLPs. For example, if a user says "yes" when asked if they have started their assignment, then this should be noted so that they are not asked again, or so that appropriate action can be taken by other nodes that may respond in the future. Similarly, the overcoming of a hurdle in the DLP may be noted so that this is not tested again.

```
public Response checkFollowupResponse(.....)
{response = null
 Get index of next state node from string.
 Get followuppattern RE and check it against the user input.
 If a match then
   get followup node and check that it can be considered based on evaluation fields
   If it can be considered then
     get the response, weight, next states,etc from possibly Switch node hierarchy
     If a non-null response exists then
       Ensure that any post-processing of user state variables is done by tidyUp
       Ensure that User's history is updated - node has been visited- by tidyUp
     Endif
   Endif
 Endif
return response
}
```

Figure 4.59 Pseudo code for `checkFollowupResponse` method

Figure 4.60 shows the algorithm for the `activeQuery` method. The pro-active responses have two forms: a generalised "did you know" response for giving useful facts about the unit, and a normal response such as "have you started your assignment yet?". When a `UnitTopic` is chosen for a pro-active response, there is an equal probability of the response coming from either source. Therefore a random "did you know that xxx" (typically retrieved from a cached file), or a random response from one of the Active nodes is returned. Again, the evaluation and the post-processing of the user state is done.

```
public Response activeQuery(....)
{
   response = null
   If not disabled for didYouKnow type active queries
     See if this graph Topic has any didYouKnow type active queries
       If so, randomly choose to return a random didyouknow

   If not chosen then
     if this graph Topic has any active queries then
       Get index of random active state node
       check that it can be considered based on evaluation fields
       if it can be considered then
         get the response plus weight, next states,etc from this node
         Ensure that any post-processing of user state variables is done by tidyUp
         Ensure that User's history is updated
       Endif
     Endif
   Endif
   return response
}
```

Figure 4.60 Pseudo code for `activeQuery` method

These three methods are efficient in processing the general Domain Knowledge held in the various arrays. The overall system allows a non-computing professional to set up complex Domain Knowledge Topics with common point-and-click skills. The nomenclature used - VHML, REs, and the meaning of the evaluation fields - needs to be learnt but this is similar to mastering any new tool. This issue is further discussed on page 185.

DLPs can be easily constructed, allowing a lecturer to concentrate on helping the user rather than on how to create nodes or links. In the end, the processing of these network nodes is transparent to the Topic developer.

Directed Learning Path Processing

The algorithms of Figures 4.58, 4.59 and 4.60 produce a response to the user input if a match occurs. Which node produces the actual response returned is complicated by the functionality of the Topic Builder Switch node.

The DLP traversal of Figure 4.61 is a recursive descent algorithm that finds the next appropriate node in the DLP. That is, it will look at all possible paths from that Switch node to find the next un-visited node that also 'evaluates' to be true.

```
Examines the response string of the successfully matched node.
If no response string has been set for this node then
  For each of the Next state nodes
    examine the node to see if this is a Switch node, set found flag if so.
  Endfor
Endif
If no response found and no Switch node, then  return null
If no Switch node found then
  return the response string of this node.
Else
  traverse the DLP represented by this Switch node, return the appropriate response
Endif
```

Figure 4.61 Pseudo code for node traversal algorithm

This traversal algorithm must accommodate three cases. Initially the user will start the DLP and will probably have done so by asking for help. Once on the path, users may respond to questions from the nodes and hence the path must be traversed to find the appropriate followup node. Finally, the user may simply ask for help at a later time and the path is traversed again triggered by this initial request. In this case, the appropriate node in the DLP corresponding to the next step in the learning process will supply the response.

In each of these cases, as the user advances along the DLP, the node will be marked as visited and any post-processing will take place. The user's history, saved in a file at the end of each session, contains the list of visited nodes.

Note also that the evaluation criteria means that users may be given hurdles to pass before they can move on. Also the post-processing may be used to block off entire sub-paths. For example, the user may have arrived at this node having already completed a scheduled learning task. It would be incorrect to try to lead the user down a path they had already trod. Another example is where a user simply wants to start down the path again (perhaps by saying "I want to start again!") and hence the post-processing done by the node that recognises this request would un-visit nodes, sub-paths, or the entire DLP.

A DLP (see Figure 4.56b) can be as simple or as complex as the DLP builder requires. Typically, a simple yes-no structure can be created. That is, the node will indicate that a certain task now needs performing, and then asks the question "have you done this?" or says "go and do this!". The followup nodes will either match the "yes, i have done this" (round cornered square node in the first column of Figure 4.56b) or will match "no" (node to the right of the responding node in Figure 4.56b). Obviously other patterns can also be used to match user responses such as "ok", or "later", or "thank you" (see the small yellow node numbered 32 in Figure 4.56b) and these simply make the graph more complex.

Since **each** Next state node is searched, a DLP can branch to form different paths to guide the user. This accommodates different learning styles and paces.

Directed Learning Path Issues

The DLP system was successful in guiding students along the learning path - a quote from the evaluation of the second case study:

> *This was THE best thing about the Mentor System. I couldn't have done without it.*

However, the issue of hurdles remains. That is, a user should be able to prove to the system that they have worked their way along the path, not just said that they have done everything that is asked of them, hoping that there is a solution at the end of the path. Currently, the DLP system can ask questions about various tasks that the user should have performed (see the interconnected nodes at the bottom of Figure 4.56). The user cannot go past this section of the DLP until they have given the right answer.

The issues with these hurdles are two fold. Informal discussions with students have indicated that if the system asks too many questions as the user travels down the path, then it becomes less attractive as a source of information and as a helper. Also, users do not always understand or see the importance of some information until **after** they have passed a certain point. Things may 'click into place'. The balance between checking a user's real progress and the need to be seen as helpful and open, is difficult to determine on a per user basis. This issue is seen to be necessary for providing better help to students and is a future area for research.

The second issue is that users may not be able to answer even simple questions if they have arrived at that point in the path by using some supplied information but not other. For example, the system may ask: "what was the output when you typed xxxx?". This assumes that the student has arrived at this point by following the previous suggestions. This is not useful and may be unanswerable to someone who has taken a different, perhaps better, or perhaps more individual approach to the problem.

The node that must recognise the answer to this question needs to be very complex because the possible answers may range from the correct answer (in many varied forms) to a response that says "I didn't do it that way!", again in many forms. Since this could happen at each node or at each hurdle node in a DLP, the system could easily become unwieldy for the DLP builder.

The current solution to this issue is to err on the side of simplicity for the DLP builder. One possible solution for future research may be to define and encapsulate a hurdle node as a separate type of node and to build in specialist knowledge that can help the builder to cater for the aforementioned issues.

The Topic Builder application allows for the point-and-click development of Topics by non-computing people. However, it is naive to assume that non-computing people would be happy in creating complex Perl-5 REs to match the possible user input. One possible solution to this issue is for the DLP developer to type in typical questions and then let the Topic Builder application generate suitable Perl-5 REs based upon some heuristic that took into account synonyms, common spelling and typographical errors, and even common abbreviations or SMS use.

4.2.7.3. Other Educational Paradigm Issues

One Educational issue is that the system has not been tested with other paradigms. The Topic Builder application has quite specific buttons to help build a DLP, and the system methods to process the network data files generated, perform traversals that understand the DLP structure. Support for different paradigms within the DM system, as well as through new applications similar to the Topic Builder, is seen as future work.

For example, another paradigm that has been seen to be beneficial between a student and another (expert) user, is that the expert user plays dumb and gets the student to explain their problem to him/her and in doing so, the student's inner reflection about the problem helps them to understand their errors. This expert is typically called a Duty Programmer in Computing disciplines.

Therefore, it would be interesting to implement a 'Stupid Mentor' paradigm, where the system has knowledge but only provides it to help a user move forward in their own explanation of a particular problem. This could be accommodated within the existing structure of the system and the network builder application. However, as for the creation of new 'hurdle' nodes described in the previous section, the system may benefit from a more specialist approach.

This task is quite complex from a Natural Language Processing point of view and may show shortfalls in the RE pattern matching that is used in the ***Mentor*** System. Pattern matching is good at determining short user requests, but may not be suited for understanding complex descriptions of a user's problems. Results of a recent inconclusive study (Aleven, Popescu, Ogan & Koedinger 2003a; Popescu *et al* 2003a) have indicated that this may be a major endeavour for any system.

4.3. Limitations

The developed system has some limitations, although several of the initial limitations / delimitations of Chapter Three have been relaxed or catered for by the developed system:

(1) The system was initially planned to only be available in a Unix-based environment that the student group would use. The client-server architecture design relaxed this initial limitation.

(2) The system was initially planned to only be available at the university and not from a student's home computer. Requests from the formative evaluation of the case studies prompted development that relaxed this initial limitation so that the system could be accessed (with authentication) from anywhere in the world.

(3) The study was initially limited to Unix-based clients. Requests from the formative evaluation of the case studies prompted development that relaxed this initial delimitation so that the system could be accessed from Windows based clients as well.

(4) The system was not planned to accept speech input. However, given the research done by people such as Kadous & Sammut (2002a), it would appear that a speech enabled front-end could be easily developed to cater for multiple sentence estimates

being parsed by the system, and the most "reasonable" one being used to provide a Response to the user.

This designed-in extensibility also caters for multimodal input such as the emotion of the user, and will enable this to be used when the technology matures to be able to provide such input from audible or visual cues from the user. Multimodal output is also available via the use of VHML.

The limitations of the developed system are:

(1) The system can accommodate a significant number of connections or users at any one time with good response time (see Section 5.3.6 on page 258), and this was using the old server. The current production machine is a PIV running at 2.0 gigahertz or more with 1 gigabyte of main memory with Sun's Java jit or Hotspot Virtual Machine. It is estimated that this configuration should be able to cater for 2 to 3 times as many users with the same or lower response time. However, the researcher is currently the Lecturer in Charge of a unit with over 350 students. It is possible that one third of these could be connected to the system at any one time in some future scenario. The implementation language - Java - may limit the number of connected users, as well as the response time to these users. The architecture of the system could also limit the response time of the system. Evaluating this is seen as future work.

(2) The system can be accessed from the Web but the current Web interface is similar to but not identical to the `MentorClient` interface. This is more a limitation of the installed version of Java that runs in the client browser. This version must be compatible with the developed AppletClient version. Since the user may install any version of Java on their system, this limitation will always remain.

(3) Currently the Domain Knowledge consists of hand-crafted Java Topics as well as Topics automatically created by the Topic Network Builder application. This knowledge has been created by the researcher. Therefore a significant limitation of this system is that any Lecturer who wishes to get similar benefits for their students, must learn how to craft these Topics or learn how to use the Topic Network Builder application. Case Study 3 evaluated the 'ease of programming' the system (see Section 5.3.7 on page 259). These results came from second year computing students, and it can be seen that their level of programming skill is all that is required to build good Domain Knowledge.

However, for most potential non-computing DK builders, this is a severe limitation. Therefore, the Topic Network Builder application is the only possibility. Although this is a typical point-and-click application for building the Topic, it still requires practice to be able to use it well. This is not a limitation - anyone who wants to use a tool, must learn how to use it first. However, one significant limitation in using the application is that DK builders would still have to know how to craft Perl-5 Regular Expressions for the pattern matching, and this requires specialist knowledge. Few non-computing DK builders would want to deal with this way of specifying

questions to be matched against. Future research could add to the Topic Network Builder application so that plain English sentences could be typed in, and the system could transform these into optimised Perl-5 regular Expressions (see Tan (2005a) for some initial work and encouraging results in this area).

(4) Similarly, the user manual for the Topic Network Builder application is very rudimentary since only the researcher has used the application. This limits the usefulness of the application for other DK builders.

(5) The system, including the Educational Support applications, has only been tested with the two paradigms of Authoritative Guidance and Directed Learning Path. Other paradigms may be able to be accommodated, but further development may need to be done for some specific paradigms. The architecture of the system may limit the types of paradigms that can be supported.

(6) The system currently assumes only one query per input: input that contains two or more questions is not catered for. Future research may address this by pre-parsing the input and breaking it into a series of questions.

4.4. Further research

This chapter has highlighted some issues that need to be addressed in the future.

(1) The heuristic that uses the first name of the connected user to determine whether they should be addressed using him/her, he/she is predominantly European biased and hence is often inaccurate for the South East Asian names typical of our user population. A better heuristic is needed for a future global environment.

(2) The area of pro-active Responses caused problems for many users who felt interrupted in their work. A facility for users to tailor / suppress topics that produce pro-active Responses was seen as a suitable solution to this.

A more important issue was the need for some Topics to be able to switch from passive to pro-active and back again based upon the user requesting some information that was not immediately available. Work-around methods already exist within the system to address this issue, but a more holistic approach should be developed for future use.

(3) Better kernel support for the processing of anaphora and ellipsis may improve the system’s ability to understand more user input.

(4) Users wanted an anthropomorphic interface: they engaged with the system through the dialogue and expected the system to have a personality of some sort. The system has hooks to allow for a personality sub-system, and this personality can modify any Response returned from a Topic at various stages of the dialogue management process. For example, one personality may reduce the response weight of a Response that mentions "cats" if the personality did not like cats. Or it could modify the wording used or the emotion associated with that Response. For a Response to be properly processed by a personality, a new class of personality-aware Topics may need to be developed that can cater for the alteration of different aspects of the Response.

(5) As for a human mentor, the system should answer relevantly - that is, a human would assume that a user in the third year of their course does not require a trivial answer to a request but one that perhaps addresses a deeper issue. One solution would require a Topic resolution mechanism that was cognisant of the student's learning level and the relevance of the current Topic given the student's current knowledge. Another solution would be to weight the responses differently dependant upon whether the response came from a unit Topic that the student was currently enrolled for, or from a past unit Topic. A simpler approach would be to enable a student to choose any Topic including those from their past units, via the client preferences.

(6) VHML has general issues that need to be addressed, not just for the marking up of Responses for the ***Mentor*** System. Addressing these issues however, is seen as being outside the domain of ***Mentor*** System research. The ***Mentor*** System can benefit from any advances in the functionality of VHML.

(7) 'Learning hurdles' in the DLP. The issues with these hurdles are two fold. Firstly, informal discussions with students have indicated that if the system asks too many questions as the user travels down the DLP, then it becomes less attractive as a source of information and as a helper. Secondly, users may not be able to answer even simple questions if they have arrived at that point in the DLP by using some supplied information but not other. The DLP hurdle may be pointless, or worse, may not be passable in these cases. The DLP support in the DM kernel could be improved to allow for more complex checks to be made at these hurdle points so as to maximise student "throughput" and learning, whilst minimising the 'unattractiveness' or obstructive nature of the hurdle.

(8) Future research should look at adding extra functionality into the DM kernel and into associated support applications such as the Topic Network Builder so that different educational paradigms can be tested.

4.5. Conclusions

This Chapter detailed the design and implementation decisions taken in developing the ***Mentor*** System. It also detailed the issues that arose from the design and implementation of the architecture of the ***Mentor*** System.

The design and development of the ***Mentor*** System had 3 main objectives:

```
1.  Could the system be developed in Java as a real-world tool
    (not just an academic test-bed)?
2.  Could it be effective in helping students with their units
    (specifically assignments in these studies)?
3.  Would students feel that it was beneficial to use it?
```

This chapter addressed the first question posed above. The system was developed and implemented using Java, and was robust and stable even when more than 60 users were connected. Of importance, a Dialogue Manager Server was implemented, of which the ***Mentor*** System was just one instantiation. In this way, the kernel of the system could be re-used in other Dialogue Management applications.

Similarly, the Knowledge Base was not static information but was designed around active Topics - Java classes that matched the user input and created the Response that was sent back to the user. This enabled the dynamic generation of information in response to a question as well as pro-active questions from the system to the user.

Finally, the KB information was designed to be marked up using a metalanguage - VHML - that allowed for multimodal output to be sent to a number of different renderers in a number of formats: as simple as a plain text console, or as complex as a talking, gesturing Virtual Human.

Chapter Five will supply further supporting evidence of the first aim in terms of extensibility, 'Response Time', and 'Ease of Programming'.

The ***Mentor*** System was designed and implemented to test the sub-hypotheses:

> A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.
>
> The ***Mentor*** System can support an effective learning paradigm.
>
> The ***Mentor*** System will benefit students.

This chapter has tested the first sub-hypothesis and shown it to be valid. Chapter Five again will supply supporting evidence in terms of extensibility, 'Response Time', and 'Ease of Programming'.

The second sub-hypothesis was also confirmed. The ***Mentor*** System and peripheral applications such as the Topic Network Builder supported the two educational paradigms of Directed Learning Support and Authoritative Guidance.

An analysis of the qualitative and quantitative data from the Case Studies of this research is detailed in the next chapter.

This study has revealed nothing if not that the use of information and communication technologies should be in the context of clearly stated educational outcomes accompanied by practical strategies for achieving them.
Glen M. Farrell
The Development of Virtual Education:
A Global Perspective, 1999

As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.
Dave Parnas

Data Analysis

5.1. Introduction

The previous chapter described the implementation issues of the ***Mentor*** System. This chapter details the evaluation stages using student feedback through questionnaire surveys.

The design and development of the ***Mentor*** System had 3 main objectives:

1. Could the system be developed in Java as a real-world tool (not just an academic test-bed)?
2. Could it be effective in helping students with their units (specifically assignments in these studies)?
3. Would students feel that it was beneficial to use it?

The system was evaluated over four consecutive University semesters with predominantly distinct population of users: second and third year university students enrolled in the Computing discipline. The users (approximately 20% females) were aged 19-21 years on average, and were regular students in terms of their academic abilities. Of relevance to these studies is that a significant number of users did not have 'English as a first language' - 57%, 33% and 47% for the last three formal studies.

These last formal case studies are referred to as SPD251.012, AGV351.021 and SPD251.022 respectively, representing the unit, year and semester in which the studies were implemented.

The students were asked to try the system, told that it may help them with their assignments and then encouraged to use it for the semester. The trialing of the system was therefore voluntary and up to the individual choice of each student. Students formally evaluated the system in the last three case studies at the end of each semester using a 6-page questionnaire (Appendices A, B and C). Student metric data - their textual inputs and their responses to the output - were also recorded in an easily parseable format (Appendix G.12 on page 377).

While it is agreed that it is not always appropriate to rely on student evaluation of a system tested in their own course, students were given the choice of using the system or not, and the researcher was interested in whether the students felt that the system was of benefit to them in their studies.

It should be noted that no control group was possible for these studies. Also, due to confidentiality requirements, no tracking of assessment results was allowed that might differentiate those who used the system from those who did not.

Finally, it should be pointed out that the study was not trying to determine what educational paradigm would emerge from the system but whether imposing a proven teaching method template onto a software system was possible, and whether it was effective and beneficial to the students using it.

5.2. Participant Demographics

5.2.1. Gender

In the three case studies, the gender distribution of the ***Mentor*** System users was typical of the enrolment patterns in Computing related degrees. See Table 5.1 and Table 5.2

Year	Electrical & Computer Eng.	Computer Science
2000	7.5	17.1
2001	7.6	16.5
2002	9.2	15.4

Table 5.1 : University percentage distribution of females enrolled in Computing related degrees

Study	% females enrolled
SPD251.012	18
AGV351.021	22
SPD251.022	21

Table 5.2 : Percentage of females enrolled in each case study

5.2.2. Age

In the three case studies, the age distribution of the ***Mentor*** System users was typical for student enrolments in second and third year units in Computer Science.

5.2.3. Course of Study

In the three case studies, the course of study distribution of the ***Mentor*** System users was as expected for these Computing units. Students have a strong computing background and most have had three semesters of Computing for the SPD studies (second year, second semester unit) and four semesters for the AGV study (first semester, third year).

5.2.4. Average Grade of the Participants

In the three case studies, the average grade distribution of the ***Mentor*** System users enrolled in the three case studies (Figure 5.1) is similar, with the slightly lower grades for those enrolled in the third study possibly being due to the increased number of repeat students.

5.2.5. Previously Enrolled in case study Unit

Across all three case studies, the ‘previously enrolled in the unit’ distribution was similar. Although the third study had significantly more repeat students, the majority of these had not used ***Mentor*** before.

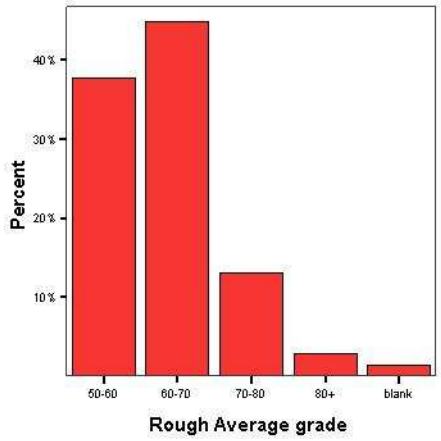


Figure 5.1a : SPD251.012

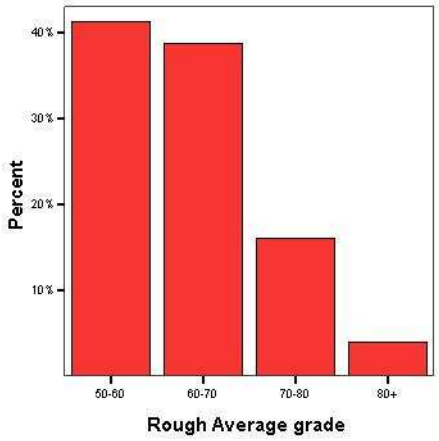


Figure 5.1c : SPD251.022

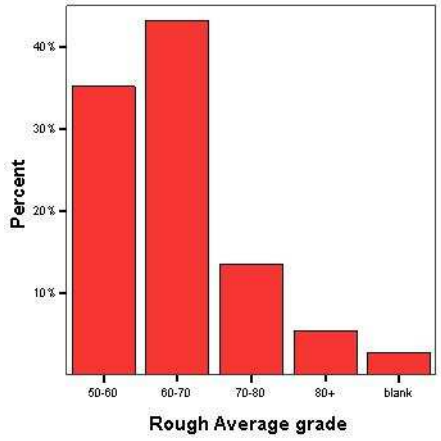


Figure 5.1b : AGV351.021

Figure 5.1 a,b and c: Response distribution for the three case studies regarding *'Average Grade of the Participants'*

5.2.6. Previously used the *Mentor* System

Figure 5.2 shows the distribution of users who had previously used ***Mentor*** during the case studies. The distribution of results in the first study (Figure 5.2a) is not possible - the users who indicated that they had previously used the system could not have done so. It is assumed that users were confused by the question as a different lecturer-student mentoring system was also in place within the Department of Computing. In the following case studies, subsequent questionnaires, and the accompanying verbal explanations with the questionnaires helped remove this ambiguity.

Although it was not possible due to privacy and confidentiality concerns, it would have been interesting to explore how the users who had previously used the system re-evaluated it given the changes that were made due to the formative and summative feedback from the successive case studies.

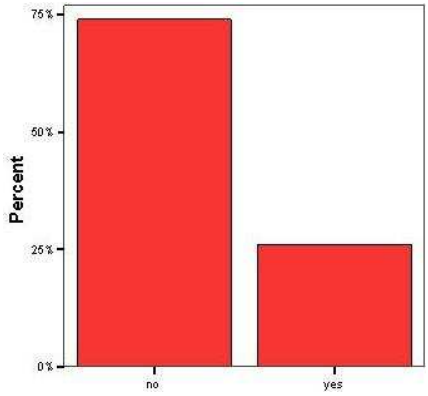


Figure 5.2a : SPD251.012

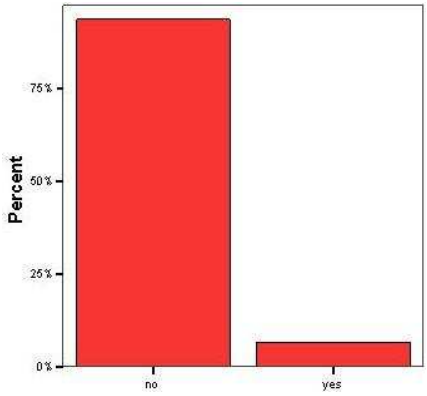


Figure 5.2c : SPD251.022

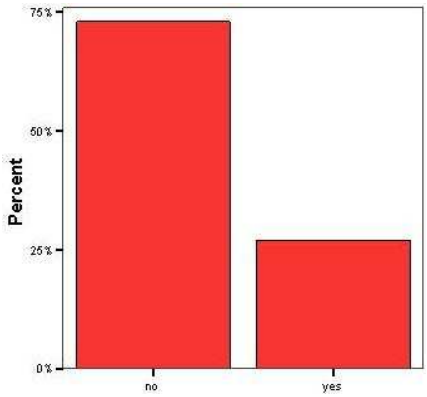


Figure 5.2b : AGV351.021

Figure 5.2a is not possible since the system could not have been used by these students.
Figure 5.2b shows that some of the users from the first study were also part of the second study.
Figure 5.2c shows that some of the users from the second study were also part of the third study.

Figure 5.2 a,b and c: Response distribution for the three case studies to the question *'Have you previously used the **Mentor** System?'*

5.2.7. English as a first language

In the three case studies, the 'English as a first language' distribution of the users was typical of the enrolment patterns in the Computing discipline (Figure 5.3 and Table 5.3). Many users did not have English as their first language (57%, 33% and 47% respectively) and since the system tries to match responses against correct English, if users do not use correct English, this poses a serious problem that even the generality of Regular Expression (RE) pattern matching often cannot overcome. The system can cope with standard abbreviations, such as the first entry in Quote 5.1, but has trouble understanding the intent of the second due to the poor English.

how r u
yap! waht to do u idoit?

Quote 5.1 : Examples of incorrect English input to the **Mentor** System

The biggest problem appears to occur due to the lack of distinction between plural and singular in many Asian languages and also the missing articles such as 'a' or 'the'. The latter often disambiguates a question/request and the former is often just not catered for in the RE. Making the RE pattern too general means it matches wrong questions. However, making it too specific means that it will not match at all.

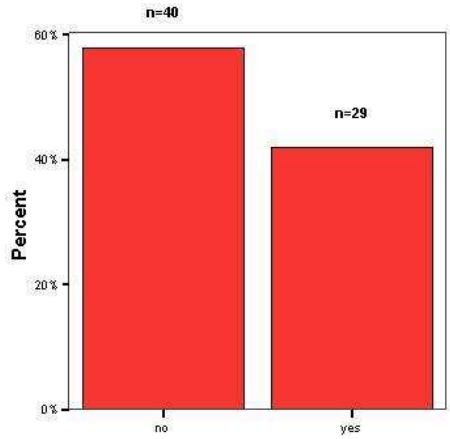


Figure 5.3a : SPD251.012

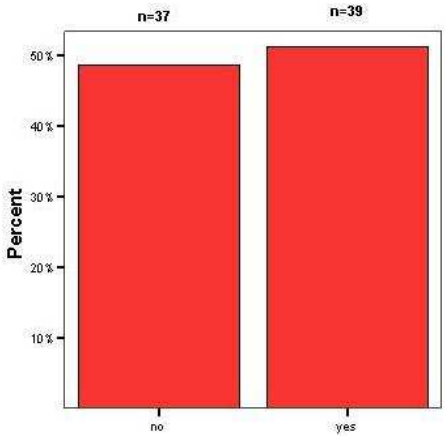


Figure 5.3c : SPD251.022

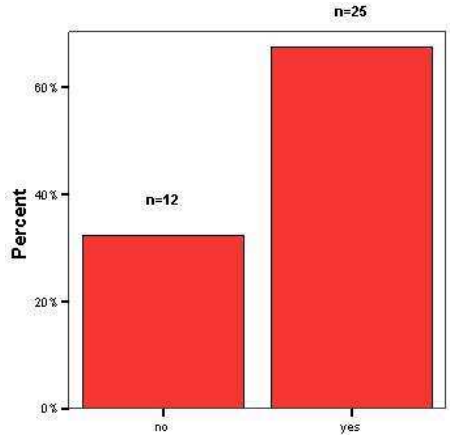


Figure 5.3b : AGV351.021

Study	Non-English as 1st language	
	Number	Percentage
SPD251.012	50	57
AGV351.021	12	33
SPD251.022	37	47

Table 5.3 : Number and Percentage of users who did not have English as their first language in each case study

Figure 5.3 a,b and c: Response distribution for the three case studies to the question *'Is English your first language?'*

5.2.8. Student Identity mark

For the purpose of contacting the students for possible follow-up interviews, the following request was included in the questionnaires for the second and third formal studies:

```
For some participants, I may ask if I can do a follow-up interview. This would be
to get more information or to clarify your responses.  This would be purely voluntary
on your part. If you are happy to be re-interviewed, then put some unique identifying
mark or picture in the box on the next page - something that you could recognise but
that does not identify you to me. In this way I can send email to all SPD students
saying I would like to re-interview the students who had these marks (and then show
the mark as an image). This would only be for a small number of participants and even
at that stage, you could decline to be re-interviewed.
Choose a mark that you can remember and that is unique.

If you don't want to be considered for another interview, leave it
blank. If you want to be considered, but want to remain anonymous,
put a mark. If you want to be considered and don't care if I know
who you are, simply put your name or your mark.
For example, I might use:
```

The second study was implemented with students who enrolled in a third year, first semester unit. In this study, only 15/37 (40%) of the users did not wish to be identified (Figure 5.4a). The final study used students enrolled in a second year, second semester unit. This study had 48/77 (62%) who did not wish to be identified (Figure 5.4b). This could indicate that third year first semester students were less concerned about anonymity in the second study than the second year second semester students in the third study.

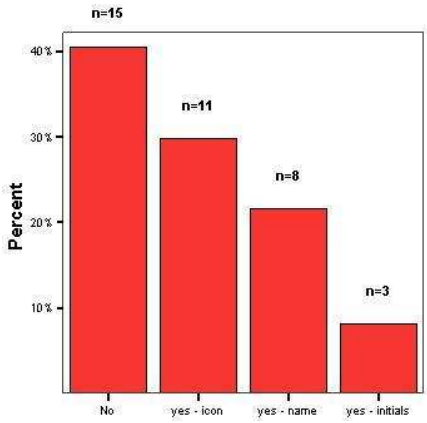


Figure 5.4a : AGV351.021

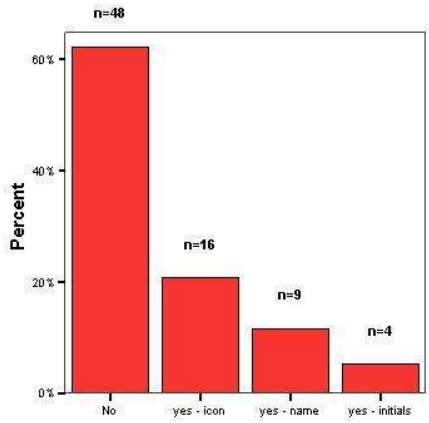


Figure 5.4b : SPD251.022

Figure 5.4 a and b: Response distribution for the last 2 case studies to the request for a *'student identity mark'*

5.2.9. Conclusion

In most of the recorded responses, across all of the case studies, the user distribution did not differ from the norm of student enrolments in any significant way. The small percentage of females in the study however does pose a problem for any attempts to generalise the results to other disciplines.

The most significant problem that emerged given the enrolment patterns, is the large number of non-English-as-a-first-language users. This is also apparent from their input to the ***Mentor*** System as well as in their comments in the three formal evaluations. The Regular Expressions used in the ***Mentor*** System therefore have been progressively modified over the three studies to create a good balance between recognising 'poor English' and not over-generalising the pattern matching (see discussion in Section 5.3.5.1 on page 242).

5.3. The Case Studies' Relationship to the Hypotheses

5.3.1. Introduction

The case studies were designed to test the sub-hypotheses:

A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.

The ***Mentor*** System can support an effective learning paradigm.

The ***Mentor*** System will benefit students.

Various topics and domain knowledge relevant to the specific Computing units were developed for each case study. The topics contained responses to typical assignment related questions, hints, statically and dynamically data-mined information, references to online material as well as pro-active questions and DLPs for tackling the assignments.

An initial informal pilot study was carried out in Semester one, 2001 to test the initial design, development and configuration parameters. The results from this pilot study, and from subsequent studies were used to confirm the first sub-hypothesis.

The first formal study was seen as a way of obtaining formative and summative evaluation information on the system so that the issues raised could be addressed in the second formal study. By providing a suitable framework for the user interface and management, and for the Dialogue Management and Knowledge base, it was hoped that the system design and implementation could be improved for the later studies.

The second study concentrated on creating substantial domain knowledge to support the user, together with complex REs to enable the users to ask about this knowledge. Many sources of knowledge were data-mined to provide actual and online information via hot links in the responses to the user.

The third study concentrated more on developing a complex Directed Learning Path to guide users through the development of their assignment. It would also evaluate the extensibility, responsiveness and usability of the designed system. All studies used the Authoritative Guidance paradigm to cater for complex queries that required specific answers.

The Directed Learning Path is essentially a step-by-step guide with checks. Users may enter this path either by asking, or the system may pro-actively ask if they want help. Traversing the path may be as simple as answering "yes" or "no", or as complex as the system asking the user at some stage questions like: "what problems did you find when ....." and then choosing a different path according to their answer. The path also ‘checks’ to ensure that the user really has performed the necessary preceding steps. This is not always successful but users came to realise its importance.

The Path is built using the domain knowledge and educational expertise of the lecturer and the various forms of assessment direct its structure. The system guides the users but more importantly demonstrates and reinforces a successful problem management strategy.

The Authoritative Guidance is based on the principle that users may want to learn in their own way but ultimately require expertise in that area. The ***Mentor*** System attempts to immediately answer most user questions and those that it cannot are forwarded to the Lecturer-in-charge of the unit, who can create a relevant answer and have that returned to the user. The next time the user logs onto the system, or when they have a free moment, it asks if they would like an authoritative answer to their question.

5.3.2. Case Study 0: AGV351, Sem 1, 2001

An informal pilot study was implemented in semester 1 of 2001 with a third year group of users. The purpose of this study was to identify and solve operational and logistic problems with the system. Due to the system’s multi-user, multi-threaded, client-server

architecture, exhaustive testing of the system in isolation only by one user - the researcher - was not indicative of its normal usage.

Informal feedback from the users was used as formative evaluation to improve the system operation and functionality. This testing was mainly concerned with discovering and fixing user interface problems rather than Knowledge base or Dialogue Manager problems. However, several user expectations were uncovered that were not catered for in the initial Dialogue Manager design. All of these problems were addressed in time for the first formal evaluation of the system in the following semester.

5.3.3. Case Study 1: SPD251, Sem 2, 2001

In each study, users were asked via a 5 point Likert scale: *'Do you think that it was beneficial or helpful to use the* ***Mentor*** System*?'* (1=>not at all, 5=>very beneficial). Assuming a value of 3 or above is a positive indication of benefit, Figure 5.5 and Table 5.4 show that users did not feel that they benefited from the ***Mentor*** System, especially given the standard deviation (σ) value. σ normally represents the population standard deviation whereas s represents the sample standard deviation. In this thesis however, σ is used to represent the sample standard deviation.

However, for a first study, the results were viewed as positive in that a purely software based mentoring system had scored a Mean of 2.58 with a Median of 3, in helping users. The cumulative frequency values in Table 5.5 were also positive. Overall, the first case study indicated that the system may become more effective by incorporating the improvements based upon the feedback from other questions.

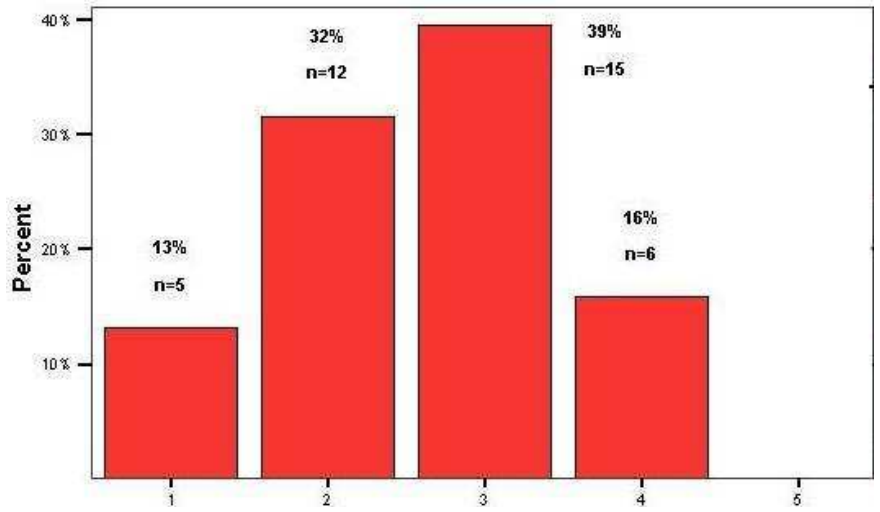


Figure 5.5 : SPD251.012: Response distribution on a 5 point Likert Scale to the question: *'Do you think that it was beneficial or helpful to use the* ***Mentor*** System*?'*

# users	Mean	Median	σ
38	2.58	3.000	0.92

Table 5.4 : SPD251.012: Response statistics to the question: *'Do you think that it was beneficial or helpful to use the **Mentor** System?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	87	55	16	0

Table 5.5 : SPD251.012: Relative cumulative frequency statistics to the question: *'Do you think that it was beneficial or helpful to use the **Mentor** System?'*

Given the hypotheses, it was necessary to understand why the users had or had not chosen to use the system, what they used it for, what its problems or deficiencies were and what its strengths were, as well. Of importance, was whether the system gave the users enough information in answering their queries; whether the Directed Learning Path was seen as useful; and what aspects of their studies were helped most by the system. Other questions (see Appendix A) recorded the users' responses to these issues.

5.3.3.1. SPD251.012: User issues

The total number of users enrolled in the unit was 96, of which 69 answered the questionnaire administered at the end of the final lecture. Of these, 65% used the **Mentor** System (see Figure 5.6).

The quantitative comments on why the system was not used were categorised (see Figure 5.7). Unfortunately only 23 responses were made and of these 5 (22%) indicated that the system '*was useless*'.

Figure 5.8 shows the categorised reasons why the system was used and as expected, the majority of usage was for assignment help. It was encouraging that users had also used it for fun.

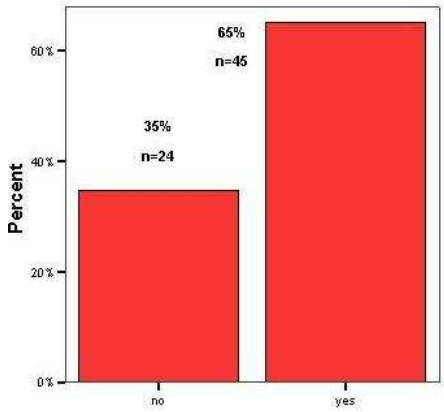


Figure 5.6 : SPD251.012: Response distribution to the question: *'Did you use the **Mentor** System in semester 2, 2001?'*

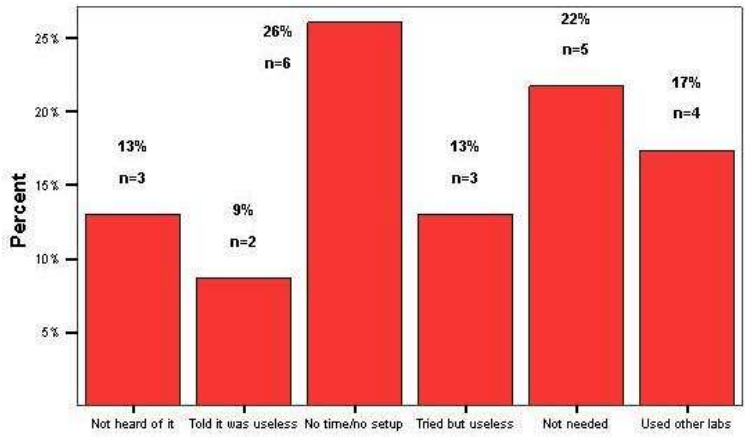


Figure 5.7 : SPD251.012: Recoded response distribution for reasons why the ***Mentor*** System was not used

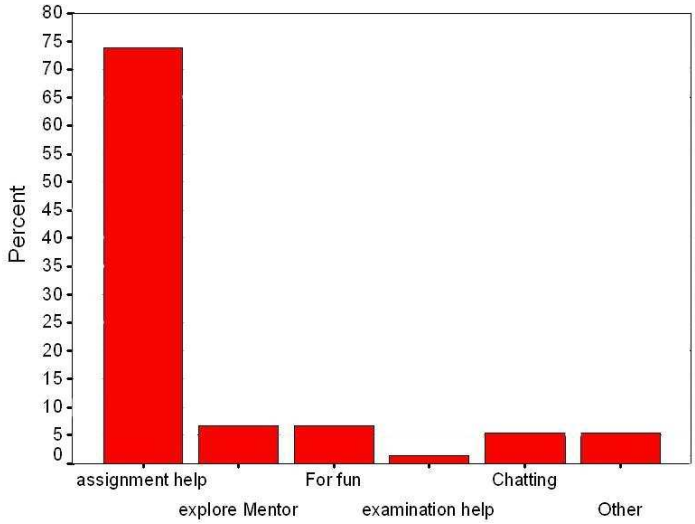


Figure 5.8 : SPD251.012: Recoded response distribution for reasons why the ***Mentor*** System was used

Consistent with responses about the benefit of the system, users also felt that it was not effective or useful for help on their assignments (see Figure 5.9, Table 5.6 and Table 5.7). Subsequent questionnaire responses were used to determine how and where it had failed the users.

Figure 5.10 indicates that the system was not beneficial because it simply did not understand the question and/or when it did, it did not provide enough or correct information. It is important to note that those who did not use the system, or who did not participate in the evaluation, may have decided not to, simply because of word of mouth that it was not effective initially.

As planned, the ‘open comment’ part of all the questions in the evaluation was a useful source of formative feedback for improvement. It can be seen from the subsequent studies, that as users see the benefit in the system, they make more, and more useful comments.

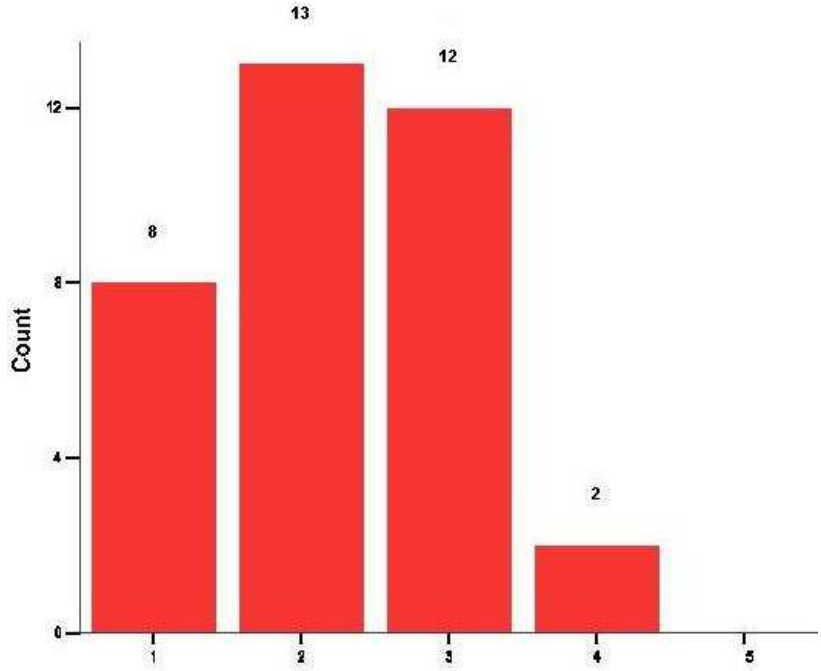


Figure 5.9 : SPD251.012: Response distribution on a 5 point Likert Scale to the question: *'How effective or useful was it for this purpose?'*

# users	Mean	Median	σ
35	2.229	2.000	0.877

Table 5.6 : SPD251.012: Response statistics to the question: *'How effective or useful was it for this purpose?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	77	40	6	0

Table 5.7 : SPD251.012: Relative cumulative frequency statistics to the question: *'How effective or useful was it for this purpose?'*

Quote 5.2 is accurate in that the Lecture-in-Charge[12] will give users information that makes them think or look for the answer rather than 'just the answer'. However, users' perceptions of the system was that it should be more direct in its answers. This observation has been repeated in each case study.

Figure 5.11 also indicates the same problem of lack of understanding and/or information. Students also found it annoying that the system could not be killed or disabled even though it iconified itself. This posed a difficult problem since the system was designed to be ever-present so that it could offer pro-active advice.

The last part of Quote 5.3 was seen as damning evidence that the system must improve if it was to become effective (for typical feelings about the HCI usefulness of the paper clip, see "Why the paper clip sucks" [HREF 62]).

[12] The Lecturer-in-Charge was the researcher.

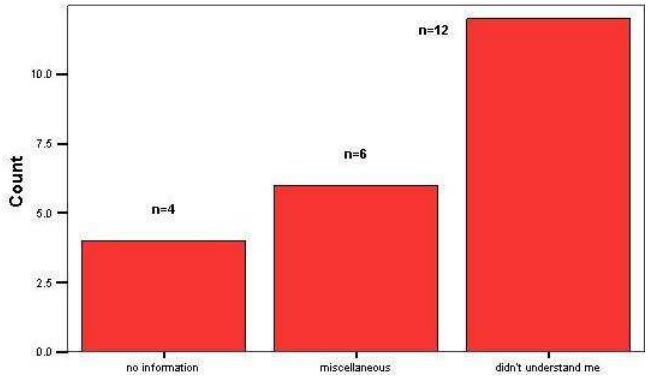


Figure 5.10 : SPD251.012: Recoded response distribution of comments on *'the least beneficial aspect'* of the ***Mentor*** System

lack of specific information/detailed information. mimiced talking to andrew[13]

Quote 5.2 : SPD251.012: *'least beneficial aspect'* comment

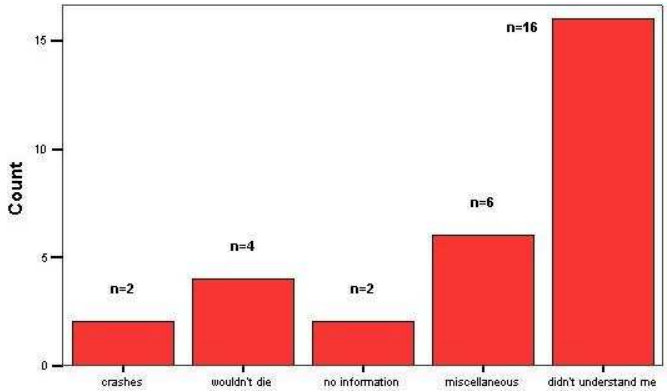


Figure 5.11 : SPD251.012: Recoded response distribution of comments on *'the most annoying aspect'* of the ***Mentor*** System

it is not (yet) user friendly, if you implement this the way iCQ works (with chatting mode) then this gonna be very useful. How about a user want to cancel/undo
you can't kill it!
sometimes, it ask silly question. Remind me of microsoft word's "paper clip"

Quote 5.3 : SPD251.012: *'most Annoying aspect'* comments

[13] 'andrew' was the Lecturer in Charge of the unit and is the researcher.

When asked about the most obtrusive aspects, users also commented on the immortality of the system as well as the fact that the pro-active prompts were annoying (Quote 5.4). Since the system was designed to be unobtrusive, this problem needed fixing.

it popped up once in awhile in the middle of something
popping out when not required

Quote 5.4 : SPD251.012: *'what aspects were obtrusive'* comments

Figure 5.12 summarises the overall problems with the first case study: the Dialogue Manager and interface needed improvement. It was comforting that a small number of (possibly naïve) users thought that there was nothing wrong with it.

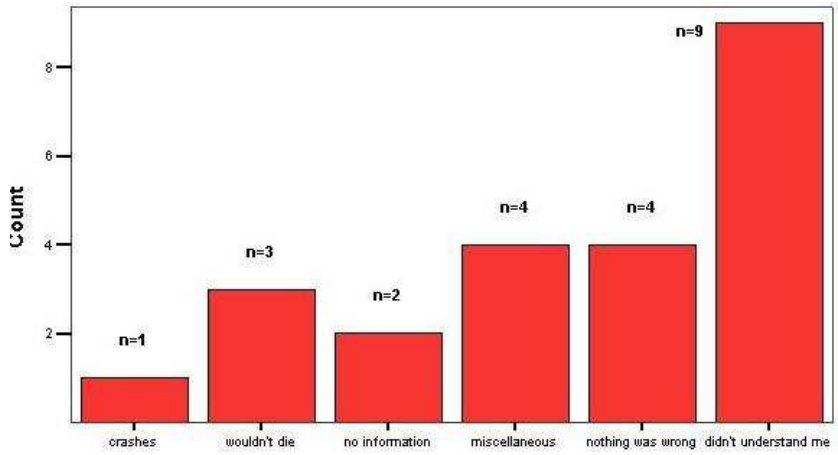


Figure 5.12 : SPD251.012: Recoded response distribution of comments on *'what was wrong'* with the ***Mentor*** System

Hard to get what you want out of it.
I would like something that could give simple answers not point to places where i could find 1000 answers to questions i dont have.
i would like to be able to ask a human beings a question through it if possible

Quote 5.5 : SPD251.012: *'what was wrong with it'* comment

Quote 5.5 encapsulates users' feelings about the system and about learning in general and these needed to be addressed for the subsequent studies.

The user metric data - their inputs and responses to the system outputs - were analysed to see whether the system was failing to pick up legitimate input, whether it was not accurate enough, or whether the users' input was simply too difficult to determine and hence the system would never be effective.

It was seen from typical user input that in certain cases, users simply exceeded the Knowledge base boundaries - they explored the system because of its newness (Quote 5.6). One solution to this was just to increase the Knowledge base. This was possible but its open endedness was not deemed to be within the scope of the project. The metric data was analysed to see where the knowledge base could be increased in a bounded manner.

what is rush hour 2
what is the meaning of life?
can u help me find good wallpaper ?
apa kabar ?[14]

Quote 5.6 : SPD251.012: example user input data

There were also long questions (Quote 5.7) - not an English problem but unreasonable to expect the system to determine intent and hence understanding:

Besides using a CGI webpage to POST data to a CGI-perl script, is it possible for me to write a script, run it on the command line, and it'll post the data to the CGI-perl script?

Quote 5.7 : SPD251.012: 'long question' user input data

Quote 5.7 has a Flesch Reading Ease (FRE)[15] of 62 and a Flesch-Kincaid Grade Level (FKGL)[16] of 12.0. The inputs of Quote 5.8 are also difficult for an NLP (FRE=45, FKGL=8.1 and FRE=61, FKGL=8.1 respectively):

in part2 the user must add a "job completion" comment. Another "job completion status" comment is optional. If no "job completion status" comment, the "job completion" comment is used for "job completion status" comment. Is it right?

ok. In the assignment specification, you wrote that "user must add a comment about job completion..., add a comment about the job completion that will be logged with the job completion status...". So there are two comments added by a user.

Quote 5.8 : SPD251.012: 'very long question' user input data

The user metrics also showed that a lack of correct English caused many problems. The system tries to match against correct English. If users do not use this, then it poses a serious problem that even the generality of RE pattern matching often cannot overcome. There were also simply some bad english questions (Quote 5.9).

hey, who a u?	hrmz ure silli
now at urself	u just get fixed, don't u ?

Quote 5.9 : SPD251.012: example user input data

To try to overcome this problem for the next study, some system-wide RE's were predefined to help match commonly used, badly worded or misspelt user queries. For example, WHAT_STRING was predefined to be "what|wot|wat|what". This, and other

[14] 'apa kabar?' is indonesian for 'how are you'.

[15] Rates text on a 100-point scale; the higher the score, the easier it is to understand the document. For most standard documents, aim for a score of approximately 60 to 70.

[16] Rates text on a U.S. grade-school level. For example, a score of 8.0 means that an eighth grader can understand the document. For most standard documents, aim for a score of approximately 7.0 to 8.0.

extensions, allowed a match for example, of any of the user queries:

```
wot is OpenGL
wat is Open GL
waht is Open-GL
```

The second point raised in Quote 5.5 (page 203) is one that is the subject of much educational debate and its correctness is not the domain of this thesis. As can be seen from Quote 5.2 (page 202), the system implemented a learning strategy consistent with the researcher's twenty years of tertiary level teaching. What is relevant is whether this approach to learning used was seen as effective in the context of the ***Mentor*** System.

Any dialogue between a student and the lecturer is modified by the lecturer's experience in recognising whether or not a student is grasping the explained concept. This recognition is based upon prior knowledge of the student, the history of interactions with the student, the way in which the student phrases the question, and even the student body language. The lecturer can also get subtle feedback by noting the timing of the student response. Some of these avenues are not open to a software based dialogue system.

Therefore, erring on the side of 'more information' was seen as a possible counter to this lack of multi-modal input. Informal feedback was obtained from several users across different academic grade levels to see what balance was needed between encouraging users to think rather than just being given the answer, and making the system attractive and beneficial to all users. This 'simple answer' expectation of responses from the ***Mentor*** System exists in all three case studies.

Finally, the third point raised in Quote 5.5 (page 203) - that of direct questioning of the lecturer or other humans - was seen as support for the Authoritative Guidance paradigm that was implemented half way through this study. From this and other user responses, it became obvious that the 'delayed answer' to their requests (see Section 5.3.3.5 on page 210) was not seen as coming from the Lecturer-in-Charge, so this was reinforced through a change in the delayed response wording for subsequent studies.

5.3.3.2. SPD251.012: Beneficial Aspects of the system

As can be seen in Figure 5.13 and Quote 5.10 users did not totally condemn the system. The availability of the system responses, and also the guidance provided were seen as beneficial and this feedback drove subsequent improvement of the system.

The initial text-based interface was not capable of using hot-links for embedded URLs. This meant that users could get the address of a website but would have to cut and paste this into a browser. Subsequent interface development allowed for hyper- and multi-media displaying of information in the responses (Figure 5.31, Section 5.3.4.1 on page 219). This allowed for images, as well as for URLs to be 'hot' - clicking on them opened the page in a new or an existing browser.

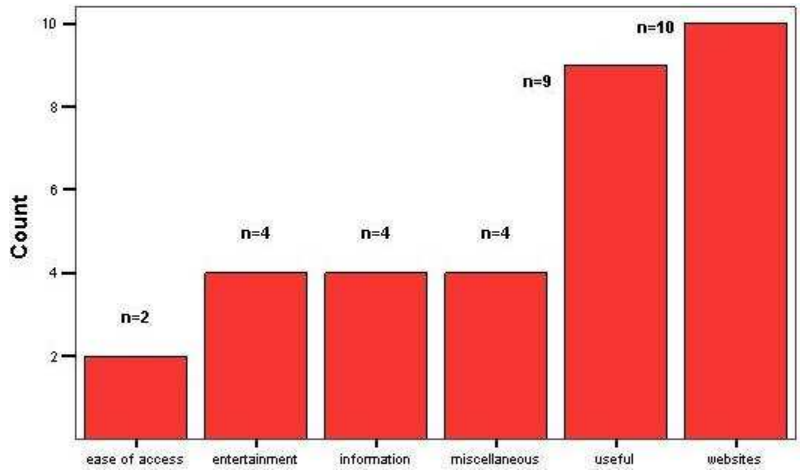


Figure 5.13 : SPD251.012: Recoded response distribution of comments on what were the *'most beneficial aspects'* of using the ***Mentor*** System

helping with simple questions rather than emailing the lecturer. Good for breaking up long coding sessions
low level guidance when got completely stuck
when it told me step by step which places on the net I needed to visit in order to complete the assignment.
to help students in improving the chance to get good marks in assignment.
Available all the time even if it told me little or nothing I didn't already know
weather updates were good, was funny, some useful links for assignments etc.

Quote 5.10 : SPD251.012: *'most Beneficial aspect'* comments

5.3.3.3. SPD251.012: 'How did it help you' Evaluation

Users were asked via a 5 point Likert scale (1=>not at all, 5=> a lot):
Do you think that

- *help from the **Mentor** System improved your time management for the assignment?*
- *help from the **Mentor** System improved the design development of your assignment?*
- *the **Mentor** System helped in the solving of code or design problems for your assignment?*
- *help from the **Mentor** System improved the presentation quality of your assignment?*
- *help from the **Mentor** System improved the overall quality of your assignment?*

The first four questions were expected to be ranked with decreasing order of improvement, with the fourth one expected to have little or no effect on improvement. This was asked to try to determine if the respondents were just 'circling values' rather than giving their considered opinions.

Figure 5.14 and Table 5.8 support this expected trend and certainly indicate that the users understood that the system was not aimed at improving their 'presentation quality'. Note the significantly smaller standard deviation for the fourth response - there is very little

spread in the respondent's values. Therefore, the evaluation questionnaire appears to be validated since it was producing the expected *information* even if the response values were not supporting the hypotheses.

The overall reaction (Figure 5.15, Table 5.9) is consistent with that for Figure 5.5 and Table 5.4 (Section 5.3.3 on page 199). The system was not helpful in improving their assignment for these question categories.

Figure 5.16 shows the same data but with missing values included - a significant number of users simply did not answer these questions. This may be attributed simply to the fact that the system was seen as so unhelpful that it was not worth commenting on.

Of importance to the study though was that it was necessary to stress to future evaluation participants that they should fill in all questions so that a better understanding of their attitudes could be obtained.

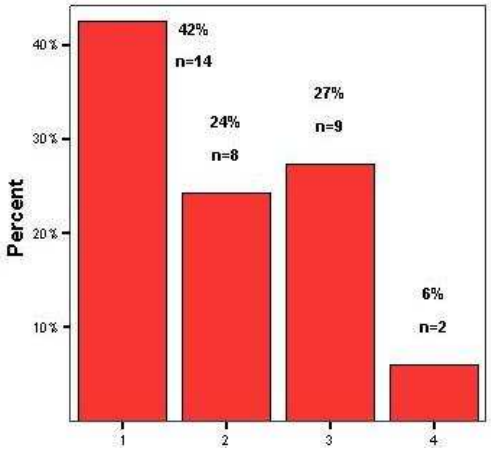


Figure 5.14a : Time

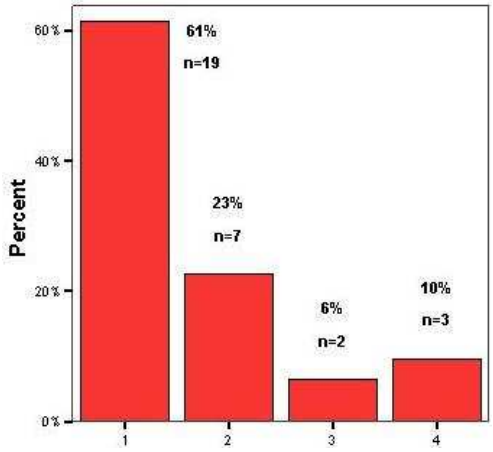


Figure 5.14c : Problems

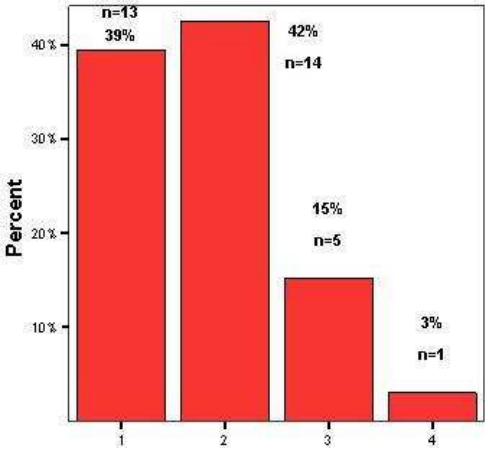


Figure 5.14b : Design

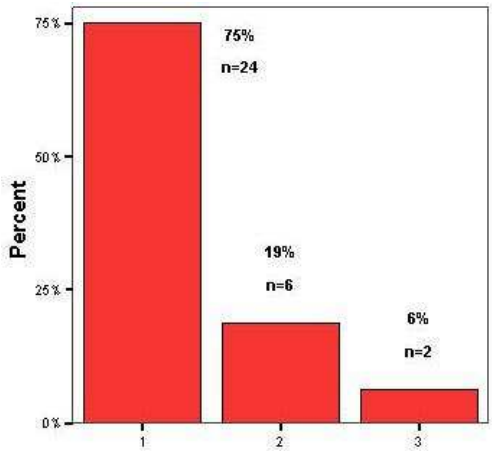


Figure 5.14d : Presentation

Figure 5.14 a,b,c and d: SPD251.012: Response distribution for the categories of *'perceived improvement'*

Figure	Mean	Median	σ
a	1.97	2.00	0.98
b	1.82	2.00	0.81
c	1.65	1.00	0.98
d	1.31	1.00	0.59

Table 5.8 : SPD251.012:Response statistics for each of the four categories of Figures 5.14 a,b,c and d for the '*How did the **Mentor** System improve ...*' Evaluation

# users	Mean	Median	σ
30	1.93	2.00	1.05

Table 5.9 : SPD251.012: Response statistics to the question: *'Do you think that help from the **Mentor** System improved the overall quality of your assignment?'*

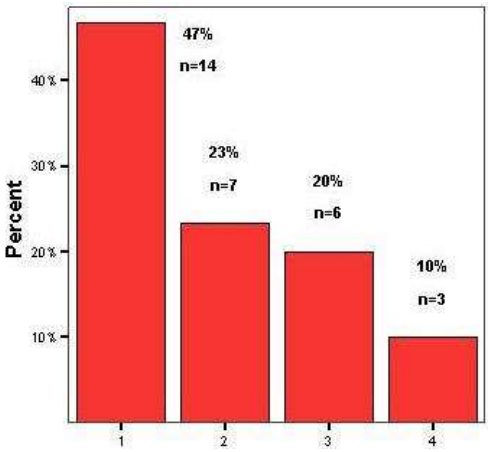


Figure 5.15 : SPD251.012:Response distribution to the question: *'Do you think that help from the **Mentor** System improved the overall quality of your assignment?'*

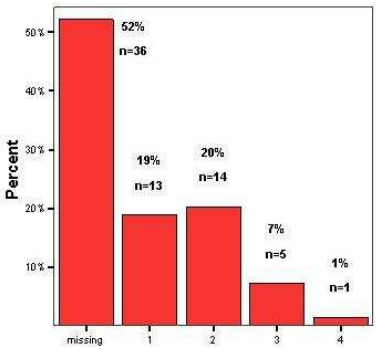


Figure 5.16a : SPD251.012

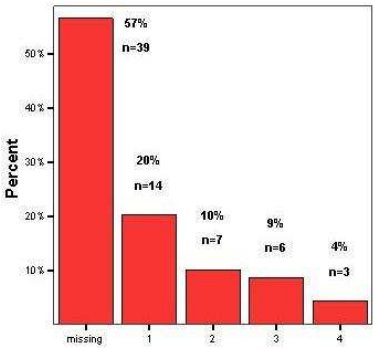


Figure 5.16c : SPD251.012

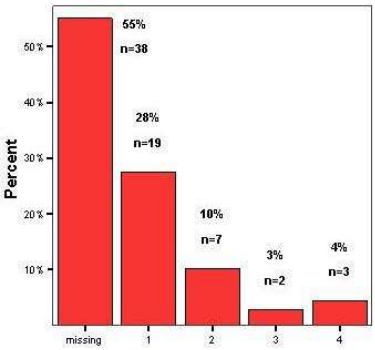


Figure 5.16b : SPD251.012

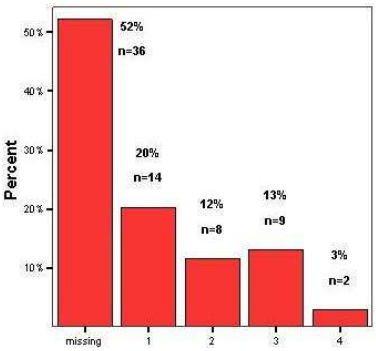


Figure 5.16d : SPD251.012

Figure 5.16 a,b,c and d: SPD251.021: Response distribution for the categories of '*perceived improvement*' with **missing** values included

5.3.3.4. SPD251.012: Directed Learning Path Evaluation

Over 60% of the respondents did not find the prompts 'obtrusive'[17] (Figure 5.17).

[17] Perhaps 'intrusive' would have been a better word to use in the questionnaire. Both words caused problems for non-English speakers. Later evaluations verbally indicated to users what was meant here.

However, in this and later studies, users found that the *way* in which the system indicated that it had information was very annoying. From this, it was assumed that the annoyance of the popping up of the interface was balanced by the usefulness of the information. Since the system was planned to integrate into the user's working environment, it was important to reduce the annoyance whilst improving the usefulness.

Comparing Table 5.10 with Table 5.4 (page 199), it can be seen that users rated the DLP lower than the overall benefit, with the slightly higher standard deviation indicating that there was a diverse attitude to its usefulness. Note that two users did find it helped 'a lot' (Figure 5.18). The user metrics were analysed to see if they could indicate why the divergence existed. Some problems with the pro-active prompts sub-system were uncovered and fixed, and it was seen that this may have affected certain users.

This analysis, along with informal feedback, was also used to determine better ways to help users via the prompting and directing mechanism. Some users felt that the system 'nagged' them whereas others found its 'concern' a positive feature. Some users found that being prompted just after connecting to the system was annoying whereas others preferred this. The timing of the prompts became a crucial issue in trying to balance these two opposing views.

Therefore, the prompting sub-system was modified so that users who reacted negatively when asked if they wanted help were prompted less and less frequently dependent upon each prompt response. If they later asked for help, then the prompt timing was suitably reduced. Those who wanted the prompts were notified more frequently. This 'learnt' timing matched the user's requirements.

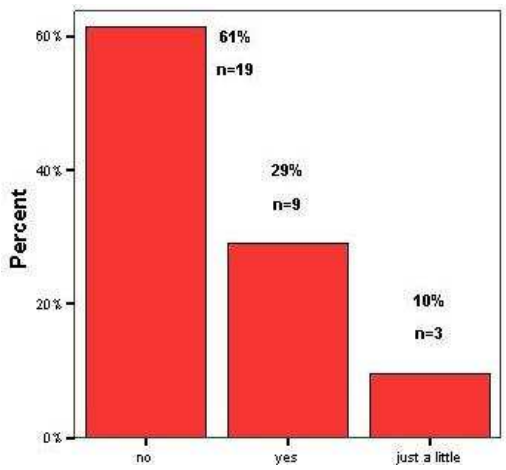


Figure 5.17 : SPD251.012:Response distribution to the question:*'Did you find the prompts obtrusive?'*

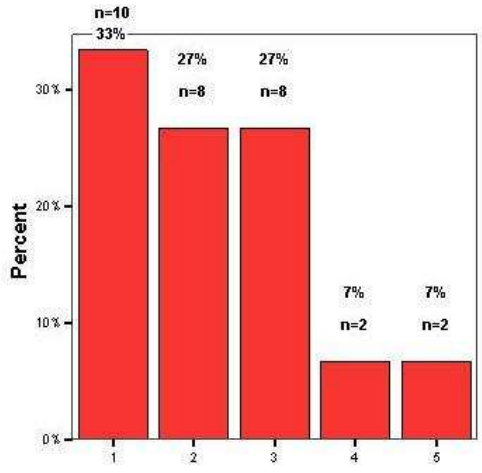


Figure 5.18 : SPD251.012:Response distribution to the question:*'Did you find the prompts helped with the assignment?'*

# users	Mean	Median	σ
30	2.27	2.00	1.2

Table 5.10 : SPD251.012: Response statistics to the question: *'Did you find the prompts helped with the assignment?'*

5.3.3.5. SPD251.012: Authoritative Guidance Evaluation

If the system could not answer a user's request, it was forwarded to the Lecturer-in-Charge (LiC) of the unit who could then provide an authoritative answer to relevant questions. The user would be notified when they next connected to the system or via a pro-active prompt when there was a lull in their dialogue with the system. This facility was developed and introduced half way through this study as a planned addition to the information-providing mechanism.

Initially this feedback was done manually by inspecting 'unanswered questions' files, but the turn-around time was seen as being too slow. Therefore the system was upgraded to include automatic periodic notification to the LiC of questions, the popping up of a GUI to manage the answering logistics and the facility of saving and re-using answers to commonly asked questions. The turn-around time was reduced to less than a minute (Figure 5.19).

This evaluation of the helpfulness of the system highlighted the known problem of linearity of the discrete Likert scale and how users might rank answers to questions: was the ranking value '3' really an attitude half-way between 'not at all' and 'a lot'? Was a value greater than '3' a positive indicator of benefit? Note that users ranked the helpfulness of answers from the LiC (the researcher) as being only '2' (Figure 5.20, Table 5.11).

It was felt that subsequent studies should explore this ranking further to try to establish a base line for the overall ranking of the system. As the turn-around time dropped due to the automation, the sole factor in evaluating this aspect may simply be the relevance of the information returned from the authoritative source.

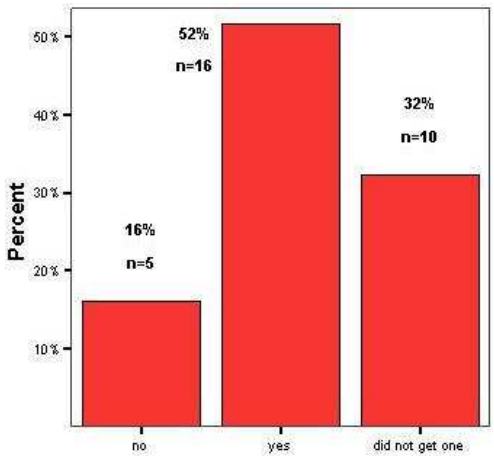


Figure 5.19 : SPD251.012: Response distribution to the question: *'Did you find these answers timely?'*

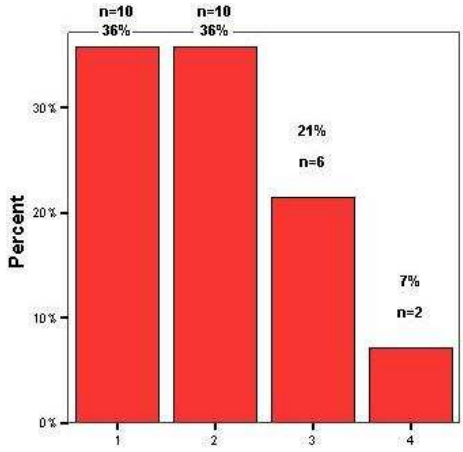


Figure 5.20 : SPD251.012: Response distribution to the question: *'Did you find these answers helped you with the assignment?'*

# users	Mean	Median	σ
28	2.00	2.00	.94

Table 5.11 : SPD251.012: Response statistics to the question: *'Did you find these answers helped you with the assignment?'*

5.3.3.6. SPD251.012: Information Retrieval Evaluation

Users were asked via a 5 point Likert scale (1=>not at all, 5=> a lot):
Was the ***Mentor*** System useful

- *in getting access to relevant information or knowledge?*
- *in your subsequent enquiries for more information or knowledge?*
- *in your subsequent enquiries for deeper information or knowledge?*

These questions were designed to determine if users were getting access to relevant information through the DM as the problem of retrieval versus relevance is always important in information accessing. The user metrics were also analysed to see how the information was being asked for, how and what information was missed and what relevance the returned information had to the request. The 'english as a first language' issue was again seen to be a problem here. For subsequent studies, it was seen that the REs of the DM would need to be improved to better determine the intent of the request, and that the Knowledge base would need to be enlarged.

The questions were expected to be ranked with decreasing order of usefulness, with the last one expected to have little or no effect on usefulness. Again, this was asked to determine if the respondents were just 'circling values'. Figure 5.21 and Table 5.12 support this expected trend.

The overall reaction is consistent with that for Figure 5.5 and Table 5.4 (page 199). The System was not as effective as it should have been in providing relevant information to the users' enquiries. However, a mean of 2.61 for usefulness in getting access to information was seen as an encouraging, if not positive result. Quote 5.11 was also encouraging - the system worked - just not well enough!

5.3.3.7. SPD251.012: Other Domain Knowledge

Figure 5.22 shows the categorised 'Other Domain Knowledge' comments. Responses in this and the 'General Comments and Improvements' questions were also used indiscriminately to indicate the major deficiencies in the system.

Most Domain Knowledge (DK) comments indicated that knowledge on other computer languages and on other units would be useful. As noted before, some requested DK was not within the scope of this thesis, but the framework of the system allowed some FAQ lists related to the requested DK to be data-mined for subsequent studies with very little extra effort but with a large gain for users.

Also users wanted more *specific* information. Unfortunately, some wanted just the answers, not pointers to where to look (sixth quote in Quote 5.12, page 213). This again is a reflection of the learning paradigm of the researcher.

The quotes of Quote 5.12 (page 213) are indicative of the overall attitude of users. The third is a good example of the English problem with the fifth being encouraging.

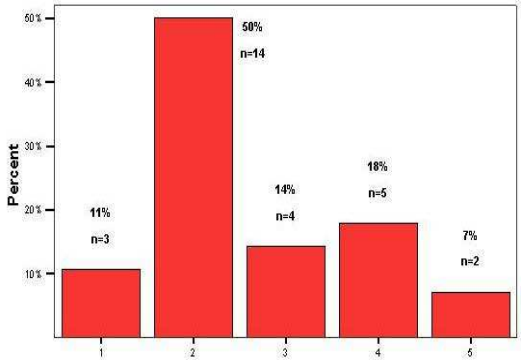


Figure 5.21a : Relevant Knowledge

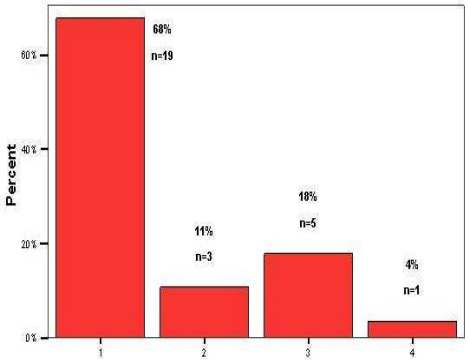


Figure 5.21c : Deeper Knowledge

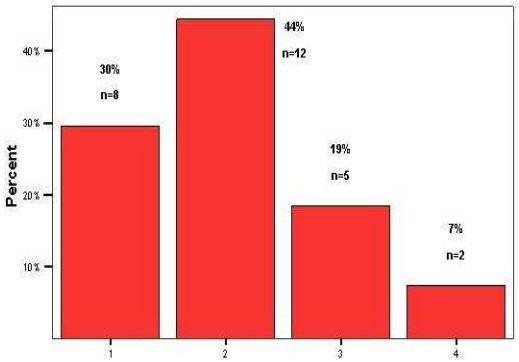


Figure 5.21b : More Knowledge

Fig.	Mean	Median	σ
a	2.61	2.00	1.1
b	2.04	2.00	0.9
c	1.57	1.00	0.9

Table 5.12 : SPD251.012: Response statistics for the *'perceived Depth of Knowledge'*

Figure 5.21 a,b and c: SPD251.012: Response distribution for the *'perceived Depth of Knowledge'*

it hid it well at times, but it did actually know quite a lot.

Quote 5.11 : SPD251.012: *'Depth of Knowledge'* comment

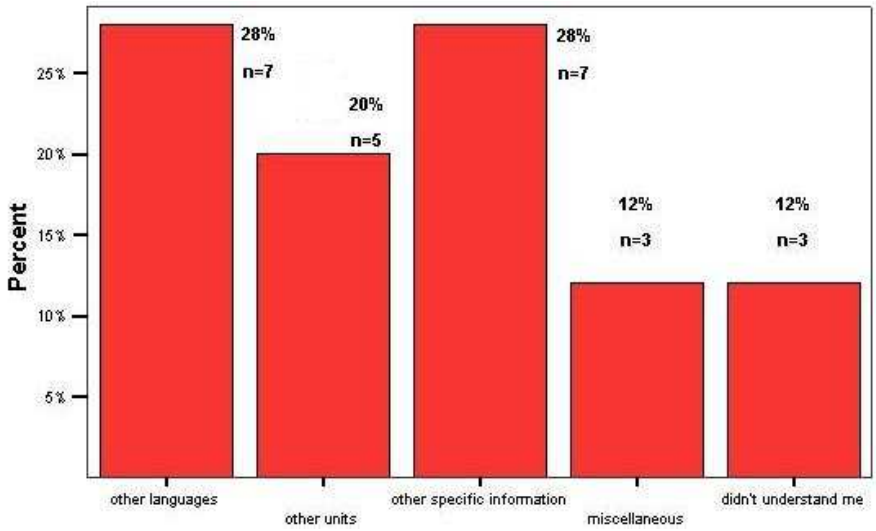


Figure 5.22 : SPD251.012: Recoded response distribution of comments on *'what other Domain Knowledge would be useful* in the ***Mentor*** System'

code architecture. specific of problems (e.g. a copy of the perl FAQ). Code guidelines
more extensive knowledge about different programming languages
know human intelligent in understanding a question ask.
undertand[sic] the question!
how about "infecting" this new system for other units, too.
be more specific rather than only giving suggestion where to look for info. Should give info directly.

Quote 5.12 : SPD251.012: *'other Domain Knowledge'* required comments

5.3.3.8. SPD251.012: General Comments and Improvements

Figure 5.23 shows that the 'good idea' perception balances out the 'did not work/useless' perception. The user comments, (examples in Quote 5.13 and Quote 5.14) provided useful feedback for development of the system for the subsequent studies. Unfortunately, only about 25% of the respondents made comments.

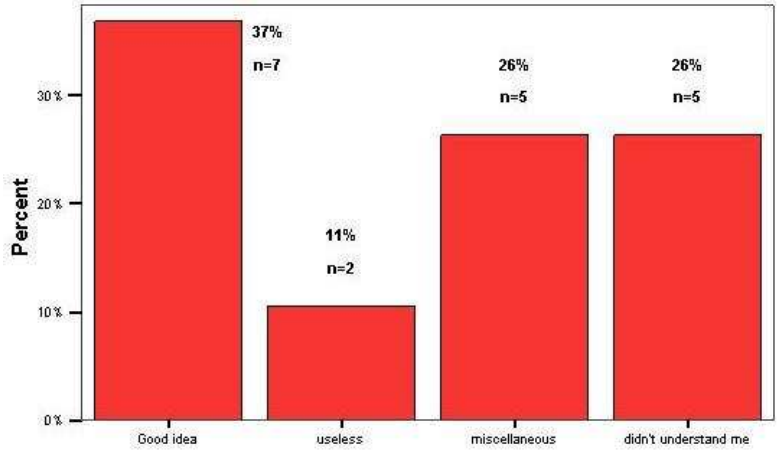


Figure 5.23 : SPD251.012: Recoded response distribution of *'General'* comments on the ***Mentor*** System'

overall it was very helpful, thanks
Usefull; A very good idea! It would have been good if active earlier
It was completely useless to me, but I do hope so see some improvements in the future :)
I find such tools mostly time wasting
hard to communicate with but its a good idea as international students too shy to ask questions.
I think that the system redirect user to web pages more than actually answering the question.
Mentor System doesn't know what I typed
add more conversation inside so the mentor system understand more what the user ask them. if possible the mentor system understand what will the user will be ask. I mean that mentor system can understand a user with english as second language

Quote 5.13 : SPD251.012: *'General'* comments on the ***Mentor*** System'

more freedom to go along my lines of questioning and not the mentors, this was its worst feature
When it came back with new information (an answer to a past question) it would often answer the same question more than once. This could get annoying. Also at times it needed an exact string where a keyword may have been better.
when you type help it should give you basic user information
comment nicer. speak normal casualy. close it if I want to.
be made into a pda sort of thing
more useful links. put some animation :)
more specific, very much larger DB. greater scope of phrasing understanding
It did not understand a lot of keywords
try to understand the question and give relevant answer
Possible larger direct question knowledge base; eg. FAQs

Quote 5.14 : SPD251.012: *'Improvement'* comments on the ***Mentor*** System'

5.3.3.9. SPD251.012: Conclusion

This case study, whilst not a success with respect to the hypothesis, was encouraging.

The system design and implementation had problems as could be seen from the formative evaluation of the user metrics and the summative evaluation from the questionnaire. The system was robust enough however, that many of these problems could be fixed even during the 5 week duration of the study.

The framework supported the DLP and AG paradigms and although these were not seen as very effective for the users, the results indicated that extra development of the DM and the KB could remedy many of the issues raised.

Finally, although the users did not see the system as being beneficial, the initial feedback was seen as encouraging and positive, and confirmed that the system could have a useful future as a mentor in an educational environment.

The formative and summative evaluation of the first study was used to develop the system further in the long break between the second semester 2001 and first semester 2002. The next study concentrated on improving the system's knowledge base, and of getting access to it, and on improving the integration of the DLP and AG paradigms.

5.3.4. Case Study 2: AGV351, Sem 1, 2002

Figure 5.24 and Table 5.13 indicate that users found the ***Mentor*** System to be marginally beneficial (given the standard deviation value). Nearly 25% of the users ranked the benefit as '5' as opposed to no responses of '5' for the first study. Note as well that nearly 75% of the users ranked the system as '3' or above (Table 5.14). This was seen as a developmental improvement (Table 5.13) and as a very positive outcome.

The high standard deviation value indicated that not all users agreed on the level of benefit. So, as for the previous study, it was necessary to determine the strong and weak points of the system, as well as the effectiveness of various aspects of the supported educational paradigms.

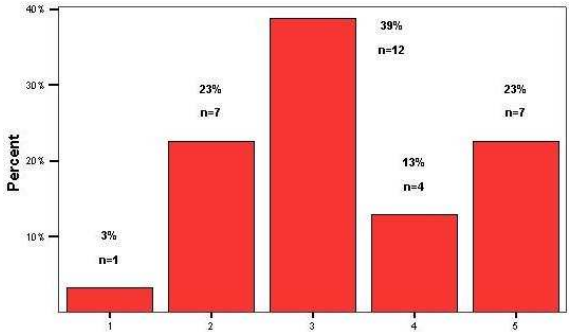


Figure 5.24 : AGV351.021: Response distribution on a 5 point Likert Scale to the question: *'Do you think that it was beneficial or helpful to use the **Mentor** System?'*

Study	# users	Mean	Median	σ
SPD251.012	38	2.58	3.000	0.92
AGV351.021	**31**	**3.29**	**3.000**	**1.16**

Table 5.13 : AGV351.021: Response statistics to the question: *'Do you think that it was beneficial or helpful to use the **Mentor** System?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	97	74	35	23

Table 5.14 : AGV351.021: Relative cumulative frequency statistics to the question: *'Do you think that it was beneficial or helpful to use the **Mentor** System?'*

5.3.4.1. AGV351.021: User issues

The number of users enrolled in the unit was 51, of which 37 answered the questionnaire. Of these, 31 (84%) used the ***Mentor*** System (Figure 5.25). All the comments on why the system was not used are shown in Quote 5.15.

Figure 5.26 shows the categorised reasons why the system was used and as expected, the majority of usage was for assignment help. Users continued to use it for reasons other than this though, notably for 'entertainment'.

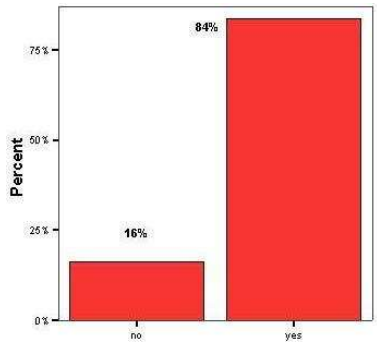


Figure 5.25 : AGV351.021: Response distribution to the question: *'Did you use the **Mentor** System in semester 1, 2002?'*

I find it easy to discover what I need to know by looking on web. Only info I can't find out is specific to assignment eg "Are we allowed to use system X"
Had my own web page with all required info,web FAQs more very beneficial. Never had the need for it.
Used it before in the previous semester but it did not help much. Also, prefer personal interaction rather than talking to a computer.
Wasn't shown how, somehow it was missed in the tutes.

Quote 5.15 : AGV351.021: Reasons why ***Mentor*** System was not used

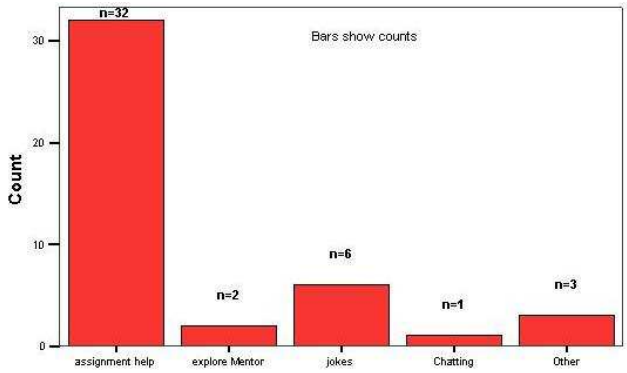


Figure 5.26 : AGV351.021: Recoded response distribution for reasons why ***Mentor*** System was used

Consistent with their responses about the benefit of the system, users also felt that it was marginally effective for help on their assignments (see Figure 5.27, Table 5.15). Of importance, 33% ranked its effectiveness '>=4' (Table 5.16). Subsequent questionnaire responses were used to determine how and where the system could be improved further.

It can be seen that the '*least beneficial*' and '*most annoying*' aspect is still the lack of understanding and/or the returning of irrelevant answers (Figures 5.28, 5.29, Quotes 5.16 and 5.17). Quote 5.16 indicates the lack of deep knowledge and the atypical response in Quote 5.17 re-iterates the 'learning' paradigm conflict mentioned in the previous study. Also the new interface was not seen as an improvement - users wanted greater control over its look and feel, and its action when it received a pro-active prompt (Figures 5.29, 5.30, Quotes 5.17 and 5.18).

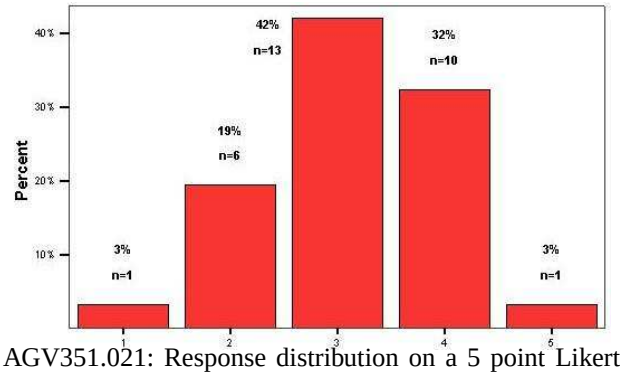


Figure 5.27 : AGV351.021: Response distribution on a 5 point Likert Scale to the question: *'How effective or useful was it for this purpose?'*

Study	# users	Mean	Median	σ
SPD251.012	35	2.229	2.000	0.877
AGV351.021	**31**	**3.13**	**3.000**	**0.88**

Table 5.15 : AGV351.021: Response statistics to the question: *'How effective or useful was it for this purpose?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	97	77	35	3

Table 5.16 : AGV351.021: Relative cumulative frequency statistics to the question: *'How effective or useful was it for this purpose?'*

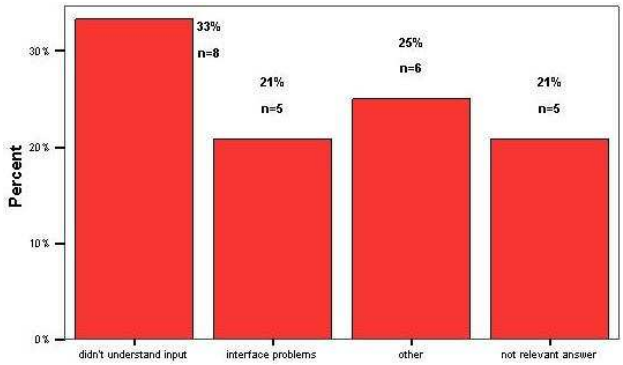


Figure 5.28 : AGV351.021: Recoded response distribution of comments on the *'least beneficial aspect'* of the ***Mentor*** System

> You had to ask lots of questions to get something out of it, especially at the late stages of the assignment, getting anything but basic information was hard.

Quote 5.16 : AGV351.021: *'least beneficial aspect'* comment

It was encouraging that the response rate for 'open comments' had increased (Table 5.17). These were seen as a positive reaction from users who felt they were contributing towards something of long term benefit rather than just an experiment (last quote in Quote 5.22, page 222).

Question	SPD251.012		AGV351.022	
	# / 69	%	# / 31	%
"least beneficial"	22	32%	21	67%
"most annoying"	28	40%	30	97%
"obtrusive"	17	25%	15	48%
"what was wrong"	22	32%	24	77%

Table 5.17 : AGV351.021: Comparison of response rates for comments

The entire user comments for the '*most annoying*' and '*most obtrusive*' questions can be found in Appendix E. These indicate that interface issues have overtaken 'lack of understanding' as the key issue. A new interface was designed to address these issues (see Figure 5.31) and to provide better hyper-media functionality. Given that users wanted more control over how the interface behaved when a pro-active prompt was sent to them, this functionality was designed into the new interface (Figures 5.32 and 5.33).

Users could then define the look and size of the interface as well as having control over whether the interface popped up, beeped, flashed, etc when it wanted attention. Users could also disable various topics such as the `jokesTopic` (i.e. asked them every so often if they wanted to hear a joke) and the `didYouKnow` topic (i.e. asked if they wanted to hear interesting but irrelevant facts). These had been introduced for 'entertainment'.

The new interface was used and evaluated in the final study. It was assumed that this would reduce or remove this source of problems in using the system.

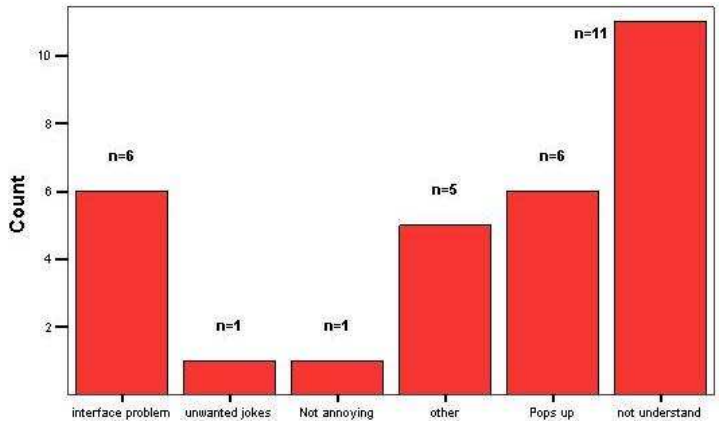


Figure 5.29 : AGV351.021: Recoded response distribution of comments on the *'most annoying aspect'* of the ***Mentor*** System

> Can't kill the buggery thing through normal operations. Give typical Marriot[18] answers - go look at ((insert obscure site)) Just answer the question MORE directly (not entirely) A stupid question to you may be a hard one to me!

Quote 5.17 : AGV351.021: *'most annoying aspect'* comment

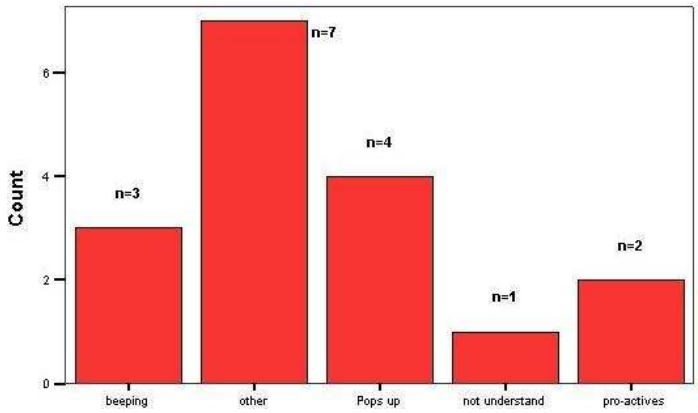


Figure 5.30 : AGV351.021: Recoded response distribution of comments on what *'aspects were obtrusive'* in the ***Mentor*** System

> Well the way it didn't go away. and Yes it flashed and sometimes i think it poped up.

Quote 5.18 : AGV351.021: *'what aspects were obtrusive'* comment

[18] 'Marriot' was the Lecturer in Charge of the unit and is the researcher.

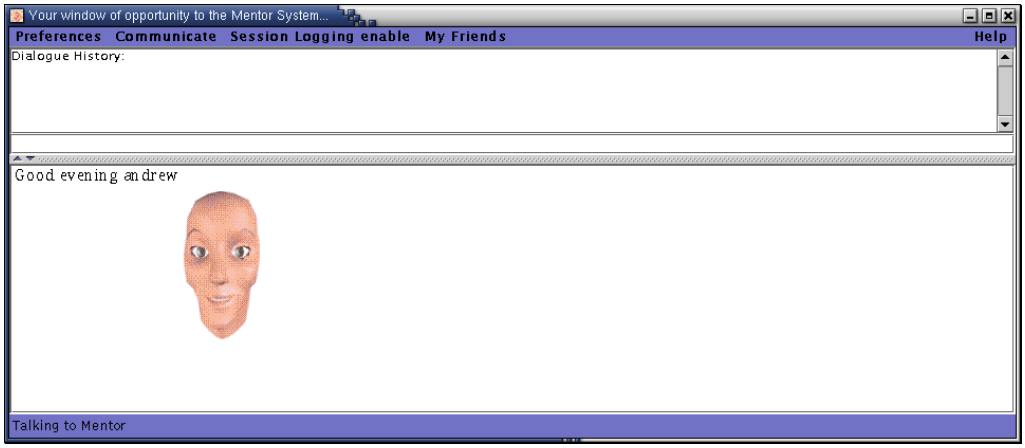


Figure 5.31 : ***Mentor*** System HTML text-based interface

Figure 5.32 : ***Mentor*** System HTML text-based interface preferences

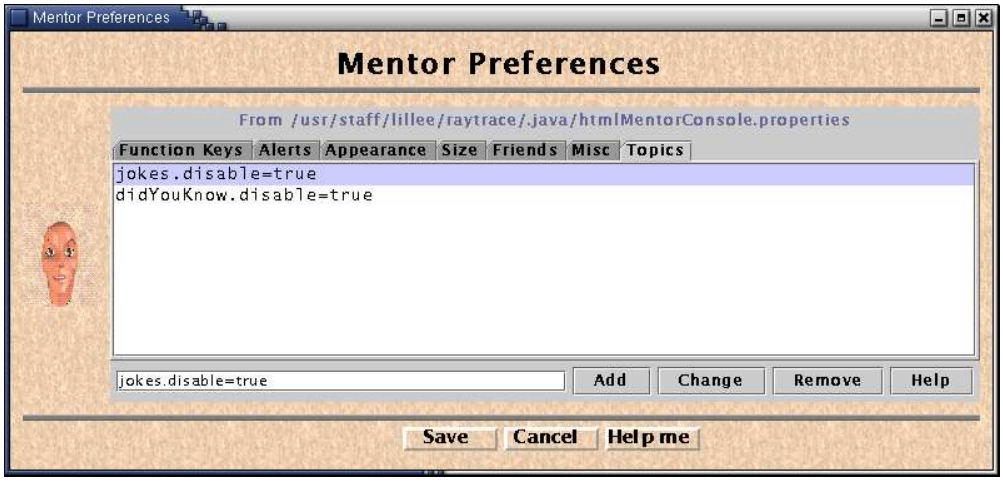


Figure 5.33 : ***Mentor*** System HTML text-based interface preferences GUI

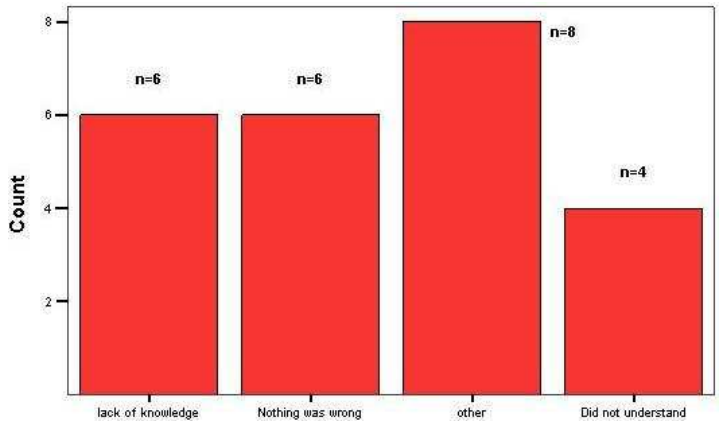


Figure 5.34 : AGV351.021: Recoded response distribution of comments on *'what was wrong'* with the ***Mentor*** System

unable to use it from home.
Quite slow.... can't really be helped so this wasn't a big issue. Maybe as more people start to use this it will get worse.
Didn't seem to cover all types of questions. Just involves adding more stuff to the database I guess??
Nothing wrong. so far

Quote 5.19 : AGV351.021: *'what was wrong with it'* comments

The '*what was wrong with it*' comments (examples in Quote 5.19) reflect the overall good and bad points of the study, but of importance, six of the users specifically said that there was nothing wrong with the system. Users also commented that they would like the system to be available when they worked at home. Due to the evolving technical structure of the network security within our Department, this was relegated to being future work.

Overall however, the system still did not understand the input all the time. The user input data were analysed and it was found that English or incorrect typing was still a problem (Quote 5.20).

waht is Red Greed Blue (RGB)

Quote 5.20 : AGV351.021: example of poor English in user input data

Users were also exploring the system more (see user input metrics in Table 5.18, greyed out text is metric formatting data) - user expectation rose with the system's ability to understand and answer, and hence exceeded it.

Finally, the user comments reflected the fact that some users had also started thinking about the system and about what it 'did' to help them. For example, in response to '*What was wrong with it*?', Quote 5.21 was an encouraging observation.

Not much. It didn't know everything, but thats impossible anyway.
Small knowledge base? But can't really fault it much since it's not human.

Quote 5.21 : AGV351.021: *'what was wrong with it'* comments

```
<input date="..." seconds="...">
who r u
</input>
<response responseCode="0" topicName="who" ...>
Mentor
</response>
<input date="..." seconds="...">
what r u doing
</input>
<response responseCode="0" topicName="dialogueManager" ...>
Fred, I am Mentor. I was developed by raytrace@cs.curtin.edu.au.
I am a large scale software-based Dialogue Manager
                        :
I hope to give assistance to students in their studying.
You can find out more about me from
   <a href="http://www.mentor.computing.edu.au/">
        http://www.mentor.computing.edu.au/</a>
</response>
```

Table 5.18 : AGV351.021: example of exploratory user input data

The user response rate for '*what was the most beneficial aspect*' was 97% (30/31). Many responses were detailed and covered several points. Users felt that the 'answers/information' and the 'websites' were the most beneficial aspects of the system (Figure 5.35, Quote 5.22). The enhanced DK came from data-mining several static as well as dynamic sources of information relevant to that unit. The Web sites were hand picked by the Authoritative source (the LiC) so that specific, relevant sites that were not overly general or misleading were presented to users. Users also believed that the assistance given by the DLP in starting the assignment and in guiding them through the various steps was very helpful (Quote 5.22). This was a positive result and seen as supporting evidence of the success of the development aim for this study.

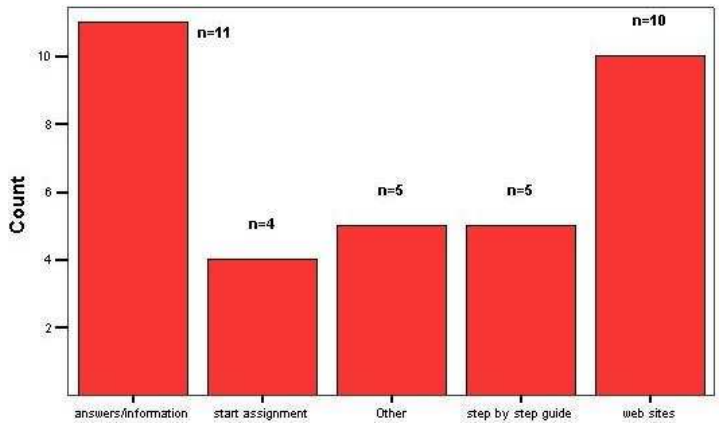


Figure 5.35 : AGV351.021: Recoded response distribution of comments on the *'most beneficial aspects'* of the ***Mentor*** System

Gave a process / suggested set of actions/steps for the assignments, and had useful documentation.
gives very quick response on simple assignment related questions by pointing a link or some reference I should read.
Helped with a way to start the assignments and an easy way to find out where the resources relevant to the assignment could be found.
Its pretty easy to use,straight forward. Also seems very friendly. Jokes and conversation are good
jokes were ok
• Knowing that we are contributing to something that may one day be a greatly usefull tool. • Staff and student database access was interesting • Active hyperlinks were cool

Quote 5.22 : AGV351.021: *'most beneficial aspect'* comments

5.3.4.2. AGV351.021: 'How did it help you' Evaluation

Again, users were asked via a 5 point Likert scale about improvements in time management, design development, code or design problems solving, and in the overall quality of their assignment due to the ***Mentor*** System. For this study, they were not asked about improvements to the presentation quality.

The first three questions were expected to be ranked with decreasing order of improvement and Figure 5.36 and Table 5.19 support this expected trend. However, the 'overall' ranking is not consistent with that for the '*Do you think it was beneficial or helpful to use...*' question (Figure 5.24 and Table 5.13, Section 5.3.4 on page 215). The system was not helpful in improving their assignment for these question categories.

It is a positive result that the mean of the responses has increased for the second study's '*categories of improvement'* but so has the σ (Table 5.19). The increase in the mean is marginal and within the spread indicated by the σ values. The second study had only half the number of respondents so this does have an effect on generalising the results.

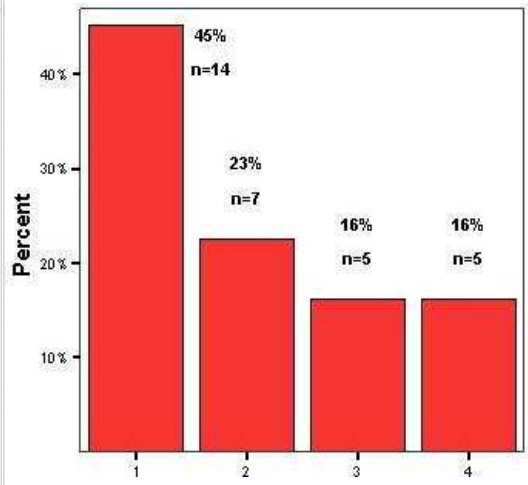


Figure 5.36a : Time

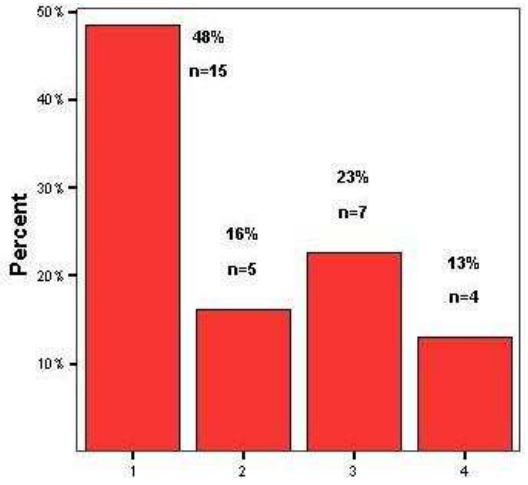


Figure 5.36b : Design

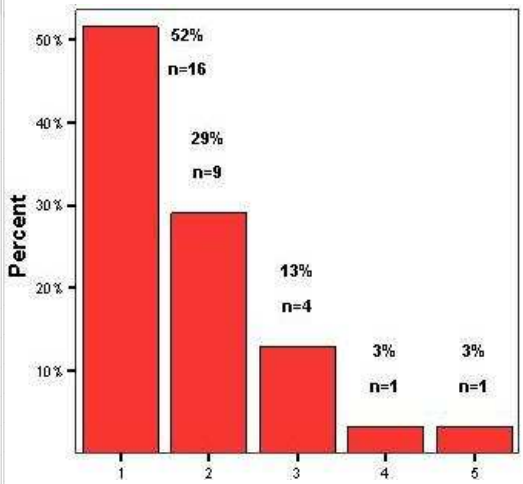


Figure 5.36c : Problems

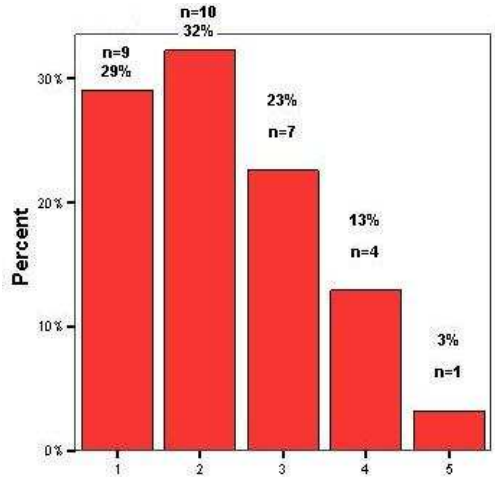


Figure 5.36d : Overall

Figure 5.36 a,b,c and d: AGV351.021:Response distribution for the categories of *'perceived improvement'*

However, the first study had many missing values for these responses. If those values are assumed worse case values of '1' for that study, for all those who said "yes they used the system" (65% of respondents), then the new statistics (Table 5.20) at least show how the system has improved even though users still are not benefiting enough from it .

Figure	SPD251.012			**AGV351.021**		
	Mean	Median	σ	**Mean**	**Median**	σ
Time	1.97	2.00	0.98	**2.03**	**2.00**	**1.14**
Design	1.82	2.00	0.81	**2.00**	**2.00**	**1.13**
Problems	1.65	1.00	0.98	**1.77**	**1.00**	**1.02**
Overall	1.93	2.00	1.05	**2.29**	**2.00**	**1.13**

Table 5.19 : AGV351.021:Response statistics for each of the 4 categories of Figures 5.36 a,b,c and d for the '*How did the* ***Mentor*** System *improve ...*' Evaluation

Figure	SPD251.012			**AGV351.021**		
	Mean	Median	σ	**Mean**	**Median**	σ
Time	1.71	1.00	0.94	**2.03**	**2.00**	**1.14**
Design	1.60	1.00	0.78	**2.00**	**2.00**	**1.13**
Problems	1.44	1.00	0.87	**1.77**	**1.00**	**1.02**
Overall	1.62	1.00	0.96	**2.29**	**2.00**	**1.13**

Table 5.20 : AGV351.021:Adjusted statistics for each of the 4 categories of Figures 5.36 a,b,c and d for the '*How did the* ***Mentor*** *System improve ...*' Evaluation

Another explanation for the discrepancy between the 'benefit' and the 'improvement' values is that the 'benefit' is from factors in the study other than those evaluated. For example, the DLP help may not be seen by users as directly helping them with their assignment (although it was planned to do so). Similarly, the 'benefit' may be due to the users feeling that any extra help is beneficial. Finally, the effect may be due to the system

just being a placebo - users felt that they must have benefited simply because they used the system. However, the negative rating of the first study does not support this conjecture.

Informal follow-up feedback from users indicated that it was not easy for them to categorise the way in which the system had helped. But that it definitely had! This could explain the anomalous ranking of the overall category as well as the high σ value.

This aspect of the evaluation indicated that there was a need for improvement in the help that the user received from the system and that the final evaluation questionnaire should try to clarify the categorisation.

5.3.4.3. AGV351.021: Directed Learning Path Evaluation

The majority of users did not find the prompts obtrusive (Figure 5.37) - although it must be noted that the way in which the prompts notified the users was seen as a big problem of the system (Section 5.3.4.1 on page 217).

Figure 5.38 indicates that the users did not feel that the DLP prompts were helpful. The system had improved from the first study but only marginally. Again, if the missing values are assumed worse case (for all those who actually used the system) then the values (see Table 5.21) show a a significant improvement over the SPD251.012 result. Even if not adjusted, both figures are quite respectable for computer based help with a specific task. The DLP seemed useful but still not enough. However Quote 5.23 was very encouraging and gratifying. A follow-up interview with this user indicated that they felt that they would not have passed the unit without this help. Inspection of the assessment results indicated that the user passed but would have done so regardless of how well they had done in the ***Mentor***-assisted assignment.

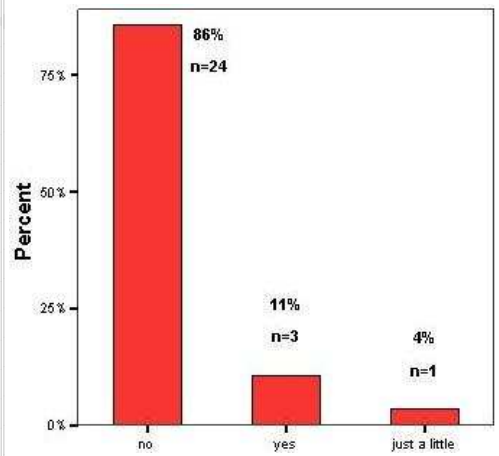


Figure 5.37 : AGV351.021:Response distribution to the question: *'Did you find the prompts obtrusive?'*

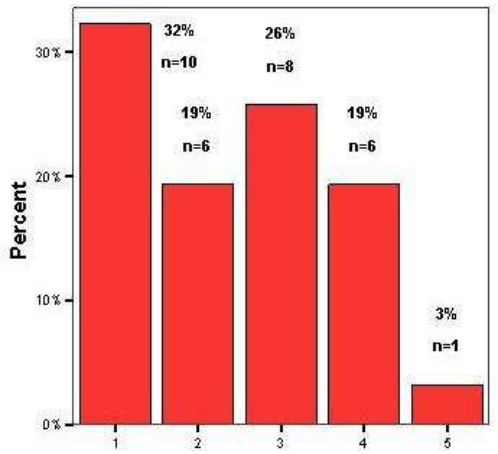


Figure 5.38 : AGV351.021:Response distribution to the question: *'Did you find the prompts helped with the assignment?'*

> This was THE best thing about the Mentor System.
> I couldn't have done without it.

Quote 5.23 : AGV351.021: '*How did the prompts help you*' comment

SPD251.012			SPD251.012 Adj			**AGV351.021**		
Mean	Median	σ	Mean	Median	σ	**Mean**	**Median**	σ
2.27	2.00	1.2	1.84	1.00	1.14	**2.42**	**2.00**	**1.23**

Table 5.21 : AGV351.021: Response statistics to the question: *'Did you find the prompts helped with the assignment?'*

5.3.4.4. AGV351.021: Authoritative Guidance Evaluation

Figure 5.39 indicates that most users felt that the *'delayed answers'* that they received were timely (not everyone would get one of these). Based on the evaluation of the first study, this study implemented a new feature to speed up the return of these delayed answers. Strangely, users felt that the return speed was better for the first study. One explanation may be that the system returned the answer so quickly (often within a matter of minutes) that it was not seen as a 'delayed' answer even though it was presented to the user as an "I have received an authoritative answer for your question 'XXXX'".

Notice that the '*helpfulness*' rating is only 2.5 even though the answers were given by the LiC[19] (Figure 5.40, Table 5.22). This again indicated a possible non-linear ranking.

The first quote in Quote 5.24 indicates that this delayed answer mechanism was put to other uses and also that the user understood the limitations of the system's understanding. The second quote either shows appreciation for the AG given or the user incorrectly assumed that the ***Mentor*** System just thought some more before answering! The final quote indicates that the user was not aware of the AG mechanism (although users were told before they used the system that their inputs were logged and monitored). However, compare this with the '*General Comment*' in Quote 5.25 for the same user.

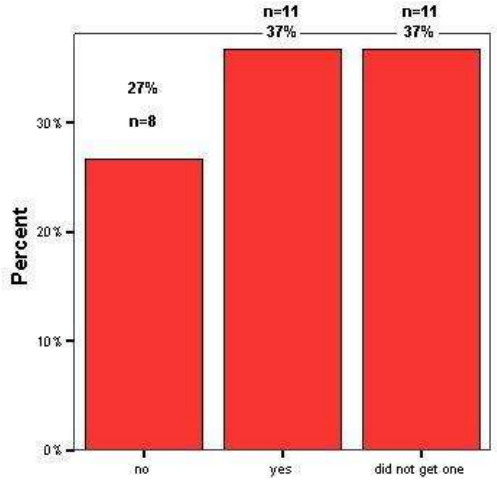


Figure 5.39 : AGV351.021: Response distribution to the question: *'Did you find these answers timely??'*

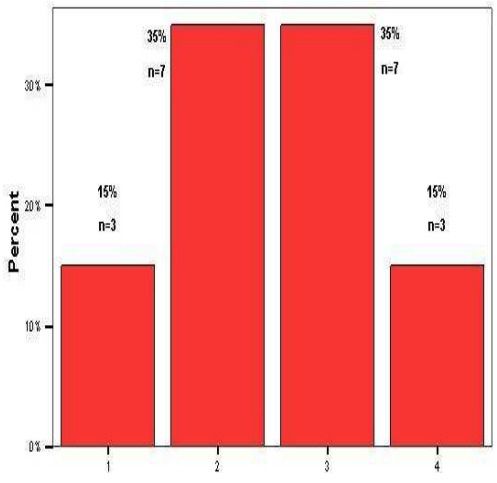


Figure 5.40 : AGV351.021: Response distribution to the question: *'Did you find these answers helped you with the assignment?'*

[19] The researcher.

SPD251.012			**AGV351.021**		
Mean	Median	σ	**Mean**	**Median**	σ
2.00	2.00	0.94	**2.50**	**2.50**	**0.95**

Table 5.22 : AGV351.021: Response statistics to the question: *'Did you find these answers helped you with the assignment?'*

There was one time the system gave me a broken link to some URL, so i left a message telling that the link is not working properly. Obviously the system doesn't understand what i typed in, but within the same day, the link was fixed =)
This part is quite impressive Good!
I didn't ask the tough questions because at the time I didn't know Andrew[20] was reading them or would answer personally.

Quote 5.24 : AGV351.021: *'How the delayed answers helped'* comments

Having a human review & answer the logs is a 2 edge sword. It was cool when really authoritative answers came back the next day. It was equally disquiting to know that the lecturer was "looking over my shoulder".

Quote 5.25 : AGV351.021: comment from *'General'* comments on the system

5.3.4.5. AGV351.021: Information Retrieval Evaluation

Again, users were asked about the usefulness of ***Mentor*** in getting access to *relevant* information/knowledge, *more* information/knowledge and *deeper* information/knowledge. Figure 5.41a and Table 5.23a shows that it was nearly successful in fulfilling the needs of users for access to *relevant* information about assignments (as planned) but is less useful for *more* or *deeper* information (as expected).

It was disappointing that this study's response rankings (Table 5.23) were only marginally higher than for the first study. In the second study an online book was data-mined to create a valuable source of indexed information in a special ***Mentor*** topic. Figure 5.42 (see page 229 for this figure plus description of it) shows the extra Domain Knowledge and Directed Learning Path information added to the normal unit topic. This should have increased the ranking for the answers to the question about getting access to relevant information. It had increased but only by a small amount.

Figure	SPD251.012			**AGV351.021**		
	Mean	Median	σ	**Mean**	**Median**	σ
a	2.61	2.00	1.1	**2.87**	**3.00**	**1.02**
b	2.04	2.00	0.9	**2.10**	**2.00**	**1.06**
c	1.57	1.00	0.9	**1.57**	**1.00**	**0.69**

Table 5.23 : AGV351.021:Response statistics for each of the 4 categories of Figures 5.40 a,b,c and d for the '*How did the **Mentor** System improve* ...' Evaluation

[20] 'Andrew' was the Lecturer in Charge of the unit and is the researcher.

The results for Figure 5.41 a, b and c represent 100%, 96% and 90% respectively of the respondents who said "yes they used the system". The first study had a very poor return rate for values for these questions - only 28 out of the 45 participants. Assuming worse case values for the missing values, we get the statistics of Table 5.24. Comparing this Table with Table 5.23, this study can be seen as a developmental improvement only.

Figure	SPD251.012 adj.			**AGV351.021**		
	Mean	Median	σ	**Mean**	**Median**	σ
a	2.00	2.00	1.19	**2.87**	**3.00**	**1.02**
b	1.62	1.00	0.86	**2.10**	**2.00**	**1.06**
c	1.35	1.00	0.77	**1.57**	**1.00**	**0.69**

Table 5.24 : AGV351.021:Adjusted Response statistics for each of the 4 categories of Figures 5.40 a,b,c and d for the '*How did the* ***Mentor*** *System improve ...*' Evaluation

Unfortunately this online book data-mining meant that there was really no followup states in the Domain Knowledge. This in turn implied that there was no mechanism to cater for followup enquiries. That is, more knowledge. Because of this (in certain cases) extra text that said roughly "ask me again and I will tell you more" was appended to the data-mined information response. Inspection of the user input metrics indicated that a few users took advantage of this new feature.

The quotes about getting access to '*more*' knowledge (Quote 5.26) reflect the fact that this data-mined knowledge may have been shallow. '*Deeper*' knowledge (Figure 5.40c) should not be there except for a few cases. Only 28/31 users responded, either due to the questions being at the end of the questionnaire or because they could not see how the system could address this problem. The majority agreed that '*deeper*' knowledge was not there (see typical quotes in Quote 5.27).

When I became more specific with my questions, it did not understand.
More specific questions I asked, it could not answer me.
Would occasionally give more info

Quote 5.26 : AGV351.021: *'more knowledge'* comments

You seem to think mentor is some deep source of information. It knew only the major web sites and some FAQ locations.
Just found google groups for searches more effective.
I think it only knows a limited amount of information when prompted for more, it gives the same tips given before.

Quote 5.27 : AGV351.021: *'deeper knowledge'* comments

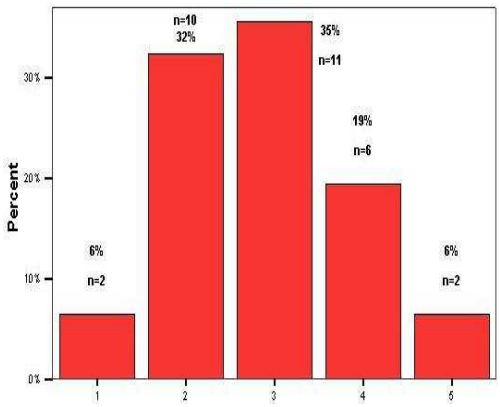


Figure 5.41a : Relevant Knowledge

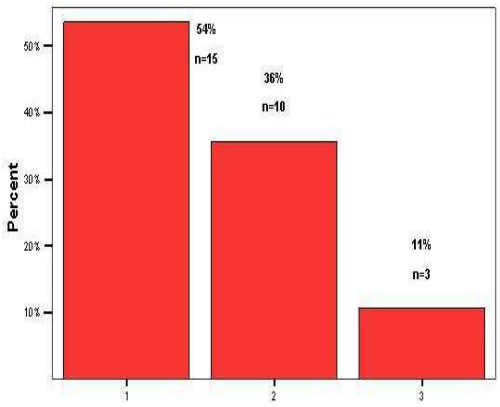


Figure 5.41c : Deeper Knowledge

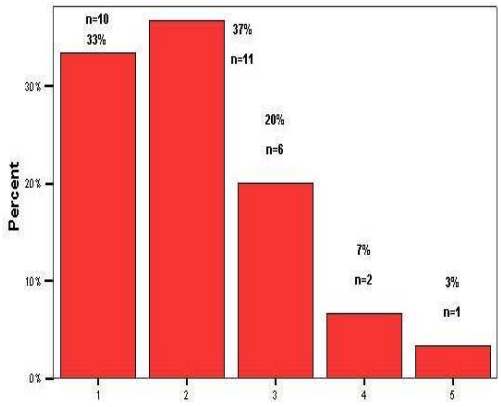


Figure 5.41b : More Knowledge

Figure 5.41 a,b and c: AGV351.021: Response distribution for the *'perceived Depth of Knowledge'*

Figure 5.42 montages various aspects of the Domain Knowledge and Directed Learning Path network that was created using a Topic Builder application developed for this study. The image in the top left hand corner shows the GUI for the entire AGV351 unit (this unit had an old name of CG351). The Assessment sub-node has been exploded in the middle image on the top row. It can be seen that the assessment knowledge and paths were further broken down into the Common, Assignment 1, Assignment 2 and Examination sub-nodes. As can be seen from the exploded view of the Common sub-node, these nodes contained still more sub-nodes. This hierarchical break-down of the course was seen as a logical division of information. In this case, the 'GTK' sub-sub-node can be seen to contain just simple 'user-request/system-response' nodes (upper right hand corner image). Over 150 of these types of nodes were developed.

The image in the lower left hand corner of Figure 5.41 shows an exploded view of the Assignment 1 sub-node. This sub-node encapsulates the Directed Learning Path for assignment 1 as well as specific 'user-request/system-response' nodes for questions. A similar hierarchy can be seen in these sub-nodes (middle image on bottom row) and some of these sub-sub-nodes contain the actual DLP.

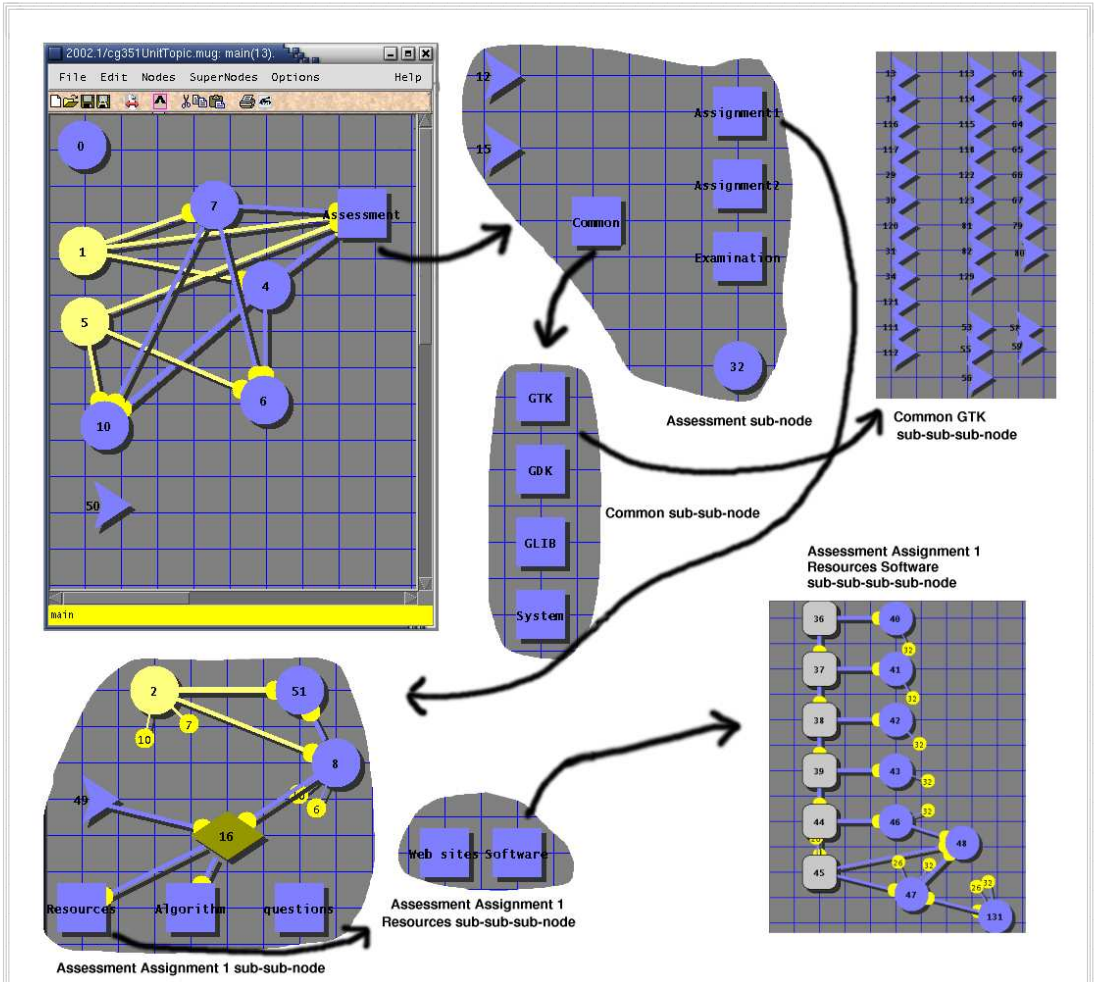


Figure 5.42 : Graph Network representing AGV351.021 Domain Knowledge

Although the system was only ranked as an improvement over the previous study, the development of this Topic Builder application was not wasted as it formed the super-structure for the final study. Analysis of the user input metrics showed that students did get answers to their questions - the Domain Knowledge was there. Students still wanted more flexibility in how they could ask the questions as well as getting a breadth and depth to the responses.

The image in the lower right hand corner of Figure 5.42 shows one part of the DLP for this study. Although not scheduled for evaluation in the final study, future work could add extra followup nodes to cater for the ‘more’ and ‘deeper’ knowledge type requests. For example, node 36, which currently has two followup nodes (37 and 40) could have another node which caters specifically for a "*tell me more*" request or a context specific request based upon what node 36 has returned to the user as a response.

5.3.4.6. AGV351.021: Other Domain Knowledge

Similar responses to the previous study but with the majority of requests for the C language - required for the assignment but taught in a previous unit (Quote 5.28). It was thought that knowledge for the examination - expected to be beneficial to users in their

overall studies - would be requested, but few requested it. Conversely, there were many requests for help with examinations in the user input metrics.

The last quote in Quote 5.28 is of concern from an educational point of view - again certain users only wanted the answers that gave an immediate benefit.

Users also wanted the DK to expand to other relevant areas concerned with this and other units. Users felt that since it had been beneficial in this restricted domain, then it should be beneficial in helping them with other units. This is true to a certain extent but the DK experts for different units may have different perspectives on what is important. For example, encapsulation of code as modules is seen as important in programming and certainly in the programming language C. This advice would be given by the system for any general purpose programming done in units simply based upon this good design principle. But for real time programming or for real-time visualisation of 3D graphics, any non-trivial encapsulation of code would have unacceptable procedure call overheads and hence would not be considered good practice - what is needed is real-time behaviour.

Therefore a conflict between the domain experts may occur and this would need to be resolved based upon the context of the request. Since the Topics which encapsulate this domain knowledge may be prepared separately, the two domain experts may not even realise that the system can give 'incorrect' or conflicting information. Currently no mechanism exists within the system to detect or resolve this situation. This is seen as future work.

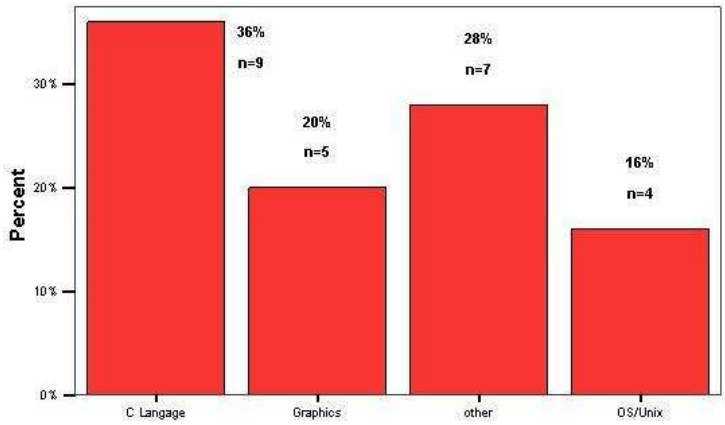


Figure 5.43 : AGV351.021: Recoded response distribution of comments on *'what other Domain Knowledge would be useful'* in the ***Mentor*** System

- general c commands (e.g. convert string to float in C)
Maybe the C language or Java language Even broader general things like exam timetable for CS department units.
It should have topics not only related to assignments. But also topics covered in the unit. This help in the revision as well.
Those how are sufficient for me to complete my assignment.

Quote 5.28 : AGV351.021: *'other Domain Knowledge'* required comments

5.3.4.7. AGV351.021: General Comments and Improvements

It was noticeable that the users were making more comments and more detailed comments (Table 5.25). It can be seen that the number of comments was roughly the same for the two studies and for the two categories but the second study had half the number of participants. The number of words per comment was also nearly double that for the first study. Users wanted to be involved in the improvement process.

The comments provided valuable feedback for improvements for the subsequent study and also provided encouragement that the system was effective and that users appreciated the effort involved (Quote 5.29).

Study	# users	General Comments			Improvements		
		# comments	%	words/ comment	# comments	%	words/ comment
SPD251.012	69	19	27.5%	12.3	17	24.6%	11.5
AGV351.021	**37**	**18**	**48.6%**	**22.2**	**23**	**62.2%**	**23.8**

Table 5.25 : AGV351.021: Comparison of ’quantity’ and ’detail’ of user comments from Case Studies

The first quote in Quote 5.29 is repeated in the other studies. It shows a lack of understanding of knowledge/information versus data. It is obvious that a search engine will return data about a certain topic but the quantity and quality of the information in the data is unknown or may not be helpful. Getting one Web page may be just as useless as getting ten thousand. The one Web page may contain little, misleading or incorrect information. Getting ten thousand Web pages does not help solve a problem. The AG paradigm ensures that the user gets relevant information not just data, simply because an expert has sifted through the data to get or choose the best or most relevant information. It is this information that can then be passed on to the user in the correct context.

The second quote (again typical of concerns raised in this and the last study) shows awareness of the problem of using technology that removes or distances users from their primary ‘carer’. The researcher agrees with the user’s concerns and feels that this issue needs to be addressed. Interesting future research would be to see the advantages and disadvantages of two studies that stressed or de-stressed the ‘delayed answer’ aspect of the AG paradigm. That is, in one case, it pushed the concept that the LiC was always there ‘caring’ for the users by monitoring their problems. In the other, it indicated that everything that the users typed was being scrutinised. This may indicate a good compromise between feelings of ‘caring’ and ‘surveillance’.

The final quotes show that the users have an emerging understanding of the complexity of the system as well as an appreciation of the effort involved. This was also indicated through informal feedback given by users during the study. Of interest, is that user’s started seeing the system as a primary source of information and help not just as an annoying last resort. Finally, the last quote is high praise indeed!

The system should link to search engine, and grep some useful infor and display to clients next time.
Personally, I don't like the idea of having less human interaction through the use of mentor system. Although, most lecturers have tight schedules and looking for a way to increase their efficiency in helping users.
o More info about how to use the mentoring system. o Maybe have some other forms of interaction besides textual based. More user friendly
I think it is generally a difficult system to implement as you have to cater for almost all questions in that unit. Knowledge base must be very large.
If the mentor system could be used from home that would be great.
It is a great work to aid students with their work especially during after office hours where lecturers are uncontactable.
I really like the learning path it had to guide me through each of the assignments. It was a big help.
Good concept.
Keep up the good work! :)
It's a good idea, lots of potential, already come pretty far.
Is good can be very helpful, generally better than the paper clip.

Quote 5.29 : AGV351.021: *'General'* comments on the ***Mentor*** System

Typical '*Suggestions for improvements*' are shown in Quote 5.30 and were used to improve the system for the final study. Some issues had already been addressed (i.e. an improved user interface), some comments suggested future work given the limited time for preparing for the final study (e.g. the 'Ask Jeeves' feature).

The final part of the last quote was seen as another useful feature for the future, that had definite learning possibilities. Many students like the concept of a newsgroup for a unit where questions and answers may be posted by the students. Educationally this has advantages but two main disadvantages:

- some students will just consume the answer rather than think about the problem and how to arrive at the answer, and
- some students will post incorrect answers which can propagate to others unless the newsgroup is constantly monitored by an Authority.

The AG paradigm of the system could provide correct, relevant answers and only to users who had done the work and had walked the 'path' to the answer.

5.3.4.8. AGV351.021: Conclusion

Users found the ***Mentor*** System to be marginally beneficial, with nearly 25% of the users ranking the benefit as '5' (as opposed to no responses of '5' for the first study), and nearly 75% of the users ranking the system as '3' or above. Consistent with these responses, users also felt that it was marginally effective or useful for help on their assignments, with 33% of users ranking its effectiveness '>=4', and 77% ranking its effectiveness '>=3'.

The lines on this feedback form are too small ^v
Get rid of that fucking beeping
Maybe some options for the user to decide on the interface eg. user set the colour of the interface.
Its much better if can access the mentor system in much user friendly manner.
Used at home with linux would be a big improvement. If it could run through the tutes with us and also extra examples to run through with it.
Provide some related areas of interest for a given query. Not just the same answer.
e-mail notification when mentor system finds an answer.
Try to add some more useful hints esp. when assignments are nearly due. because as we progress (in doing assignments) more hints are needed to do more difficult stuff in the assignment. Comments: generally it's very useful =)
either increase the database manually or grep "suitable" inform on the web to answer those FAQ.
Basically just give more specific info instead of "look at this" kinda stuff because we already know how to use google and read e-mail!
Needs more control over what it can do. A scrolling mechanism would be nice. A list of topics you can ask it about would be helpful. Being able to really close it, not just minimise it and i would have used it but i hate having a untidy desktop.
1) better documentation on how to use the system 2) add in scrolls 3) able to reprint previous messages. 4) speed up the response time as if it takes too long to response, no body wants to use it. 5) improve the AI as I feel it is still not as smart to answer some questions
- Could have an "Ask Jeeves" approach ie if can't understand a ques give a list of possible questions - After each answer as "Would you like to know more about XYZ" - customisable interface.
- improve the interpretation of user questions. - Could also provide maybe a message board type functionality where questions students asked (and there answers) can be viewed by all students. However these questions can only be answered by the lecturer.

Quote 5.30 : AGV351.021: *'Suggestions for improvement'* comments

The response rate for 'open comments' had increased, and this was seen as a positive reaction from users who felt they were contributing towards something of long term benefit rather than just an experiment. The users also showed an understanding of the complexity of the system as well as an appreciation of the effort involved, and started seeing the system as a primary source of information and help, not just as an annoying last resort. Users also commented that they would like the system to be available when they worked at home.

For the DLP Evaluation, users did not feel that the DLP prompts were helpful, although nearly 50% of them indicated a value '>=3' in ranking the prompt's help on the assignment. However, this was a significant improvement over the previous case study. For the AG Evaluation, students felt that the delayed answers were timely and that they marginally helped with the assignment. Of interest, the mean rank of the '*helpfulness*' evaluation was only 2.5 even though the answers were given by the Lecturer in Charge.

In all aspects, the second case study showed an improvement over the first one. Six of the users specifically said that there was nothing wrong with the system. The formative and summative evaluation of the second study was used to develop the system further in the short break between the first semester 2002 and second semester 2002. The final study concentrated on improving the system's Domain Knowledge about assignments, and on improving the DLP paradigm.

5.3.5. Case Study 3: SPD251, Sem 2, 2002

Figure 5.44 and Table 5.26 give confidence to the conclusion that users found it to be beneficial (given the standard deviation value and the number of participants). Nearly 50% of the users ranked the benefit above '3' whereas only 13% ranked it less than '3'. (Table 5.27). This was an improvement over both previous studies and a very positive outcome (Table 5.26).

The standard deviation value is smaller than the previous studies and is a good indication that users' opinions are starting to converge. However, not all users agreed on the level of benefit. So it was necessary to determine how to improve the system so as to improve the benefit to all participants.

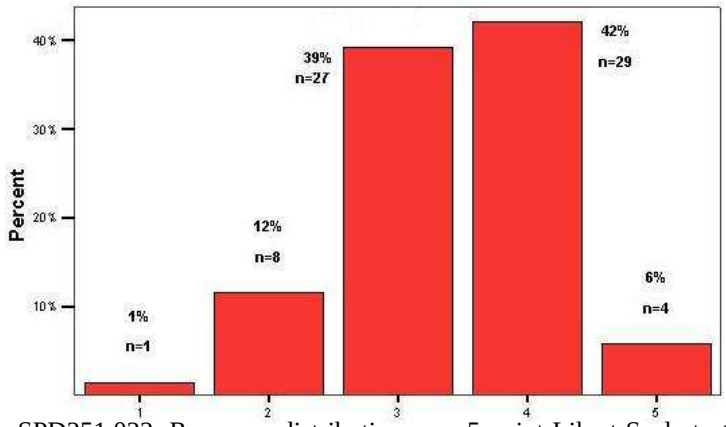


Figure 5.44 : SPD251.022: Response distribution on a 5 point Likert Scale to the question: *'Do you think that it was beneficial or helpful to use the* ***Mentor*** System*?'*

Study	# users	Mean	Median	σ
SPD251.012	38	2.58	3.000	0.92
AGV351.021	31	3.29	3.000	1.16
SPD251.022	**69**	**3.39**	**3.000**	**0.83**

Table 5.26 : SPD251.022: Response statistics to the question: *'Do you think that it was beneficial or helpful to use the* ***Mentor*** *System?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	99	87	48	6

Table 5.27 : SPD251.022: Relative cumulative frequency statistics to the question: *'Do you think that it was beneficial or helpful to use the* ***Mentor*** *System?'*

5.3.5.1. SPD251.022: User issues

The number of users enrolled in the unit was 96, of which 77 answered the questionnaire. Of these, 94% used the system (Figure 5.45). Assuming that the users who were enrolled but not did not answer the questionnaire did not use the system, then 74% of all users in the unit used the system to help them in their assessment.

It is important to note that in order to evaluate one aspect of the research, users had to write code to create a topic for the ***Mentor*** System for one of their assignments but the questionnaire specifically asked them to evaluate the system based upon their use for getting help on the assignment rather than on their programming.

All the comments on why the system was not used are shown in Quote 5.31. There are too few non-users to make any correlation conclusions (Table 5.28) but perhaps brighter users did not feel the need for it (similar to comments in the AGV351.021 study).

The first user was not part of the study but was obviously in the laboratory when the questionnaires were handed out and filled it in anyway. This is typical - many past users and unrelated students use the system even if not enrolled for the evaluated unit.

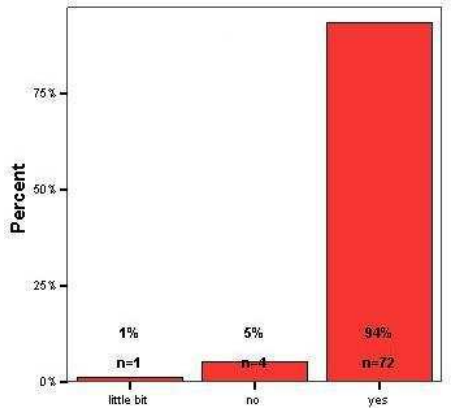


Figure 5.45 : SPD251.022: Response distribution to the question: *'Did you use the **Mentor** System in semester 2, 2002?'*

Dont have spd this sem However, I found the Mentor System for unit such as CG
Problems where it outputs "I know nuffink about spd at times" Did not recognise many types of questions Didn't seem to have enough knowledge Managed to work out how to tackle assignment2 myself
Wasn't aware I could until I read ass 2 spec later on in the semester, then friends seemed a better source of info.
I rather send email to the lecturer because it is easier to ask that way. (I can't make a complicated question to mentor)
I didn't find it very helpful, it seemed to have dificulty understanding questions. It was also annoying when I couldn't close it as it just minimises.

Quote 5.31 : SPD251.022: Reasons why the ***Mentor*** System was not used

Is english your first language	Gender of user	Age of user in years	Previously used *Mentor*	Previously enrolled in spd251	Enrolled course[21]	Rough Average grade
no	male	24	no	yes	IT	60-70
yes	male	19	no	no	DD	70-80
yes	male	19	no	no	CS	60-70
yes	male	-	no	no	CS	80+
yes	female	19	no	no	CS	80+

Table 5.28 : SPD251.022: Statistics on the six[22] who did not use the system

Figure 5.46 shows the categorised reasons why the system was used and as expected, the majority of usage was for assignment help. Users continued to use it for reasons other than this though. Most entries in 'Other reasons' were concerned with other aspects of the unit (programming in C, Perl).

[21] IT = Information Technology, CS = Computer Science, DD = Double Degree in CS and another discipline

[22] One user left no information other than 'did not use it'.

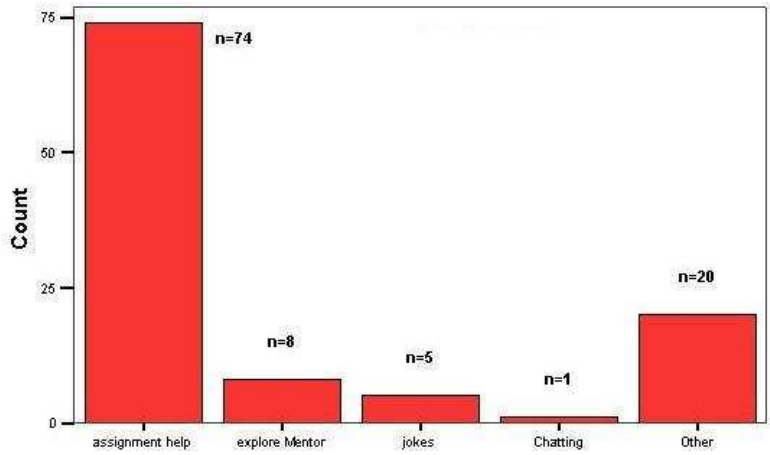


Figure 5.46 : SPD251.022: Recoded reasons why ***Mentor*** System was used

Development time was critical in this study - being only the few weeks between semesters. The DK development had concentrated on the assignment and the DLP paradigm had been improved to see how effective this could be. Figure 5.47, Table 5.29 and Table 5.30 shows that the resulting system was very effective in helping users with their assignment.

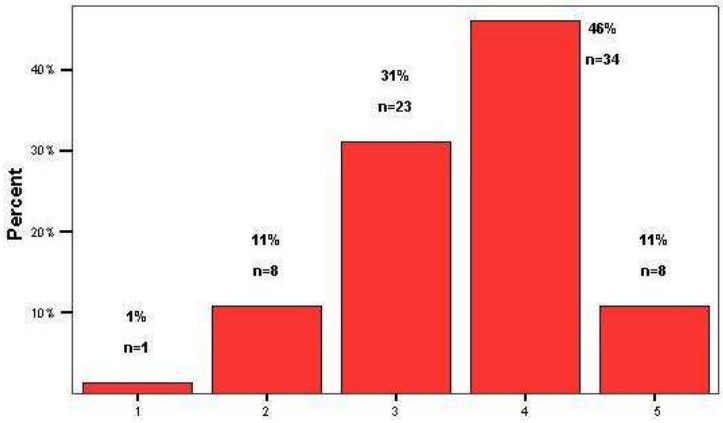


Figure 5.47 : SPD251.022: Response distribution on a 5 point Likert Scale to the question: *'How effective or useful was it for this purpose?'*

Study	# users	Mean	Median	σ
SPD251.012	35	2.23	2.000	0.877
AGV351.021	31	3.13	3.000	0.880
SPD251.022	**74**	**3.54**	**4.000**	**0.879**

Table 5.29 : SPD251.022: Response statistics to the question: *'How effective or useful was it for this purpose?'*

%≥1	%≥2	%≥3	%≥4	%=5
100	99	88	57	11

Table 5.30 : SPD251.022: Relative cumulative frequency statistics to the question: *'How effective or useful was it for this purpose?'*

Table 5.30 shows the effectiveness of using the system for the three case studies. The mean increases each time with the standard deviation remaining reasonably constant (but notice that the last study had twice as many participants).

Figure 5.48 indicates the responses regarding '*what was the least beneficial aspect*' of the system. Of these, 8% specifically said that there was nothing that was least beneficial. If blank comments are included, 42% indicated no '*least beneficial aspect*'.

Roughly 9% indicated that they were annoyed that they had to repeat all the questions in the DLP each time they re-connected to the ***Mentor*** System. That is, the path would have a number of questions, hints, etc that it would give to the user to help them in the development of their solution to the assignment. Normally, when a user leaves the system, ***Mentor*** notes where they are in the path and hence can continue from there when the user later re-connects to the system and requests further help. The 9% had to start at the beginning of the path each time. The problem was forseen because of the **programming of the topics** that the users had to do for their assignment. Users were told as part of the assignment specification, to use two ***Mentor*** client interfaces to the system - one for giving them help and one for testing. The users who met this problem did not do this.

The above users, who only used a 'testing' client interface, would 'kill' it when they changed their code and needed to re-run the new client. Since the Java version at that time did not have a mechanism to trap or catch this 'kill' request, their connection to the system died without recording where they were up to in the DLP. This is not a problem in normal use but became apparent when users used the same interface for 'testing' as well as for 'help' in this particular study. It did though have a significant negative impact on user evaluation of the system. This issue is repeated in many of the later question responses (with higher percentages).

Figure 5.48 also indicates that 16% of the users wanted the system to have other unit information such as due dates or unit outline information. The system did in fact have this information but updating it to reflect any new information was also a problem. Since the Java Virtual Machine *ClassLoader* caches loaded classes, it is not always possible to 'update' the system 'on the fly'. So, replacing a topic class that has corrected information cannot be done whilst the system is running. When the system is being used by 10-15 users from 6 in the morning until 2-3 the next morning each day, it is difficult to fix problems.

The system was modified so that after 3 in the morning, when no users were using the system, it would re-boot itself so that any new classes could be used in the updated version. However, as the assignment deadline approached, this lull in usage simply did not occur. Then the ***Mentor*** System administrator[23] had to connect to the system as a privileged user, issue a message via the system to all connected users that it was about to be re-booted and then ask the system to reboot.

[23] The researcher.

Roughly 17% were still not happy that the system could not understand all their questions. This is present in all of the three studies. There is no real solution to this except the building of very large Domain Knowledge databases or crafting more exacting REs.

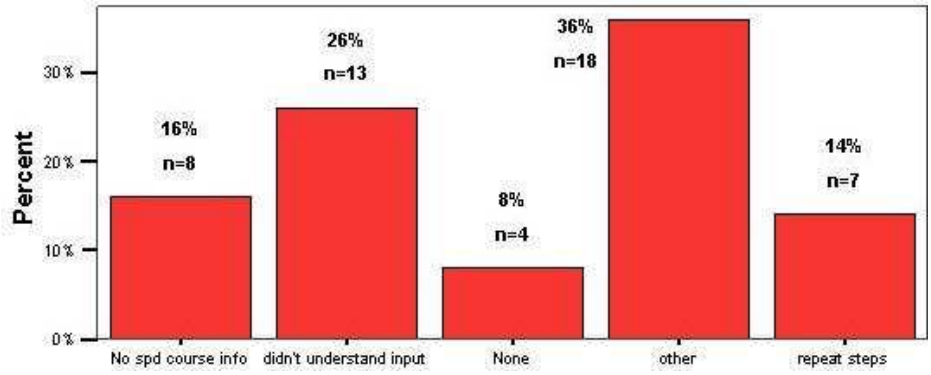


Figure 5.48 : SPD251.022: Recoded response distribution of comments on the *'least beneficial aspect'* of the ***Mentor*** System

sometimes when I answer "yes" to the Mentor System, it will generate another questions and again I answer "yes" and this will continue until mentor halts and says "try this <...> And come back to me again when you've done this". And then I have to go through the above process again.
If there was a problem along the way of completing the necessary steps in the assignment, the mentor system did not always provide a wide variety of solutions.
It can't repeat the answer that it gave previously when told. As sometimes students might be careless or didn't think carefully and wish to go back to the answer /question that mentor system provide previously.
It was really really slow. If you had trouble with something in between steps there is no facility for help.
No real downside... I'll never knock back help on an assignment :)
Most of the time, the Mentor System was interrogating instead of stopping for a moment to ask if I had any questions at the point of time.
Some of the links displayed in the system did not work.

Quote 5.32 : SPD251.022: *'least beneficial aspect'* comments

Similar responses were seen regarding the '*most annoying aspect*' (Figure 5.49) but with 8% specifically saying it was not annoying. However, 8% said 'popups getting in their way when not wanted' was the most annoying, 8% said 'unrelated pro-active questions' and 6% said 'unwanted jokes'. All of these could be controlled by the client interface preferences menu (Figures 5.32 and 5.33 Section 5.3.4.1 on page 219) but users did not seem to read the specified ***Mentor*** Help Web pages nor use the help menu on the interface. This problem could be solved with better user education.

Figure 5.49 also indicates that 23% found the need to repeat the previous steps in the DLP annoying - again users should have read the help pages. Also, 21% found its lack of Domain Knowledge/ability to understand questions annoying.

It was again encouraging that users gave more response comments. The values (Table 5.31) were comparable with the AGV351.021 study - third year students - and significantly

better than the first study of comparable students regarding experience and age. The conducting of the evaluation questionnaire stressed the need for answering all questions and for making as many open comments as the users liked.

Question	SPD251.012		AGV351.022		SPD251.022	
	# / 69	%	# / 31	%	**# / 69**	**%**
"least beneficial"	22	32%	21	67%	**49**	**71%**
"most annoying"	28	40%	30	97%	**56**	**81%**
"obtrusive"	17	25%	15	48%	**37**	**54%**
"what was wrong"	22	32%	24	77%	**46**	**66%**

Table 5.31 : SPD251.022: Comparison of response rates for comments

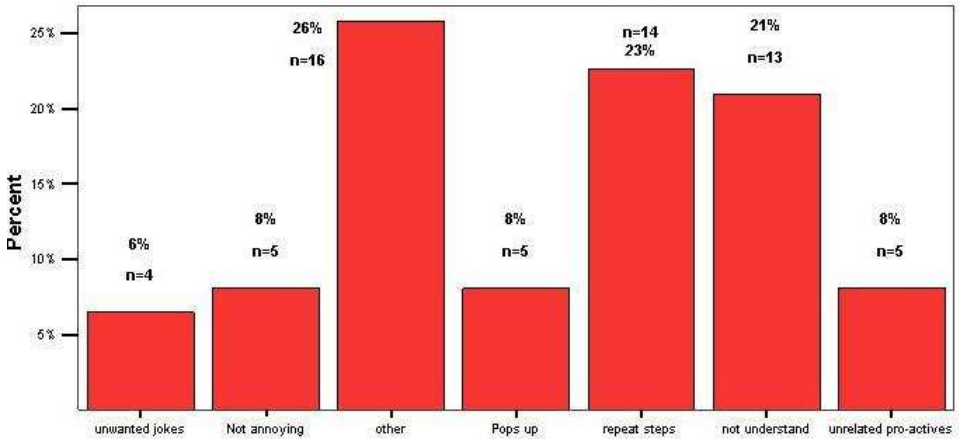


Figure 5.49 : SPD251.022: Recoded response distribution of comments on the *'most annoying aspect'* of the ***Mentor*** System

the un-related pro-active questions from the Mentor System like "Do you want to hear jokes" etc...
o) Trying to ask the right questions to get the right info. o) The jokes o) When the system initially didn't know the answer but four/five days later it found the answer and bugs you to hear the response/
That the system would not backtrack when you reached a certain level of questioning.
Not that courteous. ie it will be good to say "see you around" or "nice talking to you" instead of "bye" <- I find bye offensive
the pro-active questions popping up when I didn't want them too, also when it couldnt understand me, que? Also it kept calling me Dave[24] now and then
Didn't know trivial things like due dates, reg expressions etc. It only provided links which were irritating because it meant more time was spent surfing than getting the short info

Quote 5.33 : SPD251.022: *'most Annoying aspect'* comments

[24] For variety, one system response for not understanding was "I am sorry Dave, I did not understand that" as an oblique reference to the film '2001: A Space Odyssey' by Stanley Kubrick based on the book 'The Sentinel' by Arthur C. Clarke. The system even understood questions about this and could point to a website dedicated to this.

As can be seen in Quote 5.33 there was a great diversity of comments about the '*most annoying aspect*'. Some were beneficial for system improvement, others were bewildering (quote four in Quote 5.33). As for previous responses, these issues were considered part of the summative evaluation for the research and noted as future work where relevant.

The '*most obtrusive*' aspects (Figure 5.50) were due to 'popping up with a pro-active question' (14%), 'repeat steps' (6%) and just 'popups' (16%). Of interest, 4% specifically said it was not obtrusive, and 52% made no comment about this aspect.

The first two quotes of Quote 5.34 do not seem to indicate 'obtrusiveness' but were probably just general comments about bad aspects of the system.

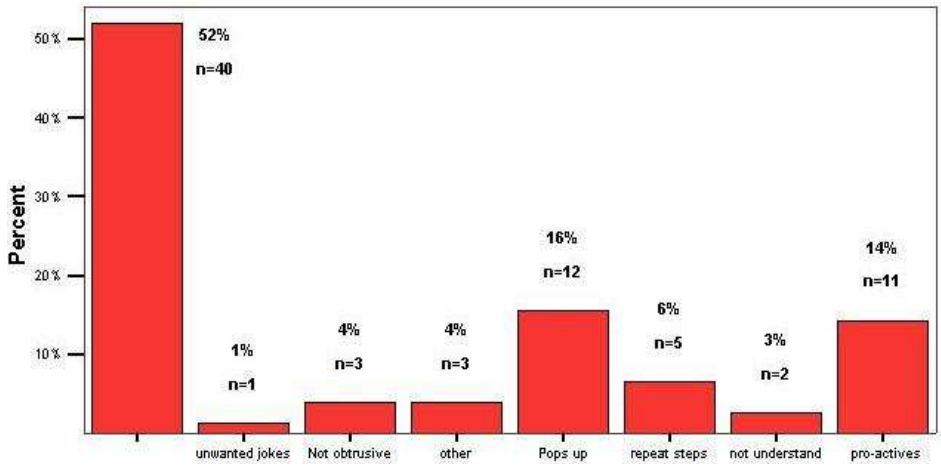


Figure 5.50 : SPD251.022: Recoded response distribution of comments on what '*aspects were obtrusive'* in the ***Mentor*** System

When going through the assignment steps, a user must progress from begining to end to receive help on the required part. At the same time, occasionally you could not return to areas which you have passed and then decided you need help with.
sometimes it didn't understand my question, however it was good that the next day, I could find the answer, cause the mentor database has been updated quite frequently.
pro-active part. But jokes were funny
Randomly popping out. But was fixed after I set the preferences.
Only proactive topics until I found out how to disable them. It is a good feature though.

Quote 5.34 : SPD251.022: *'what aspects were obtrusive'* comments

Some users experienced initial problems with the interface 'popping up' but successfully used the preferences system to control it. Others did not do this. This indicates that a better initial demonstration of the system could ameliorate this problem. Note that the final quote in Quote 5.34 also indicates that better initial education of the user about the system functionality may reduce user problems.

Figure 5.51 indicates that 42% of respondents did not indicate a problem when asked '*what was wrong*' with the system, and, of more importance for this study, 23% specifically said that there was nothing wrong with the system.

However, 16% and 10% respectively indicated that the main thing was that it did not understand their questions or lacked knowledge about the question that was asked. This, as well as the need to repeat the DLP steps, were seen as the two dominant problems with this final study. Typical response comments (Appendix F) supported this.

Some atypical comments are shown in Quote 5.35. The first quote shows typical English as well as coherency problems. The intent of the comment is not obvious.

A solution to the third quote is easily supported in the ***Mentor*** System framework - the nodes in the DLP (see Section on Topic Network Builder on page 176) could be turned into 'entry nodes' as well so that users may reach that part of the DLP directly. This was seen as future work.

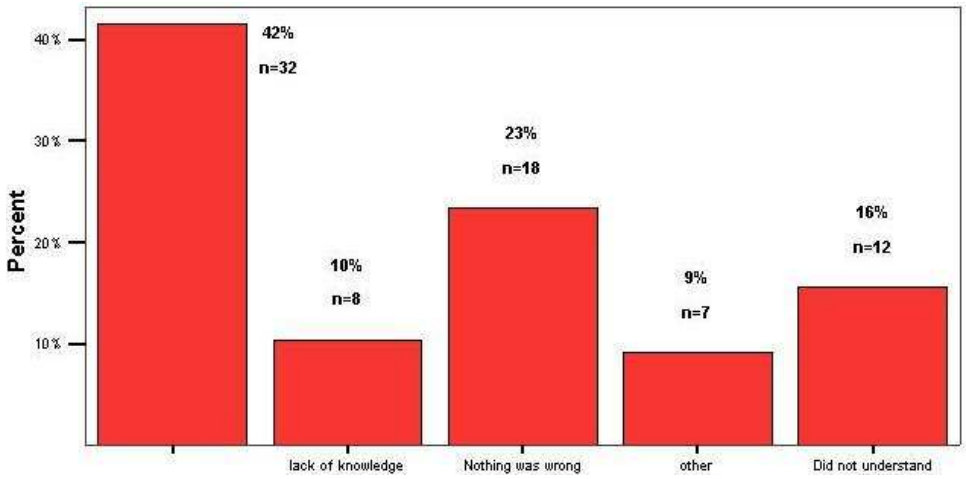


Figure 5.51 : SPD251.022: Recoded response distribution of comments on *'what was wrong'* with the ***Mentor*** System

I think it depends on the person's point of view. Let say person A wants to know about this question X, but in fact, there is no description or explanation at all. But in person B point of view, it might be helpful to understand and use it. Let's say I to keep say "yes" to get the answer what I really want about, for example. I couldn't simply just go and say "How to do pro-active". I think it is just the matter of how all possibilities we can find out of "about spd assignment2". questionaires.
Nothing seemed wrong
Can't jump to a certain stage of the assignment to ask for help
In a real conversation, we are interested in getting different kinds of answers for a topic. For example I am interested in why we need to fork() but usually the Mentor System answers it as what is a fork(), which sometimes proves to be not beneficial.
It said the same thing, that is only provided a small skeleton frame work to start, but one you had problems and debugging which took the most time then mentor was pretty useless, there should be a trouble shooting topic or something. Maybe "trouble with spdass2", "what trouble?" <then problem to mentor here>.
The only thing is that I don't really know what it knows about SPD!! If I asked "what do you know", it say the topic "Systems Programming 251", but I'm not sure what I can actually ask about SPD 251.

Quote 5.35 : SPD251.022: *'what was wrong with it'* comment

The fourth and fifth quote indicate a lack of accuracy in the RE's but also hint at the problem of repetition of answers. The design of the responses has tried to vary the content even in minimal ways. However, contrary to what you would expect, these variations may be detrimental in terms of user acceptance. This has recently been discussed in the article 'Repeat or Not Repeat' in Speech Technology magazine. In conclusion, the article says:

> *Repetition is the friend of the wise designer. Not only is it the basis of all learning, using it serves to limit the expectations of the user by reminding him that he is talking to a (stupid) machine.*
> *Repetition implies predictability and predictability assures the user that he is in control.*
>
> Dr. Walter Rolandi, The Voice User Interface Company,
> http://www.speechtechmag.com/pub/industry/1946-1.html

This rise in user expectation became very apparent when the user input data were analysed. Users expected the system to be able to understand them as though they were communicating with a friend. Given that typically 50% of the students did not have English as a first language, this meant that they expected the system to understand their native language or English modified by their native grammar/style.

Personalised parsing of user input could be catered for based on the nationality of the user by ignoring words with no semantic meaning:

aaargh! it's not working
erh! i'm not miss leh! hheeheee! dun wan to tell u leh! yeas! i really luv u leh!
"nothing la" means i have nothing to say
makan u lah! day!

Quote 5.36 : SPD251.022: Language specific user input

The first example in Quote 5.36 shows a typical English word that can be discarded, the next three show the use of 'la' language common in Singapore and Malaysia where the 'la' or 'leh' is just added, often at the end of sentences, with no semantic meaning. The fifth example shows how a student is trying to help the system by indicating what 'la' means. Unfortunately, as an AI system becomes more effective in understanding, users expect even more from it. The last example shows multi-language user input!

This usage highlights a significant problem that emerged in this study: many students, in talking to ***Mentor***, would indicate their nationality and hence store their preferred greeting language in their Context. The next time they were greeted, students believed that the system understood their entire language!

This was compounded by a system design error that used greetings like "ciao" and "ca va?". So, many users asked "can you speak XXX?" where XXX was French, Italian, Chinese, Japanese and Indonesian. Worse still, users also simply used foreign language phrases expecting the system to understand. In total, users entered 70 foreign language phrases, some incorrect, some mixed language (Quote 5.37).

For the researcher, the first three double column entries are recognisably Indonesian, although ‘nasi lemak’ is in fact just a popular food dish. The next three are Mandarin and/or Cantonese (many students are from Singapore, Malaysia or Hong Kong). The next is ‘English’ Japanese and the following 4 are French (or nearly so). Both French and Italian (the next three lines) were very popular because they were featured as part of the greetings.

apa kabar?	apa tuh?
selamat pagi	loe suka apa?
kasar loe	nasi lemak
ni how man?	ni ke yi jiang hua yi, man
wo ai ni	wo bu ming bai yi wei
wo he hao	jiang hua yu
Sayonara	sayonara means bye
au revoir	bona petite
cava?	comment appellez vous?
quelle heure est-il?	quelle temps est-il?
vous parlez francais	
tu italiano	capisco
come stai?	ciao, im good!
come se llama, est puebloe?	
dev dave	dev hei
dev hoho	dev oi oioi
man mpage	

Quote 5.37 : SPD251.022: Foreign Language user input

The researcher is not certain about the phrase concerning llamas, it even has a Spanish ring to it! The next two are unclassifiable by the researcher although they could be Italian, pidgin Italian or just plain garbage. The last, given the preceding user input context could have been a foreign phrase, but was in fact just a request for the man page for a Unix utility.

The personalisation of the responses to cater for nationality raised user expectations about the system’s ability to understand and these could not be met. This was a short term disappointment for users but had a long term benefit in that users started to realise that the system was not ‘all powerful’ and grounded their expectations accordingly. It also became apparent from user input that the students were thinking about the problems of Natural Language Parsing and Dialogue Management:

do you really have to comb through all this stuff?
oh to be a mentor....

Quote 5.38 : SPD251.022: Reflective user input

This inner reflection was also seen as a positive outcome of the system research.

Overall students felt that the Directed Learning Path - the starting help and the step by step guide - were the most beneficial aspect of the system (52%) whilst 29% felt that the

answers/information about the assignment was most beneficial (Figure 5.52). This is as planned for this study.

Their comments (Quote 5.39, full comments in Appendix F) indicated success for the paradigm but it was important to quantify how successful the system had been. As for previous evaluations, subsequent questions had targeted this aspect. There were still issues (e.g. third quote in Quote 5.35, page 241) that needed to be addressed.

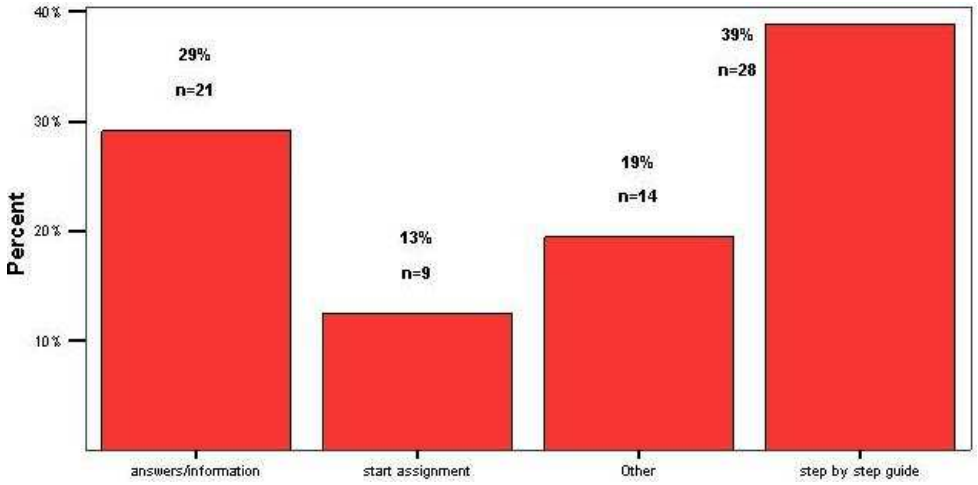


Figure 5.52 : SPD251.022: Recoded response distribution of comments on the *'most beneficial aspects'* of using the ***Mentor*** System

Provided a good work progression and ensured that the students were aware of each step and completed them.
I was given short and clear answers. If system did not have answer, I knew I would get it later (system has: I've asked me a question, now I know the answer)
o answer my questions, even the stupid ones. o give useful information on assignment 2
for the assignment 2, the MS was good in that it showed me clearly where and how I should go about in starting the assignment.
It was active in its conversation and prompted if I had do this or not, and if I had not, it had prompted me to the correct direction.
The immediate reply to simple questions that usually aren't immediately replied by tutors ... etc
Official and up-to-date info delivered on demand.
For assignment, it was giving me information I needed and the 'delayed' responses were great as well.
fast responses, and it knew the context of my questions. also the naughty jokes were good!
it help me how to organize my self when I was doing assignment 2.

Quote 5.39 : SPD251.022: *'most Beneficial aspect'* comments

5.3.5.2. SPD251.022: 'How did it help you' Evaluation

Again, users were asked about improvements in their assignment due to the ***Mentor*** System with the first three questions expected to be ranked with decreasing order of improvement. Figure 5.53 and Table 5.32 support this expected trend.

The overall reaction is marginally consistent with that for Figure 5.24 and Table 5.13 (Section 5.3.4 on page 215). The system was not helpful in improving their assignment for these question categories although the median value for the 'time' and the 'design' aspects is '3'. Overall however, the users indicated that it was beneficial.

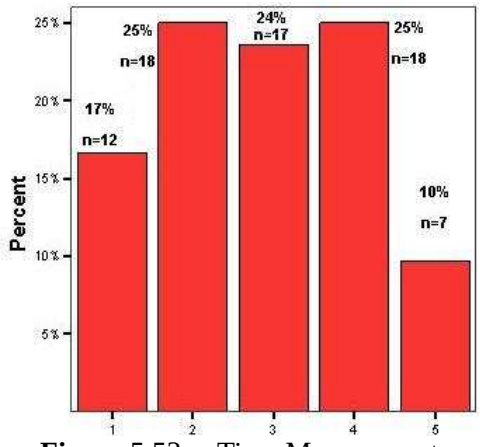


Figure 5.53a : Time Management

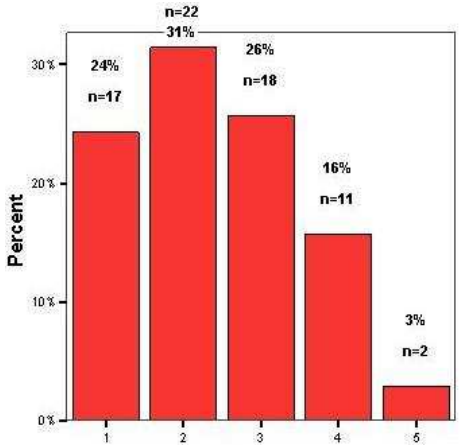


Figure 5.53c : Problem solving

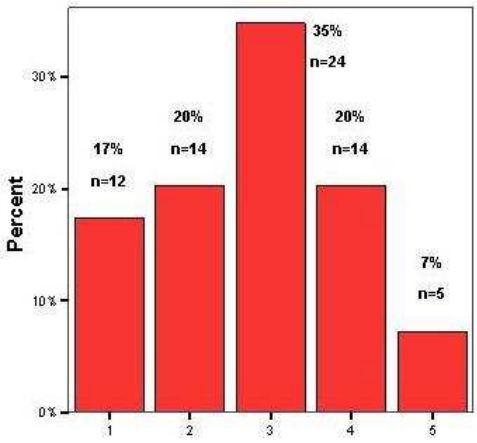


Figure 5.53b : Design

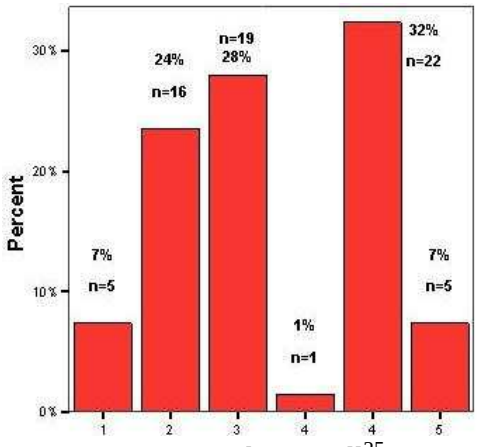


Figure 5.53d : Overall[25]

Figure 5.53 a,b,c and d: SPD251.022:Response distribution of *'perceived improvement to user's assignment'* for the categories 'Time Management', 'Design Development', 'Problem Solving', and 'Overall'

It is important to note (as can be visually seen in Figures 5.53 (a-d)) that the spread of answers in the four categories is large (high standard deviation values). Users had very mixed feelings about the level of improvement in the categories.

[25] One user specifically entered 3.5 rather than use the Likert scale.

Figure	SPD251.012			AGV351.021			**SPD251.022**		
	Mean	Median	σ	Mean	Median	σ	**Mean**	**Median**	σ
Time	1.97	2.00	0.98	2.03	2.00	1.14	**2.86**	**3.00**	**1.25**
Design	1.82	2.00	0.81	2.00	2.00	1.13	**2.80**	**3.00**	**1.17**
Problems	1.65	1.00	0.98	1.77	1.00	1.02	**2.41**	**2.00**	**1.110**
Overall	1.93	2.00	1.05	2.29	2.00	1.13	**3.10**	**3.00**	**1.07**

Table 5.32 : SPD251.022:Response statistics for each of the four categories of Figures 5.53 a,b,c and d for the *'perceived improvement to user's assignment'* Evaluation

The mean of the responses has increased for the various categories of improvement for the three studies but so has the standard deviation. Users were 'unanimous' in their condemnation of the system in the first study but very variable in their praise for the last two studies.

This may be due to the fact that a users' learning needs are varied - a good teacher uses different techniques to help a student. The users could have used the various aspects of the system in differing amounts. Alternatively, the help from the system for this final study may have been seen by users as being very varied. It could also be due to a misunderstanding of the question categories (see self-contradictory quote in Quote 5.40).

> It doesn't really have anything to do with time management. It might have helped by saying you do this first then this then proactive.

Quote 5.40 : SPD251.022: *'How did it improve Time Management'* comment

The first group of quotes in Quote 5.41 indicate that perhaps the 'help' had gone too far in reducing the amount of thinking that a student had to do. The aim of the system is not just the short term completion of an assignment but the inculcation of learning techniques that will help students throughout their learning lifetime. Perhaps the second group quote is closer to this requirement of the system. It is what a mentor does - try to reduce the wasted time of the mentee by using the experience and knowledge of the mentor.

Two ends of the spectrum are indicated by the third group whilst the fourth group quote indicates that the balance between 'nagging' and 'concern' mentioned in the analysis of the first case study may have been reached. Perhaps the ***Mentor*** System will always need to be seen in this way if it is to be effective. Perhaps this is the 'necessary evil' that is lecturers.

A comment on reaching a balance between the system 'spoon feeding' versus 'encouraging' learning can also been seen in the final quote of Quote 5.42. This user was a Double-Degree student with a rough average grade of 50-60.

In comparing the meaning of the quotes for the '*improvement in design and development*' question (Quote 5.42), one possible explanation for the spread of values is that the users' academic background would not be exactly the same for their different courses of study (first quote in Quote 5.42). Some of the users may have been given a more formal

definition of the software life-cycle and the nomenclature associated with it. This may have influenced the meaning they placed on the individual categories but not influenced their overall evaluation.

There is also support in Quote 5.42 for the difficulty in classifying 'how it helped' as well. The second quote could just as easily have been in the '*how did it improve time management*' section.

save me time from searching for info myself.
It guided through step by step. Don't have to waste time thinking or searching for answer / useful information
The mentor guided me to complete my assignment, so I didn't waste my time understanding the assignment.
It directed me quite well. ie. It told me exactly what I needed to do and what I needed to find out so I wasn't wasting time reading about things I didn't need.
I have poor time management at the best of times, no amount of prompting is going to help that.
If I didn't know how to do something then I would go to the mentor system and I would know the answer straight away.
By asking "Have u started the assignment". Really annoying. but quite helpful

Quote 5.41 : SPD251.022: *'How did it improve Time Management'* comments

knoledge about software engineering
Since it gave a step by step approach in doing the assignment, my assignment development was done in the same way making it neater and better designed.
I think someone who used mentor to do ass, and someone who didn't, the work would be pretty much the same.
The recommendations given were helpful, but the design and development were something I had to figure out for myself, and I did prefer it that way.

Quote 5.42 : SPD251.022: *'How did it improve design and development'* comments

5.3.5.3. SPD251.022: Directed Learning Path Evaluation

Figure 5.54 shows that nearly equal amounts of users felt that the DLP prompts were obtrusive as opposed to unobtrusive. This is a major problem. To explain a possible reason for this, it is necessary to look at the evaluation results for the questions '*what was the most annoying*' (page 238) and '*what aspects were obtrusive*' (page 240).

Based upon the formative evaluation of the second study, more active prompts to provide information about other parts of the course were added to the Domain Knowledge. These, plus the "*do you want to hear a joke*" and "*did you know*" prompts, along with the "*do you want help with your assignment*" prompts made up the bulk of the pro-active prompts. Unfortunately users did not seem to differentiate between the Directed Learning Path prompts and other ones.

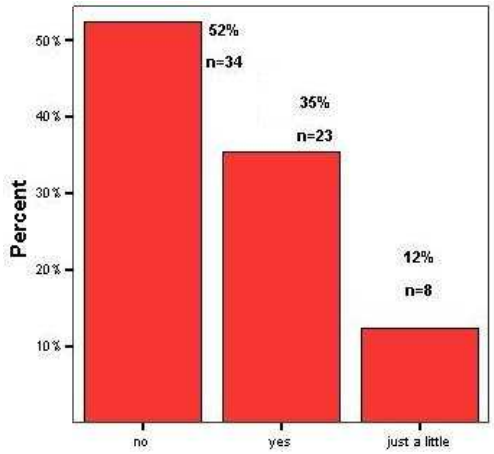


Figure 5.54 : SPD251.022:Response distribution to the question: *'Did you find the prompts obtrusive?'*

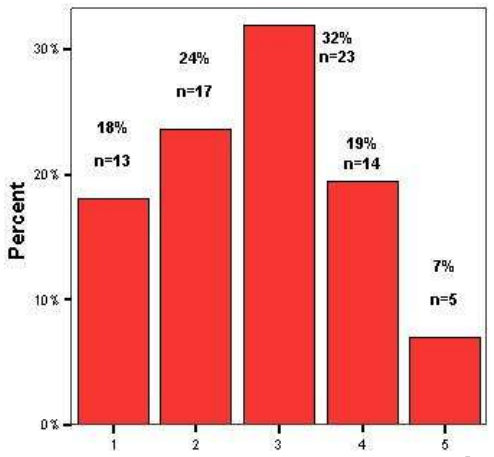


Figure 5.55 : SPD251.022:Response distribution to the question: *'Did you find the prompts helped with the assignment?'*

SPD251.012 Adj			AGV351.021			**SPD251.022**		
Mean	Median	σ	Mean	Median	σ	**Mean**	**Median**	σ
1.84	1.00	1.14	2.42	2.00	1.23	**2.74**	**3.00**	**1.17**

Table 5.33 : SPD251.022: Response statistics to the question: *'Did you find the prompts helped with the assignment?'*

Roughly 30% of users felt that the pro-actives and popups were the most obtrusive aspects and roughly 22% indicated that jokes, popups and unrelated pro-active questions were the most annoying aspects. It has already been indicated that users were told that these popups, jokes and unrelated pro-actives could be turned off via the interface. Again, better user education may reduce this problem in the future.

A failure of this study has been in not anticipating that this problem may have swamped legitimate evaluation of the DLP prompts' effectiveness and obtrusiveness. The original text for this evaluation question was:

> *Sometimes the Mentor System would actively prompt you and suggest learning paths to guide and help with the development of your assignments. Did you find that these prompts or guidance helped you with the assignments?*

A better question would have removed the word 'prompt' and concentrated on the Directed Learning Path aspect.

Figure 5.55 and Table 5.33 indicate that even though the prompts may have been seen as obtrusive, they were seen as almost helpful to the students with a mean of 2.74 (median 3) but with a very large standard deviation value.

5.3.5.4. SPD251.022: Authoritative Guidance Evaluation

Figure 5.56 indicates that the Authoritative answers from the Lecturer in Charge arrived in a timely fashion for most users. It should be noted that many students would ask questions starting late Friday evening and throughout the weekend. These would not be answered until the Monday morning.

Figure 5.57 and Table 5.34 indicate that although the system has improved, it has not really been helpful to the users and that there is a considerable variation in the user's ranking of the helpfulness. Since the delayed answers came direct from the LiC, this result seemed anomalous. To understand the anomaly, these questions were analysed and categorised. Figure 5.58 shows that although the largest category of answered questions (29%) were assignment related, they were not the majority of questions. Users explored the system as well (20%), asked questions about other units or topics(18%) and made general unrelated statements (15%). Users again had trouble with the preferences (2%) and many requests were to the system to 'go away' or stop nagging them with pro-active requests (9%).

In hindsight, the increased functionality of the system should have been reflected in a change to the evaluation questions to be more specific in targeting the usefulness or helpfulness of the delayed responses relating only to the assignment. However, the system is designed to help students in more aspects of their university life than just assignments. The typical quotes from Quote 5.43 (page 250) do reflect the overall helped provided.

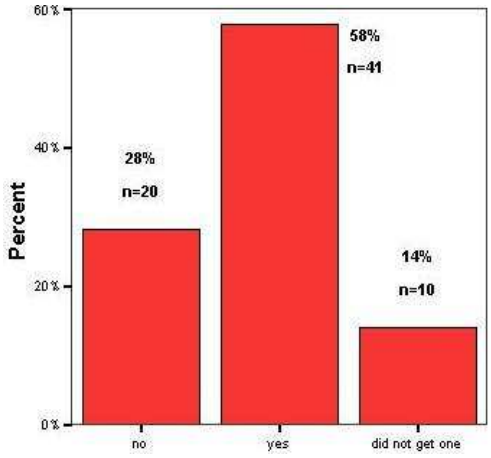


Figure 5.56 : SPD251.022: Response distribution to the question: *'Did you find these answers timely??'*

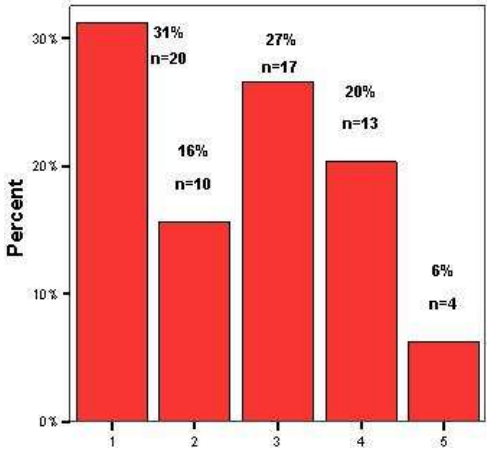


Figure 5.57 : SPD251.022: Response distribution to the question: *'Did you find these answers helped you with the assignment?'*

SPD251.012			AGV351.021			**SPD251.022**		
Mean	Median	σ	Mean	Median	σ	**Mean**	**Median**	σ
2.00	2.00	0.94	2.50	2.50	0.95	**2.55**	**3.00**	**1.29**

Table 5.34 : SPD251.022: Response statistics to the question: *'Did you find these answers helped you with the assignment?'*

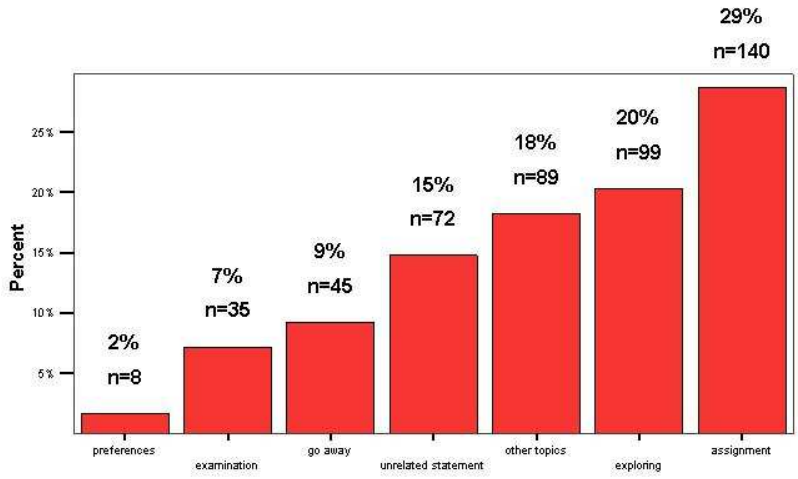


Figure 5.58 : SPD251.022: Recoded categorisation of the user questions answered by the LiC

The delayed answers were beneficial in the fact that if a problem or question couldn't be answered, you would get one eventually and it could be useful, However, if you miss the opportunity to answer 'yes' at the right time, you miss the answer altogether
This is very good!
this answering technique is a bit time consuming
Understand that I must go search for answers myself.
The were encouraging but often vague, the question may need to be followed up for better understanding.
They did help me but none of my q's were about the assignment.
Sometimes they were ok, other times completely useless.
Actually, they don't help much but they are really good! I'd say one of the better features since they are often more accurate
One thing abt delayed messages was that it showed the system "cares" not really but when the answers reached me, it did remind me that I had some portions of the assignment untouched.
If I can not found the answer by myself, it would be great to hear it from mentor
As these answers were obviously from a human, they tackled my problems directly.

Quote 5.43 : SPD251.022: *'How the delayed answers help'* comments

Question	# users	Mean	Median	σ
helped with assignment	64	2.55	3.00	1.29
usefulness of response	60	3.13	3.00	1.13

Table 5.35 : SPD251.022: Response statistics to the questions: *'Did you find these answers helped you with the assignment?'* and *'How would you rate the usefulness of the delayed response?'*

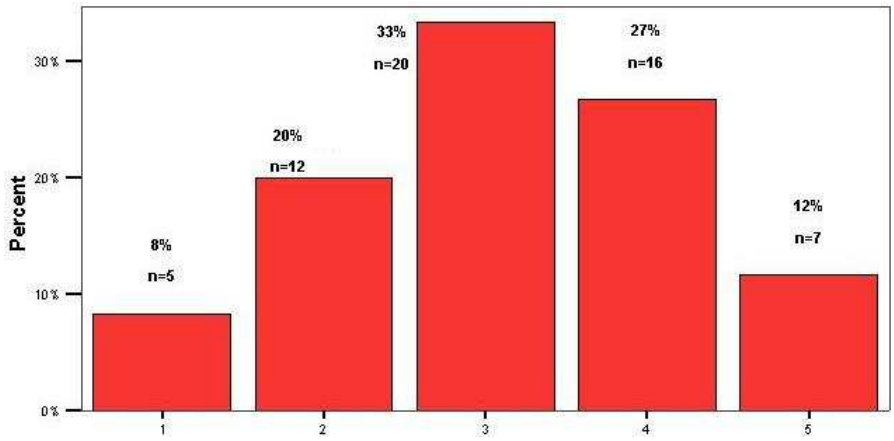


Figure 5.59 : SPD251.022: Response distribution to the question: *'How would you rate the usefulness of the delayed response?'*

It can be seen by comparing Figure 5.57 and Figure 5.59, and the values in Table 5.35 that a subtle difference in user perception of the AG aspect of the system exists. The questionnaire gave the following information:

> *The Mentor System provided you with two sources of information. It could give you an initial* **immediate** *answer but if it did not understand you immediately, it would often give you a* **delayed** *answer at a later date.*

The questionnaire then asked amongst other things:

Did you find these **delayed** *answers helped you with the assignments?*

1	*2*	*3*	*4*	*5*
Not at all				*a lot*

How would you rate the usefulness of the **immediate** *answers:*

1	*2*	*3*	*4*	*5*
Not very useful				*very useful*

How would you rate the usefulness of the **delayed** *answers:*

1	*2*	*3*	*4*	*5*
Not very useful				*very useful*

The difference in ranking (Table 5.35) supports the previous conjecture that the system provides help on many aspects of a user's studies not just assignments. Users ranked the responses as being marginally useful (mean of 3.13, median of 3) but with a large variation (σ = 1.13).

The questions that asked for the ranking of the **immediate** response *vs* the **delayed** response were aimed at providing a baseline for the 5 point Likert scale. That is, does a mean ranking greater than '3' indicate that users felt *marginally positive*, *positive* or *overwhelmingly positive* about the particular aspect of the system. Although other variables such as delay time affected the ranking, it is very interesting to note that **immediate** responses from the system (Figure 5.60 and Table 5.36) were ranked significantly higher (mean 3.46, median 4 and σ of 0.95) than the **delayed** answers (Figure 5.57 and Table 5.34 direct from the LiC (mean 3.13, median 3. σ of 1.13).

The ranking of the **immediate** response is consistent with the overall *benefit* of the system (Table 5.26, page 234) and the *effectiveness* of the system for assignment help (Table 5.29, page 236).

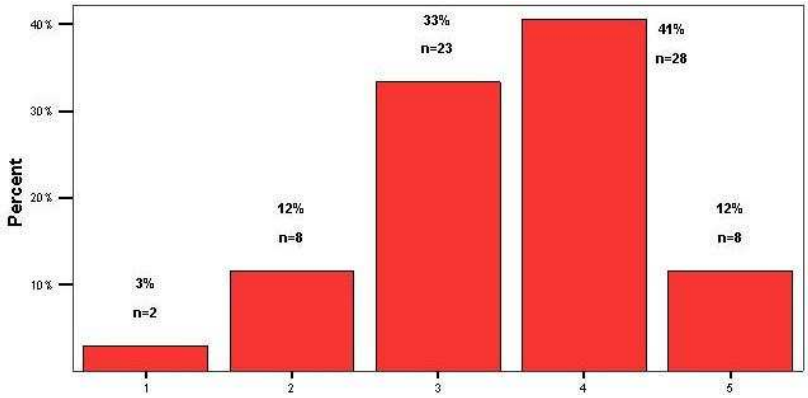


Figure 5.60 : SPD251.022: Response distribution to the question: *'How would you rate the usefulness of the immediate response?'*

# users	Mean	Median	σ
69	3.46	4.00	.95

Table 5.36 : SPD251.022: Response statistics to the question: *'How would you rate the usefulness of the immediate response?'*

5.3.5.5. SPD251.022: Information Retrieval Evaluation

Again, users were asked via a 5 point Likert scale about the usefulness of the ***Mentor*** System in getting access to relevant information/knowledge, more information/knowledge and deeper information/knowledge.

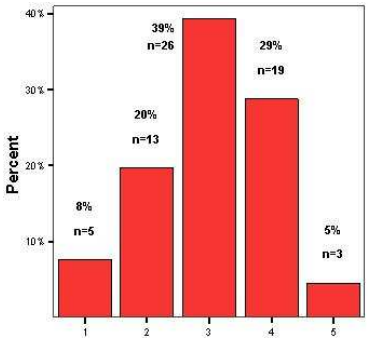


Figure 5.61a : Relevant Knowledge

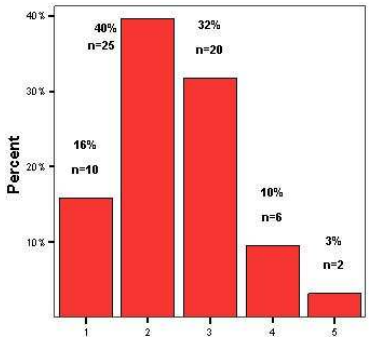


Figure 5.61c : Deeper Knowledge

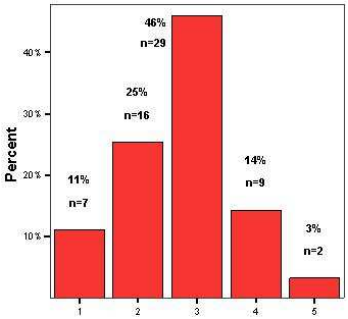


Figure 5.61b : More Knowledge

Fig.	Mean	Median	σ
a	3.03	3.00	.996
b	2.73	3.00	.954
c	2.44	2.00	.980

Table 5.37 : SPD251.022: Response statistics for the *'perceived Depth of Knowledge'*

Figure 5.61 a,b and c: SPD251.022: Response distribution for the *'perceived Depth of Knowledge'*

Figure 5.61 and Table 5.37 indicate that users felt the system was marginally useful in getting access to relevant information or knowledge to help them finish their assignment. It was nearly useful in getting access to **more** information or knowledge. These were seen as positive results. Although ranked higher than expected, it was not useful for **deeper** knowledge. The mean of 2.44 is significantly higher than for the previous studies.

Table 5.38 shows the developmental improvement of the system over the three studies. This is another positive result of the final study.

The last category only had 28/69 open comments (see Appendix F) but some of these indicated that they felt that the system would 'only' give them shallow information and that they had to figure the rest out for themselves (in keeping with DLP and LiC attitude to teaching). Others felt that the delayed answers from the AG removed the need for deeper information from the system. Neither of these were planned outcomes of the final system.

A useful future study may be to determine how users felt about the teaching paradigm of the system that requires them to think/search for solutions not just be given them. Since this paradigm is consistent with that of the LiC, it would be useful to see what attitudes users had to using the system with a LiC who used a different or conflicting teaching style.

Type	SPD251.012			AGV351.021			**SPD251.022**		
	Mean	Median	σ	Mean	Median	σ	**Mean**	**Median**	σ
Information	2.61	2.00	1.13	2.87	3.00	1.02	**3.03**	**3.00**	**0.99**
More Info	2.04	2.00	0.90	2.10	2.00	1.06	**2.73**	**3.00**	**0.95**
Deeper Info	1.57	1.00	0.92	1.57	1.00	0.69	**2.44**	**2.00**	**0.98**

Table 5.38 : SPD251.022: Comparison of rankings to *'perceived Depth of Knowledge'*

The aim of the final study was to evaluate the usefulness of a complex Directed Learning Path. However, the success of the second study due to the improved Domain Knowledge, motivated an increase in the DK for this study as well. Figure 5.62 is a montage of the topic nodes for this study showing an increased complexity and granularity of the DLP questions/responses (top left and right corners) as well as more question/response nodes (bottom left) and with more detailed responses(bottom right).

Although the preparation time for the final study was reduced (the few weeks between semesters), the added DLP complexity and increase in question nodes was accomplished by building on the existing solid base. The results indicate that this preparation was successful.

5.3.5.6. SPD251.022: Other Domain Knowledge

Users had no general preference for other domain knowledge (Figure 5.63 and Quote 5.44). The number of comments was quite large (50/69 = 72%) and the amount of comments was also high (average 10 words per comment). These comments provided good formative feedback for future use of the system.

Based upon this response, the 'other topics' category of Figure 5.58 (page 250) that represents questions that users asked about other units or topics, and the fact that users attempted to use ***Mentor*** after the end of the final study, it can be seen that users wanted more

of ***Mentor***. This could be due to the fact, as one user put it, that it is "*better than nothing*" (Quote 5.49, page 262) or as this study has shown, users do benefit from using the system.

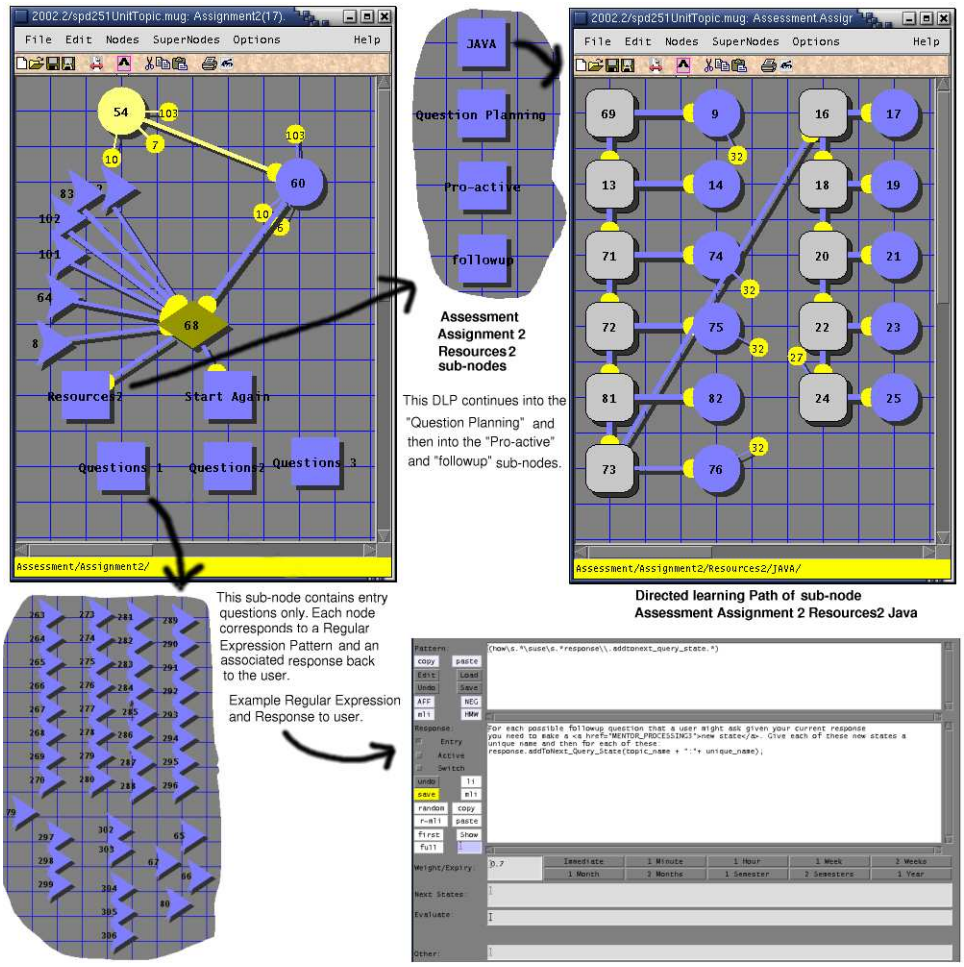


Figure 5.62 : Graph Network representing SPD251.022 Domain Knowledge

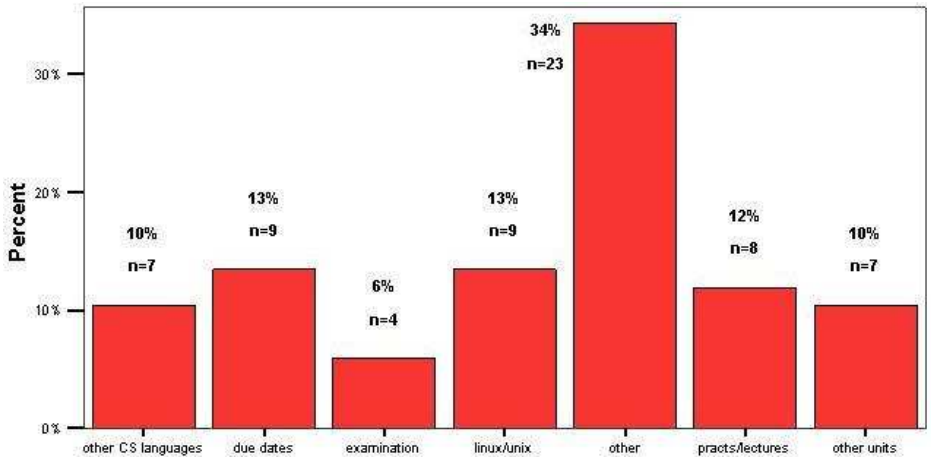


Figure 5.63 : SPD251.022: Recoded response distribution of comments on *'what other Domain Knowledge would be useful'* in the ***Mentor*** System

Other units would be great.. More information on SPD content, further places to go for the information you require.
Any, all areas of info could & probably would be used.
All of the CS units, Java, C++, C, Perl, Shell Scripting! General UNIX stuff as well
CS, is about Java, C++, Perl and XML. If mentor could have a series of useful 'quotes', 'tricks n tips' about this languages, then as a cs student I would find it more useful. How about a 'lookup' or 'advisor' for 'Unix' commands. I am still having problems with certain 'Unix' commands. if the mentor could help in that. CS students would love it.
Current units that I am doing. Basically other infor on other units.
lecturer/tutor availability, as well as contact details.
important dates and places in the school of computing and timetable of tutorial and their location perhaps
Assignment due dates. The meaning of life. This Saturday's Lotto results. Where I put my other sock

Quote 5.44 : SPD251.022: *'other Domain Knowledge'* required comments

5.3.5.7. SPD251.022: General Comments and Improvements

It was noticeable that the users were still making detailed comments (Table 5.39) and that they wanted to be involved in the improvement process. Again the comments indicated that the system was effective and that users appreciated the effort involved (Quote 5.45).

Study	# users	General Comments			Improvements		
		# comments	%	words/ comment	# comments	%	words/ comment
SPD251.012	69	19	27.5%	12.3	17	24.6%	11.5
AGV351.021	37	18	48.6%	22.2	23	62.2%	23.8
SPD251.022	**77**	**28**	**36.4%**	**16.3**	**35**	**45.4%**	**18.1**

Table 5.39 : SPD251.022: Comparison of 'quantity' and 'detail' of user comments from Case Studies

Was generally entertaining and very useful source of information. Was essential for the assignment and I have really enjoyed exploring its function, use and capabilities.
-> very interesting on the first time use... -> get bored after that
It is a good system. When trying to change the mentor settings, things get quite unstable and very slow. Sometimes it was a chore to change the settings.
The system is fine. The chat window is hard to configure. The faces (graphics) are a bit scary
I think that is a very good Idea for the University!
good idea, but should be smarter.
No, but I'm sure there'll be lotsa progress in future.
Mentor System can be good way to learn. But now sometimes it can't understand what the user asking. So computing school students should know more about it. May be from the first year Teaching staff can show the limit of it that what student can ask to the mentoring system. Actually first we should find out where we want to use it. Then we can make huge database about on it and click on it.
Good potential for learning!

Quote 5.45 : SPD251.022: *'General'* comments on the ***Mentor*** System

Of interest to the study is the last two comments in Quote 5.45. Educationally, there is a big difference between effective teaching and effective learning. The ***Mentor*** System can be seen to be effective at teaching the material - it makes information available at any time, in a

useful way and lets users progress at their own pace. The entire system - the Dialogue Manager, the Domain Knowledge, the interface and the support tools - is a good environment for developing and delivering effective teaching material.

But it is the user who determines if it is effective in helping them to learn. Two of the objectives of the system were to resolve the questions:

- Could it be effective in helping students with their units (specifically assignments in these studies)?
- Would students feel that it was beneficial to use it?

That is, the objectives are user oriented rather than lecturer oriented: would the students be helped, would the students feel that it is beneficial? The evaluation questionnaires have always tried to elicit this information from the users. Given the overall ranking of the various aspects of this final study and the comments made, it would appear that the system has met these objectives. The ***Mentor*** System is a good environment for developing learning experiences.

Quote 5.46 shows typical comments about improving the system and they are similar to the second study. These comments will be used as formative feedback for future improvements to the system.

The first three comments in Quote 5.46 are concerned with ‘courtesy’. This may seem strange but it is quite a common need in the Embodied Conversational Agents field. That is, software programs that converse with a user and may have a computer generated form such as a talking head. Some users are starting to ‘engage’ with the ***Mentor*** System and expect it to be more human like. So, these users are expecting that the system display human traits such as courtesy. This is not unrealistic nor hard to add to the system. The system must become affective to improve and become more effective.

As noted before however, as the system improves in various ways, the user expectations increase. It is important for the users to be grounded in their expectations of what is simply a software based mentoring system. It is not human, it cannot replace a lecturer or tutor.

5.3.5.8. SPD251.022: Conclusion

In most aspects of this case study, the ***Mentor*** System was successful. Nearly 70 users evaluated the system, and 87% ranked its benefit as being ‘>=3’. The mean value of the responses was 3.39, as opposed to 2.58 for the first case study and 3.29 for the second. In terms of how effective the system was in helping specifically with their assessment, 88% ranked it as being ‘>=3’, with a mean of 3.54 compared to 2.23 and 3.13 respectively for the first two studies.

The negative aspects of this case study had also decreased, with 42% indicating that there was no least beneficial aspect, and 42% indicating that there was nothing wrong with the system. The negative aspects reported were concerned with adding more functionality to the system so that students could get more from it. However, 17% still wanted the system to better understand their questions. Users still reported on annoying aspects of the system, but these were mainly concerned with poor user education on tailoring the system.

Of importance, students gave significantly more qualitative comments about the system, perhaps as they started to see the system as a useful feature in their learning, or as they started to engage with the system. This last supposition is supported by the fact that users explored the system more, and even started using their native language in conversing with the system.

Overall students felt that the DLP was the most beneficial aspect of this case study. Of interest, the improvements in 'time management' did not reflect this benefit as markedly as the benefit 'overall', with roughly 68% ranking the improvement at '>=3'. Unfortunately, for the helpfulness of the DLP prompts, only 58% ranked it as '>=3', and the mean of the ranking was only 2.74. This was an improvement over the first two case studies but disappointing.

Finally, in terms of information retrieval, the users were satisfied with getting access to relevant knowledge through the system (74% '>=3' and a mean of 3.03), but as expected getting access to more or deeper knowledge was not catered for.

For saying thank you, more possible words could be added (e.g: "thx")
More courteous.
Get rid of the scary face. Sometimes when i try to click on the input box it won't select. Sometimes when i type in there it doesn't register. Sometimes when i press backspace it stuffs up
Fixing some of the links from the pro-active responses, also adding further information could always be useful.
It could be better where the lectures are up to for each unit and give relevant help by assuming that the user might not have knowledge of later chapters.
o- cut and paste input into the MS (or did I miss it). o- easier options o- input & output shown together on main display would make conversation easier to trace instead of seperate screen for user /MS. user: hi. MS: g'day user:how are you? MS: good
(1) A nice feature or command which would put the Mentor to 'sleep' til it is being asked to 'wake-up' => will stop the annoying pop-up. (2)'Tricks and tips' for better programming. A lot of programming books have them and they seem to make an impact
Maybe a few answers on latest news or current university events may be good. Sporting news perhaps. maybe also a detailed weather outlook for the next few days maybe useful.
It might be good if the Mentor System can be accessed from anywhere (e.g. using Web-based) and catch the possibility error that may occurs if the user type in a rubbish questions
Would be nice if we could scroll through history with arrow keys, i.e, to enter previous , ^<-
A console based version that could be acessed via telnet would be very useful, so I could use it to study at home.

Quote 5.46 : SPD251.022: *'Improvement'* comments for the ***Mentor*** System

5.3.6. 'Response Time' Evaluation

In this study, users were asked via a 5 point Likert scale: *'How would you rate the response time of the **Mentor** System?' (i.e. the time it took between you asking a question and getting an initial response)* (1=>Very unsatisfactory, 5=>very satisfactory). Users could also add open comments.

This was seen as a way to determine if the implemented system was functioning in a reasonable and useful manner regarding interaction and it was hoped that it would in part, support sub-hypothesis one.

Assuming a value of '3' or above is a positive indication of satisfactory response time, Figure 5.64 and Table 5.40 show that users overwhelmingly felt that the response time was satisfactory, especially given the standard deviation value.

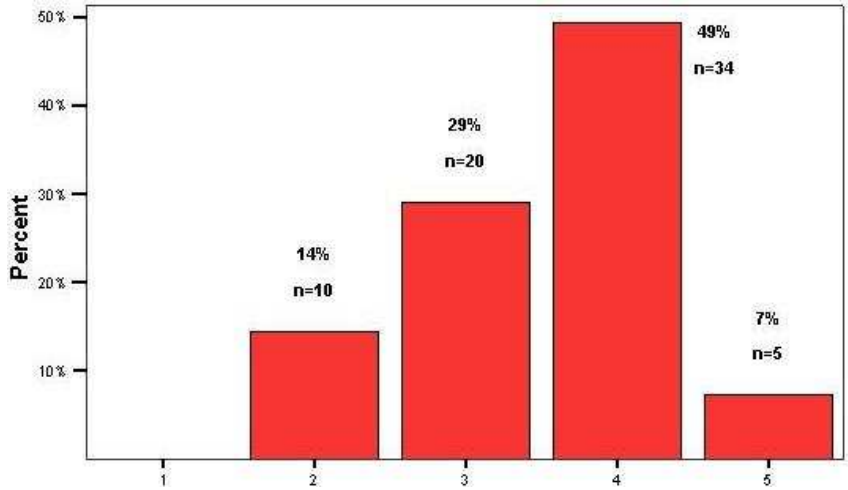


Figure 5.64 : SPD251.022: Response distribution on a 5 point Likert Scale to the question: *'How would you rate the response time of the **Mentor** System?'*

# users	Mean	Median	σ
69	3.49	4.00	.834

Table 5.40 : SPD251.022: Response statistics to the question: *'How would you rate the response time of the **Mentor** System?'*

Depended upon the time of use - when a lot of users were using the system it ran VERY SLOWLY sometimes, to the point where it became unusable.
sometimes a little retarded.
A tad slow at times. rare though.
It is that bad, Sometimes it makes me wonder which question it was answering.
Mainly great.
Sometimes it was quick and at times a bit slow but at an acceptable slowness.
It was varying, but I guess more server power = more money :?) . But the times were acceptable.

Quote 5.47 : SPD251.022: *'Response Time'* comments

The researcher felt that the system had a good response time at most times but towards the due date for the assignment, 30+ users were using the ***Mentor*** System at the same time in a very intensive manner. This would correspond to approximately 2 tutorials worth of users.

The ***Mentor*** server at that time was a Silicon Graphics (SGI) O2 with 256 megabytes of main memory running at 180 megahertz (roughly comparable to a PII 200 megahertz machine) using SGI's Java 1.1.3 Just-in-Time (jit) translation of Java byte-code into native code. The current production machine is a PIV running at 2.0 gigahertz or more with 1 gigabyte of main memory with Sun's Java jit or Hotspot Virtual Machine. It is estimated that this configuration should be able to cater for 2 to 3 times as many users with the same or lower response time.

5.3.7. 'Ease of Programming the *Mentor* System' Evaluation

In this study, users were asked via a 5 point Likert scale: *'How would you rate the ease of use in adding Domain Knowledge by* **programming** *the* ***Mentor*** *System?'* (1=>Very difficult, 5=>very easy). Users could also add open comments.

This was seen as a way to determine if the system could be programmed/extended easily by a relatively naïve user and it was hoped that it would in part, support sub-hypothesis one.

Assuming a value of 3 or above is a positive indication of ease of adding domain knowledge, Figure 5.65 and Table 5.41 show that users overwhelmingly felt that it was easy to do this, especially given the standard deviation value.

As with any complex system, it was difficult at first for users to know how to build new topics (see first quote group in Quote 5.48). Once they understood the nomenclature, the modules and the data flow across the client-server communication channel, they then realised that the knowledge acquisition and the programming of the state-based question-response mechanism was quite time consuming and tedious (second group of quotes).

Adding a new topic to increase the knowledge of the system was straightforward but tedious: copy an old topic, change the relevant module names, then add the patterns and responses. This method was not suitable for any large-scale DK development, especially so when considering the production of information about units.

Also, mentoring is not just having knowledge about a particular topic - it is concerned with guidance based upon expertise, timely reminders, the tailoring of a path through the obstacles to help the mentee reach the final goal. An astute user (last quote) realised that a better solution to the knowledge capture and representation problem was perhaps to use a graphical user interface. This is exactly the mechanism that the system developer used. See Section on Topic Network Builder on page 176 for more information on the mechanism used to create the DK questions and answers as well as the Directed Learning Paths.

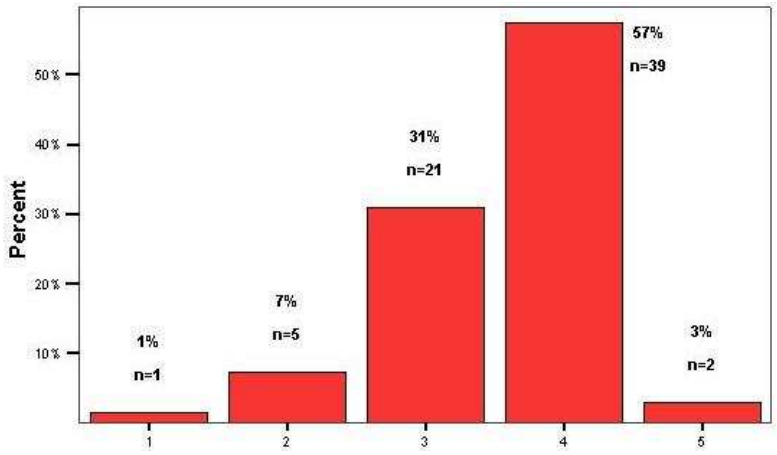


Figure 5.65 : SPD251.022: Response distribution on a 5 point Likert Scale to the question: *'How would you rate the ease of use in programming the **Mentor** System?'*

# users	Mean	Median	σ
68	3.53	4.00	.743

Table 5.41 : SPD251.022: Response statistics to the question: *'How would you rate the ease of use in programming the **Mentor** System?'*

Easy, only the REs were harder to grasp because it was difficult to accomodate all types of responses.
Adding follow up states seemed complex at first.
Once you understand the topic files, it was quite easy to get things running correctly
Once I got the feel for how the system worked Java wise(in the code) it was easy enough to implement my code, even though it was complex.
Had a bit of a tedious learning curve
adding knowledge for the followup questions/input was a bit tedious. Has to be a better way to make the system more intelligent without have too much tedium.
I would think that some kind of database tool for constructing networks of responses/states/inputs would be better than straight programming. Perhaps this tool could generate java source code, compile it and include it in the MS's knowledge. I generally found programming the state-by-state behaviour to be extremely complex and interwoven.

Quote 5.48 : SPD251.022: *'Ease of Programming'* comments

5.3.8. 'Availability' Evaluation

In the final study, users were asked to answer the following:

Assuming that the Mentor System was given appropriate knowledge for other future units in your course, would you rather it was available or not available for helping you in those units?

□ *Available* □ *Not Available* □ *Don't Care*

They could also make open comments on this as well. Figure 5.66 shows the response distribution to this. Note that not one user said that 'they did not want it', although there were 10% missing values. Using the values specified and taking into account the accompanying comments that the users made, an '*only if*' and '*could not hurt*' category were added to the Availability responses. The '*only if*' category was used if the person indicated 'yes' they wanted it available but put some kind of condition on it being available (such as 'must get better'). The '*could not hurt*' category was used if the person said to make it

available but indicated that it could not hurt to do so. Figure 5.65 shows this recoded response.

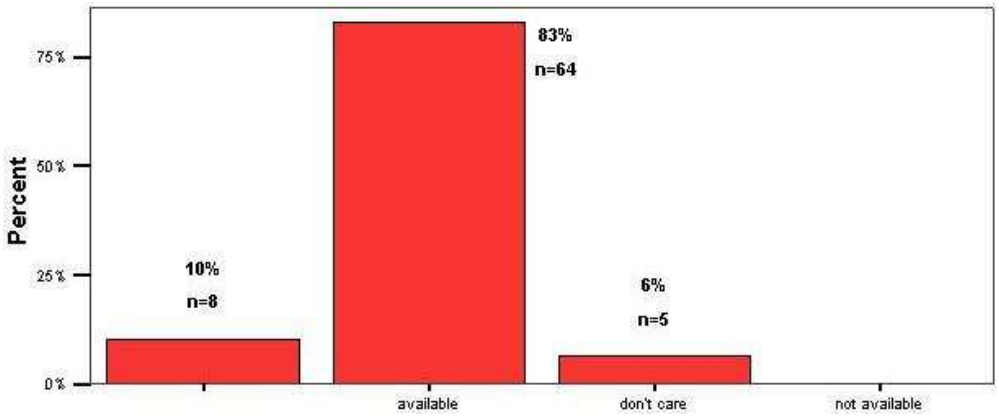


Figure 5.66 : SPD251.022: Original response distribution to the question: *'Would you rather the **Mentor** System was available or not in the future for other units?'*

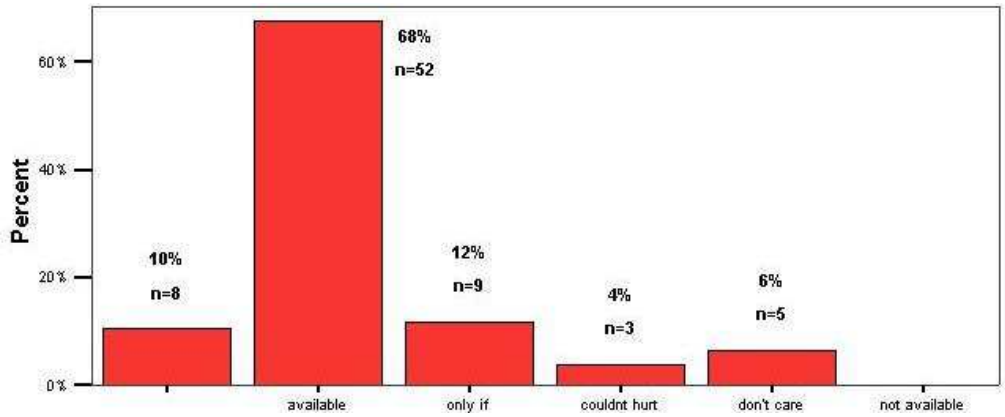


Figure 5.67 : SPD251.022: Recoded response distribution to the question: *'Would you rather the **Mentor** System was available or not in the future for other units?'*

More than two thirds of respondents wanted it *unconditionally*. This is supported by the fact that in the semester subsequent to the final study, users attempted to access the system for other units. This was seen as one of the more positive results of the research.

The entire user comments can be found in Appendix F, with representative comments shown in Quote 5.49.

The first group of comments is typical of the entire study and are relevant in the context of explaining the user's response to the Availability question.

The second group is typical of the '*only if*' category. It is surprising that a second year Computer Science student had not understood by the end of the study that the lecturer had approved because the lecturer provide the domain knowledge.

The third group is representative of the '*couldn't hurt*' category. The degree referred to is considered 'taxing'. The comments shown in the fourth group represent the various ways in which users felt that the system could help - on demand information, out-of-office hours help, etc. However, the last quote is disturbing in that this user believes that they could bypass the domain knowledge expert and only use the **Mentor** System.

This aspect, and the concerns of the fifth group, are shared by the researcher. The system should not be seen as a primary cost-cutting replacement for access to lecturers/tutors. It is simply a tool that can provide secondary or complementary information to help users (see final quote group).

It was very helpful for the assignment but somewhat less helpful for general info on the unit.
Starting an assignment is the most difficult phase. The mentor system made it really easy.
Provided that the Mentor System is provided with ample information, this provides a very open source for questions students have that may not be able to be answered by other sources.
as long as it give useful info and helpful for the unit and the tutor/lecturer approve it.
Every little bit of help counts. Even if it wasn't useful for that unit, we could simply ignore it. It should be available even with the slight chance it could be helpful.
We need all the help we can get in succeeding in our degrees in a very taxing school.
better than nothing The MS will be useful for getting on-demand info at random instead of browsing through long site-maps for the info.
Yes, it may help during off-office hours, hints for exercise and much more.
Because a lot of new students need to know more about the units they take. And if the Mentor System exist, they can ask to the Mentor whenever they wants. Don't have to ask their tutor or lecturer
I don't want lecturers/tutors say go and have a look at in Mentor system/ why didn't you ask the mentor first?
It would prevent students from approaching lecturers about basic topics.
But then it would be better to get help from the lecturer concerned than to ask the Mentor System because your questions may be too complex for the Mentor System
Any extra help in the form of extra study tools is always good.

Quote 5.49 : SPD251.022: *'Availability'* comments

5.3.9. Summary of Case Studies

In each study, users were asked via a 5 point Likert scale: '*Do you think it was beneficial or helpful to use the Mentor System?*' (1=> not at all, 5=>very beneficial). Table 5.42 shows the summarised results. Assuming a value of '3' or above is a positive indication, it can be seen that **the second and third studies were marginally beneficial** (given the standard deviation value). This aspect also improved over the three case studies.

Study	# responses	Mean	Median	σ
SPD251.012	38	2.58	3.00	0.92
AGV351.021	31	3.29	3.00	1.16
SPD251.022	69	3.39	3.00	0.83

Table 5.42 : *'Was it beneficial or helpful?'*

Table 5.43 shows the summarised results for the question: '*How effective or useful was it for this purpose?*'. It can be seen that **the second and third studies were marginally effective** (given the standard deviation value). Also, the system became more effective when problems with the client interface and lack of domain knowledge were addressed.

Study	# responses	Mean	Median	σ
SPD251.012	35	2.23	2.00	0.88
AGV351.021	31	3.13	3.00	0.88
SPD251.022	74	3.54	4.00	0.88

Table 5.43 : '*How effective was it*'

Table 5.44 shows the summarised results for the question: '*Do you think that help from the **Mentor** System improved the overall quality of your assignment*'. It can be seen that **only the third study was considered marginally helpful in this respect**.

Study	# responses	Mean	Median	σ
SPD251.012	35	1.93	2.00	1.05
AGV351.021	31	2.29	2.00	1.10
SPD251.022	74	3.10	3.00	1.07

Table 5.44 : '*perceived improvement to user's assignment Overall*'

Table 5.45 shows the summarised results for the question: '*Was the **Mentor** System useful in getting access to relevant information or knowledge?*'. It can be seen that **only the third study was considered marginally effective**.

Study	# responses	Mean	Median	σ
SPD251.012	35	2.61	2.00	1.13
AGV351.021	31	2.87	3.00	1.02
SPD251.022	74	3.03	3.00	0.99

Table 5.45 : '*useful in getting access to relevant information or knowledge?*'

Therefore it can be seen that the final **Mentor** System **was effective** and **did benefit** the users. This is also supported by qualitative feedback from the evaluations.

In the third case study, users were asked if they would rather have the system available or not for future units and allowed to make a free text comment. These data were recoded into the categories shown in Figure 5.68. Overwhelmingly, students wanted it.

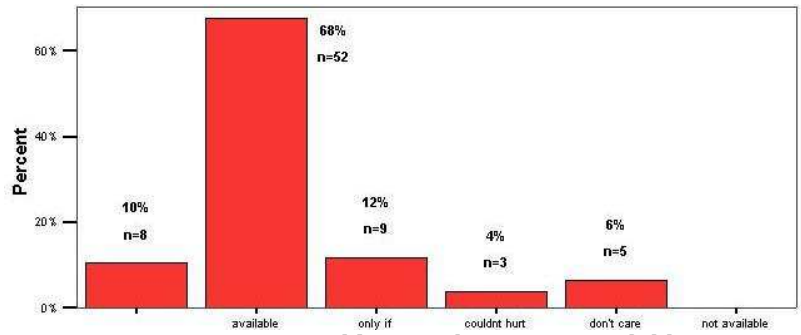


Figure 5.68 : '*Would you rather it was available*'

To test support for the learning paradigms, users were asked questions about the Directed Learning Path and the Authoritative Guidance paradigms. For the DLP, Table 5.46 shows the summarised results for the question: *'Did you find the prompts helped with the assignment?*'. It can be seen that even though a ranking of 2.74 is respectable for a software only based system, **students did not consider the DLP helpful in this respect**. Typical qualitative evaluation on the '*most beneficial aspect*' of the system did however indicate that students found the DLP to be very helpful: This was THE best thing about the Mentor System. I couldn't have done without it. Table 5.44 supports this conjecture, since the majority of help that the system gave to students in the last study was from the DLP.

Study	# responses	Mean	Median	σ
SPD251.012	35	1.84	1.00	1.14
AGV351.021	31	2.42	2.00	1.23
SPD251.022	74	2.74	3.00	1.17

Table 5.46 : *'DLP prompts helped with the assignment?'*

Similarly for the AG, Table 5.47 shows the summarised results for the question: *'Did you find the answers helped with the assignment?*'. The specific values are not that important since the '*answers*' came from the Lecturer in Charge if the system could not answer immediately. Typical qualitative feedback from the evaluation indicated that the AG paradigm was useful: This is very good!

Study	# responses	Mean	Median	σ
SPD251.012	35	2.00	2.00	0.94
AGV351.021	31	2.50	2.50	0.95
SPD251.022	64	2.55	3.00	1.29

Table 5.47 : *'AG answers helped with the assignment?'*

Of more importance are the values of Table 5.48 which compares the usefulness of delayed answers direct from the Lecturer in Charge against the immediate answers from the system. The Lecturer in Charge was the researcher. These two questions were included in the final study to try to determine whether the help given by the ***Mentor*** System was better or worse than help from for example a tutor or a Duty programmer. As can be seen, there is a flaw in either the evaluation question, the assumed linearity of the ranking, or in the participant's understanding of the question.

Answer	# responses	Mean	Median	σ
delayed	60	3.13	3.00	1.13
immediate	69	3.46	4.00	0.95

Table 5.48 : *'SPD251.022 AG answers usefulness?'*

This result may explain the discrepancy between the qualitative feedback and the quantitative rankings of the effectiveness/usefulness of help provided by the system with respect to the DLP and AG paradigms. Similarly there is a discrepancy between the ranking of '*usefulness*' of a delayed answer (3.13) and the ranking of how that answer '*helped*' (2.55), although the answers sent by the LiC were specifically aimed at helping the student with their assignment.

Chapter Four showed that the system architecture was able to support the DLP and AG paradigms. However, given the conflicting qualitative and quantitative feedback, the educational paradigms supported by the ***Mentor*** System are **not conclusively proved to be effective in helping students**.

The system seems to be effective and beneficial but it is far from perfect as shown by some of the qualitative data given in Quote 5.50. There is obviously a need for better understanding of user questions. Given the large number of users who do not have English as a first language, the system needs to address the lack of distinction between plural and singular in many Asian languages and also the non-use of articles like 'a' or 'the'. Given that nearly 50% of the users did **not** have English as their first language, the results may be seen as better than expected.

One can take comfort from the last two examples of Quote 5.50 in that some students did realise that the system was just a computer program not omniscient!

Sometimes if I asked a more specific question it didn't know what I was talking about.
can't seem to understand questions properly.
It didn't know everything, but thats impossible anyway.
Small knowledge base? But can't really fault it much since it's not human.

Quote 5.50 : Quantitative evaluation responses?

The problem where users started conversing with the system using their native language, supports the opinion that Natural Language systems should not pretend to be smarter than they are. They must be able to understand what they 'dish out'.

It was also important to determine if the system could be scaled and applied to a real-world environment. The third case study typically had 20+ users at any one time and peaked at 34. The users were asked: '*How would you rate the response time of Mentor?*' (1=> very unsatisfactory, 5=>very satisfactory). The response time is how long the system takes to answer a question. Table 5.49 indicates that students seemed happy with the response time although qualitative responses indicated that it got slow '*when the assg due date is close*'.

# users	Mean	Median	σ
69	3.49	4.00	0.834

Table 5.49 : Response Time of the ***Mentor*** System

Similarly, it was important to know if 'normal' users could extend the system through the development of local topics. Users were asked: '*How would you rate the ease of use in adding Domain Knowledge by programming the Mentor System?*' (1=> very difficult, 5=>very easy). Table 5.50 indicates that students found it easy to extend the system.

# users	Mean	Median	σ
68	3.53	4.00	0.743

Table 5.50 : Ease of Programming of the ***Mentor*** System

The system response time was good with a non-trivial number of users, and the system was extendable by ‘normal’ users so that they could program their own Domain Knowledge. This supports the aim that **the system could operate as a real-world tool** not just as an academic test-bed. The system helped real students in a real environment.

5.4. Limitations

The evaluation has also shown some limitations:

(1) The number of participants, whilst sufficient for the analysis of the Case Studies, is not large enough for any statistically useful breakdown of the participants into sub-groups. For example, sub-grouping to correlate enrolled degree vs. attitude, or age vs. attitude would produce sub-group numbers that did not yield results of any statistical significance. Further studies with more participants may enable useful correlations to be determined.

Specifically, the percentage males vs. females in the computing discipline does not allow the results to be generalised to specify female ‘benefits’ or ‘effectiveness’ in other disciplines. Further studies need to be carried out to see if different approaches should be adopted for different genders.

However, it is believed that of more importance is the underlying mechanism that enables this effectiveness for any specific user - the problem is not gender specific, but learner specific. A good human mentor or tutor will change their style when dealing with different users, and this is often based upon the user’s knowledge level **and** their personality. The system currently has facilities for imposing a personality style on the responses but further work has to be done to determine good mentoring personalities, and how these would adapt to help users to learn. See Xiao *et al* (2005a) and Xiao (2006a) for some initial research in using personalities in the ***Mentor*** System.

(2) As mentioned in Chapter Four, page 188, the system currently assumes only one query per input: input that contains two or more questions is not catered for. It can be seen from some user input that students felt that the system could understand long, convoluted sentences.

(3) The Unit Controller and Knowledge Domain builder for the Case Study units was the researcher. The system has not been evaluated by other lecturers, although students are keen for them to do so. This may not affect the outcomes of the research but has an impact on the balance of effort vs. return for another lecturer developing Domain Knowledge and using the system.

5.5. Further research

This chapter has highlighted some issues that need to be addressed in the future.

(1) In the computing discipline at the university where this research was carried out, a significant number of students do not have English as their first language - 57%, 33% and 47% for the last three formal studies. This poses a serious problem for any

DM system that assumes correct spelling and grammar. Future research should look at the use of synonyms and typical spelling errors to improve the patterns used to match the user input. Systems such as WordNet (Miller, Beckwith *et al* 1990a; Miller, Fellbaum & Miller 1997a) could be used for this. See Tan (2005a) for some initial results.

(2) English is evolving, and more users were entering SMS-type queries. Future research should look at the rules for SMS abbreviations so as to improve the patterns used to match the user input. See Tan (2005a) for some initial results.

(3) The area of pro-active Responses suffered from some users being happy with certain responses (for example help information relevant to their study) but not to others (for example, asking if they want to hear a joke when they are rushing to meet a deadline). As mentioned in Chapter Four, a facility for users to tailor / suppress topics that produce pro-active Responses was seen as a suitable solution to this.

(4) Better processing of user input was needed. That is, it must understand the user's input. See discussion on this in Chapter Four, page 188.
Users also wanted more Domain Knowledge - help in different units.

(5) Users wanted an anthropomorphic interface: see discussion on this in Chapter Four, page 188.

(6) A better understanding of the user's immediate attitude to interacting with the system would be beneficial. If the system could detect through multimodal input that the user was happy, annoyed, angry, or bored with the Responses, it could modify its behaviour accordingly to address the problem, especially if the system also had a personality.

5.6. Conclusion

This Chapter described the results of the users' evaluation of their use of the ***Mentor*** System.

The design and development of the ***Mentor*** System had 3 main objectives:

```
1.  Could the system be developed in Java as a real-world tool
    (not just an academic test-bed)?
2.  Could it be effective in helping students with their units
    (specifically assignments in these studies)?
3.  Would students feel that it was beneficial to use it?
```

Chapter Four addressed the first question posed above, with this Chapter supplying supporting evidence in terms of extensibility, 'Response Time', and 'Ease of Programming'. This Chapter also addressed the questions posed in the last two objectives. The system **was** effective in helping students with their units, and students **did** feel that it was beneficial to use it.

The ***Mentor*** System was designed and implemented to test the sub-hypotheses:

> A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.
>
> The ***Mentor*** System can support an effective learning paradigm.
>
> The ***Mentor*** System will benefit students.

The first sub-hypothesis was tested and shown to be valid in Chapter Four, with this Chapter again supplying supporting evidence in terms of extensibility, 'Response Time', and 'Ease of Programming'.

The second sub-hypothesis was tested. The ***Mentor*** System supported the two educational paradigms of Directed Learning Support and Authoritative Guidance, but it was inconclusive as to whether these were effective in helping the students in their learning.

The third sub-hypothesis was tested and shown to be valid by both qualitative and quantitative evaluation by users.

The conclusions and the implications of this research are described in the next chapter.

Each problem that I solved became a rule
which served afterwards to solve other problems.
Rene Descartes
Discours de la Methode

Then it's me and my machine
For the rest of the morning
For the rest of the afternoon
And the rest of my life
James Taylor
Millworker

Conclusions and Implications

6.1. Introduction

Bloom (1984a) indicated that two standard deviations difference in student benefit is possible in relation to the results of one-on-one tutoring vs. one-on-many. Fletcher (2003a) indicated that this could represent a lifting of the performance of a typical student from the mid-level 50th percentile to the 98th!

This thesis has successfully argued that a software based mentoring system could be designed and implemented that would support a mentoring educational paradigm and that it would be beneficial to the students using it. The ***Mentor*** users in the Case Studies reported that its one-on-one help as being effective and beneficial, and the Directed Learning Path being especially useful in helping students with their assignments.

Due to privacy concerns, it was not possible in this study to determine whether users **actually** benefited in their learning and retention of the course content. Future Case Studies using the ***Mentor*** System should be conducted to measure quantitative benefits to student learning as predicted by Bloom and by Fletcher.

The following sections discuss the conclusions concerning the research objectives and hypotheses, the conclusions about the research problem in general, the implication for further use of software based systems in education, the limitations of the research and finally, areas for future research.

6.2. Conclusions about each research question or hypothesis

The design and development of the ***Mentor*** System had 3 main objectives:

1. Could the system be developed in Java as a real-world tool (not just an academic test-bed)?
2. Could it be effective in helping students with their units (specifically assignments in these studies)?
3. Would students feel that it was beneficial to use it?

The conclusions in Chapter Four affirmed the first objective question, and Chapter Five supplied supporting evidence in terms of extensibility, 'Response Time', and 'Ease of Programming'. The system was implemented using Java, and was robust and stable even when more than 60 users were connected. The Dialogue Manager Server was extensible, the Knowledge Base was designed around active Java Topics, and the entire system could be, and was used in dynamic applications such as a talking, gesturing Virtual Human.

The conclusions in Chapter Five also affirmed the questions posed in the last two objectives. The system **was** effective in helping students with their units, and students **did** feel that it was beneficial to use ***Mentor***.

Using these objectives as a base, the ***Mentor*** System was designed and implemented to test the hypothesis:

A software-based *Mentor* System system can benefit University students.

This was expressed as 3 sub-hypotheses:

A software-based ***Mentor*** System can be designed and implemented using a suite of extensible, cooperating, network based computer programs and intelligent user interfaces.

The ***Mentor*** System can support an effective learning paradigm.

The ***Mentor*** System will benefit students.

The first sub-hypothesis was tested and shown to be valid in Chapter Four. The user evaluation of Chapter Five supplied supporting evidence in terms of extensibility, 'Response Time', and 'Ease of Programming'.

Chapters Four and Five also reported on the testing of the second sub-hypothesis. The ***Mentor*** System and peripheral applications such as the Topic Network Builder effectively supported the two educational paradigms of Directed Learning Support and Authoritative Guidance, but it was inconclusive as to whether these were effective in helping the students in their learning.

The third sub-hypothesis was tested and shown to be valid in Chapter Five through the qualitative and quantitative evaluation by users.

6.3. Conclusions about the research problem

The final reflection of the Action Research cycle....

Broadly, the research problem was "can this system be built and be of benefit to the users." The answer to this has been shown to be a definite, but qualified "yes".

The significant contributions of this research to the areas of Dialogue Management and Tutorial Dialogue systems are:

(1) the design, development, implementation and evaluation of a Dialogue Manager that uses **regular expression pattern matching** that is adequate for the mentoring task. The DM process is supported by the Knowledge Base being extensible Java code rather than just plain data. The Case Study qualitative results in Chapter Five showed the limitations of this simple technique, but also its power.
Therefore a researcher could use the results of this study to build a better mentoring system, knowing that pattern matching will suffice for the parsing of the user input. This study has also shown the limitations of the technique. For example, the DM is sufficient for question-answer type interactions but not discriminating enough for explanation-analysis with the user. It also suffers the same limitations as other forms of dialogue management when conversing with users who do not have a good grasp of English. The study also indicated where future research exploration should occur so as to remove or reduce these limitations.

(2) the design, development, implementation and evaluation of a large scale, extensible, multi-threaded, client-server **Java** application that can process a significant number of users in real time. There are significant benefits in using Java for the coding of a computer assisted learning system, and this study has shown that Java is capable of scaling to a real world environment.
Therefore a researcher could use the results of this study to build a new large scale Java application, confident that it could scale easily.

(3) the design, development, implementation and evaluation of an **extensible** Dialogue Manager that allows a user to add their own knowledge and management tasks through the use of easily coded Java Topics. An extensible DM provides a better research platform for dialogue management research development as has been shown by the use of the generic DM in other non-mentoring applications such as those developed for Talking Head systems.
Therefore a researcher could use the results of this study to design a new DM system, knowing the benefits and limitations of using this extensible approach.

(4) an **analysis** of three Case Studies that detailed qualitative / quantitative evaluation data of the usage of the system by an appropriate sized population of students. This analysis has shown that the system did benefit the students who used it, and in what areas it excelled and in what areas it was deficient.
Therefore a researcher could use this detailed analysis to improve relevant areas in both the underlying Java system, as well as in the educational paradigms that can be supported.

(5) a description and analysis of the use of the **Virtual Human Markup Language** in the marking up of the Domain Knowledge so as to ensure a consistent rendering of the response from the ***Mentor*** System regardless of the output display (plain text, graphical user interface, talking head). VHML has been shown to be very successful in conveying hyper- and multi-media information as well as emotion/gestures to users. The Dialogue Manager sub-system of ***Mentor*** has been used successfully in many other DM applications that require emotional responses.

VHML ensures that future research using ***Mentor*** can adapt to changing rendering needs, and could for example, support the emotional and gestural needs of a Virtual Tutor/Lecturer.

Although not discussed in this thesis, analysis of user responses has motivated the development of a **personality** module for the ***Mentor*** kernel (Xiao *et al* 2005a). This personality module hooks into the Dialogue Management "user input analysis" pipeline at various stages. The module can alter the returned response at any of these stages, either by minimally modifying the response, changing the response entirely, or adding emotional/gestural VHML tags to the response. In keeping with the extensible nature of the DM system, a user can load their own personality module at client connection time and this will over-ride the default server-side personality.

(6) a description and analysis of two implemented **paradigms** of learning. The usefulness of the Directed Learning Path and the Authoritative Guidance paradigms were discussed, as well as issues arising from their implementation using the ***Mentor*** System. Other possible paradigm candidates for implementation were discussed.

(7) an analysis of the **issues** arising from the use of a software based mentoring system in an educational environment. For example:

- students accept even trivial help: students felt that even if the response from ***Mentor*** was not exactly what they were asking for, it was useful in helping them.
- holding hands vs. nagging: there was a fine line between ***Mentor*** being seen as caring by reminding students about approaching deadlines, and nagging them all the time to start work.
- students expect it to be too smart: if ***Mentor*** greeted students with phrases from their native language, the students expected to be able to converse with ***Mentor*** in their native language. Students also expected that the complexity of their input to the system should be equivalent to the complexity of the response received from the system.
- English is evolving: students started using SMS as part of their dialogue.
- students over-rated the help of the system: students thought that the immediate response from ***Mentor*** was more useful than a delayed response from the Lecturer in Charge of the unit. This was true even from students who were programming their own Domain Knowledge into the system and who should have known its limitations.

(8) **Information dissemination** through the research results published in various books and at conferences (see Preface Chapter).

Finally, as has been previously stated: the system must work in the researcher's learning environment, not just as an academic exercise. As such, a significant pragmatic contribution that this research has made has been in **helping students**. As can be seen in Chapter 5, student responses to the system have, to a very large extent, been positive and students were grateful for the benefits that ***Mentor*** provided.

6.4. Implications for theory

The parent theories for this research are from the Software Agents / Intelligent User Interfaces and the Educational Technology areas. The specific research problem areas are in Dialogue Management and Tutorial Dialogue Systems. Underlying this research is the enabling technology of an Object Oriented Java based client-server system.

6.4.1. Java

In 1996, in the initial phase of this mentoring research, Java code execution was slow and the language was in its infancy. This thesis has shown that Java is quite capable of meeting the requirements of a real world software based mentoring system and providing extensibility that will enable the system to grow to meet future needs and functionality. Java provided a suitable environment for design and development and caused few implementation problems. This research has shown that Java is a suitable language for future Educational Technology development even in medium scale applications.

6.4.2. Dialogue Management and Tutorial Dialogue Systems

The simple Regular Expression Pattern Matching Dialogue Manager performed well in the educational domain of question-answer mentoring. This research has shown the benefits of this approach, along with its limitations.

The development of the Domain Knowledge was relatively easy yet very effective due to the knowledge being embodied in dynamic Java class Topics. A further benefit of this approach was that users could easily create their own Domain Knowledge and have it loaded into the server when they connected to it. One drawback in the use of Java Topics was that it restricted this effectiveness to educationalists with programming knowledge, although this research has shown that this knowledge need not be at an expert level. Suitable support applications also hid some of this needed Java programming knowledge but with a subsequent loss of functionality in the resulting Topic.

A client-server approach to the mentoring has been beneficial in that many of the initial limitations / delimitations of the proposed system were relaxed or removed by using this technique. Also, the Dialogue Management system was developed to allow for extensibility and flexibility in application, and has been shown to be very effective in other DM areas, especially those requiring emotion/gestural responses.

The Tutorial Dialogue aspect of the ***Mentor*** System was enabled by the support for educational paradigms on top of the Dialogue Management. Two specific paradigms were implemented and discussed, and others were investigated. For example, it was seen that a 'Stupid Mentor' paradigm would not work well: pattern matching is good at determining

short user requests, but may not be suited for understanding complex descriptions of a user's problems. However, a future use of the system could be to simply ask revision questions and comment on the correctness or incorrectness of the user's answers. Research in this area could also enable greater understanding in the further development of the Directed Learning Path support so that better "hurdle" strategies could be devised. The underlying framework can support many different paradigms for learning. Of importance, the pro-active nature of ***Mentor*** meant that it was **not** a passive learning partner.

This research has shown that a developer of a new Tutorial Dialogue system can use the techniques developed or implemented in this study, aware of their advantages and disadvantages as well as their limitations.

6.4.3. A software based mentoring system

The literature from the parent and research theories areas identified unexplored or novel research issues in the Dialogue Management and Tutorial Dialogue Systems area. The literature also indicated that any developed software based mentoring system should have the following attributes / functionality:

- Relevant teaching and learning strategy (e.g. educational paradigms of Core, Moore, Zinn & Wiemer-Hastings (2000a), Eugenio (2001a), Rosé, Moore *et al* (2001a), and state based dialogues of Mauldin (1994a), Whalen & Patrick (1990a)). It should help the student learn not just answer questions.

  ✓ The ***Mentor*** System's Directed Learning Path does '*lead students through a line of reasoning point by point.*': "This was THE best thing about the Mentor System. I couldn't have done without it."

  Mentor's state based dialogue framework allows for dialogue, not just answers.

- Proactiveness (e.g. through hints such as in Zhou *et al* (1999a), or the mixed initiative of Keim *et al* (1997a), or the '*coaching*' of Bratt *et al* (2005a)). The system should not just provide information but should also actively guide the user, through the asking of questions or in the offering of advice.

  ✓ The ***Mentor*** System supports the 'teacher as guide' (Vygotsky 1978a) concept by allowing the system to take the initiative in starting a dialogue and then allowing "*the cultivation of individual initiative*" (R Schank & Cleary 1994a) as the student subsequently explores the problem domain.

  In being proactive, ***Mentor*** will *'Suggest rather than act'* (Ehlert 2003a).

- Effectiveness in helping users not only in learning but also in other daily tasks and problems. The information given to users must be relevant and also effective in helping them solve their problems.

  ✓ The Case Studies indicated quantitatively and qualitatively that students found the system to be beneficial: 'Good potential for learning!', 'I really like the learning path it had to guide me through each of the assignments. It was a big help'.

- Expertise (e.g. the human guidance of Poleretzy *et al* (1999a), the '*perceived intelligence*' of Isbister (1995a)). The system should have relevant domain knowledge

and this expertise should also be seen as coming from an authoritative source.

✓ The ***Mentor*** System's Domain Knowledge is built by the Lecturer to handle specific bounded-domain queries, and the Authoritative Guidance paradigm allows for '*human interaction ... so as to ensure a quality interaction experience* ' (Poleretzy *et al* 1999a).

"Provided that the Mentor System is provided with ample information, this provides a very open source for questions students have that may not be able to be answered by other sources."

- Adaptability. The system must be able to adapt its interaction behaviour for different users. It should also be able to adapt the level and amount of interaction.

  ✓ ***Mentor***'s Domain Knowledge is dynamic Java code rather than static data, and hence can adapt to an individual user's learning needs. The Java code can use database information about a user's performance for various assessment components to prompt them into revising specific areas, and tailor the interaction for that area.

  The hurdles of the Directed Learning Path can adapt their response to reflect the current level of understanding that the user has, or to address a misconception that the user has.

  The personality sub-system can adapt the ***Mentor*** System's response so as to better match the required student-tutor relationship.

- Extensibility (e.g. rendering in different forms - the "*animated pedagogical agents*" of C Elliott *et al* (1997a), André *et al* (1997a), and L Johnson *et al* (1998a), or the Research Methods Tutor of Wiemer-Hastings *et al* (2005a)).

  ✓ The user can easily extend the ***Mentor*** System so as to augment its facilities. This can be done by them creating new Domain Knowledge that is loaded at connect time, new delivery personalities, and even different interfaces to the Dialogue Manager.

  VHML allows for the response to be rendered as text, sent to a Web page or even delivered by a Virtual Human.

  The ***Mentor*** System can handle a 'real world' numbers of users.

- Systemic in its data uptake as well as in its usage (e.g. Norrie & Gaines (1995a), or the increasing ITS use of the Semantic Web in AIED2005 (2005a)).

  ✓ The Case Studies indicated that many students felt that the ***Mentor*** System did *provide open access to a learning environment for all people, in all places at all times.*

  The Topic Network Builder support application allows for the building of Domain Knowledge by anyone, currently at the expense of learning Regular Expression nomenclature.

6.5. Limitations

The developed system has some limitations, although several of the initial limitations / delimitations outlined in Chapter Three have been relaxed or catered for by the developed system. The client-server architecture meant that the system was available world-wide, and more importantly, from a student's home computer running most operating systems. The system was not planned to accept speech input. However, the designed-in extensibility of the system catered for multimodal input such as the emotion of the user, and will enable this to

be used when the technology matures to be able to provide input from audible or visual cues from the user. Multimodal output was also available via the use of VHML.

The system accommodated a significant number of connections or users at any one time with good response time. However, the researcher is currently the Lecturer in Charge of a unit with over 350 students. It is possible that one third of these could be connected to the system at any one time in some future scenario. The implementation language - Java - may limit the number of connected users, as well as the response time to these users. The architecture of the system could also limit the response time of the system.

The system can be accessed from the Web but the current Web interface is similar to but not identical to the `MentorClient` interface. This is more a limitation of the installed version of Java that runs in the client browser. This version must be compatible with the developed AppletClient version. Since the user may install any version of Java on their system, this limitation will always remain.

The system, including the Educational Support applications, has only been tested with the two paradigms of Authoritative Guidance and Directed Learning Path. Other paradigms may be able to be accommodated, but further development will need to be done for some specific paradigms. The architecture of the system may limit the types of paradigms that can be supported.

Currently the Domain Knowledge consists of hand-crafted Java Topics as well as Topics automatically created by the Topic Network Builder application. Therefore a significant limitation of this system is that any Lecturer who wishes to achieve similar benefits for their students, must learn how to craft these Topics or learn how to use the Topic Network Builder application. It was shown that a second year computing level of programming skill was all that was required to build the Java code. However, for non-computing Domain Knowledge builders the Topic Network Builder application is the only possibility. Although this is a typical point-and-click application, one significant limitation in using the application is that Domain Knowledge builders would still have to know how to craft Perl-5 Regular Expressions for the pattern matching, and this requires specialist knowledge. Current research on using plain English questions in the Topic Network Builder application combined with an automatic English-to-Regular-Expression translator is promising (Tan 2005a).

Similarly, the Unit Controller and Domain Knowledge builder for the Case Study units was the researcher. The system has not been evaluated by other lecturers, and although this has not affected the outcomes of the research it has an impact on the balance of effort vs. return for another lecturer developing Domain Knowledge and using the system.

The number of participants, whilst sufficient for the analysis of the Case Studies, is not large enough for any statistically reliable analysis of the participants into sub-groups. For example, sub-grouping to correlate enrolled degree vs. attitude, or age vs. attitude would produce sub-group numbers that did not yield results of any statistical significance. Further studies with more participants may enable useful correlations to be determined.

Specifically, the percentage of males vs. females in the computing discipline does not allow the results to be generalised to specify female ‘benefits’ or ‘effectiveness’ in other disciplines. Further studies need to be carried out to see if different approaches should be adopted for different genders.

However, it is believed that of more importance is the underlying mechanism that enables this ‘effectiveness’ for any specific user - the problem is not gender specific, but learner specific. A good human mentor or tutor will change their style when dealing with different users, and this is often based upon the user’s knowledge level **and** their personality. The system currently has facilities for imposing a personality style on the responses but further work has to be done to determine good mentoring personalities, and how these would adapt to help users to learn.

Finally, the system currently assumes only one query per input: input that contains two or more questions is not catered for. It can be seen from some user input that students incorrectly felt that the system could understand long, convoluted sentences.

These limitations do not detract from the research nor its findings. They do point to areas of future research that should be investigated.

6.6. Implications for future research.

In the computing discipline at the university where this research was carried out, a significant number of students did not have English as their first language. This posed a serious problem for the DM system that assumed correct spelling and grammar. In a similar way, English is evolving, and many users entered SMS-type queries. Future research should look at the use of synonyms and typical spelling errors, as well as rules for SMS abbreviations to improve the patterns used to match the user input.

The area of pro-active Responses caused problems for many users who felt interrupted in their work. A facility for users to tailor / suppress topics that produce pro-active Responses was seen as a suitable solution to this problem.

A more important issue was the need for some Topics to be able to switch from passive to pro-active and back again based upon the user requesting information that was not immediately available. Work-around methods already exist within the system to address this issue, but a more holistic approach should be developed for future use.

Future research should investigate better kernel support for the processing of anaphora and ellipsis, as well as adding extra functionality into the DM kernel and into associated support applications such as the Topic Network Builder so that different educational paradigms can be tested.

Users wanted more Domain Knowledge, and better processing of the user input. That is, they wanted it to know more and to understand more. And as for a human mentor, the system should answer relevantly - that is, a human would assume that a user in the third year of their course does not require a trivial answer to a request but one that perhaps addresses a deeper issue. One solution would require a Topic resolution mechanism that was cognisant of the student’s learning level and the relevance of the current Topic given the student’s

current knowledge. Another solution would be to weight the responses differently dependant upon whether the response came from a unit Topic that the student was currently enrolled for, or from a past unit Topic. A simpler approach would be to enable a student to choose any Topic including those from their past units, via the client preferences.

Users wanted an anthropomorphic interface: they engaged with the system through the dialogue and expected the system to have a personality of some sort. The system has hooks to allow for a personality sub-system, and this personality can modify any Response returned from a Topic at various stages of the dialogue management process. For example, one personality may reduce the response weight of a Response that mentions "cats" if the personality did not like cats. Or it could modify the wording used or the emotion associated with that Response. For a Response to be properly processed by a personality, a new class of personality-aware Topics may need to be developed that can cater for the alteration of different aspects of the Response.

A better understanding of the user's immediate attitude to interacting with the system would be beneficial. If the system could detect through multimodal input that the user was happy, annoyed, angry, or bored with the responses, it could modify its behaviour accordingly to address the problem, especially if the system also had a personality.

One criticism of the ***Mentor*** System is that its use is limited to those lecturers who can understand its complexity. That is not totally accurate but currently the Domain Knowledge consists of hand-crafted Java Topics as well as Topics automatically created by the Topic Network Builder application. Therefore a significant limitation of this system is that any Lecturer who wishes to get similar benefits for their students, must learn how to craft these Topics or learn how to use the Topic Network Builder application. It was shown that a second year computing level of programming skill was all that was required to build the Java code. However, for non-computing Domain Knowledge builders the Topic Network Builder application is the only possibility. Although this is a typical point-and-click application, one significant limitation in using the application is that Domain Knowledge builders would still have to know how to craft Perl-5 Regular Expressions for the pattern matching, and this requires specialist knowledge. Current research on converting plain English questions into Regular Expressions seems promising (Tan 2005a). The Topic Network Builder application is being redeveloped to include the functionality of this English->Regular Expression sub-system as well as an improved modern GUI. This will open up the use of ***Mentor*** to any Lecturer.

Mentor does not provide a perfect solution but is one more step towards helping students to learn more effectively. The ***Mentor*** System has met its design objectives, and students eagerly await its use in other units:

We become what we behold.
We shape our tools and
then our tools shape us.
Marshall McLuhan

References

The following references contain Uniform Resource Locators (URLs) that were valid on 1st March 2003. The ↵ symbol used in the references indicates a continuation of an URL that will not fit on one line.

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.

AAAI 2001a, *Intelligent User Interfaces*, AAAI, 22(4): Winter 2001, 85-94. Online at http://www.aaai.org/Library/Magazine/Vol22/22-04/vol22-04.html.

Abdu, D & Bar-Ner, O 1996a, '*Software Agents: A General Overview*', Originally at http://t2.technion.ac.il/~s3180501/agent.html, 1996. Online at http://www.mentor.computing.edu.au/archives/abdu_bar-ner_agent.html.

Ackerman, M S, Starr, B & Pazzani, M 1997a, 'The Do-I-Care Agent: Effective Social Discovery and Filtering on the Web.', in *Proceedings of RIAO'97 (Computer-Assisted Information Searching on the Internet)*, Montreal, June 25-27 1997, pp. 17- 31. Online at http://www.ics.uci.edu/~ackerman/pub/97b23/dica.riao.html.

Aho, A.V, Sethi, R & Ullman, J.D 1986a, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA

AIED2005 2005a, 'SW-EL'05: Applications of Semantic Web Technologies for E-Learning', in *In AIED2005 - Supplementary Proceedings of the 12th International Conference on Artificial Intelligence in Education*, eds. Lora Aroyo & Darina Dicheva, Amsterdam, July 18th, 2005

Aleven, V & Koedinger, K.R 2000a, 'The Need for Tutorial Dialog to Support Self-Explanation', in *Building Dialogue Systems for Tutorial Applications, Papers of the 2000 AAAI Fall Symposium*, eds. C. P. Rosé & R. Freedman, AAAI Press, Menlo Park, CA, 2000, pp. 65-73, Technical Report FS-00-01. Online at http://www-2.cs.cmu.edu/~aleven/aleven-koedinger-aaaif00.ps.gz.

Aleven, V, Popescu, O & Koedinger, K R 2001a, 'A Tutorial Dialogue System with Knowledge-Based Understanding and Classification of Student Explanations', *In Working Notes of the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, August 5, 2001, Seattle. Online at http://www-2.cs.cmu.edu/~aleven/AlevenEtAlIJCAI2001WS.pdf.

Aleven, V, Popescu, O & Koedinger, K R 2001b, 'Pedagogical Content Knowledge in a Tutorial Dialogue System to Support Self-Explanation', in *Papers of the AIED-2001 Workshop on Tutorial Dialogue Systems*, ed. V. Aleven, San Antonio, Texas, May 19-23 2001, pp. 59-70. Online at http://www-2.cs.cmu.edu/~aleven/AlevenEtAlAIED2001Dial-WS.pdf.

Aleven, V, Popescu, O & Koedinger, K R 2002a, 'Pilot-Testing a Tutorial Dialogue System that Supports Self-Explanation', in *Proceedings of Sixth International Conference on Intelligent Tutoring Systems, ITS 2002*, eds. S. A. Cerri, G. Gouardères & F. Paraguaçu, Springer Verlag., Berlin, 2002, pp. 344-354

Aleven, V, Popescu, O, Ogan, A & Koedinger, K R 2003a, 'A Formative Classroom Evaluation of a Tutorial Dialogue System that Supports Self-Explanation', in *AIED2003 Workshop Proceedings*, ed. Vincent Aleven, School of Information Technologies, University of Sydney, Sydney, Australia, 18 July 2003, ISBN 1 86487 572 0

Allen, J F 1987a, *Natural language understanding*, Benjamin/Cummings, Menlo Park, CA

Allen, J F, Byron, D K, Dzikovska, M, Ferguson, G, Galescu, L & Stent, A 2001a, *Toward Conversational Human-Computer Interaction*, AAAI. Online at http://www.aaai.org/Library/Magazine/Vol22/22-04/vol22-04.html.

Ambite, J.-L & Knoblock, C A 1995a, 'Reconciling Distributed Information Sources', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at http://www.isi.edu/info-agents/papers/ambite95-models.pdf.

Anderson, J, Corbett, A, Koedinger, K R & Pelletier, R 1995a, 'Cognitive Tutors: Lessons Learned', *Journal of the Learning Sciences*, vol. 4, no. 2, 1995, p.167−207

André, E, Rist, T & Müller, J 1997a, 'WebPersona: A Life-Like Presentation Agent for Educational Applications on the World-Wide Web.', in *Proceedings of the workshop "Intelligent Educational Systems on the World Wide Web", 8th World Conference of the AIED Society*, Kobe, Japan, Aug 18-22 1997. Online at http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Andre/Andre.html.

Arafa, Y, Kamyab, K & Mamdani, E 2002a, 'Character Animation Scripting Languages: A Comparison', in *Special IST 5th Framework meeting on Representation Formats/Languages*, European Union IST, Bologna, Italy, 18 July 2002

Armstrong, R, Freitag, D, Joachims, T & Mitchell, T 1995a, 'WebWatcher: A Learning Apprentice for the World Wide Web', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, Mar 27-29 1995. Online at http://www-2.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/webagent-plus.ps.Z.

Asada, M, Uchibe, E & Hosoda, K 1995a, 'Agents that learn from other competitive agents', *Workshop on Agents that Learn from Other Agents, International Machine Learning Conference*, July 1995, Tahoe City, USA. Online at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/ml95w1/asada.ps.gz.

Astikainen, M, Kautto, V, Siitonen, J, Sirén, M & Turpeinen, M 1996a, '*PNA - Personal News Assistant*', 1996. Online at http://www.niksula.cs.hut.fi/~vka/pna/UserManual.html.

Baecker, R 1991a, 'New paradigms for computing in the nineties', *Proceedings of Graphics Interface 1991*, 1991, Morgan Kaufmann, Los Altos,CA, pp.224-231

Balabanovic, M & Shoham, Y 1995a, 'Learning Information Retrieval Agents: Experiments with Automated Web Browsing', in *AAAI-95 Spring Symposium on Information Gathering from Heterogenous, Distributed Environments*, 1995, Originally at http://robotics.stanford.edu/people/marko/papers/lira.ps. Online at http://www.mentor.computing.edu.au/archives/Balabanovic.ps.gz.

Balabanovic, M, Shoham, Y & Yun, Y 1997a, 'An Adaptive Agent for Automated Web Browsing', *Journal of Image Representation and Visual Communication*, vol. 6, no. 4, 1997. Online at ftp://db.stanford.edu/pub/cstr/reports/cs/tn/97/52/CS-TN-97-52.pdf.

Ball, G, Ling, D, Kurlander, D, Miller, J, Pugh, D, Skelly, T, Stankosky, A, Thiel, D, Dantzich, M V & Wax, T 1997a, 'Lifelike Computer Characters: The Persona Project at Microsoft Research', in *Software Agents,*, ed. Jeffrey M. Bradshaw, AAAI Press/The MIT Press, California, 1997, pp. 191-222

Bandura, A 1977a, '', in *Social Learning Theory*, General Learning Press, New York, 1977

Bates, J 1994a, 'The Role of Emotion in Believable Agents', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.122-125. Online at http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/ba-and-emotion.ps.

Bates, J, Loyall, A B & Reilly, W S 1992a, 'An Architecture for Action, Emotion, and Social Behaviour', Technical Report CMU--CS--92--144, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, May 1992. Online at http://www.cs.cmu.edu:80/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-92-144.ps.

Beale, R & Wood, A 1994a, 'Agent-Based Interaction', in *People and Computers IX: Proceedings of HCI'94*, British Computer Society HCI Group, August 1994, pp. 239-245. Online at ftp://ftp.cs.bham.ac.uk/pub/dist/hci/papers/agent-based_interaction.ps.Z.

Beard, S 2002a, 'Design Decisions Underlying Virtual Conversational Character Scripting Languages', in *Human Factors 2002 Virtual Conversational Character Workshop*, Melbourne, Australia, 29th November 2002. Online at http://www.vhml.org/workshops/HF2002/papers.shtml.

Beard, S & Reid, D 2002a, 'MetaFace and VHML: A First Implementation of the Virtual Human Markup Language', in *AAMAS workshop on Embodied conversational agents - let's specify and evaluate them!*, AAMAS 2002, Bologna, Italy, 16 July 2002. Online at http://www.vhml.org/workshops/AAMAS/papers.html.

Berners-Lee, T 1998a, *A roadmap to the Semantic Web*, W3C. Online at http://www.w3.org/DesignIssues/Semantic.html.

Berners-Lee, T & Fischetti, M 2000a, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*, HarperBusiness, San Francisco, ISBN: 006251587X

Berners-Lee, T, Hendler, J & Lassila, O 2001a, 'The Semantic Web', *Scientific American*, May 2001. Online at http://www.scientificamerican.com/article.cfm?articleID=↵ 00048144-10D2-1C70-84A9809EC588EF21&catID=2.

Bickmore, T W, Cook, L K, Churchill, E F & Sullivan, J W 1998a, 'Animated Autonomous Personal Representatives', *Proceeding of the 2nd International Conference on Autonomous Agents*, 1998, ACM Press, New York, USA, pp.8-15

Bigus, J P & Bigus, J 2001a, *Constructing Intelligent Agents Using Java: Professional Developer's Guide, 2nd Edition*, John Wiley & Sons, ISBN: 047139601X

Blessing, S B 1996a, '*Webtutor*', Originally at http://www.psy.cmu.edu/~sb6s/webtutor.ps, 1996

Bloom, B S 1984a, 'The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring', *Educational Researcher*, vol. 13, 1984, pp.4-16

Blumenthal, R, Meiskey, L, Dooley, S & Sparks, R 1996a, 'Reducing Development Costs With Intelligent Tutoring System Shells', *ITS'96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs*, June 10th 1996, Originally at http://advlearn.lrdc.pitt.edu/its-arch/papers/blumenthal.html. Online at http://www.mentor.computing.edu.au/archives/blumenthal.html.

Bocionek, S & Sassin, M 1993a, 'Dialog-Based Learning (DBL) for Adaptive Interface Agents and Programming-by-Demonstration Systems', Technical Report CMU-CS-93-175, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213, July 1993. Online at ftp://reports.adm.cs.cmu.edu/usr/anon/1993/CMU-CS-93-175.ps.

Boehm, B W 1981a, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ

Boguraev, B, Garigliano, R & Tait, J 1995a, 'Editorial', *Journal of Natural Language Engineering*, vol. 1, no. 1, 1995

Bonasso, R P, Firby, R J, Gat, E, Kortenkamp, D, Miller, D & Slack, M 1997a, 'Experiences with an Architecture for Intelligent, Reactive Agents', *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2, 1997, Originally at ftp://hobbes.jsc.nasa.gov/pub/korten/jetai.ps.Z. Online at http://www.mentor.computing.edu.au/archives/jetai.ps.gz.

Bowman, C M, Danzig, P B, Hardy, D R, Manber, U & Schwartz, M F 1995a, 'The Harvest information discovery and access system', *Computer Networks and ISDN Systems*, vol. 28, no. 1-2, 1995, pp.119-125. Online at http://catarina.usc.edu/danzig/harvest.ps.

Bratt, E O, Schultz, K, Peters, S, Chen, T & Pon-Barry, H 2005a, 'Empirical Foundations for Intelligent Coaching Systems', *Proceedings of I/ITSEC 2005*, 2005, Orlando, Florida

Brown, J S, Collins, A & Duguid, P 1989a, 'Situated Cognition and the Culture of Learning', *Educational Researcher*, vol. 18, no. 1, 1989, pp.32-42. Online at http://www.ilt.columbia.edu/ilt/papers/JohnBrown.html.

Brown, S M, Harrington, R A, Santos, E, Jr. & Banks, S B 1997a, 'User Models, Intelligent Interface Agents and Expert Systems.', in *Proceedings of International Workshop on Embedding User Models in Intelligent Applications (UM'97)*, Chia Laguna, Sardinia, June 1 1997. Online at http://w5.cs.uni-sb.de/~UM97/ws1-proceedings.html.

Bruner, J S 1967a, *On Knowing: Essays for the Left Hand*, Harvard University Press, Cambridge, Mass

Brusilovsky, P 1995a, 'Integrating Hypermedia and Intelligent Tutoring Technologies: From Systems to Authoring', in *Proceedings of AI-ED 95: Workshop on Authoring Shells for Intelligent Tutoring Systems*, AACE Press., Washington, USA, 1995, Originally at http://www.pitt.edu/~al/aied/brusilov.html. Online at http://www.mentor.computing.edu.au/archives/brusilov.html.

Brusilovsky, P 1995b, 'Intelligent tutoring systems for World-Wide Web', in *Poster Proceedings of Third International WWW Conference*, April 10-14 1995, pp. 42-45. Online at http://www.igd.fhg.de/www/www95/proceedings/posters/48/index.html.

Brustoloni, J C 1991a, 'Autonomous Agents: Characterization and Requirements', Technical Report CMU--CS--91--204, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, November 1991. Online at http://www.cs.cmu.edu/afs/cs/user/jcb/papers/minor.ps.

Bryan, D & Gershman, A 2000a, 'Aquarium: A novel user interface metaphor for large, online Stores.', *Proceedings of 11th International Workshop on Database and Expert Systems Applications (DEXA'00)*, 2000, DEXA

Burke, R & Hammond, K 1995a, 'Combining databases and knowledge bases for assisted browsing', *Presented at AAAI Symposium on Information Gathering in Heterogenous, Distributed Environments*, March 1995. Online at ftp://ftp.isi.edu/sims/sss95/burke.ps.Z.

Burke, R R 1997a, 'Do you see what I see? The future of Virtual Shopping', *The Journal of the Academy of Marketing Science*, vol. 25, no. 4, 1997, pp.352-360

CACM 1994a, 'Special Edition on Intelligent Agents', *Communications of the ACM*, vol. 37, no. 7, July 1994

CACM 1992a, 'Special Edition on Information Filtering', *Communications of the ACM*, vol. 35, no. 12, December 1992

Caldwell, B & Carter, E M A 1993a, *The Return of the Mentor : strategies for workplace learning*, Falmer Press, London, UK

Campbell, C P 1995a, 'Ethos: Character and Ethics in Technical Writing', *IEEE Transactions on Professional Communication*, vol. 38, no. 3, September 1995

Cengeloglu, Y, Khajenoori, S & Linton, D 1994a, 'A Framework for Dynamic Knowledge Exchange among Intelligent Agents', *Published at American Association for Artificial Intelligence (AAAI) Symposium Control of the Physical World by Intelligent Agents*, November 1994, Originally at http://users.aimnet.com/~yilsoft/publications/aaai/aaai.html

Centinia, F, Routen, T, Hartmann, A & Hegarty, C 1996a, '*Statutor: Too Intelligent By Half?*', Originally at http://www.cms.dmu.ac.uk/~centinia/jurix95.html, 1996. Online at http://www.mentor.computing.edu.au/archives/jurix95.html.

Chan, T.-W & Baskin, A B 1990a, 'Learning companion systems', in *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, eds. C. Frasson & G. Gauthier, Ablex, Norwood, NJ, 1990, pp. 6-33

Chan, T.-W & Chou, C.-Y 1995a, 'Simulating a Learning Companion in Reciprocal Tutoring Systems', in *Proceedings of Computer Support for Collaborative Learning '95*, Indiana University, Bloomington, IN, October 17-20 1995, Originally at http://www-cscl95.indiana.edu/cscl95/chan.html. Online at http://www.mentor.computing.edu.au/archives/chan.html.

Chang, D T & Lange, D B 1996a, 'Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW', *OOPSLA'96 Workshop Toward the Integration of WWW and Distributed Object Technology*, 1996, Originally at http://www.trl.ibm.co.jp/aglets/ma.html

Chavez, A & Maes, P 1996a, '*Kasbah: An Agent Marketplace for Buying and Selling Good*', January 1996. Online at http://agents.www.media.mit.edu/groups/agents/publications/kasbah-paam96.ps.gz.

Cheikes, B A 1995a, '*PROPA: An Argumentation-Based Tutor for Explanatory Analysis Tasks*', 1995. Online at http://www.mitre.org/resources/centers/it/g068/g068docs/tutor.html.

Cheikes, B A 1995b, 'Intellectual Skills and Cognitive Strategies: Can One Method Tutor Both?', in *Proceedings of AI-ED 95: World Conference on Artificial Intelligence in Education*, Washington, D.C, August 1995, pp. 445-452, Originally at http://www.cs.umbc.edu/~cheikes/papers/aied-fl.ps

Cheikes, B A 1995c, 'Should ITS Designers Be Looking For A Few Good Agents?', in *Proceedings of AI-ED'95 Workshop on Authoring Shells for Intelligent Tutoring Systems*, ed. N. Major and T. Murray and C. Bloom, Washington DC, USA, 1995

Cheikes, B A 1995d, 'GIA: An Agent-Based Architecture for Intelligent Tutoring Systems', *Proceedings of the CIKM'95 Workshop on Intelligent Information Agents*, December 1-2 1995, Baltimore, USA. Online at http://www.mitre.org/resources/centers/it/g068/giamore.html.

Cheong, F C 1996a, *Internet Agents. Spiders, Wanderers, Brokers and Bots*, New Riders Publishing, Indianapolis, USA

Cheong, F C 1992a, 'OASIS: An Agent-Oriented Programming Language for Heterogeneous Distributed Environment', Ph.D. Thesis, University of Michigan, 1992. Online at http://www.mentor.computing.edu.au/archives/OASIS.tar.gz.

Chess, D, Grosof, B, Harrison, C, Levine, D, Parris, C & Tsudik, G 1995a, 'Itinerant Agents for Mobile Computing', *IEEE Personal Communications Magazine*, vol. 2, no. 5, October 1995, pp.34-49

Chi, M T H, Siler, S A, Jeong, H, Yamauchi, T & Hausmann, R G 2001a, 'Learning from Human Tutoring', *Cognitive Science*, vol. 25, 2001, pp.471-533

Churcher, G E, Atwell, E S & Souter, C 1997a, '*Dialogue Management Systems: a Survey and Overview*', 1997. Online at citeseer.nj.nec.com/churcher97dialogue.html.

Cockayne, W R & Zyda, M 1998a, *Mobile Agents*, Prentice Hall PTR, ISBN: 0138582424

Coen, M H 1994a, 'SodaBot: A Software Agent Environment and Construction System', M.S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994. Online at ftp://ftp.ai.mit.edu/pub/users/mhcoen/aitr-1493.ps.Z.

Cohen, D K 1988a, 'Teaching practice: Plus ca change. . .', in *Contributing to educational change: Perspectives on research and practice*, ed. P. W. Jackson, Berkeley, CA, 1988, pp. 27-84

Cohen, P 1992a, 'The Role of Natural Language in a Multimodal Interface', *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1992, ACM Press, California, USA, pp.148-149

Cohen, P A, Kulik, J A & Kulik, C.-L C 1982a, 'Educational outcomes of tutoring: A Meta-Analysis of Findings', *American Educational Research Journal*, vol. 19, 1982, pp.237-248

Cohn, R 1998a, 'Bringing Back the Human Touch', *Communications News*, vol. 36, no. 1, November 16 1998, pp.32-34

Collins, A & Brown, J S 1988a, 'The computer as a tool for learning through reflection learning', in *Learning Issues for Intelligent Tutoring Systems*, eds. H. Mandl & A. Lesgold, Springer, New York, 1988, pp. 1-18

Compute-Ed 1995a, '*Compute-Ed*', Originally at http://www.education.uts.edu.au/projects/comped, 1995

Conati, C, Gertner, A S, VanLehn, K & Druzdzel, M J 1997a, 'On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks.', in *Proceedings of Sixth International Conference on User Modeling On-Line Proceeding (UM97)*, Chia Laguna, Sardinia, June 2-5 1997. Online at http://w5.cs.uni-sb.de/~UM97/abstracts/ConatiC.html.

Cooper, A & Reimann, R 2003a, *About Face 2.0: The Essentials of Interaction Design*, Wiley Publishing Inc, Indianapolis

Core, M G, Moore, J D & Zinn, C W 2001a, *Initiative Management for Tutorial Dialogue*, NAACL Workshop Adaption in Dialogue Systems, Pittsburgh, PA. Online at http://www.cogsci.ed.ac.uk/~zinn/Publications/adapt.ps.gz.

Core, M G, Moore, J D, Zinn, C W & Wiemer-Hastings, P 2000a, 'Modeling human teaching tactics in a computer tutor', *Proceedings of the ITS'00 Workshop on Modelling Human Teaching Tactics and Strategies*, 2000, Montreal. Online at citeseer.ist.psu.edu/core00modeling.html.

Cousins, S B 1994a, '*Intelligent Interface Agents*', May 1994. Online at http://euphrates.stanford.edu/cousins/papers/qual-reading-list.ps.

Cousins, S B, Frisse, M E, Chen, W & Hassan, S W 1991a, '*Approaches to Information Filtering in Medicine*', Jul 15, 1991. Online at http://euphrates.stanford.edu/cousins/papers/sbc-Info-Filt-91.ps.

Dalgarno, B 1996a, 'Constructivist Computer Assisted Learning: Theory and Techniques', in *Making new connections, Proceedings of the thirteenth annual conference of the Australasian Society for Computers in Learning in Tertiary Education*, eds. A. Christie, P. James & B. Vaughan, ASCILITE, Adelaide, Australia, 1996, pp. 143-154. Online at http://www.ascilite.org.au/conferences/adelaide96/papers/21.html.

Dam, H & Souza, S. de 2002a, 'Applying Talking Head Technology to a Web based Weather Service', *Proceedings of Virtual Conversational Characters Workshop, Human Factors 2002*, 2002. Online at http://www.vhml.org/workshops/HF2002/papers/dam.

DARPA 1993a, *Specification of the KQML Agent-Communication Language (Draft Version)*, The DARPA Knowledge Sharing Initiative External Interfaces Working Group, T. Finin, J. Weber, G. Wiederhold, M. R. Genesereth, R. Fritzson, D. McKay, J. McGuire, P. Pelavin, S. Shapiro, & C. Beck.. Online at http://www-ksl.stanford.edu/knowledge-sharing/papers/kqml-spec.ps.

Davies, W H E & Edwards, P 1995a, 'Agent-Based Knowledge Discovery', *Proceedings of AAAI Spring Symposium on Information Gathering from Distributed, Heterogeneous Environments, Stanford University, California*, Mar 1995. Online at http://www.csd.abdn.ac.uk/~pedwards/research/pubs/daviesss95.pdf.

Deerwester, S, Dumais, S T, Furnas, G W, Landauer, J K & Harshman, R 1990a, 'Indexing by Latent Semantic Analysis', *Journal of the American Society for Information Science*, vol. 41, no. 6, 1990, pp.391-407

Dix, A, Finlay, J, Abowd, G.D & Beale, R 2004a, *Human-Computer Interaction, third edn*, Addison-Wesley Pearson Education, London

Dix, A, Finlay, J & Hassell, J 1992a, 'Environments for cooperating agents: designing the interface as medium', Technical Report, HCI Group, University of York, June 1992

Dooley, S, Meiskey, L, Blumenthal, R & Sparks, R 1995a, 'Developing Usable Intelligent Tutoring System Shells', *AI-ED '95., American Association for Computing in Education*, 1995

Droms, R 1989a, '*The Knowbot Information Service*', Originally at ftp://info.cnri.reston.va.us/rdroms/KIS-id.PS, December 1989. Online at http://www.mentor.computing.edu.au/archives/KIS-id.ps.gz.

Drummond, C, Ionescu, D & Holte, R C 1994a, '*A Learning Agent that Assists the Browsing of Software Libraries*', 1994. Online at http://www.csi.uottawa.ca/~cdrummon/pubs/TR-95-12.ps.

Duchastel, P 1996a, 'Learning Interfaces', in *Advanced Educational Technology: Research Issues and Future Potential*, ed. T. Liao, Springer-Verlag, New York, 1996, Originally at http://www.nova.edu/~duchaste/interfac.html. Online at http://www.mentor.computing.edu.au/archives/duchastel_interfac.html.

Duchastel, P 1996b, 'Knowledge Interfacing in Cyberspace', *Proc. of First International Cyberspace Conference on Ergonomics*, 1996, Originally at http://www.nova.edu/~duchaste/cyberg.html. Online at http://www.mentor.computing.edu.au/archives/duchastel_cyberg.html.

Duchastel, P 1996c, 'A Motivational Framework for Web-Based Instruction', in *Web-Based Instruction: Development, Application and Evaluation*, ed. Badrul Khan, 1996, Originally at http://www.nova.edu/~duchaste/motivati.html. Online at http://www.mentor.computing.edu.au/archives/duchastel_motivati.html.

Duchastel, P & Breuleux, A 1996a, '*A Web-Based Model for University Instruction*', Originally at http://www.nova.edu/~duchaste/unimodel.html, 1996. Online at http://www.mentor.computing.edu.au/archives/duchastel_unimodel.html.

Duchastel, P & Hannah, S 1996a, 'Strategies for Applying Technology to Learning', *Proc. of International Conference on Technology and Distance Education*, Nov 1996, San Juan, Costa Rica, Originally at http://www.nova.edu/~duchaste/costarica.html. Online at http://www.mentor.computing.edu.au/archives/duschastel_costarica.html.

Duchastel, P & Spahn, S 1996a, '*Design for Web-Based Learning*', Originally at http://www.nova.edu/~duchaste/design.html, 1996. Online at http://www.mentor.computing.edu.au/archives/duchastel_design.html.

Duchastel, P & Turcotte, S 1996a, '*On-line learning and teaching in an information-rich context*', Originally at http://www.nova.edu/~duchaste/inet.html, 1996. Online at http://www.mentor.computing.edu.au/archives/duchastel_inet.html.

Ehlert, P 2003a, *Intelligent User Interfaces: Introduction and Survey*, Delft University of Technology, The Netherlands, Research Report DKS03-01 / ICE 01. Online at citeseer.ist.psu.edu/ehlert03intelligent.html.

Eichmann, D 1995a, '*Ethical Web Agents*', Originally at http://rbse.jsc.nasa.gov/eichmann/www-f94/ethics/ethics.html, Jun, 1995. Online at http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/eichmann.ethical/eichmann.html.

Eklund, J 1995a, 'Cognitive models for structuring hypermedia and implications for learning from the world-wide web', in *Proceedings of First Australian World Wide Web Conference*, 1995, pp. 111-116. Online at http://www.scu.edu.au/ausweb95/papers/hypertext/eklund/index.html.

Eklund, J 1995b, '*Adaptive Learning Environments: The future for tutorial software?*', 1995. Online at http://www.education.uts.edu.au/education/staff/john.eklund/ale.html.

Eklund, J 1993a, 'Cognitive Modelling in Intelligent Tutoring: Individualising Tutorial Dialogue', in *Proceedings of ACCE93 in Australian Educational Computing*, 1993, pp. 73-79

Eklund, J 1994a, 'Possibilities for intelligent tutorial software', in *NSW CEG Conference Paper*, Macquarie University, October 1994, pp. 325-334. Online at http://www.edfac.usyd.edu.au/staff/eklundj/ceg.html.

Elliott, C, Lester, J C & Rickel, J 1997a, 'Integrating affective computing into animated tutoring agents.', in *Proceedings of IJCAI '97 workshop on Intelligent Interface Agents: Making Them Intelligent*, Nagoya Japan, September 1997, pp. 113-121. Online at http://www.depaul.edu/~elliott/papers/ijcai97/ij.html.

Elliott, J 1991a, *Action Research for Educational Change*, Open University Press, Milton Keynes, UK

Etzioni, O 1993a, 'Intelligence Without Robots (A Reply to Brooks)', *AI Magazine*, vol. 14, no. 4, 1993, pp.7-13

Etzioni, O, Lesh, N & Segal, R 1994a, 'Building Softbots for UNIX', in *Software Agents --- Papers from the 1994 Spring Symposium (Technical Report SS--94--03)*, ed. O. Etzioni, AAAI Press, March 1994, pp. 9-16

Etzioni, O, Levy, H M, Segal, R B & Thekkath, C A 1993a, 'OS Agents: Using AI Techniques in the Operating System Environment', Technical Report UW-CSE-93-04-04, University of Washington, April 1993. Online at ftp://ftp.cs.washington.edu/tr/1993/04/UW-CSE-93-04-04.PS.Z.

Etzioni, O & Weld, D 1994a, 'A Softbot-Based Interface to the Internet', *Communications of the ACM*, vol. 37, no. 7, July 1994, See also http://www.cs.washington.edu/research/projects/softbots/www/softbots.html, pp.72-76. Online at ftp://ftp.cs.washington.edu/pub/etzioni/softbots/cacm.ps.Z.

Etzioni, O & Weld, D 1994b, 'The First Law of Robotics', in *Software Agents --- Papers from the 1994 Spring Symposium (Technical Report SS--94--03)*, ed. O. Etzioni, AAAI Press, March 1994, pp. 17-23. Online at ftp://ftp.cs.washington.edu/pub/etzioni/softbots/first-law-aaai94.ps.Z.

Eugenio, B D 2001a, 'Natural Language Processing for computer-supported instruction', *ACM Intelligence*, vol. 12, no. 4, 2001, pp.22-32. Online at http://www.cs.uic.edu/~bdieugen/PS-papers/intelligence.pdf.

Evans, J 1994a, '*John Evans Shares His Thoughts on Information*', Originally at http://www.discribe.ca/softworld94/4key1.htm, 1994

Farrow, S 1995a, '*Student Modelling for Intelligent Tutoring Systems*', 1995. Online at http://cedude.ce.gatech.edu/Students/S.Farrow/modelling/references/home.html.

Ferguson, I A 1992a, 'TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents', Ph.D. Thesis, Clare Hall, University of Cambridge, UK, November 1992

Ferguson, I A & Davlouros, J D 1995a, 'PeopleFinder: a Multimodal Multimedia Communications Tool for Interconnecting Office Staff', *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp.2059-2060. Online at ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-38382.pdf.

Ferguson, I A & Davlouros, J D 1995b, 'On Establishing Multi-sensory Multi-channel Communications among Networked Users', *Proceedings of the IJCAI Workshop on AI in Distributed Information Networks*, Aug 1995, pp.103-109. Online at ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-38381.pdf.

Ferguson, I A & Karakoulas, G J 1996a, 'Multiagent Learning and Adaptation in an Information Filtering Market', *Proceedings AAAI Spring Symposium on Adaptation, Coevolution and Learning in Multiagent Systems*, mar 1996. Online at ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-39186.pdf.

Fikes, R, Engelmore, R, Farquhar, A & Pratt, W 1995a, 'Network-Based Information Brokers', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/fikes.ps.Z.

Finin, T & Fritzson, R 1994a, 'KQML --- A Language and Protocol for Knowledge and Information Exchange', in *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, Lake Quinalt, WA, July 1994, pp. 126-136

Finin, T, Fritzson, R, McKay, D & McEntire, R 1994a, 'KQML - A Language and Protocol for Knowledge and Information Exchange', Technical Report CS-94-02, Computer Science Department, University of Maryland and Valley Forge Engineering Center, Unisys Corporation, 1994, ftp://gopher.cs.umbc.edu/pub/ARPA/kqml/papers/kbks.ps. Online at http://www.cs.umbc.edu/kqml/papers/kbkshtml/kbks.html.

Finin, T, Potluri, A, Thirunavukkarasu, C, McKay, D & McEntire, R 1995a, 'On Agent Domains, Agent Names and Proxy Agents', *CIKM '95 Workshop on Intelligent Information Agents*, December 1-2 1995, Baltimore, USA, Originally at http://www.cs.umbc.edu/~cikm/iia/submitted/viewing/finin/. Online at http://www.cs.umbc.edu/~finin/papers/ans95.ps.

Flanagan, D 1997a, *Java in a Nutshell, 2nd Edition.*, O'Reilly, Sebastopol, USA

Fletcher, J D 2003a, 'Does this stuff work? A review of technology Used to Teach', *TechKnowLogia*, vol. 5, no. 1, 2003, pp.10-14. Online at http://www.TechKnowLogia.org..

Foner, L N 1993a, '*What's an Agent Anyway? - A Sociological Case Study*', ftp://ftp.media.mit.edu/pub/Foner/Papers/What's-an-Agent-Anyway--Julia.ps.Z, May 1993. Online at http://foner.www.media.mit.edu/people/foner/Julia/.

Foner, L N 1994a, 'Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning', M.S. Thesis, Media Lab, Massachusetts Institute of Technology, Cambridge, MA, 1994. Online at ftp://ftp.media.mit.edu/pub/Foner/MS-Thesis/Document/.

Foner, L N 1995a, 'Clustering and Information Sharing in an Ecology of Cooperating Agents', in *Proceedings of 1995 Conference on Computers, Freedom,and Privacy*, March 1995, ftp://ftp.media.mit.edu/pub/Foner/Papers/CFP-95/paper.txt. Online at ftp://ftp.media.mit.edu/pub/Foner/Papers/CFP-95/paper.ps.

Franklin, S & Graesser, A 1996a, '*Is it an Agent , or just a program? A Taxonomy for Autonomous Agents*', March 1996. Online at http://www.msci.memphis.edu/~franklin/AgentProg.html.

Freedman, R 1997a, 'Degrees of Mixed-Initiative Interaction in an Intelligent Tutoring System', in *Working Notes of AAAI97 Spring Symposium on Mixed-Initiative Interaction*, eds. S. Haller & S. McRoy, Stanford, CA., 1997, pp. 44-49. Online at http://www.csam.iit.edu/~circsim/documents/rfmis97.ps.gz.

Freeman, E & Fertig, S 1995a, 'Lifestreams: Organizing your Electronic Life', *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval Cambridge, MA*, November, 1995, Originally at http://www.cs.yale.edu/homes/freeman/papers/AAAI/draft.ps. Online at http://www.mentor.computing.edu.au/archives/lifestreams.ps.gz.

Friedl, J E & Oram, A 1997a, *Mastering Regular Expressions*, O'Reilly, Sebastopol, USA

Friedl, J E.F 1997a, *Mastering Regular Expressions 1st Edition*, O'Reilly and Associates, ISBN: 1-56592-257-3. Online at http://www.oreilly.com/catalog/regex/.

Furnas, G W, Landauer, T K, Gomez, L M & Dumais, S T 1987a, 'The Vocabulary Problem in Human-System Communication', *Communications of the ACM*, vol. 30, no. 11, 1987, pp.964-971. Online at citeseer.ist.psu.edu/furnas87vocabulary.html.

Garcia-Molina, H, Hammer, J, Ireland, K, Papakonstantinou, Y, Ullman, J & Widom, J 1995a, 'Integrating and Accessing Heterogeneous Information Sources in TSIMMIS', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/hammer.ps.Z.

Gardner, H 1993a, *Multiple Intelligences: The Theory in Practice*, BasicBooks, New York, ISBN 0-465-01822-x

Garner, R 1999a, '*Programming with JFRED Ruleset Language*', 5 June 1999. Online at http://www.fringeware.com/~robitron/JRL.html.

Garner, R 1999b, '*The Tight Sponge*', 5 June 1999. Online at http://www.fringeware.com/~robitron/ts.html.

Genesereth, M R & Fikes, R E 1992a, 'Knowledge Interchange Format: Version 3.0 Reference Manual', Technical Report, Computer Science Department, Stanford University, 1992. Online at citeseer.ist.psu.edu/article/genesereth92knowledge.html.

Genesereth, M R & Ketchpel, S P 1994a, 'Software Agents', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.49-53. Online at http://logic.stanford.edu/sharing/papers/agents.ps.

Gertner, A, Conati, C & VanLehn, K 1998a, 'Procedural Help in Andes: Generating Hints using a Bayesian Network Student Model', *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998, AAAI Press, Menlo Park, Madison, p.106−111

Ginsberg, M L 1991a, 'Knowledge Interchange Format: The KIF of Death', Technical Report, Stanford University, 1991

Goldberg, D, Nichols, D, Oki, B M & Terry, D 1992a, 'Using Collaborative Filtering to Weave an Information Tapestry', *Communications of the ACM*, vol. 35, no. 12, December 1992, Originally at http://www.xerox.com/PARC/dlbx/tapestry-papers/TN44.ps, pp.61-70

Golden, K, Etzioni, O & Weld, D 1994a, 'Omnipotence without Omniscience: Efficient Sensor Management for Software Agents', in *Software Agents --- Papers from the 1994 Spring Symposium (Technical Report SS--94--03)*, ed. O. Etzioni, AAAI Press, March 1994, pp. 31-36. Online at ftp://ftp.cs.washington.edu/pub/etzioni/softbots/xii-aaai94.ps.Z.

Goodwin, R 1993a, 'Formalizing Properties of Agents', Technical Report CMU-CS-93-159, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, May 1993. Online at ftp://reports-archive.adm.cs.cmu.edu/usr/anon/1993/CMU-CS-93-159.ps.

Goodyear, P 1991a, *Teaching Knowledge and Intelligent Tutoring*, Lawrence Erlbaum, Norwood, USA

Gosling, J & McGilton, H 1995a, 'The Java Language Environment -- A Whitepaper', Technical Report, Sun Microsystems, Oct 1995. Online at http://java.sun.com/whitePaper/javawhitepaper_1.html.

Graesser, A C, Person, N K & Magliano, J P 1995a, 'Collaborative dialogue patterns in Naturalistic One-to-One Tutoring', *Applied Cognitive Psychology*, vol. 9, 1995, pp.495-522

Graesser, A C, VanLehn, K, Rosé, C P, Jordan, P W & Harter, D 2001a, 'Intelligent Tutoring Systems with Conversational Dialogue', *AI Magazine*, vol. 22, no. 4, 2001, pp.39-52.

Graesser, A C, Wiemer-Hastings, K, Wiemer-Hastings, P & Kreuz, R 1999a, 'AUTOTUTOR: A Simulation of a Human Tutor', *Journal of Cognitive Systems Research*, vol. 1, no. 1, 1999, p.35−51

Graesser, A C, Wiemer-Hastings, P, Wiemer-Hastings, K, Harter, D & Person, N 2000a, 'Using Latent Semantic Analysis to Evaluate the Contributions of Students in AUTOTUTOR', *Interactive Learning Environments*, vol. 8, no. 2, 2000, p.129−148

Granger, R H, Staros, C J, Taylor, G S & Yoshii, R 1983a, 'Scruffy Text Understanding: Design and Implementation of the NOMAD System', *Proceedings of the 1983 First Applied Natural Language Processing Conference*, 1-3 February 1983, Santa Monica, California. Online at http://acl.ldc.upenn.edu/A/A83/A83-1017.pdf.

Gravano, L & Garcia-Molina, H 1995a, 'Generalizing GlOSS for Vector-Space Databases and Broker Hierarchies', *Proc. of the 21st International Conference on Very Large Data Bases (VLDB'95)*, 1995. Online at ftp://db.stanford.edu/pub/gravano/1995/vldb95.ps.

Gravano, L, Garcia-Molina, H & Tomasic, A 1994a, 'Precision and Recall of GlOSS Estimators for Database Discovery', *Proc. of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS'94)*, 1994. Online at ftp://db.stanford.edu/pub/gravano/1994/stan.cs.tn.94.010.pdis94.ps.

Gray, R S 1995a, '*Agent Tcl: Alpha Release 1.1*', Originally at ftp://cs.dartmouth.edu/pub/agents/doc.1.1.ps.gz, Dec 1, 1995. Online at http://www.mentor.computing.edu.au/archives/doc.1.1.ps.gz.

Greer, J, McCalla, G, Cooke, J, Collins, J, Kumar, V, Bishop, A & Vassileva, J 1998a, 'The Intelligent Helpdesk: Supporting Peer Help in a University Course', in *Intelligent Tutoring Systems*, eds. B.P. Goettl, H.M. Halff, C.L. Redfield & V.J. Shute, Springer-Verlag, Berlin Heidelberg, 1998, pp. 494-503

Greer, J, McCalla, G, Kumar, V, Collins, J & Meagher, P 1997a, 'Facilitating collaborative learning in distributed organizations', *Computer Support for Collaborative Learning 97 Conference*, December, 1997, Toronto, Canada. Online at http://citeseer.ist.psu.edu/greer97facilitating.html.

Greif, I 1994a, 'Desktop Agents in Group-Enabled Products', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.100-105

Grosof, B 1995a, 'Conflict resolution in advice taking and instruction for learning agents', *Workshop on Agents that Learn from Other Agents, International Machine Learning Conference*, July 1995, Tahoe City, USA. Online at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/ml95w1/grosof.ps.gz.

Grosz, B J & Kraus, S 1996a, 'Collaborative Plans for Complex Group Action', *Artificial Intelligence*, vol. 86, no. 2, 1996, p.269−357

Grosz, B J & Sidner, C L 1986a, 'Attention, Intentions, and the Structure of Discourse', *Computational Linguistics*, vol. 12, no. 3, 1986, pp.175-204. Online at http://acl.ldc.upenn.edu/J/J86/J86-3001.pdf.

Grosz, B J & Sidner, C L 1990a, 'Plans for Discourse', in *Intentions and Communication*, eds. P. R. Cohen, J. L. Morgan & M. E. Pollack, MIT Press., Cambridge, Mass, 1990, pp. 417−444

Gruber, T R, Tenenbaum, A B & Tenenbaum, J M 1994a, 'NIKE: A National Infrastructure for Knowledge Exchange', *Enterprise Integration Technologies, Technical Report*, 1994. Online at http://www.mentor.computing.edu.au/archives/nike.html.

Guha, R V & Lenat, D B 1994a, 'Enabling Agents to Work Together', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.127-142

Guha, R V & Lenat, D B 1990a, 'Cyc: A Midterm Report', *AI Magazine*, vol. 11, no. 3, Fall 1990, pp.32-59

Guha, R V, Lenat, D B, Pittman, K, Pratt, D & Shepherd, M 1990a, 'Cyc: A Midterm Report', *Communications of the ACM*, vol. 33, no. 8, August 1990

Guilfoyle, C & Warner, E 1994a, *Intelligent Agents: The New Revolution in Software*, Ovum Ltd, London

Gustavsson, C, Strindlund, L & Wiknertz, E 2002a, 'Verification, Validation and Evaluation of the Virtual Human Markup Language (VHML)', in *Master's thesis*, Institutionen för Systemteknik, 2002. Online at http://www.ep.liu.se/exjobb/isy/2002/3188/.

Gustavsson, C, Strindlund, L & Wiknertz, E 2001a, 'Dialogue Management Tool', in *The Talking Head Technology Workshop of OZCHI2001, The Annual Conference for the Computer-Human Interaction Special Interest Group (CHISIG) of the Ergonomics Society of Australia.*, Fremantle, Australia, 20-22 November 2001. Online at http://www.ep.liu.se/exjobb/isy/2002/3188/.

Güzeldere, G & Franchi, S 1995a, 'Dialogues with Colorful Personalities of Early AI', in *Constructions of the Mind: Artificial Intelligence and the Humanities*, eds. Stefano Franchi & Güven Güzeldere, Stanford Humanities Review, 24 July 1995

Güzeldere, G & Franchi, S 1995b, 'Mindless Mechanisms, Mindful Constructions: an Introduction', in *Constructions of the Mind: Artificial Intelligence and the Humanities*, eds. Stefano Franchi & Güven Güzeldere, Stanford Humanities Review, 4 June 1995

Hammond, K, Burke, R, Martin, C & Lytinen, S 1995a, 'FAQ Finder: A Case-Based Approach to Knowledge Navigation', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/hammond.ps.Z.

Harasim, L 1993a, 'Collaborating in cyberspace: Using Computer Conferences as a Group Learning Environment', *Interactive Learning Environments*, vol. 3, no. 2, 1993, pp.119-130.

Harnad, S 1992a, 'The Turing Test Is Not A Trick: Turing Indistinguishability Is A Scientific Criterion', *SIGART Bulletin*, vol. 3(4), October 1992, pp.9-10

Harrison, C G, Chess, D M & Kershenbaum, A 1995a, 'Mobile Agents: Are they a good idea?', Research Report, IBM Research Division, March 1995, Originally at http://www.watson.ibm.com:8080/main-cgi-bin/search_paper.pl/entry_ids=7929. Online at http://www.infosys.tuwien.ac.at/Research/Agents/archive/mobagtibm.ps.gz.

Hermans, B 1996a, 'Intelligent Software Agents on the Internet', M.S. Thesis, Tilburg University, Nederlands, 9th July 1996. Online at http://www.hermans.org/agents/.

Hibbard, J 1998a, 'Web service: Ready or not', *Information Week*, November 16 1998, pp.18-20

Hirst, G 1981a, 'Anaphora in Natural Language Understanding: A Survey', in *Springer Lecture Notes in Computer Science 119*, Springer, Berlin, 1981

Hjalmarsson, A 2002a, 'Evaluating AdApt, a multi-modal conversational, dialogue system using PARADISE', *Master's Thesis in Speech Technology*, November, 2002, Department of Speech Music and Hearing, KTH Royal Institute of Technology, Stockholm, Sweden

Holic, A 2004a, 'Experiential User Interfaces For the Elderly', in *Honours Thesis*, Department of Computing, Curtin University of Technology, Perth, Western Australia, June 2004. Online at http://www.vhml.org/theses/holica.

Holt, P, Fontaine, C, Gismondi, J & Ramsden, D 1995a, 'Collaborative Learning Using Guided Discovery on the INTERNET', *Centre for Computing Information Systems and Mathematics (CCISM) Athabasca University Athabasca, Alberta Canada, T9S 1A1*, 1995. Online at http://ccism.pc.athabascau.ca/html/ccism/deresrce/papers/icce95.htm.

Holte, R C & Drummond, C 1994a, 'A Learning Apprentice for Browsing', in *Software Agents --- Papers from the 1994 Spring Symposium (Technical Report SS--94--03)*, ed. O. Etzioni, AAAI Press, March 1994, pp. 37-42. Online at http://www.csi.uottawa.ca/~cdrummon/pubs/AAAI94_Sym.ps.

Höök, K 2000a, 'Steps to take before Intelligent User Interfaces become real', *Journal of Interacting with Computers*, vol. 12, no. 4, February 2000, pp.409-426. Online at http://www.sics.se/~kia/papers/Steps_Hook_final.pdf.

Hubler, A W & Assad, A M 1995a, 'CyberProf: An Intelligent Human-Computer Interface for Asynchronous Wide Area Training and Teaching', in *Fourth International World Wide Web Conference. The Web Revolution*, Boston, Massachusetts, December 11-14 1995. Online at http://www.w3.org/pub/Conferences/WWW4/Papers/247/.

Hudson, W 2004a, '*Mental Models, Metaphor, and Design*', August 29, 2004. Online at http:www.syntagm.co.uk/design.

Huffman, S B 1994a, 'Instructable Autonomous Agents', Ph.D. Thesis, University of Michigan Dept. of Electrical Engineering and Computer Science, January 1994, Originally at ftp://ftp.eecs.umich.edu/people/huffman/papers/thesis.ps.Z. Online at http://www.mentor.computing.edu.au/archives/instructable_thesis.ps.gz.

Huffman, S B 1995a, 'Learning information extraction patterns from examples', *IJCAI-95 workshop on New Aproaches to Learning for Natural Language Processing*, August 1995, Montréal, Québec, Canada, Originally at ftp://ftp.eecs.umich.edu/people/huffman/papers/ijcai95-workshop.ps.Z, pp.246-260. Online at http://www.mentor.computing.edu.au/archives/jcai95-workshop.ps.gz.

Huffman, S B & Laird, J E 1994a, 'Learning from highly flexible tutorial instruction', *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, July 31 - August 4 1994, Seattle, USA, Originally at ftp://ftp.eecs.umich.edu/people/huffman/papers/aaai94.ps.Z, pp.506-512. Online at http://www.mentor.computing.edu.au/archives/aaai94.ps.gz.

Huffman, S B & Steier, D 1995a, 'Heuristic Joins to Integrate Structured Heterogeneous Data', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/huffman.ps.Z.

Hume, G 1995a, 'Using Student Modelling to Determine when and how to Hint in an Intelligent Tutoring System', *PhD Thesis*, May 1995, Illinois Institute of Technology, Chicago

Hume, G, Michael, J, Rovick, A & Evens, M 1993a, 'The use of hints as a tutorial tactic', *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1993, Boulder, CO, pp.563-568

Hume, G, Michael, J, Rovick, A & Evens, M 1995a, 'Controlling active learning: how tutors decide when to generate hints', *Proceedings of the 8th Florida Artificial Intelligence Research Symposium*, 1995, Melbourne Beach, FL, pp.157-161

Hume, G, Michael, J, Rovick, A & Evens, M 1995b, 'Hinting as a tactic in one-on-one tutoring', *Journal of Learning Sciences*, vol. 5, no. 1, 1995, pp.23-47

Hume, G, Michael, J, Rovick, A & Evens, M 1996a, 'The Use of Hints by Human and Computer Tutors: The Consequences of the Tutoring Protocol', *Proceedings of the 2nd International Conference on the Learning Sciences*, 1996, Evanston, IL, pp.135-142

Hutchens, J & Adler, D 1998a, 'Introducing MegaHAL', *Human-Computer Communication Workshop*, 1998, pp.271-274

Huynh, Q 2000a, 'A Facial Animation Markup Language (FAML) for the Scripting of a Talking Head', in *Honours Thesis*, School of Computing, Curtin University of Technology, Perth, Western Australia, December 2000. Online at http://www.vhml.org/theses/huynhqh.

Ibrahim, B & Franklin, S D 1995a, 'Advanced Educational Uses of the World-Wide Web', in *The Third International World-Wide Web Conference. Technology, Tools and Applications*, Darmstadt. Germany, April 10-14 1995, See also volume 27, pages 871-877, special issue of Computer Networks and ISDN Systems.. Online at http://www.igd.fhg.de/www/www95/proceedings/papers/89/paper.html.

Inlärning, K. för, Kognition och IT 1995a, '*ITS and AI References*', 1995. Online at http://www.mentor.computing.edu.au/archives/inlarning.html.

Isbister, K 1995a, 'Perceived intelligence and the design of computer characters', *Extract from M.A. thesis, presented at the Lifelike Computer Characters conference*, 28 September 1995. Online at http://www.cyborganic.com/People/kath/lccpaper.html.

Isbister, K & Layton, T 1996a, '*Intelligent Agents: a review of current literature*', 1996. Online at http://www.research.microsoft.com/research/ui/persona/isbister.htm.

ISO/IEC 1998a, *Text for ISO/IEC FDIS 14496-2 Visual*, ISO/IEC. Online at http://www.cselt.it/mpeg/working_documents.htm.

Jacobson, I, Christerson, M, Jonsson, P & Övergaard, G 1992a, *Object Oriented Software Engineering - A Use Case Driven Approach.*, Addison-Wesley, Sydney

Jacobson, M J & Levin, J A 1995a, '*Network Learning Environments and Hypertext: Frameworks for Constructing Personal and Shared Knowledge Spaces*', 1995. Online at http://www.ed.uiuc.edu/People/MJ/JETele.html.

Jacobson, M J & Levin, J A 1995b, 'Conceptual Frameworks for Network Learning Environments: Constructing Personal and Shared Knowledge Spaces', *International Journal of Educational Telecommunications*, vol. 1, no. 4, 1995, pp.367-388. Online at http://www.ed.uiuc.edu/tta/papers/JL_EdTele/.

JALN 1995a, '*Journal of the Asynchronous Learning Network*', 1995. Online at http://www.aln.org/publications/jaln/.

JEMH 1995a, '*Journal of Educational Multimedia and Hypermedia*', January 1995. Online at http://www.aace.org/pubs/jemh/.

Jennings, N R 1992a, 'A Knowledge Level Approach to Collaborative Problem Solving', in *Proc. AAAI Workshop on Cooperation Among Heterogeneous Intelligent Agents*, San Jose, USA, 1992, pp. 55-64, Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/AAAI92-WS.ps.Z. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Jennings, N R 1992b, 'Towards a Cooperation Knowledge Level for Collaborative Problem Solving', in *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, Vienna, Austria, 1992, pp. 224-228, Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/ECAI92.ps.Z. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Jennings, N R & Jackson, A J 1995a, 'Agent based meeting scheduling: A Design and Implementation', *Electronics Letters, The Institution of Electrical Engineering*, vol. 31, no. 5, 1995, Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/ElecLetters95.ps.Z, pp.350-352. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Jennings, N R, Varga, L Z, Aarnts, R P, Fuchs, J & Skarek, P 1993a, 'Transforming Standalone Expert Systems into a Community of Cooperating Agents', *International Journal of Engineering Applications of Artificial Intelligence*, vol. 6, no. 4, 1993, Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/ENG-APPL-AI-6-4.ps.Z, pp.317-331. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Jennings, N R & Wooldridge, M 1995a, '*Applying Agent Technology*', Originally at http://www.doc.mmu.ac.uk/STAFF/mike/aaij95.ps, 1995. Online at http://www.mentor.computing.edu.au/archives/aaij95.ps.gz.

JILR 1995a, '*Journal of Interactive Learning Research*', 1995. Online at http://www.aace.org/pubs/jilr/.

Joachims, T, Mitchell, T, Freitag, D & Armstrong, R 1995a, 'WebWatcher: Machine Learning and Hypertext', *Fachgruppentreffen Maschinelles Lernen, Dortmund, Germany*, Aug 1995, Originally at http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/mltagung-e.ps.Z, pp.6-12. Online at http://www.mentor.computing.edu.au/archives/joachims95webwatcher.pdf.

Johansen, D, Renesse, R. van & Schneider, F B 1994a, 'Operating system support for mobile agents', Technical Report TR94-1468, Department of Computer Science, Cornell University, 1994, Originally at http://cs-tr.cs.cornell.edu/TR/CORNELLCS:TR94-1468. Online at http://www.cs.cornell.edu/fbs/publications/tacomaHotOS.ps.

Johnson, L, Shaw, E & Ganeshan, R 1998a, 'Pedagogical Agents on the Web', *ITS98 Workshop on Intelligent Tutoring Systems on the World Wide Web*, August 1998. Online at http://www.isi.edu/isd/ADE/papers/its98/ITS98-WW.htm.

Johnson, M B 1995a, 'WavesWorld. A testbed for constructing semi-autonomous animated characters.', Ph.D. Thesis, Massachusetts Institute of Technology, School of Architecture and Planning., June 1995. Online at http://xenia.media.mit.edu/~wave/PhDThesis/outline.html.

Johnson, W L 2001a, *Pedagogical Agent Research at CARTE*, AAAI. Online at http://www.aaai.org/Library/Magazine/Vol22/22-04/vol22-04.html.

Johnson, W L 1995a, 'Pedagogical Agents in Virtual Learning Environments', in *Proceedings of International Conference on Computers and Education*, AACE, Singapore, 1995, pp. 41-48. Online at http://www.isi.edu/isd/VET/icce.ps.

Johnson, W L & Rickel, J 1996a, 'Intelligent Tutoring in Virtual Environment Simulations', in *ITS '96 Workshop on Simulation-Based Learning Technology, held in conjunction with the International Conference on Intelligent Tutoring Systems*, Montreal, June 1996, Originally at http://www.isi.edu/isd/VET/sbt-workshop.ps

Johnson, W.L, Rickel, J.W & Lester, J.C 2000a, 'Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments', *International Journal of Artificial Intelligence in Education*, no. 11, 2000, pp.47-78. Online at http://www.isi.edu/isd/rickel-old/apa.pdf.

JTATE 1995a, '*Journal of Technology and Teacher Education*', 1995. Online at http://www.aace.org/pubs/jtate.

Jurafsky, D & Martin, J H 2000a, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice-Hall, ISBN: 0-13-095069-6. Online at http://www.cs.colorado.edu/~martin/slp.html.

Kadous, M W & Sammut, C 2002a, 'Mobile Conversational Characters', in *Human Factors 2002 Virtual Conversational Character Workshop*, Melbourne, Australia, 29th November 2002. Online at http://www.vhml.org/workshops/HF2002/papers/kadous/kadous.pdf.

Kahle, D 1995a, '*Computer Mediated Communication in Distance Education - An Annotated Bibliography*', Originally at http://www.mit.edu:8001/↵ afs/athena.mit.edu/user/d/j/djkahle/www/hgse/cmcbiblio.html, January 1995. Online at http://www.mentor.computing.edu.au/archives/cmcbiblio.html.

Karakoulas, G J & Ferguson, I A 1995a, 'A Computational Market for Information Filtering in Multi-dimensional Spaces', *Proceedings of the AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, Nov 1995, pp.78-83. Online at ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-38387.pdf.

Karakoulas, G J & Ferguson, I A 1996a, 'SIGMA: Integrating Learning Techniques in Computational Markets for Information Filtering', *Proceedings AAAI Spring Symposium on Machine Learning in Information Access*, March 1996, Portland, Oregon. Online at ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-39187.pdf.

Karlsson, F & Karttunen, L 1996a, 'Language Analysis and Understanding', in *Survey of the State of the Art in Human Language Technology*, eds. Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, Victor Zue, Giovanni Battista Varile & Antonio Zampolli, Cambridge University Press, 1996, ISBN 0-521-59277-1. Online at http://cslu.cse.ogi.edu/HLTsurvey/.

Katz, B 1997a, 'Annotating the World Wide Web Using Natural Language', *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997, Montreal, Canada

Kautz, H A, Milewski, A & Selman, B 1995a, 'Agent Amplified Communication', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/kautz.ps.Z.

Kautz, H A, Selman, B, Coen, M & Ketchpel, S 1994a, 'An Experiment in the Design of Software Agents', in *Software Agents --- Papers from the 1994 Spring Symposium (Technical Report SS--94--03)*, ed. O. Etzioni, AAAI Press, March 1994, pp. 43-48

Kay, A 1990a, 'User Interface: A Personal View', in *The Art of Human-Computer Interface Design*, ed. Brenda Laurel, Addison-Wesley, Massachusetts,USA, 1990, pp. 191-207

Kay, J & Kummerfeld, B 1996a, 'User model based filtering and customisation of web pages', *Fifth International Conference on User Modelling*, Jan 2-5 1996, Hawaii. Online at http://www.cs.su.oz.au/~bob/um96-paper.html.

Kearney, P 1996a, 'Personal agents: A walk on the client side', *IEE C3 Colloquium on Intelligent Agents and their applications*, April 1996, Originally at http://www.sharp.co.uk/pk/unicom/Unicom.htm. Online at http://www.mentor.computing.edu.au/archives/Unicom.html.

Kearsley, G 1998a, *Online Help Systems Design and Implementation*, Alex Publishing Corporation, New Jersey, USA

Keim, G, Fulkerson, M & Biermann, A 1997a, 'Initiative in tutorial dialogue systems', *Symposium on Computational Models for Mixed Initiative Interactions*, 1997, AAAI, Menlo Park CA. Online at citeseer.ist.psu.edu/keim97initiative.html.

Kemmis, S & McTaggart, R 1988a, *The Action Research Planner, third edition*, Deakin University, Victoria, Australia

Kirk, T, Levy, A Y, Sagiv, Y & Srivastava, D 1995a, 'The Information Manifold', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at http://www.research.att.com/~divesh/papers/klss95-im.ps.

Kittock, J E 1993a, 'Emergent Conventions and the Structure of Multi-Agent Systems', in *Proceedings of the 1993 Santa Fe Institute Complex Systems Summer School*, 1993, Originally at http://robotics.stanford.edu/people/jek/Papers/sfi93.ps.gz

Kittock, J E 1994a, 'The Impact of Locality and Authority on Emergent Conventions: Initial Observations', in *Proceedings of the Twelfth National Conference on Artificial Intelligence AAAI'94*, Seattle, USA, July 31 - August 4 1994, pp. 420-425, Originally at http://robotics.stanford.edu/people/jek/Papers/aaai94.ps.gz

Klark, P & Manber, U 1994a, '*Developing a Personal Internet Assistant*', Originally at http://glimpse.cs.arizona.edu:1994/~paul/warmlist/paper.html, Dec 1994. Online at http://www.mentor.computing.edu.au/archives/klark_paper.html.

Klein, J 1999a, '*Computer Response to User Frustration*', 1999. Online at citeseer.nj.nec.com/klein99computer.html.

Knabe, F C 1995a, 'Language Support for Mobile Agents.', Ph.D. Thesis, Carnegie Mellon University, School of Computer Science., Dec 1995. Online at http://www.cs.virginia.edu/~knabe/dissertation.ps.gz.

Knoblock, C A 1995a, 'Integrating Planning and Execution for Information Gathering', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/knoblock.ps.Z.

Knoblock, C A & Levy, A Y 1995a, 'Exploiting Run-Time Information for Efficient Processing of Queries', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/levy.ps.Z.

Koda, T 1996a, 'Agents with Faces: A Study on the Effects of Personification of Software Agents', M.S. Thesis, MIT Media Laboratory, September 1996. Online at http://www.media.mit.edu/~tomoko/docs/ROMAN_paper.ps.

Koda, T & Maes, P 1996a, 'Agents with Faces: A Study on the Effects of Personification of Software Agents', *Proceedings of HCI'96*, 1996, The British HCI Group, pp.98-103

Krathwohl, D R 1993a, *Methods of Educational and Social Science Research: An Integrated Approach*, Longman, New York

Krenn, B 2002a, '*IST Cross-programme Concertation Meeting on Representation Formats/Languages*', 18 July 2002

Krueger, M 1999a, 'Cognitive Space', in *Humane Interfaces: Questions of method and practice in Cognitive Technology*, eds. P. Marsh, B. Gorayska & J. Mey, Elsevier, Singapore, 1999, pp. 219-228

Krulwich, B 1995a, 'Learning User Interests Across Heterogeneous Document Databases', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/krulwich.ps.Z.

Kuokka, D & Harada, L 1995a, 'Supporting Information Retrieval via Matchmaking', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/kuokka.ps.Z.

Landauer, T K, Foltz, P W & Laham, D 1998a, 'Introduction to Latent Semantic Analysis', *Discourse Processes*, vol. 25, 1998, pp.259-284

Landauer, T K, Foltz, P W & Laham, D 1998b, 'Introduction to Latent Semantic Analysis.', *Discourse Processes*, vol. 25, 1998, pp.259-284. Online at http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

Lange, D B & Chang, D T 1996a, 'Programming Mobile Agents in Java', *White Paper*, September 9, 1996, Originally at http://www.trl.ibm.co.jp/aglets/whitepaper.htm

Lange, D B & Oshima, M 1998a, *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley Professional, ISBN: 0201325829

Larsson, S, Bohlin, P, Bos, J & Traum, D 1999a, *TrindiKit manual*, The Trindi Consortium, Tech. Rep., TRINDI Deliverable D2.2

Larsson, S & Traum, D 2000a, '*Information state and dialogue management in the TRINDI dialogue move engine toolkit*', 2000. Online at citeseer.ist.psu.edu/article/larsson00information.html.

Lashkari, Y, Metral, M & Maes, P 1994a, 'Collaborative Interface Agents', in *Proceedings of the Twelfth National Conference on Artificial Intelligence AAAI'94*, Seattle, USA, July 31 - August 4 1994. Online at ftp://ftp.media.mit.edu/pub/agents/interface-agents/coll-agents.ps.

Lave, J & Wenger, E 1991a, *Situated Learning : Legitimate Peripheral Participation (Learning in Doing : Social, Cognitive and Computational Perspectives)*, Cambridge University Press, Cambridge, UK, ISBN: 0521423740

Lawler, R & Yazdani, M 1987a, *Artificial Intelligence and Education*, Ablex, Norwood, USA

Lehnert, W 1975a, 'What Makes SAM Run? Script Based Techniques for Question Answering', *Theoretical Issues in Natural Language Processing*, 10-13 June 1975, Cambridge, Massachusetts. Online at http://acl.ldc.upenn.edu/T/T75/T75-1005.pdf.

Lenat, D B 1995a, *CYC: A Large-Scale Investment in Knowledge Infrastructure*, Communications of the ACM. Online at citeseer.ist.psu.edu/lenat95cyc.htm.

Lesgold, A, Katz, S, Suthers, D & Weiner, A 1996a, '*Extending the Sherlock Learning-by-Doing Approach to Work Force (Re)training*', Originally at http://advlearn.lrdc.pitt.edu/advlearn/NeRC-proposal-excerpts.html, 1996. Online at http://www.mentor.computing.edu.au/archives/NeRC-proposal-excerpts.html.

Lester, J C, Converse, S A, Kahler, S E, Barlow, S T, Stone, B A & Bhoga, R S 1997a, 'The Persona Effect: Affective Impact of Animated Pedagogical Agents', *Proceedings of CHI '97 Conference on Human Factors in Computing Systems*, 22-27 March 1997, Atlanta, USA. Online at http://www.acm.org/sigchi/chi97/proceedings/paper/jl.htm.

Lester, J.C & Stone, B.A 1997a, 'Increasing Believability in Animated Pedagogical Agents', *First International Conference on Autonomous Agents*, February 1997., ACM, Marina del Rey, California, pp.16-21. Online at http://www4.ncsu.edu/~lester/Public/dap-aa-97.ps.gz.

Lewin, I, Rupp, C J, Hieronymus, J, Milward, D, Larsson, S & Berman, A 2000a, *Siridus System Architecture and Interface Report*, Siridus Project D6.1. Online at http://www.ling.gu.se/projekt/siridus/Publications/deliv6-1.pdf.

Li, W.-S 1995a, 'Knowledge Gathering and Matching in Heterogeneous Databases', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at ftp://ftp.isi.edu/sims/sss95/li.ps.Z.

Lieberman, H 1994a, '*Attaching Interface Agent Software to Applications*', March 1994. Online at http://lieber.www.media.mit.edu/↵ people/lieber/Lieberary/Attaching/Attaching/Attaching.html.

Lingnau, A & Drobnik, O 1995a, 'An Infrastructure for Mobile Agents: Requirements and Architecture', *Proc. 13th DIS Workshop*, September 1995, Orlando, Florida. Online at ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/13dis-paper.ps.gz.

Lingnau, A & Drobnik, O 1996a, 'Making Mobile Agents Communicate: A Flexible Approach', *1st Annual Conference on Emerging Trends and Applications in Communications (etaCOM'96)*, May 7-10, 1996, Portland, USA. Online at ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/etacom96-paper.ps.gz.

Lingnau, A, Drobnik, O & Dömel, P 1995a, 'An HTTP-based Infrastructure for Mobile Agents', *World Wide Web Journal - Proceedings of the Fourth International World Wide Web Conference*, December 11-14, 1995, Boston, USA. Online at ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/www4-paper.ps.gz.

Lohse, G L & Spiller, P 1998a, 'Electronic shopping', *Communications of the ACM*, vol. 41, no. 7, 1998, pp.81-87

Lovgren, J 1994a, 'How To Choose Good Metaphors', *IEEE Software*, vol. 11, no. 3, 1994, pp.86-88

Loyall, A B & Bates, J 1995a, 'Personality-Rich Believable Agents That Use Language', Technical Report, Mar, 1995. Online at http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/bel-agents-nlg.ps.gz.

Loyall, A B & Bates, J 1993a, 'Real-time Control of Animated Broad Agents', *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, June 18-21 1993, Boulder, USA

Maclin, R & Shavlik, J W 1994a, 'Incorporating Advice into Agents that Learn from Reinforcements', *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI '94)*, July 31 - August 4 1994, Seattle, USA. Online at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/maclin.tr94.ps.

Maes, P 1990a, *Designing Autonomous Agents*, The MIT Press: Cambridge, MA

Maes, P 1994a, 'Agents that Reduce Work and Information Overload', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.31-40

Maes, P 1995a, 'Artificial Life meets Entertainment: Interacting with Lifelike Autonomous Agents.', *Special Issue on New Horizons of Commercial and Industrial AI*, vol. 38, no. 11, November 1995. Online at http://pattie.www.media.mit.edu/people/pattie/CACM-95/alife-cacm95.html.

Maes, P, Darrell, T, Blumberg, B & Pentland, A 1996a, 'The ALIVE System: Wireless, Full-Body Interaction with Autonomous Agents', *Special Issue on Multimedia and Multisensory Virtual Worlds, ACM Multimedia Systems*, Spring 1996. Online at http://www-white.media.mit.edu/vismod/publications/techdir/TR-257.ps.Z.

Maes, P & Kozierok, R 1993a, 'Learning Interface Agents', in *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, The MIT Press: Cambridge, MA, Washington, USA, July 11-15 1993, pp. 459-465

Major, N 1993a, 'COCA - A Co-operative Classroom Assistant', Ph.D. Thesis, University of Nottingham, 1993, Originally at http://www.psyc.nott.ac.uk/aigr/research/its/COCA.html

Mark, M A & Greer, J E 1993a, 'Evaluation Methodologies for Intelligent Tutoring Systems', *Journal of Artificial Intelligence and Education*, vol. 4, 1993, pp.129-153. Online at ftp://ftp.cs.usask.ca/pub/aries/papers/mark-jaied.ps.

Marriott, A 1996a, 'Electronic Publishing Colloquium.', *International Asia Pacific Rim WWW Conference '96*, 23-28 Aug 1996, IEEE. Hong Kong and Beijing,China.. Online at http://www.cs.curtin.edu.au/~raytrace/papers/ap96/html/colloquium.html.

Marriott, A 1997a, 'The Webbing of the Grad Dip Comp', *Proceedings of ASCILITE'97*, 7-10 Dec, 1997, Perth, Western Australia.. Online at http://www.cs.curtin.edu.au/~raytrace/papers/ascilite/html/ascilite.html.

Marriott, A 2002a, 'A Facial Animation case study for HCI: the VHML-based Mentor System', in *MPEG-4 Facial Animation - The standard, implementations and applications*, eds. I. Pandzic & R. Forchheimer, John Wiley, London, 2002, pp. 219-239

Marriott, A 2003a, 'Mentor System Customisation', in *AAMAS2003 workshop on Embodied Conversational Characters as Individuals*, AAMAS 2003, Melbourne, Australia, 15 July 2003. Online at http://www.vhml.org/workshops/AAMAS2003/.

Marriott, A 2001a, 'VHML - Virtual Human Markup Language', in *Talking Head Technology workshop, OZCHI 2001 conference*, Computer-Human Interaction Special Interest Group's (CHISIG), Perth Australia, 20-22 November 2001

Marriott, A 2002b, 'VHML', in *Special IST 5th Framework workshop on VHML*, European Union IST, Bruxelles, Belgium, 5-6 February 2002

Marriott, A 2001b, '*A Java Based Mentor System*', 2001. Online at http://www.cs.curtin.edu.au/~raytrace/papers/bookonjava/mentor/.

Marriott, A 2001c, 'Yackity yack, blah blah blah blah. Parlez vous? Dialogue Management for Talking Heads', in *Keynote address at OZCHI 2001 conference*, Computer-Human Interaction Special Interest Group's (CHISIG), Perth Australia, 20-22 November 2001

Marriott, A 1999a, 'A Lifelong Mentor System', in *Teaching in the Disciplines/ Learning in Context. Proceedings of the 8th Annual Teaching Learning Forum*, eds. K. Martin, N. Stanley & N. Davison, The University of Western Australia, Perth, Australia, 3-4 February 1999. Online at http://lsn.curtin.edu.au/tlf/tlf1999/contents.html.

Marriott, A 2004a, 'Mentor: "lotsa progress in future"', in *Proceedings of "Transforming Knowledge into Wisdom: Holistic Approaches to Teaching and Learning"*, Higher Education Research and Development Society of Australasia (HERDSA), Miri, Sarawak, July 4-7, 2004

Marriott, A 2004b, ‘You, by Proxy’, in *Proceedings of "Beyond the Comfort Zone", ASCILITE 2004 conference*, ASCILITE 2004, Perth, Western Australia, December 5-8, 2004

Marriott, A & Armarego, J 1997a, ‘Online Courses: A Continuing Education Support Network.’, *Proceedings of Australasian Joint Regional Conference of GASAT and IOSTE*, 5-8 Dec, 1997, Perth, Western Australia.. Online at http://www.cs.curtin.edu.au/~raytrace/papers/gasat/html/gasat.html.

Marriott, A & Beard, S 2002a, ‘VHML Standardisation’, in *Special IST 5th Framework meeting on Representation Formats/Languages*, European Union IST, Bologna, Italy, 18 July 2002

Marriott, A & Beard, S 2003a, ‘gUI. Specifying Complete User Interaction’, in *Life-Like Characters. Tools, Affective Functions and Applications*, ed. H. Prendinger, Springer-Verlag, Berlin, Germany, 2003, to be published

Marriott, A, Beard, S, Haddad, H, Pockaj, R, Stallo, J, Hyunh, Q & Tschirren, B 2001a, ‘The Face of the Future’, *Journal of Research and Practice in Information Technology*, vol. 32, August/November 2001, Australian Computing Society, pp.231-245

Marriott, A, Beard, S, Stallo, J & Huynh, Q 2001a, ‘VHML - Directing a Talking Head’, in *Active Media Technology LNCS 2252*, eds. J. Liu, P. C. Yuen, C. Li, J. Ng & T. Ishida, Springer-Verlag, Hong Kong, 2001, pp. 90-100

Marriott, A, Holic, A & Reid, D 2006a, ‘The Multi-modal Mentor System’, in *IUI workshop on Effective Multi-modal Dialogue Interfaces*, IUI2006, Intelligent User Interfaces Conference, Sydney, Australia, 29 January 2006. Online at http://www.iuiconf.org/.

Marriott, A & Ng, J 1997a, ‘The Village University’, *Proceedings of the University Teaching and Learning for Tomorrow’s World:The Asia-Pacific Experiences*, Aug, 1997, Malang Indonesia.. Online at http://www.cs.curtin.edu.au/~raytrace/papers/unibra/html/unibra.html.

Marriott, A & Pockaj, R 2001a, ‘The Virtual Lecturer - Distance Learning via a Talking Head’, in *Proceedings of Teaching and Learning Forum*, Curtin University of Technology, Perth, Australia, 7-9 February 2001. Online at http://cea.curtin.edu.au/tlf2001/program.html.

Marriott, A, Pockaj, R & Parker, C 2001a, ‘The Face of E-commerce’, in *Internet Commerce and Software Agents: Cases, Technologies and Opportunities*, eds. S. M. Rahman & R. J. Bignall, Idea Group Publishing, New York, 2001, pp. 290-315

Marriott, A & Shortland-Jones, B 2003a, ‘The Mentor System’, in *"Tutorial Dialogue Systems: With a View Towards the Classroom". Workshop Proceedings of 11th International Conference on Artificial Intelligence in Education*, eds. Carolyn Penstein Rosé & Vincent Aleven, AIED2003, Sydney, Australia, 18 July 2003, ISBN 1 86487 572 0

Marriott, A & Shortland-Jones, B 2004a, *Mentor: Really annoying. but quite helpful*, Teaching and Learning Forum, Perth, Western Australia

Marriott, A & Stallo, J 2002a, 'VHML- Uncertainties and Problems... A discussion', in *AAMAS workshop on Embodied conversational agents - let's specify and evaluate them!*, AAMAS 2002, Bologna, Italy, 16 July 2002. Online at http://www.vhml.org/workshops/AAMAS/.

Marriott, A, von Konsky, B & Ng, J 1997a, 'Education on the line...', *Third Hong Kong Web Symposium: Publishing on the line...*, 7-10 May 1997, Hong Kong.. Online at http://www.cs.curtin.edu.au/~raytrace/papers/ap97/html/ap97.html.

Mauch, J E & Birch, J W 1993a, *Guide to the Successful Thesis and Dissertation: Notes for Students and Their Supervisors*, Marcel Dekker, New York

Mauldin, M L 1994a, '*Chatterbots, Tinymuds, And The Turing Test: Entering The Loebner Prize Competition*', Jan 24, 1994. Online at http://www.lazytoad.com/lti/pub/aaai94.html.

Maulsby, D 1994a, 'Instructible Agents', Ph.D. Thesis, Stanford University, Stanford, CA 94305, June 1994. Online at ftp://ftp.cpsc.ucalgary.ca/pub/users/maulsby.

Maybury, M T 1998a, 'Intelligent user interfaces: an introduction', *Proceedings of the 4th International Conference on Intelligent User Interfaces*, 1998, Los Angeles, California, ISBN:1-58113-098-8, pp.3 - 4. Online at http://portal.acm.org/ft_gateway.cfm?id=291081&↵ type=pdf&coll=portal&dl=ACM&CFID=37248475&CFTOKEN=9878993.

Mayfield, J, Finin, T, Narayanaswamy, R, Shah, C, MacCartney, W & Goolsbey, K 1995a, 'The Cycic Friends Network: Getting Cyc agents to reason together', *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, December 1-2 1995, Baltimore, USA. Online at http://www.cs.umbc.edu/~finin/papers/cfn95.ps.

Mayfield, J, Labrou, Y & Finin, T 1996a, 'Evaluation of KQML as an Agent Communication Language', *Intelligent Agents Volume II--Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95) Lecture Notes in Artificial Intelligence Series.*, vol. 1037, 1996, Springer-Verlag, Heidelberg, ISBN 3-540-60805-2, pp.347-360. Online at http://www.cs.umbc.edu/~mayfield/pubs/atal95.ps.

Mayfield, J, Labrou, Y & Finin, T 1995a, 'Desiderata for Agent Communication Languages', in *Proceedings of the AAAI Symposium on Information Gathering from Heterogeneous, Distributed Environments, AAAI-95*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29, 1995. Online at http://www.cs.umbc.edu/kqml/papers/desiderata-acl/root.html.

McArthur, D 1992a, 'Some Possible Futures for Artificial Intelligence in Mathematics Education', in *Proceedings of Fifth Annual Conference on Technology in Collegiate Mathematics*, Chicago, U.S.A, November 1992. Online at http://www.rand.org/hot/mcarthur/Papers/future.html.

McArthur, D, Lewis, M W & Bishay, M 1993a, '*The Roles of Artificial Intelligence in Education: Current Progress and Future Prospects*', DRU-472-NSF, November 1993. Online at http://www.rand.org/hot/mcarthur/Papers/role.html.

McCarthy, A 2000a, '*Natural Language Processing*', 2000. Online at http://www.xciv.org/~andy/nlp/chomsky.pdf.

McCarthy, J 1987a, 'Generality in Artificial Intelligence', *Communications of the ACM*, vol. 30, no. 12, 1987, pp.1030-1035.

McElligott, M & Sorensen, H 1993a, 'An Emergent Approach to Information Filtering', *Abakus. U.C.C. Computer Science Journal*, vol. 1, no. 4, December 1993, Originally at http://odyssey.ucc.ie/filtering/ABAKUS93.ps

McElligott, M & Sorensen, H 1994a, 'An Evolutionary Connectionist Approach to Personal Information Filtering', in *INNC 94 (Fourth Irish Neural Network Conference)*, University College Dublin, September 1994, pp. 141-146, Originally at ftp://odyssey.ucc.ie/pub/filtering/INNC94.ps. Online at http://www.mentor.computing.edu.au/archives/innc94.pdf.

McKay, D, Pastor, J, McEntire, R & Finin, T 1996a, 'An architecture for information agents', *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, May 29-31 1996, AAAI, Edinburgh, Scotland, ISBN 0-929280-97-0. Online at http://www.cs.umbc.edu/kqml/papers/arpi.ps.

McTear, M F 2002a, 'Spoken Dialogue Technology: Enabling the Conversational User Interface', *ACM Computing Surveys (CSUR) (ISSN:0360-0300)*, 2002, ACM Press, New York, NY, USA, pp.1-85

Means, B, Blando, J, Olson, K, Middleton, T, Morocca, C, Remz, A & Zorfass, J 1993a, *Using Technology to Support Education Reform*, US Government Printing Office., Washington, USA

Menczer, F, Belew, R K & Willuhn, W 1995a, 'Artificial life applied to adaptive information agents', in *Proceedings of AAAI Spring Symposium on Information gathering from heterogeneous, distributed environments. AAAI, 1995*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, March 27-29 1995. Online at http://www-cse.ucsd.edu:80/users/rik/papers/aaai-sss95/info-spiders.ps.

Menezes, J 1999a, 'Retail only the tip of E-commerce iceberg', *Computing Canada*, vol. 25, no. 24, 1999, pp.11-13

Mercer, N 2000a, *Words and Minds: How We Use Language to Think Together*, Routledge, ISBN: 0415224756

Metral, M 1993a, *Design of a Generic Learning Interface Agent*, MIT Media Lab. Online at ftp://ftp.media.mit.edu/pub/agents/interface-agents/generic-agents.ps.Z.

Microsoft 1998a, 'Microsoft SQL Server White Paper', *The Journal of the Academy of Marketing Science*, 1998. Online at http://microsoft.com/sql/70/whpprs/eqp.htm.

Miller, G A, Beckwith, R, Fellbaum, C, Gross, D & Miller, K J 1990a, 'Introduction to WordNet: an on-line lexical database', *International Journal of Lexicography*, vol. 3, no. 4, 1990, pp.235-244

Miller, G A, Fellbaum, C & Miller, K J 1997a, '*Five papers on WordNet*', July 1997. Online at ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps.

Minsky, M 1985a, 'Why People Think Computers Can't', *AI Magazine*, vol. 3, no. 4, 1985. Online at http://web.media.mit.edu/~minsky/papers/ComputersCantThink.txt.

Mitchell, T, Caruana, R, Freitag, D, McDermott, J & Zabowski, D 1994a, 'Experience with a Learning Personal Assistant', *Communications of the ACM*, vol. 37, no. 7, July 1994, pp.81-91

Moshman 1982a, 'Exogenous, endogenous and dialectical constructivism', *Developmental Review*, vol. 2, 1982, pp.371-384

Moukas, A & Hayes, G 1996a, 'Synthetic Robotic Language Acquisition by Observation', in *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB'96)*, Cape Cod, USA, September 9-13, 1996, (submitted)

Muller, N J 1996a, '*Systems Management: The Emerging Role of Intelligent Agents*', Originally at http://www.ddx.com/agents.shtml, 1996. Online at http://www.mentor.computing.edu.au/archives/ddx_agents.shtml.

Murphy, E 1997a, '*Constructivism From Philosophy to Practice*', 1997. Online at http://www.stemnet.nf.ca/~elmurphy/emurphy/cle.html.

Murray, I & Arnott, J 1993a, 'Toward the Simulation of Emotion in Synthetic Speech: A Review of the Literature on Human Vocal Emotion', *Journal of the Acoustical Society of America*, vol. 2, 1993, pp.1097-1108

Murray, I, Arnott, J & Rohwer, E 1996a, 'Emotional stress in synthetic speech: Progress and future directions', *Speech Communication*, vol. 20, 1996, pp.85-91

Murray, J 1997a, *Hamlet*, Free Press, Simon and Schuster, New York

Nakabayashi, K, Koike, Y, Maruyama, M, Touhei, H, Ishiuchi, S & Fukuhara, Y 1995a, 'A Distributed Intelligent-CAI System on the World-Wide-Web', *International Conference on Computers in Education*, December 5 - 8, 1995, Singapore, Originally at http://calat.tas.ntt.co.jp/leaflet/eng/papers/icce95/

Nass, C, Steuer, J, Tauber, E R & Reeder, H 1993a, 'Anthropomorphism, Agency & Ethopoeia: Computers as Social Actors', in *INTERCHI'93 Conference Proceedings*, ACM/SIGCHI and IFIP, April 1993. Online at ftp://casa.stanford.edu/pub/papers/casa-chi93.ps.

Neal, L & Ingram, D 1999a, 'Asynchronous distance learning for corporate education: experiences with Lotus LearningSpace', *Proceedings of HCI (2) 1999*, 1999, pp.750-754

Neffenger, J 1997a, 'Results of first-ever JVM server benchmark revealed', *JavaWorld*, December 1997. Online at http://www.javaworld.com/javaworld/jw-12-1997/jw-12-volanomark.html.

Ng, J & Marriott, A 1994a, 'World Wide Web based Distance Learning', *International Networking: Education, Training and Change '94*, Sept, 1994, Edith Cowen University. Perth, Western Australia. Online at http://www.cs.curtin.edu.au/~raytrace/papers/ecu/ecu.ps.

Ng, J & Marriott, A 1995a, 'Developing, Assessing and Maintaining a Global Network Academy Computer Graphics Course.', *International Asia Pacific Rim WWW Conference '95*, 18-21 Sept 1995, IEEE. Sydney, Australia.. Online at http://www.csu.edu.au/special/conference/apwww95/papers95/amarriot/amarriot.html.

Ng, J & Marriott, A 1996a, 'A Survey of users of a Web-based Computer Graphics Course', *IEEE Computer Graphics and Applications - special edition on Computer Graphics Education.*, 1996, IEEE. Perth, Western Australia. Online at http://www.computer.org/pubs/cg&a/cged/ng/.

Niemiec, R & Walberg, H J 1987a, 'Comparative effects of computer-assisted instruction: A synthesis of reviews', *Journal of Educational Computing Research*, vol. 3, 1987, pp.19-37

NOIE 1994a, *Style Manual for Authors, Editors and Printers*, National Office for the Information Economy. Online at http://www.noie.gov.au/projects/egovernment/Better_Information/style_manual.html.

Noot, H & Ruttkay, Z 2003a, 'Style in Gesture', *Proc. of Gesture'2003*, 2003, Springer-Verlag LNCS Series

Norrie, D H & Gaines, B R 1995a, '*The Learning Web: A System View and an Agent-Oriented Model*', 1995. Online at http://ksi.cpsc.ucalgary.ca/articles/LearnWeb/EM95J/EM95J.html.

Nwana, H S 1996a, 'Software Agents: An Overview', in *Knowledge Engineering Review*, Intelligent Systems Research,Advanced Applications & Technology Department,BT Laboratories, September 1996, pp. .1-40. Online at http://www.cs.umbc.edu/agents/introduction/ao/.

Oates, T, Prasad, M V N, Lesser & R, V 1994a, 'Cooperative Information Gathering: A Distributed Problem Solving Approach', *UMASS Technical Report 94-66*, 1994. Online at ftp://ftp.cs.umass.edu/pub/lesser/oates-94-66.ps.

O'Connor, D C 1995a, 'A System to Facilitate Teaching and Learning with Network-based Interactive Multimedia', *DAGS95: Electronic Publishing and the Information Superhighway*, May 30-June 2, 1995, Boston, USA, Originally at http://awi.aw.com/DAGS95/Papers/oconnor.html. Online at http://www.mentor.computing.edu.au/archives/oconnor.html.

OMG 1992a, *Object Analysis and Design, Volume 1: Reference model Draft 7.0*, Object Management Group

Ordille, J J 1995a, 'Information Gathering and Distribution in Nomenclator', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, Mar 27-29 1995. Online at ftp://ftp.isi.edu/sims/sss95/ordille.ps.Z.

Palmer, M & Finin, T 1990a, 'Workshop on the Evaluation of Natural Language Processing Systems', *Computational Linguistics*, vol. 16, no. 3, September 1990

Pandzic, I & Forchheimer, R 2002a, *MPEG-4 Facial Animation - The standard, implementations and applications*, John Wiley, London

Papert, S 1988a, 'One AI or Many?', in *The Artificial Intelligence Debate: False Starts, Real Foundations*, ed. Stephen R. Graubard, MIT Press, Cambridge, MA, 1988

Parr, J M & Fung, I 2004a, *A Review of the Literature on Computer-Assisted Learning, particularly Integrated Learning Systems, and Outcomes with Respect to Literacy and Numeracy*, New Zealand Ministry of Education, ISBN: 0-477-05196-0. Online at http://www.minedu.govt.nz/index.cfm?layout=document&documentid=5499↵ &CFID=3740596&CFTOKEN=81582330.

Patel, A, Kinshuk & Russell, D 2000a, 'Intelligent Tutoring Tools for Cognitive Skill Acquisition in Life Long Learning', *Educational Technology & Society*, vol. 3, no. 1, 2000, ISSN 1436-4522

Patel, A, Kinshuk & Russell, D 2002a, 'Implementing Cognitive Apprenticeship and Conversation Theory in Interactive Web-Based Learning Systems', in *Sixth Multi-Conference on Systemics, Cybernetics and Informatics*, eds. N. Callaos, M. Loutfi & M. Justan, International Institute of Informatics and Systemics, Orlando, Florida, July 14-18, 2002,, pp. 523-528, ISBN 980-07-8150-1

Patrick, A & Whalen, T 1992a, 'Field testing a natural-language information system: usage characteristics and users' comments', *Interacting with Computers*, vol. 4, no. 2, 1992, Butterworth-Heinemann Ltd, New York, USA

Pazzani, M, Nguyen, L & Mantik, S 1996a, '*Learning from hotlists and coldlists:Towards a WWW information filtering and seeking agent*', jan 1996. Online at http://www.ics.uci.edu/˜pazzani/Coldlist.html.

Pea, R 1993a, 'Seeing what we build together: Distributed multimedia learning environments for transformative communications', *The Journal of the Learning Sciences*, vol. 3, no. 3, 1993, pp.285-299

Pelachaud, C, Ruttkay, Z & Marriott, A 2003a, *Embodied Conversational Characters as Individuals*, Workshop for the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems, Melbourne, Australia. Online at http://www.vhml.org/workshops/AAMAS2003/..

Perry, C 1998a, 'A Structured Approach to Presenting Theses: Notes for Students and Their Supervisors', *Australasian Marketing Journal*, vol. 6, no. 1, 1998, Revised on 5 September 2002, pp.63-86. Online at http://www.scu.edu.au/schools/gcm/ar/art/cperry.html.

Plantec, P 1999a, '*The zen of scripting verbots*', 5 June 1999. Online at http://www.vperson.com/verbot30tt.html.

Poleretzy, Z, Cohn, R & Gimnicher, S M 1999a, 'The Call Center & E-commerce Convergence', *Call Center Solutions*, vol. 17, no. 7, 1999, pp.76-89

Popescu, O, Aleven, V & Koedinger, K R 2003a, 'A Knowledge-based Approach to Understanding Students' Explanations', in *AIED2003 Workshop Proceedings*, ed. Vincent Aleven, School of Information Technologies, University of Sydney, Sydney, Australia, 18 July 2003, ISBN 1 86487 572 0

Postel, J 1981a, 'Internet Control Message Protocol', Request for Comments (Standard) RFC 792, Internet Engineering Task Force, September 1981. Online at ftp://ds.internic.net/rfc/rfc792.txt.

Project, T 1995a, '*Information Resources Related to the TAP Project*', Originally at http://jupiter.sas.ntu.ac.sg:8000/research/ai_its.html, 1995. Online at http://www.mentor.computing.edu.au/archives/TAP_its.html.

Project, T 1995b, '*TAP: A Software Architecture for an Inquiry Dialogue-based Tutoring System*', Originally at http://jupiter.sas.ntu.ac.sg:8000/research/tap.html, 1995. Online at http://www.mentor.computing.edu.au/archives/TAP_tap.html.

Psotka, J 1995a, '*Immersive Tutoring Systems: Virtual Reality and Education and Training*', Originally at http://205.130.63.7/its.html, 1995

Psotka, J, Massey, L D & Mutter, S A 1988a, *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum, Hillsdale, USA

Rabkin, B & Tingley, M 1999a, 'Tech-savvy customers want quick response', *National Underwriter*, vol. 103, no. 40, 1999, pp.12-21

Rao, R, Card, S K, Jellinek, H D, Mackinlay, J D & Robertson, G G 1992a, 'The Information Grid: A Framework for Building Information Retrieval and Retrieval-Centered Applications', in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)*, ACM Press, Monteray, USA, November 15-18 1992, Originally at http://pubweb.parc.xerox.com/hypertext/dlbx/uir-papers/uist92-final.ps. Online at http://www.mentor.computing.edu.au/archives/uist92-final.ps.gz.

Raphael, B 1964a, *SIR: A Computer Program for Semantic Information Retrieval*, MIT Laboratory for Computer Science, Cambridge, MA, MIT-LCS-TR-002. Online at http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-002.pdf.

Reeves, B & Nass, C 1996a, *The Media Equation*, Cambridge University Press

Reichgelt, H, Shadbolt, N, Paskiewicz, T, Wood, D & Wood, H 1996a, '*EXPLAIN: On implementing more effective tutoring systems*', Originally at http://www.psyc.nott.ac.uk/aigr/research/its/EXPLAIN/new-paper.html, 1996

Reilly, W S 1996a, 'Believable Social and Emotional Agents.', Ph.D. Thesis, School of Computer Science,Carnegie Mellon University, Pittsburgh, PA., May 1996. Online at http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-96-138-1sided.ps.gz.

Reilly, W S & Bates, J 1992a, 'Building Emotional Agents', Technical Report CMU--CS--92--143, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, May 1992. Online at http://www.cs.cmu.edu:80/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-92-143.ps.

Reilly, W S & Bates, J 1995a, 'Natural Negotiation for Believable Agents', Technical Report, Jun, 1995. Online at http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-95-164.ps.gz.

Reinhardt, A 1995a, 'New Ways to Learn', *Byte Magazine*, March 1995, BYTE

Reiser, B, Kimberg, D, Lovett, M & Ranney, M 1992a, 'Knowledge representation and explanation in GIL, an intelligent tutor for programming', in *Computer Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches*, eds. J. Larkin & R. Chaby, Lawrence Erlbaum Associates., Hillsdale, NJ, 1992, pp. 111-149

Reynolds, C J 2001a, '*The Sensing and Measurement of Frustration with Computers*', 2001. Online at http://www.media.mit.edu/~carsonr/pdf/sm_thesis.pdf.

Rhodes, B J & Maes, P 1995a, 'The Stage as a Character Automatic Creation of Acts of God for Dramatic Effect', *Presented at the AAAI '95 Spring Symposium on Interactive Story Systems*, March 1995, Stanford University, USA. Online at http://rhodes.www.media.mit.edu/people/rhodes/Papers/aaai95.html.

Rhodes, B J & Starner, T 1996a, 'Remembrance Agent: A continuously running automated information retrieval system', *Presented at AAAI-Spring Symposium '96: Acquisition, Learning and Demonstration: Automating Tasks for Users*, March 1996, Stanford University, USA. Online at http://rhodes.www.media.mit.edu/people/rhodes/Papers/remembrance.html.

Rich, C, Sidner, C L & Lesh, N 2001a, *COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction*, AAAI. Online at http://www.aaai.org/Library/Magazine/Vol22/22-04/vol22-04.html.

Richardson, R & Smeaton, A F 1995a, 'Using WordNet in a Knowledge-Based Approach to Information Retrieval', Technical Report CA-0395, Dublin, Ireland, 1995. Online at citeseer.ist.psu.edu/richardson95using.html.

Rickel, J & Johnson, W L 1997a, 'Intelligent Tutoring in Virtual Reality: A Preliminary Report', in *Proceedings of the Eighth World Conference on AI in Education*, Kobe, Japan, August 18-22 1997. Online at http://www.isi.edu/isd/VET/ai-ed97.pdf.

Rickel, J & Johnson, W L 1997b, 'Mixed-Initiative Interaction between Pedagogical Agents and Students in Virtual Environments', in *AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, Stanford University, USA, March 24-26 1997. Online at http://www.isi.edu/isd/rickel/mii97.ps.

Rist, T, Pelachaud, C, Ruttkay, Z, Marriott, A & Vilhjalmsson, H 2002a, *Embodied conversational agents - let's specify and evaluate them!*, Workshop for the First International Joint Conference on Autonomous Agents & Multi-Agent Systems, Bologna, Italy. Online at http://www.vhml.org/workshops/AAMAS/.

Ritter, S & Blessing, S B 1995a, 'Controlling content: Towards a domain general authoring framework for intelligent tutors', *In N. Major, T. Murray, and C. Bloom (Eds.), Workshop on Authoring Shells for Intelligent Tutoring Systems*, 1995, Originally at http://nersp.nerdc.ufl.edu/˜sbless/abstracts.html#Controller, pp.78-81

Rosé, C P 2000a, 'Facilitating the Rapid Development of Language Understanding Interfaces for Tutoring Systems', *Proceedings of the AAAI Fall Symposium on Building Tutorial Dialogue Systems*, November 3-5, 2000, North Falmouth, Massachusetts. Online at http://www.cs.cmu.edu/˜cprose/dialsymp.ps.

Rosé, C P 1997a, 'Robust Interactive Dialogue Interpretation', *Ph.D. Dissertation*, 1997, School of Computer Science, Carnegie Mellon University. Online at http://www-2.cs.cmu.edu/˜cprose/dissertation.ps.gz.

Rosé, C P & Aleven, V 2002a, 'ITS2002', in *Proceedings of the ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems*, San Sebastion, Spain, June 4 2002. Online at http://www-2.cs.cmu.edu/˜aleven/ITS2002DialogueWS/ITS2002W6Proceedings.pdf.

Rosé, C P, Moore, J D, VanLehn, K & Albritton, D 2001a, 'A comparative evaluation of socratic versus didactic tutoring', *23rd Annual Conference of the Cognitive Science Society*, August 2001, Edinburgh, Scotland. Online at http://www.lrdc.pitt.edu/pdf%20files/cog.pdf.

RSA Laboratories 1993a, '*RSAREF(TM): A Cryptographic Toolkit for Privacy-Enhanced Mail*', Jan 1993. Online at http://www.rsasecurity.com/products/bsafe/bsafe.html.

RSS 2002a, *RSS at Harvard Law*, RSS. Online at http://blogs.law.harvard.edu/tech/rss.

Ruttkay, Z 2003a, '*Personal email.*', 11 Aug 2003

Ruttkay, Z, Moppes, V. van & Noot, H 2003a, 'The jovial, the reserved and the robot', *Proceedings of AAMAS2003 workshop on Embodied Conversational Characters as Individuals*, 2003. Online at http://www.vhml.org/workshops/AAMAS2003/papers.html.

Ruttkay, Z, Pelachaud, C, Poggi, I & Noot, H 2003a, 'Exercises of Style for Virtual Humans', in *Advances in Consciousness Research Series*, eds. L. Canamero & R. Aylett, John Benjamins Publishing Company, 2003

Sarkar, A 2003a, '*CMPT 825 Natural Language Processing*', 2003. Online at http://www.cs.sfu.ca/~anoop.

Schank, R & Cleary, C 1994a, *Engines for Education*, Lawrence Erlbaum Associates, Inc, Hillsdale, NJ. Online at http://www.engines4ed.org/hyperbook/.

Schank, R & Edelson, D 1990a, 'A Role for AI in Education: Using Technology to Reshape Education', *Journal of Artificial Intelligence in Education*, vol. 1, no. 2, 1990, pp.3-20

Schank, R C 1972a, 'Conceptual Dependency: A Theory of Natural Language Understanding', *Cognitive Psychology*, vol. 3, no. 4, 1972, pp.pages 532-631

Schank, R C 1993a, 'Learning via Multimedia Computers', *Communications of the ACM*, vol. 36, no. 5, May 1993, pp.54--56

Schank, R C & Abelson, R P 1977a, *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*, L. Erlbaum, Hillsdale, NJ

Schank, R C, Goldman, N M, Rieger, C & Riesbeck, C K 1973a, 'MARGIE: Memory, analysis, response generation, and inference on English', *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI)*, August 1973, Stanford, USA, pp.255-261

Schank, R C, Goldman, N M, Rieger, C & Riesbeck, C K 1975a, 'Inference and paraphrase by computer', *Journal of the ACM*, vol. 22, no. 3, 1975, pp.309-328

Schank, R C & Kass, A 1987a, 'Natural Language Processing: What's Really Involved?', *Theoretical Issues in Natural Language Processing 3*, 7-9 January 1987, Las Cruces, New Mexico. Online at http://acl.ldc.upenn.edu/T/T87/T87-1023.pdf.

Scriven, M 1991a, *Evaluation Thesaurus, 4th ed*, Sage Publications, Newbury Park, CA

Self, J 1988a, *Artificial intelligence and human learning : intelligent computer-aided instruction*, Chapman and Hall, London, ISBN:0412166100

Shardanand, U & Maes, P 1995a, 'Social Information Filtering: Algorithms for Automating 'Word of Mouth'', *Proceedings of the Conference on Human Factors in Computing Systems (CHI-95C) Conference*, May 7-11 1995, ACM, Denver, USA. Online at http://lcs.www.media.mit.edu/groups/agents/papers/ringo/chi-95-paper.ps.

Sheth, B D 1994a, 'A Learning Approach to Personalized Information Filtering', M.S. Thesis, MIT Media Lab, January 1994. Online at ftp://ftp.media.mit.edu/pub/agents/interface-agents/news-filter.ps.Z.

Shoham, Y 1991a, 'AGENT0: A simple agent language and its interpreter', in *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Anaheim, CA, July 14-19 1991, pp. 704-709

Shulman, L S 1986a, ‘Paradigms and Research Programs in the Study of Teaching’, in *Handbook of Research on Teaching: A Project of the American Educational Research Association, 3rd Edition*, ed. Merlin C. Wittrock, Macmillan Publishing Co, New York, 1986, pp. 3-36

Shulman, S P 1995a, ‘The Role of Intelligence in a Multimedia Tutoring System for Basic Computer Hardware’, *Journal of Computer-Based Instruction*, 1995, Originally at http://alaspooryorick.ee.wits.ac.za/scorpion/paper.html

Shute, V.J & Psotka, J 1996a, ‘Intelligent Tutoring Systems: Past, Present and Future’, in *Handbook of research on Educational Communications and Technology*, ed. D. Jonassen, Scholastic Publications, 1996, pp. 570-600

Sleeman, D & Brown, J.S 1982a, *Intelligent Tutoring Systems*, Academic Press, London

Sloman, A 1996a, ‘What sort of architecture is required for a human-like agent?’, *Invited talk at Cognitive Modeling Workshop, AAAI96*, August 1996, Portland USA. Online at http://www.cs.bham.ac.uk/~axs/misc/agent.architecture/agent.architecture.html.

Smeaton, A F 1991a, ‘Using Hypertext for Computer Based Learning’, *Computers & Education*, vol. 17, no. 3, 1991, pp.173--179

SMIL2.0 2001a, *Synchronized Multimedia Integration Language (SMIL 2.0)*, W3C. Online at http://www.w3.org/TR/smil20..

Soller, A, Lesgold, A, Linton, F & Goodwin, B 1999a, ‘What Makes Peer Interaction Effective? Modeling Effective Communication in an Intelligent CSCL’, in *Working Papers of the {AAAI} Fall Symposium on Psychological Models of Communication in Collaborative Systems*, ed. Susan E. Brennan and Alain Giboin and David Traum, American Association for Artificial Intelligence, Menlo Park, California, 1999, pp. 116-124. Online at http://citeseer.ist.psu.edu/208051.html.

Song, H, Franklin, S & Negatu, A 1996a, ‘SUMPY: A Fuzzy Software Agent’, *Proceedings of the ISCA Conference on Intelligent Systems*, June 1996, Reno USA. Online at http://www.msci.memphis.edu/~franklin/sumpy.html.

Sparks, R, Dooley, S, Meiskey, L & Blumenthal, R 1999a, ‘The LEAP Authoring Tool: Supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations’, *International Journal of Artificial Intelligence in Education*, vol. 10, 1999, pp.75-97. Online at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_10/sparks.html.

Sprinthall, R C, Schmutte, G T & Sirois, L 1991a, *Understanding Education Research*, Prentice Hall, Englewood Cliffs, USA, ISBN 0-13-945973-1

Stallo, J 2000a, ‘Simulating Emotional Speech for a Talking Head’, in *Honours Thesis*, School of Computing, Curtin University of Technology, Perth, Western Australia, December 2000. Online at http://www.vhml.org/theses/stalloj.

Standardization, I O. for 1985a, *Information processing: text and office systems: Standard Generalized Markup Language (SGML).*, ISO, Geneva, Switzerland, Cover title. ISO/DIS 8879. Draft international standard. ISO/DIS 8879

Stefani, A & Strapparava, C 1998a, 'Personalizing Access to Web Sites: The SiteIF Project.', in *Proceedings of 2nd Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98*, Pittsburgh, USA, June 20-24 1998. Online at http://www.contrib.andrew.cmu.edu/~plb/HT98_workshop/Stefani/Stefani.html.

Steier, D 1995a, 'Comparable Datasets in Performance Benchmarking', in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, eds. C. Knowblock, A. Levy, S-S. Chen & G. Wiederhold, AAAI, Stanford University, Mar 27-29 1995. Online at http://www.pwcglobal.com/extweb/service.nsf/↵ 8b9d788097dff3c9852565e00073c0ba/↵ 2d398d3f70dd2c26852566ab005a87ee/$FILE/Tr34.pdf.

Steier, D, Huffman, S B & Hamscher, W C 1995a, 'Meta-information for knowledge navigation and retrieval: What's in there', *Working notes of the 1995 AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, Oct 1995. Online at http://www.pwcglobal.com/extweb/service.nsf/↵ 8b9d788097dff3c9852565e00073c0ba/↵ 2d398d3f70dd2c26852566ab005a87ee/$FILE/Tr38.pdf.

Streitz, N 1988a, 'Mental Models and Metaphors: Implications for the Design of Adaptive User-System Interfaces', in *Learning issues for intelligent tutoring systems*, eds. H. Mandl & A. Lesgold, Springer-Verlag, New York, 1988

Sycara, K, Decker, K, Pannu, A, Williamson, M & Zeng, D 1996a, 'Distributed Intelligent Agents', *IEEE Expert*, vol. 11, no. 6, Dec 1996, citeseer.nj.nec.com/sycara96distributed.html, pp.36-46

Takeda, H, Iino, K & Nishida, T 1995a, 'Ontology-supported agent communication', *In Working notes of 1995 AAAI Spring Symposium on Information Gathering in distributed environments*, 1995, pp.160-165. Online at ftp://ftp.isi.edu/sims/sss95/takeda.ps.Z.

Tan, S H 2005a, 'Rexpression: Improving Matching for Script Based Dialogue Managers', in *Honours Thesis*, School of Computing, Curtin University of Technology, Perth, Western Australia, December 2005. Online at http://www.vhml.org/theses/rex.

Trindi Consortium 2001a, *The Trindi Book*, The Trindi Consortium, Task Oriented Instructional Dialogue, LEA-8314, Department of Linguistics, Göteborg University,Sweden

Turing, A M 1950a, 'Can a Machine Think?', *Mind*, October 1950, pp.433-460

VanLehn, K, Freedman, R, Jordan, P, Murray, C, Osan, R, Ringenberg, M, Rosé, C P, Schulze, K, Shelby, R, Treacy, D, Weinstein, A & Wintersgill, M 2000a, 'Fading and Deepening: The Next Steps for ANDES and Other Model-Tracing Tutors', in *In Intelligent Tutoring Systems: Fifth International Conference, ITS 2000*, eds. G. Gauthier, C. Frasson & K. VanLehn, Springer-Verlag., Berlin, 2000, pp. 474−483

von Konsky, B 1996a, 'Using the World Wide Web as a Delivery Mechanism for Distributed Educational Multimedia', in *Proceedings of Third Interactive Multimedia Symposium*, Perth, Western Australia, January 1996, pp. 203-212. Online at http://www.cs.curtin.edu.au/~bvk/iimms96/iimms96.html.

Vygotsky, L S 1978a, *Mind in Society*, Harvard University Press, Cambridge, MA

Wærn, A 1996a, 'Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction', *Ph.D. Thesis*, 1996, SICS Dissertation Series 20, Stockholm, Sweden, ISBN 91-7153-456-3

Wærn, Y, Hägglund, S, Löwgren, J, Rankin, I, Sokolnicki, T & Steinemann, A 1990a, 'Communication Knowledge for Knowledge Communication', *Research report, LiTH-IDA-R-90-17*, 1990, Linköpings University, Sweden

Wall, L & Schwartz, R L 1992a, *Programming Perl, 2nd Edition*, O'Reilly Associates, Inc.

Watt, S, Zdrahal, Z & Brayshaw, M 1995a, 'Multiple Agent Systems for Configuration Design', *TR number: KMI-TR-11. Published in the proceedings of AISB'95. Sheffield, UK*, April 1995. Online at http://kmi.open.ac.uk/kmi-abstracts/kmi-tr-11-abstract.html.

Weaver, W 1955a, 'Translation', in *Machine Translation of Languages*, eds. W. N. Locke & A. D. Booth, John Wiley & Sons, New York, 1955, pp. 15-23

Weischedel, R M 1983a, 'Handling Ill-Formed Input: Session Introduction', *Proceedings of the 1983 First Applied Natural Language Processing Conference*, 1-3 February 1983, Santa Monica, California. Online at http://acl.ldc.upenn.edu/A/A83/A83-1014.pdf.

Weizenbaum, J 1976a, *Computer Power and Human Reason*, W.H. Freeman and Company, New York, USA

Wells, J & Fuerst, W 2000a, 'Domain-Oriented Interface Metaphors: Designing Web Interfaces for Effective Customer Interaction', *33rd Hawaii Conference on System Sciences*, 2000, Hawaii

Wentland, E J & Smith, K W 1993a, *Survey Responses: An Evaluation of their Validity*, Academic Press, Inc., San Diego, California

Whalen, T 1999a, '*RE: Any recent "COMODA" type projects?*', Personal email communication, April 28 1999

Whalen, T 1996a, '*Computational Behaviorism Applied to Natural Language*', 1996. Online at http://debra.dgbt.doc.ca/chat/chat.theory.html.

Whalen, T & Patrick, A 1990a, 'COMODA: a conversational model for Database Access', *Behavior and Information Technology*, vol. 9, no. 2, 1990, Taylor and Francis, New York, USA, pp.93-100

White, J E 1996a, ‘Mobile Agents White Paper’, White Paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040, 1996, Originally at http://www.genmagic.com/Telescript/. Online at http://www.mentor.computing.edu.au/archives/White-Telescript.pdf.

Wiemer-Hastings, P, Allbritton, D & Arnott, E 2005a, ‘Initial results and mixed directions for Research Methods Tutor’, in *In AIED2005 - Supplementary Proceedings of the 12th International Conference on Artificial Intelligence in Education*, Amsterdam, 2005

Wiley, D L 1998a, ‘Beyond information retrieval’, *Econtent*, vol. 21, no. 4, 1998, pp.18-22

Wilkie, G 1993a, *Object-Oriented Software Engineering - The Professional Developer’s Guide*, Addison-Wesley, Sydney

Wills, S & McNaught, C 1996a, ‘Evaluation of Computer Based Learning in Higher Education’, *Journal of Computing in Higher Education*, vol. 7, no. 2, 1996, pp.106-128. Online at http://ncode.uow.edu.au/info/orig/JCHEeval.html.

Winograd, T 1972a, *Understanding Natural Language*, Academic Press, New York

Winograd, T 1971a, *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, MIT AI Technical Report 235

Winograd, T & Flores, C F 1986a, *Understanding Computers and Cognition: a new foundation for design*, Addison Wesley, Reading, MA

Winston, P 1987a, ‘Artificial Intelligence: A Perspective’, in *AI in the 1980s and Beyond: an MIT Survey*, eds. W. Eric L. Grimson & Ramesh S. Patil, MIT Press, Cambridge, MA, 1987, pp. 2-3

Wonisch, D & Cooper, G 2002a, ‘Interface Agents: preferred appearance characteristics based upon context’, *Virtual Conversational Characters Workshop, (Human Factors 2002)*, 2002, Melbourne, Australia

Wood, A 1993a, ‘*Desktop Agents*’, April 1993. Online at ftp://ftp.cs.bham.ac.uk/pub/dist/hci/papers/desktop_agents.

Wood, A 1994a, ‘Towards a Medium for Agent-Based Interaction’, Thesis Proposal PR-94-15, University of Birmingham, School of Computer Science, October 1994. Online at ftp://ftp.cs.bham.ac.uk/pub/dist/hci/papers/medium_for_abi_PR-94-15.ps.Z.

Wood, A 1994b, ‘Agent-Based Interaction’, PhD Progress Report PR-94-4, University of Birmingham, School of Computer Science, May 1994. Online at ftp://ftp.cs.bham.ac.uk/pub/dist/hci/papers/agent-bi_PR-94-4.ps.Z.

Wooldridge, M 1992a, ‘The Logical Modelling of Computational Multi-Agent Systems’, Ph.D. Thesis, Department of Computation, UMIST, Manchester, UK, October 1992. Online at http://www.csc.liv.ac.uk/˜mjw/pubs/thesis.ps.gz.

Wooldridge, M & Jennings, N R 1995a, '*Intelligent Agents: Theory and Practice*', Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/KE-REVIEW-95.ps, 1995. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Wooldridge, M & Jennings, N R 1995b, 'Agent Theories, Architectures, and Languages: A Survey', in *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, ed. M. Wooldridge and N. R. Jennings, Springer-Verlag: Heidelberg, Germany, January 1995, pp. 1-39, Originally at ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/ECAI94-WS.ps.Z. Online at http://www.iam.ecs.soton.ac.uk/publications/byauthor/nrj.html.

Woolf, B 1992a, 'Building knowledge based tutors', in *Computer assisted learning: Proceedings of the Fourth International Conference, ICCAL '92*, ed. I. Tomek, Wolfville, Nova Scotia, 1992, pp. 46-60

Woolf, B P & Hall, W 1995a, 'Multimedia Pedagogues: Interactive Systems for Teaching and Learning', *Computer*, vol. 28, no. 5, 1995, pp.74-80. Online at http://computer.org/computer/co1995/r5074abs.htm.

Wreggit, D J 1995a, '*Software Agents Using Java*', Originally at http://www.saic.alaska.net/wreggit/djw_paper.html, December 4, 1995. Online at http://www.mentor.computing.edu.au/archives/djw_paper.html.

Xiao, H 2006a, 'An Affective Personality Model for an Embodied Conversational Agent', in *Masters Thesis*, Deaprtment of Computing, Curtin University of Technology, Perth, Western Australia, March 2006. Online at http://www.vhml.org/theses/xiao.

Xiao, H, Reid, D, Marriott, A & Gulland, E K 2005a, 'An Adaptive Personality Model for ECAs', in *The First International Conference on Affective Computing & Intelligent Interaction*, LNCS, Springer-Verlag, Beijing, China, October 22-24, 2005. Online at http://www.affectivecomputing.org/2005/.

XML 1997a, *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium (W3C), Accessed April 1997. Online at http://www.w3.org/XML/.

Zhou, Y, Freedman, R, Glass, M, Michael, J A, Rovick, A A & Evens, M W 1999a, 'Delivering Hints in a Dialogue-Based Intelligent Tutoring System', *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999, Orlando, Florida

Zimmerman, P 1994a, *The Official PGP User's Guide*, MIT, prz@acm.org, Originally at http://www.pegasus.esprit.ec.org/people/arne/pgp.html. Online at http://www.mentor.computing.edu.au/archives/pgpdoc-html.tar.gz.

Zinn, C, Moore, J D & Core, M G 2002a, '*A 3-tier Planning Architecture for Managing Tutorial Dialogue*', 2002. Online at citeseer.ist.psu.edu/598920.html.

What information consumes is rather obvious:
it consumes the attention of its recipients.
Hence a wealth of information creates a poverty
of attention, and a need to allocate that attention
efficiently among the overabundance of
information sources that might consume it.
Herbert Simon

Hypertext References

The following hypertext references contain Uniform Resource Locators (URLs) that were valid on 1st March 2003.

• [HREF 1]. *Circsim*
http://www.cs.iit.edu/~circsim/

• [HREF 2]. *Atlas*
http://www.pitt.edu/~circle/Projects/Atlas.html

• [HREF 3]. *www.macquariedictionary.com.au*
http://www.macquariedictionary.com.au:8008/

• [HREF 4]. *http://www.interface.computing.edu.au/*
http://www.interface.computing.edu.au/

• [HREF 5]. *Mike Wooldridge's website*
http://www.csc.liv.ac.uk/~mjw/links/

• [HREF 6]. *agents mailing list*
http://www.cs.umbc.edu/agentslist/

• [HREF 7]. *AgentNews Webletter*
http://www.cs.umbc.edu/agents/agentnews/

• [HREF 8]. *MIT Media Laboratory*
http://agents.www.media.mit.edu/groups/agents/

• [HREF 9]. *Carnegie Mellon*
http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/oz.html

• [HREF 10]. *Agent ART Group*
http://www.csc.liv.ac.uk/research/agents/

• [HREF 11]. *http://www.iam.ecs.soton.ac.uk/*
http://www.iam.ecs.soton.ac.uk/

- [HREF 12]. *Knowledge Interchange Format (KIF)*
  http://www.cs.umbc.edu/kse/kif/
- [HREF 13]. *Knowledge Query and Manipulation Language (KQML)*
  http://www.cs.umbc.edu/kqml/
- [HREF 14]. *Aglets*
  http://www.aglets.org
- [HREF 15]. *Javaworld.com*
  www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html
- [HREF 16]. *http://vperson.com/*
  http://vperson.com/
- [HREF 17]. *http://acl.ldc.upenn.edu/*
  http://acl.ldc.upenn.edu/
- [HREF 18]. *WebWatcher Project*
  http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/
- [HREF 19]. *http://um.org/*
  http://um.org/
- [HREF 20]. *Australian Journal of Educational Technology*
  http://www.ascilite.org.au/ajet/ajet.html
- [HREF 21]. *Journal of Educational Multimedia and Hypermedia*
  http://www.aace.org/pubs/jemh/
- [HREF 22]. *Journal of Interactive Learning Research*
  http://www.aace.org/pubs/jilr/
- [HREF 23]. *Journal of Technology and Teacher Education*
  http://www.aace.org/pubs/jtate/
- [HREF 24]. *Journal of the Asynchronous Learning Network*
  http://www.aln.org/publications/jaln
- [HREF 25]. *International Journal on E-Learning (IJEL)*
  http://www.aace.org/pubs/ijel/
- [HREF 26]. *Educational Technology Review (ED-TECH Review)*
  http://www.aace.org/pubs/etr/
- [HREF 27]. *Journal of Educational Computing Research*
  http://baywood.com/journals/PreviewJournals.asp?Id=0735-6331
- [HREF 28]. *EDUCAUSE*
  http://www.educause.edu/

- [HREF 29]. *Cyc*
  http://www.cyc.com/cyc/technology/whatiscyc_dir/whatsincyc
- [HREF 30]. *Google*
  http://www.google.com/
- [HREF 31]. *http://www.stanford.edu/group/SHR/4-2/text/dialogues.html*
  http://www.stanford.edu/group/SHR/4-2/text/dialogues.html
- [HREF 32]. *SHRDLU*
  http://hci.stanford.edu/~winograd/shrdlu/
- [HREF 33]. *START*
  http://www.ai.mit.edu/projects/infolab/start.html
- [HREF 34]. *the Simon Laven Chatterbot website (http://www.simonlaven.com/)*
  http://www.simonlaven.com/
- [HREF 35]. *ftp://nl.cs.cmu.edu/usr/mlm/ftp/robot.tar.Z*
  ftp://nl.cs.cmu.edu/usr/mlm/ftp/robot.tar.Z
- [HREF 36]. *The Loebner Contest*
  http://www.surrey.ac.uk/dwrc/loebner/
- [HREF 37]. *Turing Test*
  http://cogsci.ucsd.edu/~asaygin/tt/ttest.html
- [HREF 38]. *Loebner Prize*
  http://www.loebner.net/Prizef/loebner-prize.html
- [HREF 39]. *Alice*
  http://www.alicebot.org/
- [HREF 40]. *AIML*
  http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html
- [HREF 41]. *Ella*
  http://www.ellaz.com/AI/
- [HREF 42]. *George*
  http://www.jabberwacky.com/
- [HREF 43]. *Jabberwock*
  http://www.abenteuermedien.de/jabberwock/
- [HREF 44]. *Ella*
  http://www.ellaz.com/
- [HREF 45]. *Albert One*
  http://robitron.dynip.com/Studio/Albert.html

- [HREF 46]. *Jakarta Apache project*
  http://jakarta.apache.org/oro/
- [HREF 47]. *OROMatcher project*
  http://www.savarese.org/oro/docs/OROMatcher/Syntax.html
- [HREF 48]. *http://www.regular-expressions.info/*
  http://www.regular-expressions.info/
- [HREF 49]. *http://search.cpan.org/dist/perl/pod/perlre.pod*
  http://search.cpan.org/dist/perl/pod/perlre.pod
- [HREF 50]. *Conceptual graphs*
  http://www.jfsowa.com/cg/cgstandw.htm
- [HREF 51]. *Cyc system*
  http://www.cyc.com/cyc/
- [HREF 52]. *http://lsa.colorado.edu/whatis.html*
  http://lsa.colorado.edu/whatis.html
- [HREF 53]. *WordNet*
  http://wordnet.princeton.edu/
- [HREF 54]. *Cyc*
  http://www.cyc.com/
- [HREF 55]. *www.emtech.net*
  http://www.emtech.net/learning_theories.htm
- [HREF 56]. *http://www.scu.edu.au/schools/gcm/ar/arhome.html*
  http://www.scu.edu.au/schools/gcm/ar/arhome.html
- [HREF 57]. *http://www.qual.auckland.ac.nz/action.htm*
  http://www.qual.auckland.ac.nz/action.htm
- [HREF 58]. *http://www.triangle.co.uk/ear*
  http://www.triangle.co.uk/ear
- [HREF 59]. *'Jennifer's Language Page.'*
  http://www.elite.net/~runner/jennifers/
- [HREF 60]. *http://www.vhml.org/*
  http://www.vhml.org/
- [HREF 61]. *http://www.vhml.org/workshops/AAMAS2003/*
  http://www.vhml.org/workshops/AAMAS2003/
- [HREF 62]. *"Why the paper clip sucks"*
  http://www.lclark.edu/~phan/writings/clip.html

It wasn't until late in life that I discovered how easy it is to say 'I don't know'.
W. Somerset Maugham

Theses on Agents

Ballim, A 1992a, 'ViewFinder: A Framework for Representing, Ascribing and Maintaining Nested Beliefs of Interacting Agents', Ph.D. Thesis, Université de Genève, 1992. Online at ftp://crl.nmsu.edu/pub/non-lexical/ViewFinder/ViewFinder-A4.tar.Z.

Cheong, F C 1992a, 'OASIS: An Agent-Oriented Programming Language for Heterogeneous Distributed Environment', Ph.D. Thesis, University of Michigan, 1992. Online at http://www.mentor.computing.edu.au/archives/OASIS.tar.gz.

Coen, M H 1994a, 'SodaBot: A Software Agent Environment and Construction System', M.S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994. Online at ftp://ftp.ai.mit.edu/pub/users/mhcoen/aitr-1493.ps.Z.

Dale, J 1997a, 'A Mobile Agent Architecture for Distributed Information Management', Ph.D. Thesis, Department of Electronics and Computer Science, University of Southampton, September 1997, Originally at http://www.agents.ecs.soton.ac.uk/Voyager/papers/thesis.htm. Online at http://www.mentor.computing.edu.au/archives/Voyager_thesis.ps.gz.

Ferguson, I A 1992a, 'TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents', Ph.D. Thesis, Clare Hall, University of Cambridge, UK, November 1992

Foner, L N 1994a, 'Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning', M.S. Thesis, Media Lab, Massachusetts Institute of Technology, Cambridge, MA, 1994. Online at ftp://ftp.media.mit.edu/pub/Foner/MS-Thesis/Document/.

Huffman, S B 1994a, 'Instructable Autonomous Agents', Ph.D. Thesis, University of Michigan Dept. of Electrical Engineering and Computer Science, January 1994, Originally at ftp://ftp.eecs.umich.edu/people/huffman/papers/thesis.ps.Z. Online at http://www.mentor.computing.edu.au/archives/instructable_thesis.ps.gz.

Isbister, K 1995a, 'Perceived intelligence and the design of computer characters', *Extract from M.A. thesis, presented at the Lifelike Computer Characters conference*, 28 September 1995. Online at http://www.cyborganic.com/People/kath/lccpaper.html.

Johnson, M B 1995a, ‘WavesWorld. A testbed for constructing semi-autonomous animated characters.’, Ph.D. Thesis, Massachusetts Institute of Technology, School of Architecture and Planning., June 1995. Online at http://xenia.media.mit.edu/~wave/PhDThesis/outline.html.

Knabe, F C 1995a, ‘Language Support for Mobile Agents.’, Ph.D. Thesis, Carnegie Mellon University, School of Computer Science., Dec 1995. Online at http://www.cs.virginia.edu/~knabe/dissertation.ps.gz.

Koda, T 1996a, ‘Agents with Faces: A Study on the Effects of Personification of Software Agents’, M.S. Thesis, MIT Media Laboratory, September 1996. Online at http://www.media.mit.edu/~tomoko/docs/ROMAN_paper.ps.

Major, N 1993a, ‘COCA - A Co-operative Classroom Assistant’, Ph.D. Thesis, University of Nottingham, 1993, Originally at http://www.psyc.nott.ac.uk/aigr/research/its/COCA.html

Maulsby, D 1994a, ‘Instructible Agents’, Ph.D. Thesis, Stanford University, Stanford, CA 94305, June 1994. Online at ftp://ftp.cpsc.ucalgary.ca/pub/users/maulsby.

Minar, N 1998a, ‘Designing an Ecology of Distributed Agents’, M.S. Thesis, Massachusetts Institute of Technology, September 1998. Online at http://www.media.mit.edu/~nelson/research/masters-thesis/.

Reilly, W S 1996a, ‘Believable Social and Emotional Agents.’, Ph.D. Thesis, School of Computer Science,Carnegie Mellon University, Pittsburgh, PA., May 1996. Online at http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-96-138-1sided.ps.gz.

Sheth, B D 1994a, ‘A Learning Approach to Personalized Information Filtering’, M.S. Thesis, MIT Media Lab, January 1994. Online at ftp://ftp.media.mit.edu/pub/agents/interface-agents/news-filter.ps.Z.

Thomas, S R 1993a, ‘PLACA, an Agent Oriented Programming Language’, Ph.D. Thesis, Computer Science Department, Stanford University, Stanford, CA 94305, August 1993

Turpeinen, M 1995a, ‘Agent-mediated personalised multimedia services.’, M.S. Thesis, Laboratory of Information Processing Science, Helsinki University of Technology., May 22, 1995. Online at http://www.cs.hut.fi/~mtu/thesis.html.

Tyrell, T 1993a, ‘Computational Mechanisms for Action Selection.’, Ph.D. Thesis, University of Edinburgh., 1993

Wooldridge, M 1992a, ‘The Logical Modelling of Computational Multi-Agent Systems’, Ph.D. Thesis, Department of Computation, UMIST, Manchester, UK, October 1992. Online at http://www.csc.liv.ac.uk/~mjw/pubs/thesis.ps.gz.

And what have you got, at the end of the day?
What have you got to take away?
A bottle of whisky and a new set of lies.
Blinds on the windows and a pain behind the eyes.

Scarred for life, no compensation.
Private Investigations.

Private Investigations.
Dire Straits

Colophon

This thesis was produced via the Gnu Troff system using macros originally developed by

```
Phil Dench 1992
```

and subsequently modified by

```
Andrew Marriott in 1994-2006
```

The HTML Web page production via these macros uses a Perl script originally developed by

```
Rik Harris in 1993
```

The Perl script was subsequently modified by

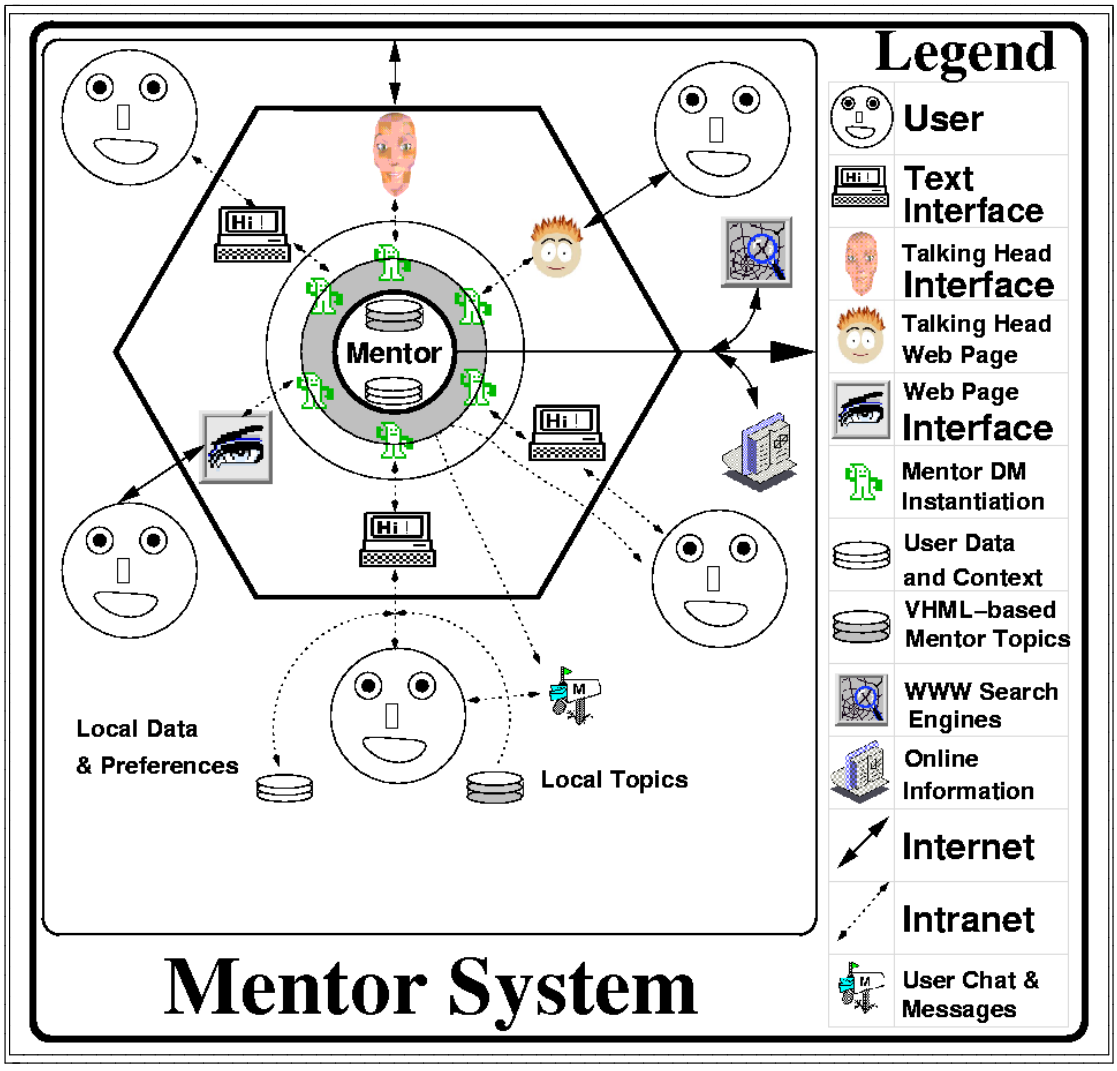
```
Joanne Ng in 1994 and
Andrew Marriott in 1994-2006
```

Several of the icons used for navigation and formatting (such as the buttons and lines) were obtained from the Internet via a general release licence. These icons remain copyright to the original artists even if not specifically stated.

If I have failed to provide adequate acknowledgement to any person’s work through omission, please accept my apologies and do not hesitate to contact me via http://www.cs.curtin.edu.au/~raytrace

Enjoy!

Mentor System Appendices



-- September 30, 2008 --

People will do better, when they know better
Anonymous

A: Case Study 1 Questionnaire: 2001 Semester two

The purpose of this questionnaire was to get valuable feedback from student users about the performance of the ***Mentor*** System in **semester 2, 2001** for the unit **Systems Programming and Design 251.**

The questionnaire, as given to the students, is on the following pages.

The questionnaire can be found on the CDROM in directory:

questionnaires/2001.2

Evaluation of the Mentor System cover sheet.
(not for students)

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System **in semester 2, 2001 for the unit Systems Programming and Design 251. This feedback will be used to improve the** Mentor System .

PhD Hypothesis**:**

A software-based system can benefit University students.

Sub-hypothesis 1

A software-based system can be designed and implemented using a suite of cooperating network based computer programs, databases and user interfaces.

Sub-hypothesis 2

The system will improve the quality of the student.

The *Mentor System* **is designed to address the following elements:**

to determine if a software-based system can be developed (from sub-hypothesis 1 above)..

to determine if the system can be scaled to a real-world, unconstrained environment (from sub-hypothesis 1 above).

to determine if the resulting system is beneficial to students using it. (from sub-hypothesis 2 above).

This questionnaire is designed to determine the user's perceptions of

- what is wrong with the Mentor System (Section 2),
- how it can be improved (Section 2)
- whether it benefits the user or not (Section 2)
- whether the system was annoying or obtrusive (Section 3)
- whether relevant information was returned by the system, whether deeper knowledge was available from the system (Section 4)
- whether features such as active prompting, delayed answering, etc are useful or not. What other features could be added. (Section 3)

It also determines

- whether the students use the system and why/why not (Section 1)

Sub-Hypothesis	***Questionnaire Question***
1: the system can be built	Not tested in this questionnaire but the responses from section 1 and 2 may indicate whether the built system is effective. Response in section 3 may help to improve the system.
2: the system will improve the quality of the student – the system will benefit the students	Section 2 may provide quantitative and qualitative responses to address this. Section 3 (*active* and *later date* questions) may provide information to help determine how the system may better help the student. Section 4 may provide information to help balance the need to provide relevant information to the students vs not simply giving them all the answers. Section 5 may provide informal feedback as to the usefulness of the system.

Evaluation of the *Mentor System*

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System in semester 2, 2001 for the unit Systems Programming and Design 251. This feedback will be used to improve the Mentor System for subsequent units. Hopefully the more feedback you can give to me, the greater will be the improvement in the Mentor System .

This exercise will take **15 minutes** of your time, and while your involvement would be helpful you do not have to participate in the exercise, **it is purely voluntary**.

Note:

- You are free to stop the questionnaire at any time
- The individual data collected will remain strictly confidential
- The questionnaires are anonymous, as there is no need to record identification

If you have any questions, then feel free to ask them during the questionnaire or via email later.

I can be contacted: Andrew Marriott: raytrace@cs.curtin.edu.au
(Or my supervisor): Mike Robey: mike@cs.curtin.edu.au

This project has been approved by the Curtin University Human Research Ethics Committee.

The committee contact details are: C/O The Secretary, Human Research Ethics Committee, Office of Research and development, Level 1, Building 100, Curtin University of Technology, GPO Box U1987, Perth 6845.

You may detach this sheet to keep it in case you need further information or would like to follow up on the study.

Thank you. Please wait until told to turn over.

Questionnaire

Section 1: **Personal details**

(Cross appropriate **O**):

Is English your first language? **O** yes **O** no

Are you **O** Male **O** Female ?

What is your age in years? _____ years

Have you ever used the Mentor System before this semester? **O** yes **O** no

Have you been enrolled in SPD 251 in a previous semester? **O** yes **O** no

Are you enrolled in **O** CS **O** IT **O** SE
O Double degree **O** Other _____

For all units, what is roughly your average grade? **O**<50 **O** 50-60 **O** 60-70
O70-80 **O** 80+

Section 2: **what did you use it for?**

Did you use the Mentor System in semester 2, 2001? **O** yes **O** no

If no, why not? _____
_____
_____
_____

If yes, for what purposes did you use the Mentor System? (E.g, assignments, exams, …)

Purpose 1: _____

How much did you use it for this purpose?

1 2 3 4 5
A little A lot

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all Very Effective

Why did you use it for this purpose?

_____
_____

Purpose 2: _____

How much did you use it for this purpose?

1 2 3 4 5
A little A lot

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all Very Effective

Why did you use it for this purpose?

_____
_____

Purpose 3: _____

How much did you use it for this purpose?

1 2 3 4 5
A little A lot

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all Very Effective

Why did you use it for this purpose?

_____
_____

If you did **not** use the Mentor System at all, then that is the end of the questionnaire.
There is room for any extra comments/suggestions you might like to make in section 7 on Page 6. Thank you.

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

If you used the Mentor System, I would appreciate your responses for the following areas:

Section 3: Was it useful?

Do you think that it was beneficial or helpful to use the Mentor System?

1 2 3 4 5
Not at all Very beneficial

What was the most beneficial aspect of the Mentor System?

_____
_____
_____
_____
_____

What was the least beneficial aspect of the Mentor System?

_____
_____
_____
_____
_____

What was the most annoying aspect of the Mentor System?

_____
_____
_____
_____
_____

Did you find the Mentor System obtrusive?
(that is, did it annoy you or get in your way when you wanted to do something)

1 2 3 4 5
Not at all Very obtrusive

If so, what aspects were obtrusive: _____

_____
_____
_____
_____
_____

What did you find wrong with the Mentor System?

_____
_____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 4: how was it useful?

Do you think that help from the Mentor System improved your time management for the assignment?

1	2	3	4	5
Not at all				**a lot**

How? _____
_____
_____
_____

Do you think that help from the Mentor System improved the design development of your assignment?

1	2	3	4	5
Not at all				**a lot**

How? _____
_____
_____
_____

Do you think that the Mentor System helped in the solving of code or design problems for your assignment?

1	2	3	4	5
Not at all				**a lot**

How? _____
_____
_____
_____

Do you think that help from the Mentor System improved the presentation quality of your assignment?

1	2	3	4	5
Not at all				**a lot**

How? _____
_____
_____
_____

Do you think that help from the Mentor System improved the overall quality of your assignment?

1	2	3	4	5
Not at all				**a lot**

How? _____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 5:how were the various parts useful?

The Mentor System actively asked you questions about the assignment.

Did you find these prompts obtrusive? **O** Yes **O** No

Did you find these prompts helped you with the assignment?

1	2	3	4	5
Not at all				**a lot**

Comments about how the prompts helped you:

_____
_____
_____
_____
_____

If the Mentor System did not understand you immediately, it would often give you an answer at a later date.

Did you find these answers timely? **O** Yes **O** No **O** Did not get one

Did you find these answers helped you with the assignment?

1	2	3	4	5
Not at all				**a lot**

Comments: _____
_____
_____
_____
_____

The Mentor System has knowledge about perl, CGI-BIN etc.
What other things could the Mentor System know about so as to better help you?

_____
_____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 6:did it help you enough?

Was the Mentor System useful in getting access to relevant information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

Was the Mentor System useful in your subsequent enquiries for **more** information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

Was the Mentor System useful in your subsequent enquiries for **deeper** information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

Section 7: general comments and suggestions for improvements.

If you have any general Comments about the Mentor System, please add them here:

_____
_____
_____
_____
_____
_____

If you have any suggestions on how the Mentor System could be improved, please add them here:

_____
_____
_____
_____
_____
_____

This ends the questionnaire. Thank you for your time and participation.

The rung of a ladder was never meant to rest upon, but only to support your weight long enough so you can reach for something higher.
Anonymous

B: Case Study 2 Questionnaire: 2002 Semester one

The purpose of this questionnaire was to get valuable feedback from student users about the performance of the ***Mentor*** System in **semester 1, 2002** for the unit **Advanced Graphics and Visualisation 351.**

The questionnaire, as given to the students, is on the following pages.

The questionnaire can be found on the CDROM in directory:

questionnaires/2002.1

Evaluation of the Mentor System cover sheet.
(not for students)

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System **in semester 1,2002 for the unit Advanced Graphics and Visualisation 351. This feedback will be used to improve the** Mentor System .

PhD Hypothesis**:**

A software-based system can benefit University students.

Sub-hypothesis 1

A software-based system can be designed and implemented using a suite of cooperating network based computer programs, databases and user interfaces.

Sub-hypothesis 2

The system will improve the quality of the student.

The *Mentor System* **is designed to address the following elements:**

to determine if a software-based system can be developed (from sub-hypothesis 1 above)..

to determine if the system can be scaled to a real-world, unconstrained environment (from sub-hypothesis 1 above).

to determine if the resulting system is beneficial to students using it. (from sub-hypothesis 2 above).

This questionnaire is designed to determine the user's perceptions of

- what is wrong with the Mentor System (Section 2),
- how it can be improved (Section 2)
- whether it benefits the user or not (Section 2)
- whether the system was annoying or obtrusive (Section 3)
- whether relevant information was returned by the system, whether deeper knowledge was available from the system (Section 4)
- whether features such as active prompting, delayed answering, etc are useful or not. What other features could be added. (Section 3)

It also determines

- whether the students use the system and why/why not (Section 1)

Sub-Hypothesis	***Questionnaire Question***
1: the system can be built	Not tested in this questionnaire but the responses from section 1 and 2 may indicate whether the built system is effective. Response in section 3 may help to improve the system.
2: the system will improve the quality of the student – the system will benefit the students	Section 2 may provide quantitative and qualitative responses to address this. Section 3 (*active* and *later date* questions) may provide information to help determine how the system may better help the student. Section 4 may provide information to help balance the need to provide relevant information to the students vs not simply giving them all the answers. Section 5 may provide informal feedback as to the usefulness of the system.

Evaluation of the *Mentor System*

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System in semester 1, 2002 for the unit Advanced Graphics and Visualisation 351. This feedback will be used to improve the Mentor System for subsequent units. Hopefully the more feedback you can give to me, the greater will be the improvement in the Mentor System .

This exercise will take **15 minutes** of your time, and while your involvement would be helpful you do not have to participate in the exercise, **it is purely voluntary**.

Note:

- You are free to stop the questionnaire at any time
- If you do participate in the questionnaire, please answer all questions fully.
- The individual data collected will remain strictly confidential
- The questionnaires are anonymous, as there is no need to record your name

For some participants, I may ask you if I can do a follow-up interview just before 2nd semester. This would be to get more information or to clarify your responses. **This would be purely voluntary on your part.** If you are happy to be re-interviewed, then put some unique identifying mark or picture in the box on the next page – something that you could recognise but that does not identify you to me. In this way I can send email to all AGV students saying I would like to re-interview the students who had these marks (and then show the mark as an image). This would only be for a small number of participants and even at that stage, you could decline to be re-interviewed. Choose a mark that you can remember and that is unique.

If you don't want to be considered for another interview, leave it blank.
If you want to be considered, but want to remain anonymous, put a mark.
If you want to be considered and don't care if I know who you are, simply put your name or your mark.

For example, I might use:

Your identity mark

If you have any questions, then feel free to ask them during the questionnaire or via email later.
I can be contacted: Andrew Marriott: raytrace@cs.curtin.edu.au
(Or my supervisor): Mike Robey: mike@cs.curtin.edu.au

This project has been approved by the Curtin University Human Research Ethics Committee.

The committee contact details are: C/O The Secretary, Human Research Ethics Committee, Office of Research and development, Level 1, Building 100, Curtin University of Technology, GPO Box U1987, Perth 6845.

You may detach this sheet to keep it in case you need further information or would like to follow up on the study.

Thank you. Please wait until told to turn over.

Questionnaire

Your identity mark

Section 1: **Personal details**

(Cross appropriate **O**):

Is English your first language? **O** yes **O** no

Are you **O** Male **O** Female ?

What is your age in years? _____ years

Have you ever used the Mentor System before this semester? **O** yes **O** no

Have you been enrolled in AGV 351 in a previous semester? **O** yes **O** no

Are you enrolled in **O** CS **O** IT **O** SE
O Double degree **O** Other _____

For all units, what is roughly your average grade? **O**<50 **O** 50-60 **O** 60-70
O70-80 **O** 80+

Section 2: **What did you use the** Mentor System **for?**

Did you use the Mentor System in semester 1,2002? **O** yes **O** no

If no, why not? _____
_____
_____
_____

If yes, for what purposes did you use the Mentor System? (E.g, assignments,other …)

Purpose 1: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

Purpose 2: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

Purpose 3: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

If you did **not** use the Mentor System at all, then that is the end of the questionnaire.
There is room for any extra comments/suggestions you might like to make in section 7 on Page 6. Thank you.

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

If you used the Mentor System, I would appreciate your responses for the following areas:

Section 3: Was the Mentor System **useful?**

Do you think that it was beneficial or helpful to use the Mentor System?

1 2 3 4 5
Not at all Very beneficial

What was the most beneficial aspect of the Mentor System?

_____
_____
_____
_____
_____

What was the least beneficial aspect of the Mentor System?

_____
_____
_____
_____
_____

What was the most annoying aspect of the Mentor System?

_____
_____
_____
_____
_____

Did you find the Mentor System obtrusive?
(that is, did it annoy you or get in your way when you wanted to do something)

1 2 3 4 5
Not at all Very obtrusive

If so, what specific aspects were obtrusive: _____

_____
_____
_____
_____
_____

What did you find wrong with the Mentor System?

_____
_____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 4: **How was the** Mentor System **useful?**

Read all of questions a), b), c) and d) before you start answering this section.

a) Do you think that help from the Mentor System improved your time management for the assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

b) Do you think that help from the Mentor System improved the design and development of your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

c) Do you think that the Mentor System helped in the solving of code or design problems for your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

d) Do you think that help from the Mentor System improved the overall quality of your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 5:**How were the various parts of the** Mentor System **useful?**

Sometimes the Mentor System would actively prompt you and suggest learning paths to guide and help with the development of your assignments.

Did you find that these prompts or guidance helped you with the assignments?

1	2	3	4	5
Not at all				**a lot**

Comments about how the prompts or guidance helped you:

_____
_____
_____
_____
_____

Did you find these prompts and guidance obtrusive? **O** Yes **O** No

If the Mentor System did not understand you immediately, it would often give you an answer at a later date.

Did you get these answers in time to help you? **O** Yes **O** No **O** Did not get one

Did you find these answers helped you with the assignments?

1	2	3	4	5
Not at all				**a lot**

Comments about how these answers helped you :

_____
_____
_____
_____
_____

The Mentor System has knowledge about OpenGL, GTK, etc.
What other topics could the Mentor System know about so as to better help you?

_____
_____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 6:**Did the** Mentor System **help you enough?**

Read all of questions a), b) and c) before you start answering this section.

a) Was the Mentor System useful in getting access to relevant information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

b) Was the Mentor System useful in your subsequent enquiries for **more** information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

c) Was the Mentor System useful in your subsequent enquiries for **deeper** information or knowledge?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____
_____
_____

Section 7: **general comments and suggestions for improvements.**

If you have any general Comments about the Mentor System, please add them here:

_____
_____
_____
_____
_____
_____

If you have any suggestions on how the Mentor System could be improved, please add them here:

_____
_____
_____
_____
_____
_____

This ends the questionnaire. Thank you for your time and participation.

To find the exact answer,
one must first ask the exact question
S. Tobin Webster

C: Case Study 3 Questionnaire: 2002 Semester two

The purpose of this questionnaire was to get valuable feedback from student users about the performance of the ***Mentor*** System in **semester 2, 2002** for the unit **Systems Programming and Design 251.**

The questionnaire, as given to the students, is on the following pages.

The questionnaire can be found on the CDROM in directory:

questionnaires/2002.2

Evaluation of the Mentor System cover sheet.
(not for students)

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System in semester 2,2002 for the unit Systems Programming and Design 251. This feedback will be used to improve the Mentor System .

PhD Hypothesis:

A software-based system can benefit University students.

Sub-hypothesis 1

A software-based system can be designed and implemented using a suite of cooperating network based computer programs, databases and user interfaces.

Sub-hypothesis 2

The system will improve the quality of the student.

The *Mentor System* is designed to address the following elements:

to determine if a software-based system can be developed (from sub-hypothesis 1 above)..

to determine if the system can be scaled to a real-world, unconstrained environment (from sub-hypothesis 1 above).

to determine if the resulting system is beneficial to students using it. (from sub-hypothesis 2 above).

This questionnaire is designed to determine the user's perceptions of

- what is wrong with the Mentor System (Section 2),
- how it can be improved (Section 2)
- whether it benefits the user or not (Section 2)
- whether the system was annoying or obtrusive (Section 3)
- whether relevant information was returned by the system, whether deeper knowledge was available from the system (Section 4)
- whether features such as active prompting, delayed answering, etc are useful or not. What other features could be added. (Section 3)

It also determines

- whether the students use the system and why/why not (Section 1)

Sub-Hypothesis	***Questionnaire Question***
1: the system can be built	Not tested in this questionnaire but the responses from section 1 and 2 may indicate whether the built system is effective. Response in section 3 may help to improve the system.
2: the system will improve the quality of the student – the system will benefit the students	Section 2 may provide quantitative and qualitative responses to address this. Section 3 (*active* and *later date* questions) may provide information to help determine how the system may better help the student. Section 4 may provide information to help balance the need to provide relevant information to the students vs not simply giving them all the answers. Section 5 may provide informal feedback as to the usefulness of the system.

***Evaluation of the** Mentor System*

The purpose of this questionnaire is to get valuable feedback from student users about the performance of the Mentor System in semester 2, 2002 for the unit Systems Programming and Design 251. This feedback will be used to improve the Mentor System for subsequent units. Hopefully the more feedback you can give to me, the greater will be the improvement in the Mentor System .

Read the following paragraph and then listen to your tutor:
This semester you used the Mentor System for two reasons:
(a) you used it to answer your spd questions / to help you complete the assignment,
(b) you used it for programming new Topics to increase its domain knowledge.
The answers you give in this questionnaire are concerned with **(a)** above. I am interested in your interaction when you said things like *help me with spd*. Only Section 7 contains a question about your use of it for programming and testing your assignment (i.e. *what is /usr/include*).

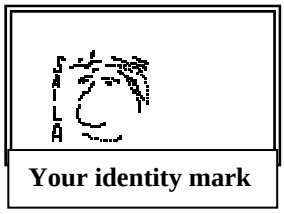
This exercise will take **15 minutes** of your time, and while your involvement would be helpful you do not have to participate in the exercise, **it is purely voluntary**.
Note:

- You are free to stop the questionnaire at any time
- If you do participate in the questionnaire, please answer all questions fully.
- The individual data collected will remain strictly confidential
- The questionnaires are anonymous, as there is no need to record your name

For some participants, I may ask you if I can do a follow-up interview. This would be to get more information or to clarify your responses. **This would be purely voluntary on your part.** If you are happy to be re-interviewed, then put some unique identifying mark or picture in the box on the next page – something that you could recognise but that does not identify you to me. In this way I can send email to all SPD students saying I would like to re-interview the students who had these marks (and show the mark as an image). This would only be for a small number of students and even at that stage, you could decline to be re-interviewed. Choose a mark that you can remember and that is unique.

If you don't want to be considered for another interview, leave it blank.
If you want to be considered, but want to remain anonymous, put a mark.
If you want to be considered and don't care if I know who you are,
simply put your name or your mark.

For example, I might use:



If you have any questions, then feel free to ask them during the questionnaire or via email later.
I can be contacted: Andrew Marriott: raytrace@cs.curtin.edu.au
(Or my supervisor): Mike Robey: mike@cs.curtin.edu.au

This project has been approved by the Curtin University Human Research Ethics Committee.

The committee contact details are: C/O The Secretary, Human Research Ethics Committee, Office of Research and development, Level 1, Building 100, Curtin University, GPO Box U1987, Perth 6845.

Detach this sheet to keep it in case you need further information or would like to follow up on the study.

Thank you. Please wait until told to turn over.

Questionnaire

Your identity mark

Section 1: **Personal details**

(Cross appropriate **O**):

Is English your first language? **O** yes **O** no

Are you **O** Male **O** Female ?

What is your age in years? _____ years

Have you ever used the Mentor System before this semester? **O** yes **O** no If yes, Unit _____

Have you been enrolled in SPD in a previous semester? **O** yes **O** no Sem/Year ___/___

Are you enrolled in **O** CS **O** IT **O** SE
O Double degree **O** Other _____

For all units, what is roughly your average grade? **O**<50 **O** 50-60 **O** 60-70
O70-80 **O** 80+

Section 2: **What did you use the** Mentor System **for?**

Did you use the Mentor System in semester 2,2002 to answer your spd questions? **O** yes **O** no

If no, why not? _____
_____
_____
_____

If yes, for what purposes did you use the Mentor System? (help on answering your spd questions, other, …)

Purpose 1: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

Purpose 2: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

Purpose 3: _____

How much did you use it for this purpose?

1 2 3 4 5
A little **A lot**

How effective or useful was it for this purpose?

1 2 3 4 5
Not at all **Very Effective**

Why did you use it for this purpose?

_____
_____

If you did **not** use Mentor at all to answer your spd questions, then that is the end of the questionnaire.
There is room for any extra comments/suggestions you might like to make in section 8 on Page 6. Thank you.

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

If you used the Mentor System, I would appreciate your responses for the following areas:

Section 3: Was the Mentor System useful in answering your spd questions?

Do you think that it was beneficial or helpful to use Mentor to help you with your spd questions?

1 2 3 4 5
Not at all Very beneficial

What was the **most** beneficial aspect of the Mentor System in answering your spd questions?

_____
_____
_____
_____

What was the **least** beneficial aspect of the Mentor System in answering your spd questions?

_____
_____
_____
_____

What was the **most** annoying aspect of the Mentor System in answering your spd questions?

_____
_____
_____
_____

Did you find the Mentor System obtrusive in answering your spd questions?
(that is, did it annoy you or get in your way when you wanted to do something)

1 2 3 4 5
Not at all Very obtrusive

If so, what specific aspects were obtrusive: _____
_____
_____
_____
_____

What did you find wrong with the Mentor System in answering your spd questions?

_____
_____
_____
_____

Assuming that the Mentor System was given appropriate knowledge for other future units in your course, would you rather it was available or not available for helping you in those units?

O Available **O** Not Available **O** Don't Care

Comments:_____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 4: **How was the** Mentor System **useful in answering your spd questions?**

Read all of questions a), b), c) and d) before you start answering this section. Your answers are concerned with usage (a) **from the first page.**

a) Do you think that help from the Mentor System improved your time management for the assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

b) Do you think that help from the Mentor System improved the design and development of your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

c) Do you think that the Mentor System helped in the solving of code or design problems for your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

d) Do you think that help from the Mentor System improved the overall quality of your assignments?

1 2 3 4 5
Not at all a lot

How? _____
_____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 5:**How were the various parts of the** Mentor System **useful in answering your spd questions?**
Your answers are concerned with usage (a) **from the first page.**

Sometimes the Mentor System would actively prompt you and suggest learning paths to guide and help with the development of your assignments.

Did you find that these prompts or guidance helped you with the assignments?

1	2	3	4	5
Not at all				**a lot**

Comments about how the prompts or guidance helped you:
_____
_____
_____

Did you find these prompts and guidance obtrusive? **O** Yes **O** No

The Mentor System provided you with two sources of information. It could give you an initial **immediate** answer but if it did not understand you immediately, it would often give you a **delayed** answer at a later date.

Did you get these **delayed** answers in time to help you? **O** Yes **O** No **O** Did not get one

Did you find these **delayed** answers helped you with the assignments?

1	2	3	4	5
Not at all				**a lot**

Comments about how these **delayed** answers helped you :
_____
_____
_____

How would you rate the usefulness of the **immediate** answers:

1	2	3	4	5
Not very useful				**very useful**

How would you rate the usefulness of the **delayed** answers:

1	2	3	4	5
Not very useful				**very useful**

What other topics could the Mentor System know about so as to better help you?
_____
_____
_____
_____

How would you rate the response time of the Mentor System ?
(i.e the time it took between you asking a question and getting an initial response)

1	2	3	4	5
Very unsatisfactory				**very satisfactory**

Comments: _____
_____
_____

The questionnaire continues on the next page. Please turn over and continue.

Questionnaire

Section 6:**Did the** Mentor System **help you enough in answering your spd questions?**

Your answers are concerned with usage (a) **from the first page.**

Read all of questions a), b) and c) before you start answering this section.

a) Was Mentor useful in getting access to relevant information or knowledge in answering your spd questions?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____

b) Was Mentor useful in your subsequent enquiries for **more** information or knowledge in answering your spd questions?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____

c) Was Mentor useful in your subsequent enquiries for **deeper** information or knowledge in answering your spd questions?

1 2 3 4 5
Not at all a lot

How?: _____
_____
_____

Section 7: **Using the** Mentor System **for programing (i.e usage (b) from first page).**

How would you rate the ease of use in adding Domain Knowledge by programming the Mentor System ?

1 2 3 4 5
Very difficult very easy

Comment: _____
_____
_____

Section 8: **general comments and suggestions for improvements.**

If you have any general comments about the Mentor System, please add them here:

_____
_____
_____
_____

If you have any suggestions on how the Mentor System could be improved, please add them here:

_____
_____
_____
_____

This ends the questionnaire. Thank you for your time and participation.

Some men see things as they are and ask why.
Others dream things that never were and ask why not
George Bernard Shaw

D: Case Study 1 Evaluation: 2001 Semester two

The 77 page SPSS output from the evaluation of the performance of the ***Mentor*** System in **semester 2, 2001** for the unit **Systems Programming and Design 251** is on the CDROM in directory:

evaluation/2001.2

Don't keep forever on the public road, going only where others have gone.
Leave the beaten track occasionally and dive into the woods.
You will be certain to find something you have never seen before.
Of course it will be a little thing, but do not ignore it.
One discovery will lead to another, and before you know it,
you will have something worth thinking about to occupy your mind,
and really big discoveries are the result of thought.
Alexander Graham Bell

E: Case Study 2 Evaluation: 2002 Semester one

The 61 page SPSS output from the evaluation of the performance of the ***Mentor*** System in **semester 1, 2002** for the unit **Advanced Graphics and Visualisation 351** is on the CDROM in directory:

evaluation/2002.1

> *The superior mind does not set his mind either for anything, or against anything; what is right he will follow.*
>
> Confucious

F: Case Study 3 Evaluation: 2002 Semester two

The 91 page SPSS output from the evaluation of the performance of the ***Mentor*** System in **semester 2, 2002** for the unit **Systems Programming and Design 251** is on the CDROM in directory:

evaluation/2002.2

```
... all flesh is as grass ...
                1 Peter 1:24-25.
```

G: Implementation Details

This appendix provides extra information about various aspects of the design and development of the ***Mentor*** System.

G.1 The Base Packages

These packages (Tables G.1 (a) and (b)) have either been totally developed during the research or have been obtained from the Internet and, in some cases, modified to suit the required functionality. Copyright has been noted for these modified classes. Tables G.1 (a) and (b) show a synopsis of the components of this package. The full documentation for these packages is available via Appendix K.

Package	Description
`Dialogue Manager`	all the classes for the Dialogue Manager kernel as well as loading, processing and saving of the XML and user variables. It also has classes for the XSLT-type transformation that converts the VHML response to the appropriate output.
`Names`	a class for dealing with Domain and Country names. Also a class for holding Gender information about a user as well as a class for holding and processing a user's full name. A class exists for taking a user name and determining the gender of the user.
`Languages`[1]	a class for converting country name to language spoken, loading up the language specific messages, etc.
`IO`[2]	classes to aid in character processing of input similar to the C scanf facility (`ASCIIInputStream`) and output similar to printf (`Format`). Also a class for reading and writing normal or zipped *Object*s such as *Hashtable*s or *Vector*s given filenames or input streams (`FileIO`). Also a File Dialog GUI class that actually works as it should (`FileLister`).
`Curses`	provides simple Interface to the well known Curses library for intelligent text-based interfaces.
`Var`	contains classes to enable Java primitives to be passed as 'var' parameters.

Table G.1a Base Packages

[1] The language dependant messages were obtained from the excellent website: 'Jennifer's Language Page', http://www.elite.net/~runner/jennifers/

[2] The `ASCIIInputStream` class is Copyright 1996 Andrew Scherpbier, modified by Andrew Marriott 1997-2006. The `Format` class is Copyright Gary Cornell and Cay S. Horstmann, Core Java (Book/CD-ROM) 1996, modified by Andrew Marriott 1997-2006.

Package	Description
awtUtil[3]	classes that extend Java's Abstract Windowing Toolkit (AWT) to provide components such as Buttons, Canvases, Windows, Panels that can have images or textured backgrounds. Also provides a Multiline label class as well as components with a 3D look to them. Two Layout Managers have also been developed.
Jar[4]	classes for loading classes from URLs, Files and JAR files as well as extracting files and information from JAR files.
QSort[5]	a class that has static methods for quicksorting arrays, vectors, password entries, etc.
Resources	three classes that help in configuring the server and clients. One class provides support for passing command line arguments into an application (GetArgs) but also blended with values from a configuration file (JavaResourceDatabase). This class enables applications to get hierarchical multi-format resources with default values from system and user configuration files. Another class allows users to specify colours for various GUI attributes using colour names from the X11 specification system.
Syslog[6]	the Syslog class that implements the Unix syslog protocol allowing Java code to log messages to a specified Unix host.
Util[7]	several utility classes to provide needed functionality for things such as Sets of symbols, executing arbitrary system applications similar to the C system and popen calls, browser support as well as general string manipulation and transformation functions.
Windows	classes that support the old-style client interface. Now deprecated due to the use of new HTML based client interface.
Topics	a class that defines the Topics Interface.
Personality	classes that support the layering of a personality onto the DM responses.

Table G.1b Base Packages

---

[3] The MultiLineLabel class is Copyright David Flanagan, 1996 O'Reilly & Associates.
The JMultiLineToolTip class is Copyright Zafir Anjum, 1997.
The Layout Manager classes are based upon example code Copyright 1995, 1996 Sun Microsystems, Inc.

[4] The class loaders are Copyright Jack Harich 1997 and Copyright John D. Mitchell, 1999.
The JarResources class is Copyright John D. Mitchell and Arthur Choi 1983.

[5] The QSort class is Copyright James Gosling and Kevin A. Smith, 1994-1996, Sun Microsystems, modified by Andrew Marriott 1997-2006.

[6] The Syslog class is Copyright 1996 Tim Endres, modified by Andrew Marriott 1996-2006.

[7] Some methods in the Util class are Copyright 1996 Jef Poskanzer.

G.2 The *Mentor* Packages

G.2.1 The `local.mentor.application` Package

Most of these information providers are grouped into two or three classes: one class will coordinate the retrieving of the information whilst the second will parse and store the information, and potentially a third may use that information to get dynamic status information. For example, the `Printers` class uses the '/etc/printcap' file to get each printer, and creates a `Printer` class for each one it finds. The `Lpq` class then uses that information to test each printer to see whether it is up or down, has jobs being printed, etc. Similarly for password information, and Unix finger information.

One important issue is that these classes are the most fragile in the system. That is, they are heavily dependant upon information in a known format in a known place. They also assume an underlying Unix environment. For example, the Unix passwd file '/etc/passwd' follows a known format and this is used to get user information. However, in a Network Information Service (NIS) system, passwords must be obtained from the NIS server by using the `ypcat` program, whereas a system that uses the Lightweight Directory Access Protocol (LDAP) must use the `getent` program to access the X.500 directory service passwords.

Similarly, the Unix netgroup file defines 'netgroups', which are sets of (host, user, domain) tuples, and these are used by the application classes to validate resources such as users and machines. Each line in the file consists of a netgroup name followed by a list of members, where a member is either another netgroup name, or a (host, user, domain) triple. Again this can be provided as a file or via the `ypcat` program.

The maintenance and accuracy of these files is normally the domain of a computing environment's technical staff and is often outside the control of a researcher. Most of the application classes have been developed to isolate this dependency from higher level classes. For example:

- The mechanism for accessing password information in the researcher's environment changed from NIS to LDAP but fortunately after the last case study had been completed.
- Unique to the researcher's environment is a file that holds information about the various units offered by our department. The format for this changed drastically once during the three case studies.
- The data-mining applications, such as the `ImportantDates` class, rely not only on the continuity of layout of the information so that it can be parsed but also in this case, on the continuity of the URL. The URL of the Web site that is used for the Important academic calendar dates has changed each year that the system has been under development.

These data mining application classes may benefit from Berner-Lees' 'Semantic Web' where XML fields can be used to specify the data in a fixed format. The use of a a Uniform Resource Identifier (URI), will remove the problem of a changing URL. A URI identifies a particular resource while a URL both identifies a resource and indicates how to locate it.

G.3 External Packages

Two external packages were used extensively in this research. Both packages were evaluated and adopted early in the research and have since been superseded due to being adopted as the standard for further development work.

Package	Description
`com.oroinc.text.regex`:	OROMatcher™ is a set of regular expression pattern matching and utility classes for Java descended from a package originally written by Daniel F. Savarese, a Ph.D. student at the University of Maryland College Park. This package has been superseded and is now part of the Jakarta Apache project under the name of 'oro'. See http://jakarta.apache.org/oro [HREF 63] for more information.
`org.xml.sax` and `javax.xml`:	The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation. The reference implementation uses Crimson, which was derived from the Java Project X parser from Sun, as its default XML parser and Xalan as its default XSLT engine. However, the pluggable architecture of JAXP allows any XML conformant implementations to be used. These packages have since been superseded with a Java core XML implementation. See http://java.sun.com/webservices/jaxp/reference [HREF 64] for more information.

Table G.2 External Packages

G.4 The *Mentor* System protocol commands

Table G.3 shows the symbolic protocols that enable efficient processing of the needed client-server functionality. For example, one protocol indicates that the data packet that follows contains a Serialised Vector of user Topics, another protocol requests that the list of this user's friends be sent from the server to the client so that the client interface can show these. Currently, these values are specified in a Protocol Constants file as part of the ***Mentor*** system, not the DM kernel.

```
sendUserProfile             ACK_SendUserProfile     NAK_SendUserProfile
NAK_ONLINEUSER              NAK_ERROR               NAK_ONLINEUSERANSWERING
NAK_WARNING                 NAK_UNKNOWNUSER         receiveUserMailMessage
receiveUserSaveMessage      sendUserSaveMessages    YouHaveMessages
receiveUserSavedMessages    deleteUserMessages      sendGeneralProfile
userWishesToTalk            transitPacket           waitingForResponse
IWillTalkToUser             popDownWishToTalk       openTalkWindow
HereIsMyMessageID           transitPacketString     incomingPacketString
popDownTalkConsole          literalString           sendLocalTopics
NetscapeHelpMessage         popUpTalkConsole        expandedWhoResult
remoteClientWindowHasOpened remoteClientURLActivate expandWhoRequest
remoteClientWindowHasClosed cancelUserSaveMessages  sendUserFriends
sendDialogueManagerBaseDir  AlertUserTalkConsole    addUserFriend
displayMessageToUser        sendDMProperties        sendDMUserVariable
setDMUserVariable           setDMProperty           setUserProperty
```

Table G.3 ***Mentor*** System protocol commands

G.5 *Mentor* GUI Dialogs

This package contains utility classes for the TalkConsole interface as well as for the interface Dialog windows.

```
ReallyThrowAwayMessages PreferencesDialog     OnlineUserDialog
UserWishesToTalkDialog  SendMessageDialog     MessagesUserDialog
WantMessagesUserDialog  TalkToMentorDialog
```

Figure `G.1 util Package classes`

The `Dialog` window classes are instantiated by the client to present messages or information to the user. The following description of some of the functionality of the `MentorClient` interface is used to show instantiations of the various `Dialog` classes.

The `MentorClient` allows a user to send messages or to report problems via email to the ***Mentor*** Maintainer. Figure G.2 shows an instantiation of the `TalkToMentorDialog` class that enables a user to send an email message along with an appropriate subject. Most `Dialog` window classes have a 'Help Me' button that opens up a browser at an appropriate help page for that `Dialog`.

Of interest, no-one has ever sent any email to the Maintainer via this mechanism. Most feedback has been through the users telling ***Mentor*** about the problem using normal dialogue.

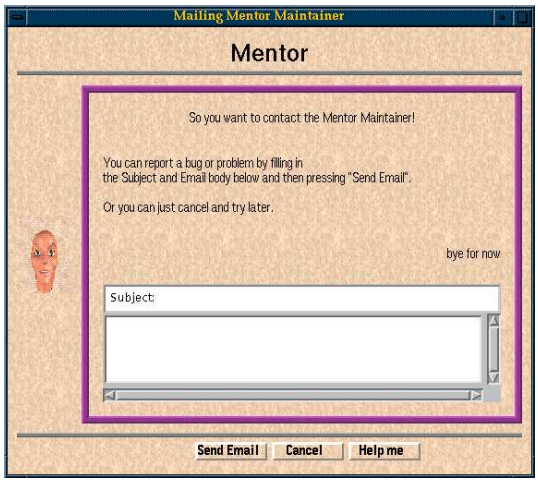


Figure G.2 : Bug Reports to ***Mentor***

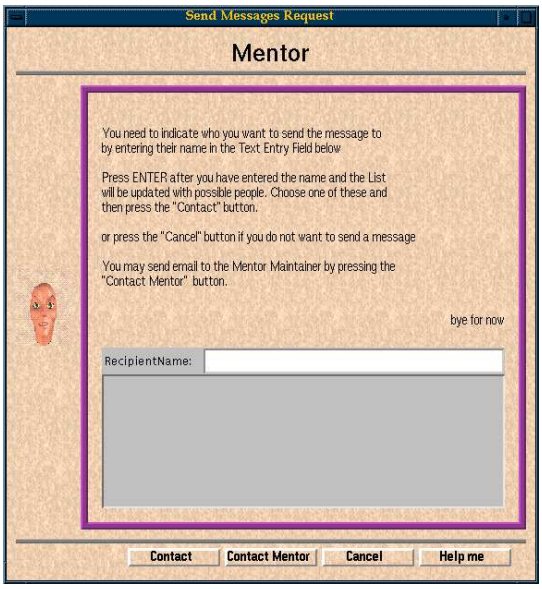


Figure G.3 : Sending Messages to other users

The system also allows a user to send messages to other users. Figure G.3 shows the `SendMessageDialog` window for this facility. Note that this is not a general email messaging system: users are specified by name - either username or via their normal name. Any naming ambiguity is resolved before the user is contacted. The username that is selected is contacted via a `UserWishesToTalkDialog` (Figure G.4). This means that the system will see if the specified user is online and if so, ask if they can accept the message.

Figure G.4 : The person to be contacted is notified

Figure G.5 : The user is informed that the person is being contacted

At the same time, the user wishing to send the message is informed (Figure G.5) that the other party is being contacted. If the person is not known then the user is also informed (Figure G.6).

Figure G.6 : Unknown User

As with any event-driven programming, the state of the interaction must be maintained and this is done by the `Dialog` registering as a `MentorMessageListener`. When the `MentorClient` receives certain data packets from the server tagged with a unique Message Id, any registered `MentorMessageListeners` are notified via their `processCommand` method.

The `Dialog` either processes the data packet, or passes it to any subsequent `Dialog`s that it may have instantiated as part of the process.

If the person is not available or does not want to be disturbed, the user is presented with the `OnlineUserDialog` shown in Figure G.7. The message can either be sent as email to the recipient or sent as a ***Mentor*** message to the recipient. It will be presented to the recipient when they next use the ***Mentor*** System.

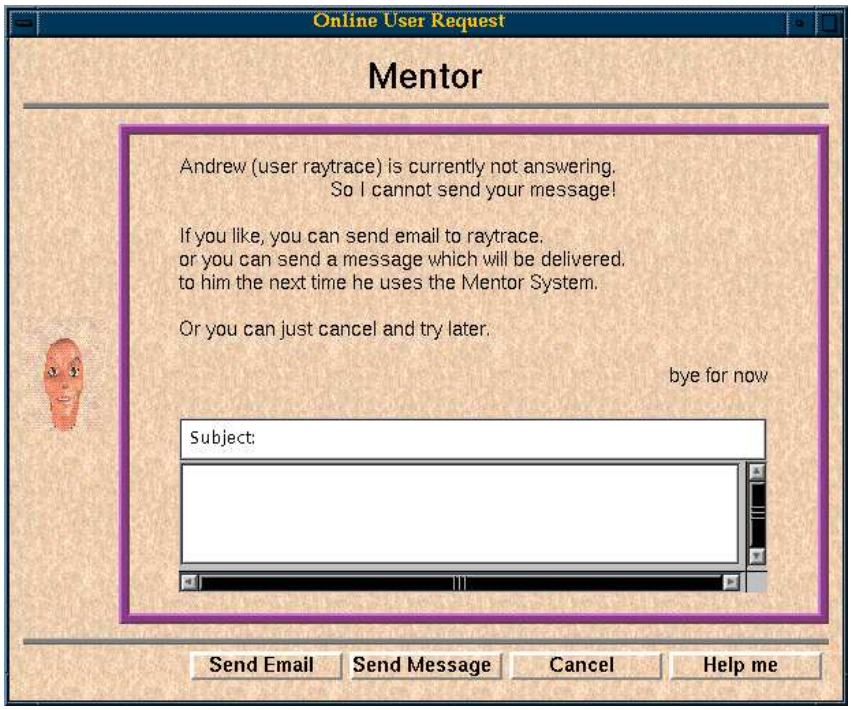


Figure G.7 : Sending a mail message to the user

If the person is available, the user will be prompted with a split `TalkConsole` window (Figure G.8) similar to the Unix `talk` program. Anything typed in will go to the other user and vice versa.

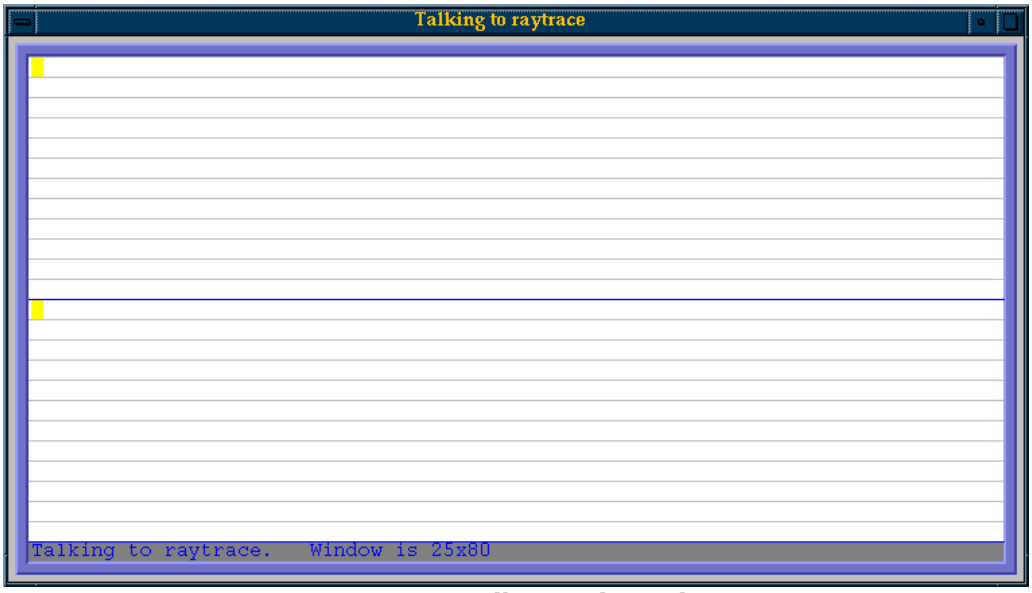


Figure G.8 : Talk Console window

When users are connected to the system, they will be periodically notified via a `WantMessagesUserDialog` (Figure G.9) about any messages sent to them by other users. The system will check when they first connect and then every few minutes after that.

Note that users do not have to read messages from other users but the Message of the Day, which is displayed within their normal interface window, is mandatory.

Figure G.9 : The user is notified that they have messages

If a user retrieves any messages sent to them, then they will be notified via a `MessagesUserDialog` similar to Figure G.10. Users may then process the messages in a typical point and click manner.

Note that the system does not store messages for users once they retrieve them. They must either save or delete them - it is not a general email system. If a user is expecting many messages, then they must make certain that they have time to process them before logging off. If they ‘Cancel’ the message processing `Dialog` and they have any unread messages, they will be notified via the `ReallyThrowAwayMessages Dialog` of Figure G.11.

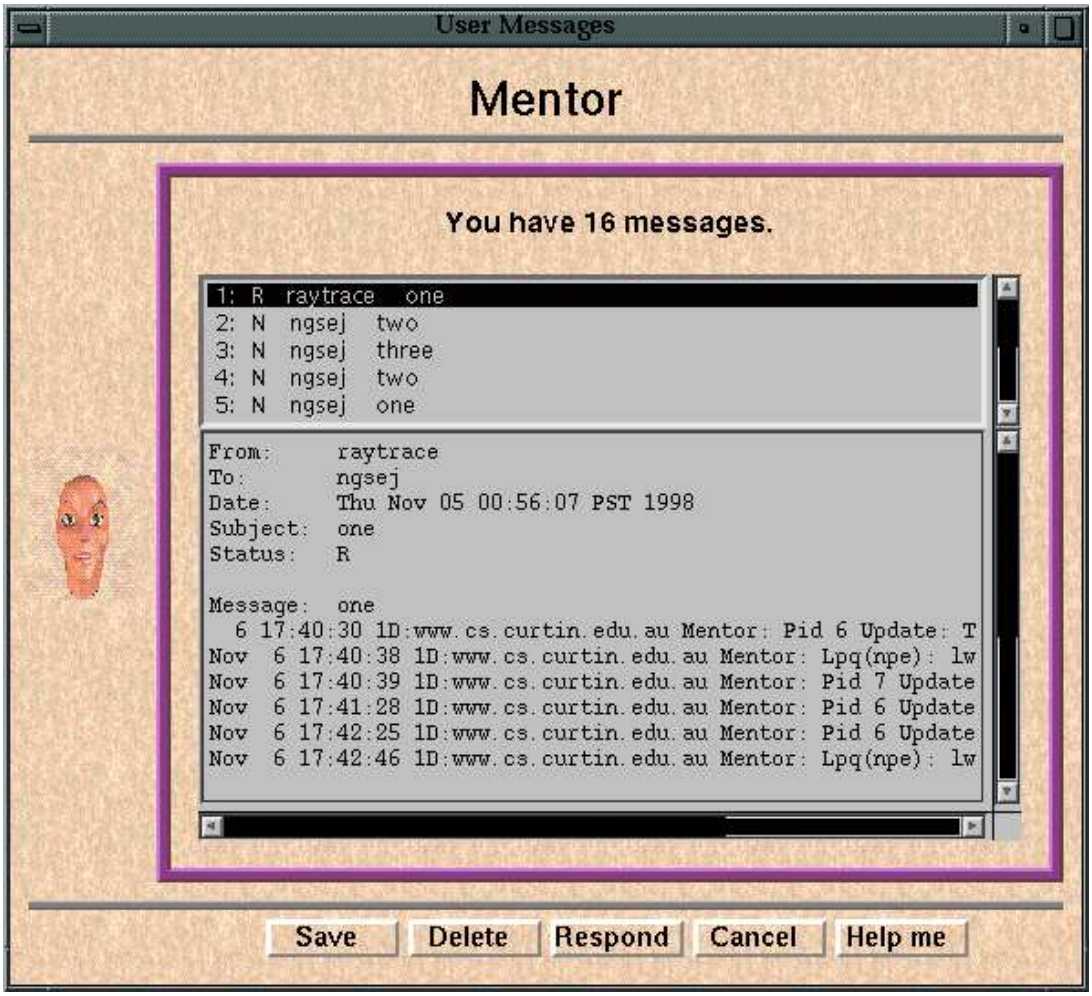


Figure G.10 : Message GUI

Figure G.11 : User is asked if they really want to discard their messages

G.6 Dialogue Manager APIs

Both the ***Mentor*** System and the Stand Alone Dialogue Manager use a well-defined Interface for all communication between the main application and the per-user instantiation of a Dialogue Manager (Figure G.12). The full documentation of these interface methods can be found via Appendix K.

```
public String getDialogueManagerWWW();
public String getDialogueBaseInformationDir();
public void sendGuruMail(String subject, String message);
public void sendEmailToUser(String from, String to, String subject, String message);
public void authenticateConnection(URLConnection connection);
public InputStream authenticateURLConnection(URL url);
public void ReloadAll();
public void ReloadResources();
public void RebootDialogueManager(DialogueManagerInterface dm);
public JavaResourceDatabase getResourceDatabase();
public Object getDaemonObject(String daemonName) throws IOException;
public void DMSErrorLog(String message);
public void DMSdebug(String message);
public FileOutputStream openDMOutputFile(String topicName, String name,
                                         boolean append);
public FileInputStream openDMInputFile(String topicName, String name);
public void removeDMFile(String topicName, String name);
public boolean DMFileExists(String topicName, String name);
public void registerMultiModal(String stimulus_type,Object topic);
```

Figure G.12 Dialogue Manager Server Interface

Similarly, all communication between the per-user DM instantiation and the Topics or Daemons is done via a Java Interface (Figures G.13 and G.14). The full documentation of these interface methods can be found via Appendix K.

```
public boolean isGuiBeingUsed();
public void wakeUpGUI(int level)
public Vector getResponseHistory();
public void addElementToResponseHistory(ResponseData responseData);
public Vector getInputHistory();
public void addElementToInputHistory(String string);
public Hashtable getUserStateHistory();
public Object getUserStateHistory(String state);
public void setUserStateHistoryVisited(String state);
public void removeUserStateHistoryVisited(String state);
public void saveSelf() throws Exception;
public void saveDialogue(String s) throws IOException;
public boolean dialogueManagerSessionIsOpen();
public void setDialogueManagerQuery_lastReceivedInput(Date date);
public Date getDialogueManagerQuery_lastReceivedInput();
public Hashtable getDialogueManagerLocalWhos();
public void setLanguageVariables();
public void setLanguageVariables(String domain, String country);
public XMLVariables getDialogueManagerVariables();
public String getDialogueManagerVariable(String key);
public void setDialogueManagerVariable(String key, String value);
public void removeDialogueManagerVariable(String key);
public void saveDialogueManagerVariables() throws Exception;
public String replaceDialogueManagerVariables(String response);
public int getClientUserId();
public String getClientUserName();
public PasswdEntry getClientPasswdEntry(String user);
public InetAddress getClientInetAddress();
public String getClientInformation(String user);
```

Figure G.13 Dialogue Manager kernel Interface 1/2

```
public Vector getNext_Query_State();
public void setNext_Query_State(Vector nextState);
public void addToNext_Query_State(String nextStateString);
public Vector getPrev_Query_State();
public void setPrev_Query_State(Vector prevState);
public void addToPrev_Query_State(String prevStateString);
public void setLastResponseTopicName(String topic);
public String getLastResponseTopicName();
public boolean DialogueManagerDebugging(String className);
public void debug(String message);
public void remoteBrowserHelp(String url);
public void sendEmailToUser(String fromName,String toName, String subject,
                                        String message)
public void sendLiteralMessage(Object obj0, Object obj)
public void sendMessage(byte protocol, Object obj0, Object obj)
public void sendMessageToUser(String message);
public void sendObjectToUser(Object MessageId, Object obj);
public void sendMessageToUserStatus(String message);
public void logClientData(String message);
public String whoIsUsingDialogueManager();
public String whoIsOnDialogueManager(String resourceName);
public String DialogueManagerTopics();
public VectorEnumeration topicsElements();
public boolean topicExists(String key);
public DialogueManagerServer getDialogueManagerServer();
public InputStream getClientInputStream()
public void close();
public String setMonitoredConnectionNumber(String key,String pid);
public String unsetMonitoredConnectionNumber(String key,String pid);
public void finaliseQueryProcess(boolean reallyDie) throws IOException;
public void pleaseKill(String key,String pidlist);
public void terminateAll(String key);
public void PsProcess();
public void WallProcess(String key,String message);
public void addDialogueManagerListener(DialogueManagerListener listener,
                                        Object callbackData);
public void removeDialogueManagerListener(DialogueManagerListener listener);
public Properties getUserProperties();
public FileOutputStream openDMOutputFile(String topicName, String name,
                                         boolean append)
public FileInputStream openDMInputFile(String topicName, String name)
public void removeDMFile(String topicName, String name)
public boolean DMFileExists(String topicName, String name)
public String getReplacementURL(String url)
public void registerMultiModal(String stimulus_type,Object topic);
public DialogueManagerPersonalityInterface
                                getDialogueManagerPersonalityInterface();
public DialogueManagerPersonalityDataInterface
                                getDialogueManagerPersonalityDataInterface();
```

Figure G.14 Dialogue Manager kernel Interface 2/2

G.7 Syslog logging as *Mentor* starts

Figures G.15a-d show the Syslog information as the system starts up. Each line shows the time, the host machine name, a process name identifier and then a general message. The figures show the different types of logging information as the system starts Daemons, loads up the Topic hierarchy, etc. It can be seen that process 0 is the `LoadDaemons` *Thread* and that in this case, the `ImportantDatesDaemon` has been suppressed. Figure G.15b shows the Syslog output from the `TopicsDaemon` - this daemon loads all system Topic classes and performs any initialisation of each of the Topics (such as for the `jokesTopic`).

```
Jun  4 11:45:05 teapot Mentor: Installed   : DialogueManagerSecurityManager
Jun  4 11:45:06 teapot Mentor:
Jun  4 11:45:06 teapot Mentor: Time 10:45  : listening on port 1024 on teapot with timeout forever and TZ = Custom
Jun  4 11:45:06 teapot Mentor: Memory      : 133107656/133955584
Jun  4 11:45:06 teapot Mentor: Pid   0 Th'd: Thread[LoadDaemons,5,Daemons]
Jun  4 11:45:06 teapot Mentor: Suppressed  : Daemon ImportantDatesDaemon
```

Figure G.15a Syslog information from the ***Mentor*** System

```
Jun  4 11:45:06 teapot Mentor: Pid   1 Th'd: Thread[TopicsDaemon,5,Daemons]
Jun  4 11:45:06 teapot Mentor: jokesTopic  : initialiseTopic
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: weather, time, greeting, who, garbage,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: polite, name, iam, youare, friends,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: arith, isa, help, helpMeWith, whatIKnow,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: dialogueManager, youAskedMe,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: admin, bye, swear,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your topics for today are: offensive, jokes, unknown,
Jun  4 11:45:07 teapot Mentor: TopicsDaemon: Your Unit topics for today are: Unit-02519, Unit-306727,
Jun  4 11:45:07 teapot Mentor: Pid   1 Up  : Thread[TopicsDaemon,5,Daemons](1)
```

Figure G.15b Syslog information from the ***Mentor*** System

Note that Figure G.15b shows normal system Topics as well as all Unit Topics but does not show the loading of Specialist Topics nor Topics that each user may develop. These per-user Topics are sent across the client-server network connection at connect time. Those classes are validated and not cached.

The Specialist Topics are only loaded (and cached) when a user connects and indicates that they want specific Specialist Topics. This enables the system to have detailed system knowledge that only certain users may get access to. For example, Figure G.16 shows a typical Talking Head schema that uses the ***Mentor*** System Dialogue Manager. Figure G.17 shows the GUI of a complex Murder Mystery application that was developed and could use either the ***Mentor*** DM or the MetaFace DM. For this application, the client specified that it wanted the Detective Topic class and did **not** want the University specific Topics nor even some of the system Topics. This functionality allows the system to be configured to cater for varied DM purposes.

```
Jun  4 11:45:08 teapot Mentor: Pid   2 Th'd: Thread[local.mentor.daemons.UnitsDaemon,5,Daemons]
Jun  4 11:45:08 teapot Mentor: Pid   2 Up  : Thread[UnitsDaemon,5,Daemons](0)
Jun  4 11:45:09 teapot Mentor: Pid   3 Th'd: Thread[local.mentor.daemons.NetgroupDaemon,5,Daemons]
Jun  4 11:45:09 teapot Mentor: Netgroup:Executing:ypcat:-k netgroup
Jun  4 11:45:09 teapot Mentor: Pid   3 Up  : Thread[NetgroupDaemon,5,Daemons](0)
Jun  4 11:45:10 teapot Mentor: Pid   4 Th'd: Thread[local.mentor.daemons.PasswdsDaemon,5,Daemons]
Jun  4 11:45:10 teapot Mentor: Pid   4 Up  : Thread[PasswdsDaemon,5,Daemons](0)
Jun  4 11:45:11 teapot Mentor: Pid   5 Th'd: Thread[local.mentor.daemons.GlobalFingerDaemon,5,Daemons]
Jun  4 11:45:13 teapot Mentor: Ping Status : www ping failed!
Jun  4 11:45:13 teapot Mentor: Pid   5 Up  : Thread[GlobalFingerDaemon,5,Daemons](2)
Jun  4 11:45:14 teapot Mentor: Pid   6 Th'd: Thread[local.mentor.daemons.RuptimeDaemon,5,Daemons]
Jun  4 11:45:14 teapot Mentor: Pid   6 Up  : Thread[RuptimeDaemon,5,Daemons](0)
Jun  4 11:45:15 teapot Mentor: Pid   7 Th'd: Thread[local.mentor.daemons.PrintersDaemon,5,Daemons]
Jun  4 11:45:15 teapot Mentor: Pid   7 Up  : Thread[PrintersDaemon,5,Daemons](0)
Jun  4 11:45:16 teapot Mentor: Pid   8 Th'd: Thread[local.mentor.daemons.NamesDaemon,5,Daemons]
Jun  4 11:45:16 teapot Mentor: Pid   8 Up  : Thread[NamesDaemon,5,Daemons](0)
Jun  4 11:45:17 teapot Mentor: Pid   9 Th'd: Thread[local.mentor.daemons.DomainNamesDaemon,5,Daemons]
Jun  4 11:45:17 teapot Mentor: Pid   9 Up  : Thread[DomainNamesDaemon,5,Daemons](0)
Jun  4 11:45:18 teapot Mentor: Pid   0 Done: Thread[LoadDaemons,5,Daemons]
Jun  4 11:45:18 teapot Mentor: Daemon state: All Daemons are now up. GC and finalisation stage finished.
```

Figure G.15c Syslog information from the ***Mentor*** System

Figure G.15c shows the initialisation process continuing with various debug messages also being output. ***Mentor*** process identifiers 1-9 are the Daemon processes, and they can provide timely up-to-date information to Topics with low latency. For example, the `PasswdsDaemon` compiles information from the Unix password file or LDAP server, and other related sources.

```
Jun  4 11:45:28 teapot Mentor: 34409: Conn : [teapot.cs.curtin.edu.au(134.7.1.156)]
Jun  4 11:45:28 teapot Mentor: 34409: Request from:teapot.cs.curtin.edu.au:
Jun  4 11:45:28 teapot Mentor: 34409: allowedHosts from:teapot.cs.curtin.edu.au breeze.cs.curtin.edu.au
Jun  4 11:45:28 teapot Mentor: 34409: allowedHosts from:blowfly.cs.curtin.edu.au scarab.cs.curtin.edu.au:
Jun  4 11:45:28 teapot Mentor: Pid  10 Th'd: Thread[local.mentor.mentord.MConnection,5,Connections]
Jun  4 11:45:28 teapot Mentor: 34409: Request   : [thread] : 5
Jun  4 11:45:28 teapot Mentor: 34409: SpecialDir: null
Jun  4 11:45:33 teapot Mentor: 34409: Connection: [closed]
Jun  4 11:45:33 teapot Mentor: Save Cntxt: /usr/staff/tango/raytrace/mentor/userContext/raytrace.context.gz
```

Figure G.15d Syslog information from the ***Mentor*** System

At this point in the Syslog output, ***Mentor*** is up and ready for use. Figure G.15d shows continuing Syslog information as a user makes a connection from a machine called 'teapot'. Notice that the system has a facility to deny connections from arbitrary machines.

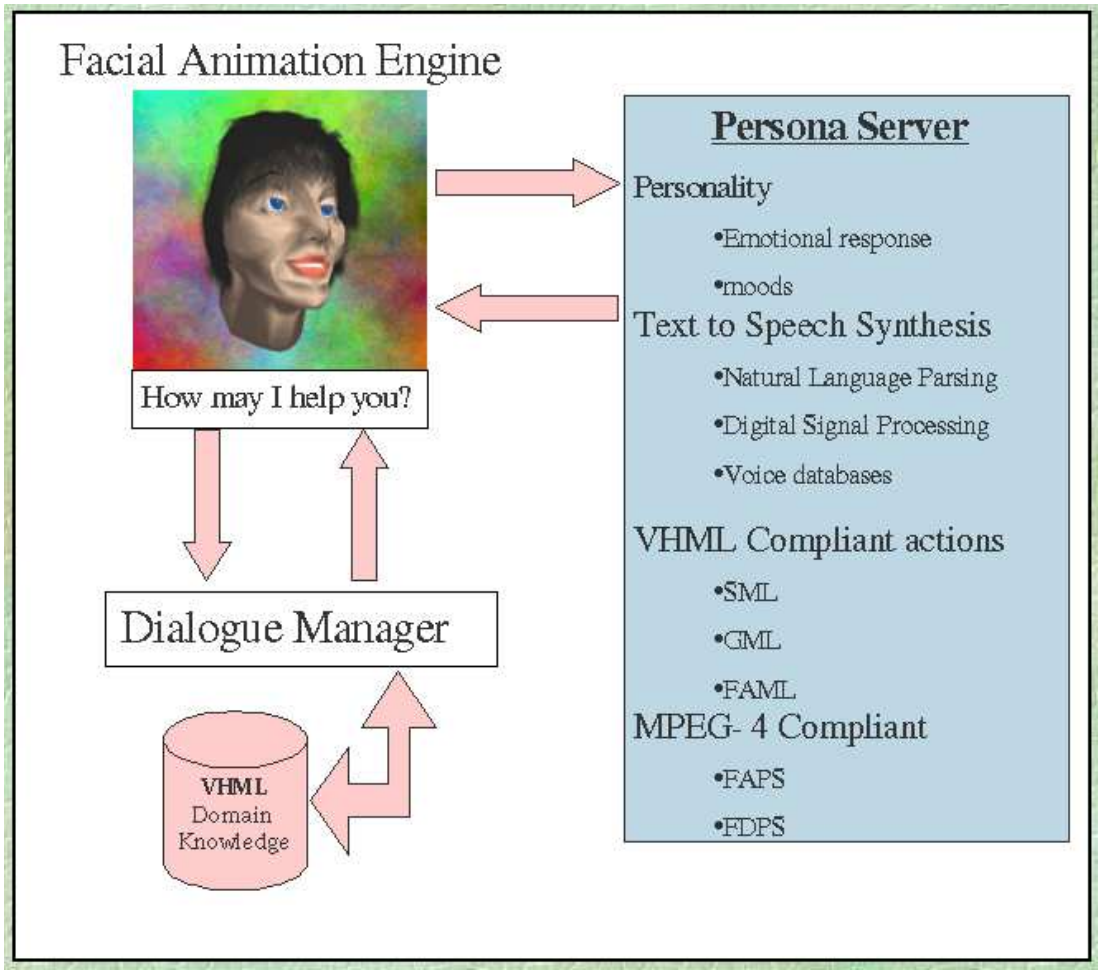


Figure G.16 : Using the ***Mentor*** DM in a Talking Head application

It can also be seen in Figure G.15d that a new `MConnection` class is instantiated for the connected user and the class is entered as process id 10. The client user's identity is verified (not shown as output) and user information is sent across the network connection. In

this case, the type of ***Mentor*** request (5 == request to display the *Thread* hierarchy) as well as any specific Specialist directory (null in this case). After the requested *Thread* hierarchy information has been sent back to the client - taking approximately 5 seconds - the connection is closed and the user's `Context` is saved for future interaction.

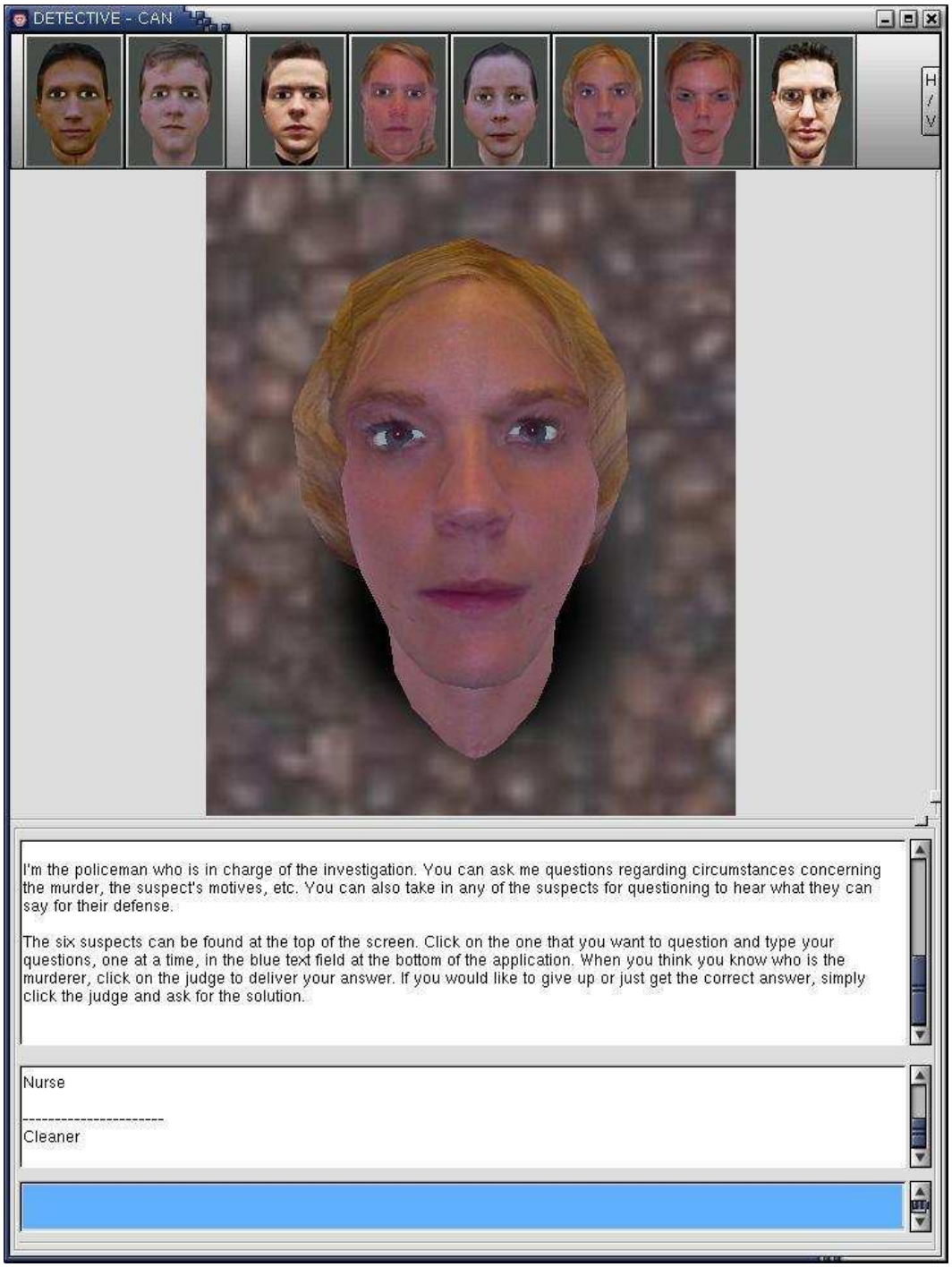


Figure G.17 : The ***Mentor*** DM in a Talking Head Mystery Detective application

G.8 *Mentor* Server Configuration

The type and amount of information that the system logs via Syslog can be controlled via settings in the ***Mentor*** Configuration file (Figures G.18a-f). This Java *Properties* file controls all the functionality of the server. The name of the configuration file that is read depends upon the instance name of the Dialogue Manager. For the ***Mentor*** System it is 'Mentor.properties', for the Stand Alone Dialogue Manager (Section 4.2.2 on page 134) that uses the kernel of the system, it is 'SADM.properties'. This flexibility through customisation allows for very different instantiations of a Dialogue Management system.

```
#Time out in seconds. Mentor will stop if no users connected after this time.
# A value of zero means never end.
timeout=0
timezoneoffset=7
# The time in minutes to wait between saving new topics.
savetopicstimeout = 1
# The time in seconds to allow people to logoff if system is going down.
graceTime = 30
```

Figure G.18a ***Mentor*** Configuration: system timing

Figure G.18a shows self-explanatory example timeout information for the periodic ***Mentor*** processes. Figure G.18b shows timing information for the pro-active responses that the system makes to the user. Based upon feedback from students, these values are quite critical as they represent a balance between the system 'nagging' the student and the system being seen as 'caring' for the student (see Section 5.3.3.4 on page 209). These values are initial values - the real values are dynamically determined by feedback from the student based upon their responses to the pro-active queries.

```
#maxminutes indicates the max time between active queries.
#minminutes indicates the min time before an active query
#initminutes indicates the initial delay before the first active query.
# Initial query is between initminutes and minminutes and then
# subsequent queries are between minminutes and maxminutes.
#For testing the following two lines are OK.
#DialogueManagerActiveQuery.maxminutes=0.1
#DialogueManagerActiveQuery.minminutes=0.2
#DialogueManagerActiveQuery.initminutes=0.1
#For production, the following values are better.
DialogueManagerActiveQuery.maxminutes=180
DialogueManagerActiveQuery.minminutes=120
DialogueManagerActiveQuery.initminutes=10
#DialogueManagerActiveQuery.debug=true

#How long do we wait for no input from the user to decide that they have gone away?
DialogueManagerQueryTimeout.maxminutes=60.0
DialogueManagerQueryTimeout.minminutes=40.0
# Testing
#DialogueManagerQueryTimeout.maxminutes=2.0
#DialogueManagerQueryTimeout.minminutes=1.0
```

Figure G.18b ***Mentor*** Configuration: user process timing

Figure G.18b also shows how debugging information for various classes of Topics may be turned on and off ('DialogueManagerActiveQuery.debug=true'). This can also be seen in Figure G.18c. Example Syslog output when various debugging flags are enabled is shown in Figure G.19 on page 370.

Within a Topic, the following debug code can be used:

```
    if(Debugging(context))
        context.dm.debug("Checking " + topic_name + ":" + context.input);
OR
    if(Debugging(dm, "DialogueManagerActiveQuery"))
        dm.debug(topic_name + "/activeQuery: choice index=" + state);
```

In the first example, the Topic name is known and hence can be used along with the Context, to derive the equivalent of the second. This debug feature is not just limited to Topics. For example, 'ImportantDates.debug' enables the debugging of this Daemon.

```
# We can only put Server Side debug stuff here.....
#JavaResourceDatabase.debug=true
#MConnection.Next_Query_State.Update.debug=true
#MConnection.Next_Query_State.debug=true
UnitTopics.debug=true
GraphTopics.debug=true
Topics.debug=true
#sendMessage.debug=true
#ImportantDates.debug=true
suppressWarnings=true
localTopics.debug = true
```

Figure G.18c ***Mentor*** Configuration: debug flags

Figure G.18d shows numeric and String values that configure the system. For example, the 'GARBAGE_VALUE' is a threshold for an algorithm which tries to detect when users just type gibberish to the system. That is, input such as 'dasadasdsas' or similar. This seems to work well but values such as for laughing at a joke ('hahaha' or 'he he he'), require tweaking of the adjacency table for consecutive letters found in typical English sentences.

```
#The value for the Garbage Topic below which is seen to be gibberish.
GARBAGE_VALUE=0.2

#What do we multiply the weighting of a response by
#if it is the next in state?
STATE_FACTOR=1.2
#Where do we look to find special topics? This is used to find
#any local topics that have been developed.
specialist_dirs=Curtin,InterFace
```

Figure G.18d ***Mentor*** Configuration: Miscellaneous values

Figure G.18e shows that the system has been designed for real-world applications. No values such as hosts, directories or administrator email addresses are hard-wired into the code. Porting the system to another site simply requires the changing of these values in the configuration file.

```
#The email address for all problems, grumbles, etc.
mentor_MAINTAINER_EMAIL=raytrace
#The email address for all ADMIN problems, grumbles, etc.
dm_SYSTEMADMIN_EMAIL=raytrace
#Should we email if a machine is down?
machineDownNotify=false

#If the host is running local then we can set it here and also set the mentor_port. This is
#normally 1 (TCP_MUX) but will probably be 1024 if the MentorDaemon was started stand alone.
mentor_host=teapot
mentor_port=1024
#mentor_host=breeze

#What is the SYSLOG machine? NULL => use the file SYSLOG
mentor_SYSLOG=teapot

mentor_WWW=www.mentor.computing.edu.au
#mentor_PORTFILE=/tmp/aa
#Where do we store/retrieve all Dialogue Manager variables, etc.
#This dir is created if need be along with sub-directories.
#dm_base=storage
mentor_base=/usr/staff/tango/raytrace/mentor

#Allowed hosts to login to system - allow at least the following.
#Put a * at the start to allow all.
Mentor.allowedHosts =    teapot.cs.curtin.edu.au  breeze.cs.curtin.edu.au \
                         blowfly.cs.curtin.edu.au scarab.cs.curtin.edu.au
```

Figure G.18e ***Mentor*** Configuration: system administration

Finally, Figure G.18f shows how it is possible to disable various Topics and Daemons. Any or all Topics or Daemons can be disabled with obvious loss of functionality but without the system dying.

```
#Do we have verbose updates from Daemons as they refresh their information
#verboseDaemons = true

# We can turn off daemons and also individual topics.....
#Turn off the Passwds
#Passwds.run=false
ImportantDatesDaemon.run=false
#Turn off the these topics
#cg351UnitTopic.run=false
#spd251UnitTopic.run=false
#didYouKnow.run=false
```

Figure G.18f ***Mentor*** Configuration: disabling Daemons and Topics

Figure G.19 shows typical debugging Syslog output as the system parses the user request "*who are you?*". Note that the Syslog timing text has been removed for brevity and that the output has been formatted for clarity.

```
... weather:who are you: against :.*(temp(erature)?|w(h)?e(a)?ther|fore?cast).*:not found
... time:who are you: against :.*\b(time|day|date)\b.*:not found
... greeting:who are you: against :.*:found
... who:who are you: against :.*\s*(who|on|around|using)\s*.*:found
... getFinalStateLessResponse Updating: Final Response to
          topic local.mentor.topics.whoTopic@6fe26 is (0.75) <dialogueManagerwhoami/>
... garbage:who are you: against :.*:found
... name:who are you: against :.*:found
                  :
... dialogueManager:who are you: against :.*:found
... admin:who are you: against :admin:not found
... bye:who are you: against :((?:\s*(?:ok|yep|ya|yes|y|yea|yeah|sure))|(?:\s*\
             (?:no|nope|na|nuh|n|nix|nay|nah)))?\s*(((bye\s*)+)|quit|close|die|exit|c\
             \s*ya|see\s*ya|cu|(c\s*u)|(see\s*(?:you|u|ya))|good\s*bye|so\s*long|\
             (shu(t|d)\s*up(pa)?\s*(?:(?:you(?:’|(?:\s*a))?re)|your|ur|ya|yor)\s*\
             (face|mouth))\|(((good\s*)?(night|nite|nighty|nightie)\s*)+)|(shu(t|d)\
             \s*up)|(.*time\s*(to|2)\s*go)|(.*hasta\s*lah?\s*(vista|pasta|vister)\
             (,?\s*bab(y|ie|e))?)|(i(’)?ll\s*be\s*back)|(((bug(ger|a|ga)?)|(piss))\s*of(f)?)\
             |(go\s*away)|adieu|ciao|au\s*revoir|fare\s*well|ttfn|exit|log\s*out|toodle\s*pip\
             |(ta\s*ta\s*(4|for)\s*now)|(.*\b(catch|cop|c|see)(\s*|-)(?:you|u|ya)\s*later)\
             |(clear|cls)|(later\b.*)).*(?:[?!.,:;]*):not found
... swear:who are you: against :.*:found
... offensive:who are you: against :.*:found
... jokes:who are you: against :.*:found
... unknownUnit:who are you: against :.*:found
... turnip:who are you: against :.*\s*turnip(s)?\s*.*:not found
... Perl_FAQ:who are you: against :.*:found
... RandCDates:who are you: against :.*\b(when|(?:wat|what|wot|waht))\b.*:not found
... TPL:who are you: against :.*\b(tute|tutorial|prac(t)?|practical|lect(ure)?)s*\b.*:not found
... venue:who are you: against :[^A-Z]*\bwhere(’)?(s| is| si| are) .*:not found
... spd251UnitTopic:who are you: against :.*:found
... unknown:who are you: against :.*:found
... processQuery Updating: Final Response to
            topic local.mentor.topics.whoTopic@6fe26 is (0.75) <dialogueManagerwhoami/>
```

Figure G.19 Syslog debug information in response to question ‘who are you?’

Each Topic checks the request against its keyword RE (Section 4.2.2 on page 134) and when this matches, and subsequent, more specific tests match, a response is generated with a suitable response_weight (5th line). Some keyword REs are quite complex - the `byeTopic` (12th line). This testing continues for all system Topics, Specialist Topics, unit Topics and user developed Topics. In this case, the returned response comes from the `whoTopic` and is the VHML tag <**`dialogueManagerwhoami`**/> which will be transformed into the DM name.

For more detail on the ‘question-answer’ process, see Section 4.2.2 on page 134.

G.9 Administrator and Control File Interface

An administration Topic was developed to accept commands that enabled an authorised user to:

reboot

This command first checked to see if there were any users (besides this user) logged in and if so, printed out a 'who' type status of the processes. The user was then asked if they really want to reboot the system, and acted accordingly on their response. All logged in users were notified that the system was going down and they were given (typically) 30 seconds to logout from the system.

die

This command was similar to the above but after verifying the command, it brought the ***Mentor*** System down in a controlled manner. Users were again notified.

reload

This asked all Daemons to reload their information. For example, the `password` Daemon would recompile password information from the Unix passwd file as well as various netgroup files. The `finger` Daemon would recheck all known machines to see who was logged on.

ps

This reported the status of all processes currently running on the system including Daemon processes. This was often used just prior to killing or rebooting the system.

kill

This command would kill the specified list of ***Mentor*** processes.

resource

This command asked the ***Mentor*** System to reload configuration information. This was typically done after the configuration files had been modified. For example, to turn on or off various Syslog debugging options.

wall

This command, similar to the Unix `wall` or write to all command, sent the specified message to all users logged into the ***Mentor*** System. It was often used to tell users why the system was going down or being rebooted.

help

This command printed out help information for the administrator.

quit or **bye**

Leave administrator mode.

This functionality was also available through a Control File interface. The system would periodically (typically every 10 seconds) examine a control file in the system file hierarchy and if it existed, it would use the contents of the file as a command to the AdminTopic. This allowed for control even if a connection to the system was not available.

G.10 The MentorClient User Preferences

Figures G.20a-f show the key-value pairs available in the MentorClient User Preferences file.

```
## FUNCTION KEYS##
F1 = Help me with
F2 = who is online\n
F3 = What is the weather?\n
F4 = when is my assignment due?\n
F5 = What is the time?\n
```

Figure G.20a ***Mentor*** Client Preferences Function Key Definition example

```
## ALERTS##
popupOnAlert = true
bringToFrontOnAlert = true
flashOnAlert = true
beepOnAlert  = true
numberOfBeeps= 5
numberOfFlashes= 10
```

Figure G.20b ***Mentor*** Client Preferences Pro-active Query Alert example

Figure G.20a shows the mechanism whereby the keyboard Function Keys can be defined by the user. Figure G.20b shows the various options available to the user to tailor how the interface will react when it is alerted that a pro-active response has been sent to it. For example, if the interface is iconified, the user can enable it to deiconify, for the icon to flash, for the computer to beep, all of these, or none of these. Since this intrusive nature of the pro-active prompts was seen as one of the major problems with the final studies (Section 5.3.5.1 on page 238), it was put under the control of the user through the Preferences file.

In a similar way, although not advertised as available, the `reallyKill` flag allows the interface to be killed instead of just iconifying when the user clicks on the Window Manager decoration ‘kill’ button. Since the system is supposed to be pro-active, killing the interface when it is not needed would render this useful aspect useless.

Figure G.20c shows how the appearance of the client interface may be adjusted although not all values are honoured on all interfaces. Similarly Figure G.20d shows the way in which the size, position and structure of the interface may be set.

```
## APPEARANCE##
inputBackgroundColour = 0xffffff
inputForegroundColour = 0x000000
              :
statusLineForegroundColour = white
responseBackgroundColour = 0x60aeff
responseForegroundColour = red
```

Figure G.20c ***Mentor*** Client Preferences Client Interface Appearance example

```
## SIZE##
rows = 25
DialogueHistoryrows = 5
columns = 80
xPosition = 80
yPosition = 30
responseHistoryLength= 100
```

Figure G.20d ***Mentor*** Client Preferences Client Interface Size example

Figure G.20e shows the mechanism that allows a user to enable/disable various Topics. This is especially useful for disabling ‘annoying’ pro-active Topics such as ‘jokes’ or the ‘didYouKnow’ Topic. These values do not represent a generic configuration mechanism. Like the timeout values in Figure G.20f, the only possible key extensions are either ‘.disable’ or ‘.timing’. This stops malicious users from setting arbitrary ***Mentor*** Properties.

```
## TOPICS##
jokes.disable=true
didYouKnow.disable=false
```

Figure G.20e ***Mentor*** Client Preferences Topics Enabling example

```
##  MISC##
DialogueManagerActiveQuery.initminutes.timing=5
DialogueManagerActiveQuery.minminutes.timing=5
DialogueManagerActiveQuery.maxminutes.timing=10
## OTHER##
```

Figure G.20f ***Mentor*** Client Preferences Timeout example

Finally, Figure G.20f indicates that there is room for future values that can be set in the preferences file via the ‘##OTHER##’ heading.

G.11 Other Clients

The network protocol interface of the ***Mentor*** System allowed different clients to connect to it. Some of these were purely for debugging, and some were for session management. These clients communicated with the server using the protocols of Figure G.3, page 357.

`QueryClient`

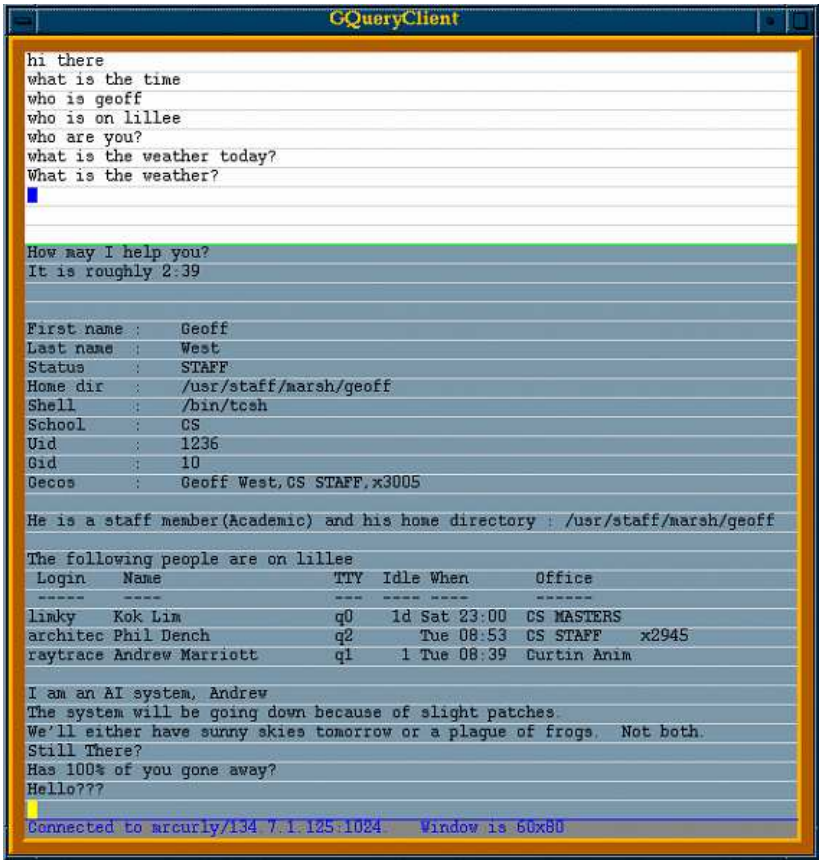


Figure G.21 : ***Mentor*** System `QueryClient`

This client was deprecated once it became obvious that users wanted more than short answer information from the system. The interface of this client was similar to the Unix talk program and could display hot links but not images and had no scrolling facility. This interface was superseded as soon as the Swing components were seen as efficient and stable on the typical client hardware platform.

`DieClient`

This text-based client requested that the ***Mentor*** System terminate. Authorisation was requested and if the supplied password was not correct, the ***Mentor*** System administrator was notified by email. If the password was correct, then the system started to shutdown in an orderly fashion. All user sessions were informed that the system was going down. Users were given the number of seconds specified in the ***Mentor*** preferences file before the session was automatically closed - their MentorClient interface was still visible but simply had a message that indicated that ***Mentor*** had closed the connection. No user information was lost as each session was asked to 'save itself' via a public method in the Java class.

Finally the ***Mentor*** System daemons were shutdown and each Topic was also asked to 'save itself' so that any stored data could be written to files.

Since the system waited for the various sub-systems to finish before it terminated, it was possible for a non-terminating sub-system to stall the termination process. For example, a network connection to a machine that had gone down. In this case, the system had to be 'hard' killed by a Unix Guru. This could have led to a loss of user data.

`KillClient`

The text-based `KillClient` simply connected to ***Mentor*** and killed the ***Mentor*** children that had been specified on the command line. Users could only kill their own processes. Users who tried to kill others' processes were asked for authorisation and if the password supplied was not correct, the ***Mentor*** administrator was notified by email.

If the password was correct, then the system killed each child similar to the `DieClient`.

`PsClient`

The text-based `PsClient` (named after the Unix ps or Process Status command) simply indicated what processes were currently running on the ***Mentor*** System. Figure G.22 shows two normal clients - process id 14 and 13 - and nine daemon processes - 1 to 9. The information associated with each daemon varies with the daemon functionality.

```
teapot % PsClient
Connecting to teapot on port 1024

Connected to teapot/134.7.1.156:1024
Pid  User     Req Thread Name         Thread Group Name      Information
--- ---------  --------------------   ------------------- -------------------
14 raytrace   3 PsClient-34413       [teapot.cs.curtin.edu.au(134.7.1.156)]
13 raytrace   9 CommandClient-34412 [teapot.cs.curtin.edu.au(134.7.1.156)]
9  null         DomainNamesDaemon    ^ local.raytrace.Names.CountryNames.gz
8  null         NamesDaemon          ^ local.raytrace.Names.{MaleNames,FemaleNames}.gz
7  null         PrintersDaemon       ^ /etc/printcap@Wed Jun 04 10:45:15 GMT+07:00 2003 Update=125000
6  null         RuptimeDaemon        ^ ruptime@Wed Jun 04 11:18:30 GMT+07:00 2003 Update=57000
5  null         GlobalFingerDaemon   ^ GlobalFinger@Wed Jun 04 11:14:51 GMT+07:00 2003: www Update=354000
4  null         PasswdsDaemon        ^ null@null
3  null         NetgroupDaemon       ^ ypcat@Wed Jun 04 10:45:09 GMT+07:00 2003
2  null         UnitsDaemon          ^ /applications/new_unit_info@Wed Jun 04 10:45:08 GMT+07:00 2003
1  null         TopicsDaemon         ^ local.mentor.topics/Topics.ser.gz
Connection closed by server.
teapot %
```

Figure G.22 Output from `PsClient`

`TopClient`

The gui-based `TopClient` was similar to the Unix `top` command and repeatedly showed the updated process id status of the ***Mentor*** System (Figure G.23). It had similar output to repetitive calls via the `PsClient` interface. The researcher often had this client running on his workstation desktop to monitor system usage.

`EchoClient`

The text-based `EchoClient` simply connected to the ***Mentor*** System and any characters typed by the user were echoed back. This client was mainly used for debugging and for testing the functionality of the system after a version upgrade.

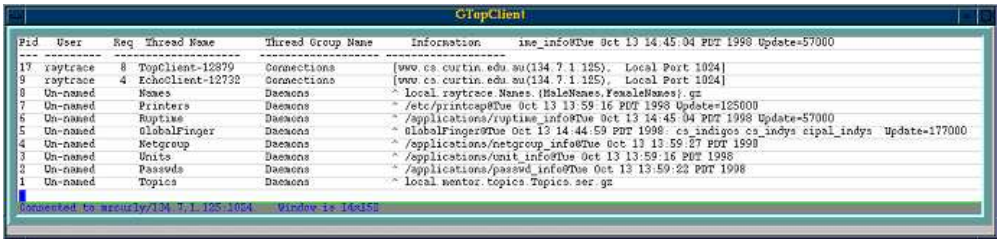


Figure G.23 : ***Mentor*** System `TopClient` for monitoring ***Mentor*** processes

`ThreadClient`

The text-based `ThreadClient` was similar to the `PsClient` but reported Java *Thread*s rather than ***Mentor*** System processes. This was useful to see the overall resource usage as users connected to the system. Figure G.24 shows the output of the **ThreadClient** running on a Linux host. The first 4 *Thread*s are Java system *Thread*s from the 'system' *ThreadGroup*. The 'main' *ThreadGroup* only has the main *Thread*, the ***Mentor*** System created its own *ThreadGroup*s: ServerThreadGroup, Connections and Daemons.

The 'Daemons' *ThreadGroup* held all the ***Mentor*** System daemon processes as well as the Java system Daemons. The ***Mentor*** System 'Connections' *ThreadGroup* held all user processes such as the `ThreadClient` shown.

```
Connected to teapot/134.7.1.156:1024
Thread Group: system  Max Priority: 10
    Thread: Reference Handler  Priority: 10 Daemon
    Thread: Finalizer  Priority: 8 Daemon
    Thread: Signal Dispatcher  Priority: 10 Daemon
    Thread: CompileThread0  Priority: 10 Daemon
    Thread Group: main  Max Priority: 10
        Thread: main  Priority: 5
    Thread Group: ServerThreadGroup  Max Priority: 10
        Thread: Server  Priority: 5
        Thread Group: Connections  Max Priority: 10
            Thread: ThreadClient-34409  Priority: 5
        Thread Group: Daemons  Max Priority: 10
            Thread: TopicsDaemon  Priority: 5
            Thread: UnitsDaemon  Priority: 5
            Thread: NetgroupDaemon  Priority: 5
            Thread: process reaper  Priority: 5 Daemon
            Thread: process forker  Priority: 5 Daemon
            Thread: PasswdsDaemon  Priority: 5
            Thread: GlobalFingerDaemon  Priority: 5
            Thread: RuptimeDaemon  Priority: 5
            Thread: PrintersDaemon  Priority: 5
            Thread: NamesDaemon  Priority: 5
            Thread: DomainNamesDaemon  Priority: 5
Connection closed by server.
```

Figure G.24 Output from `ThreadClient` on startup

Figure G.25 shows a more complex session with many users attached. The user's names have been changed in this example and the Daemon *Thread*s removed. The system was running on a Silicon Graphics host, with nearly 20 users connected to the system. It can be seen that each connection would normally start four *Thread*s:

```
    CommandClient,                CommandTimeOut,
    DialogueManagerQueryTimeout, DialogueManagerActiveQuery.
```

These *Thread*s we:w re concerned with the normal Dialogue Management session, periodic session management for messages, etc, the 'have they gone away' monitoring, and the pro-active prompting of users.

Note that user YYY (line 19 in Figure G.25) has been shown as he/she is logging out from the system. Two of his/her *Thread*s have died and the final *Thread* will be tidying up the session.

The data shown in Figure G.25 has been used for debugging - it can be seen that some Threads are not in groups of four. This shows an early error that was occurring when some Threads failed to terminate correctly when users disconnected from the system in a certain way.

```
Connected to breeze/134.7.1.223:1025
Thread Group: system  Max Priority: 10
    Thread: Clock  Priority: 12 Daemon
    Thread: Idle thread  Priority: 0 Daemon
    Thread: Finalizer thread  Priority: 1 Daemon
    Thread: Async I/O Poll thread  Priority: 11 Daemon
    Thread Group: main  Max Priority: 10
        Thread: main  Priority: 5
    Thread Group: ServerThreadGroup  Max Priority: 10
        Thread: Server  Priority: 5 Not Alive
        Thread: Server  Priority: 5
        Thread Group: Connections  Max Priority: 10
        Thread Group: Daemons  Max Priority: 10
        Thread Group: Connections  Max Priority: 10
            Thread: CommandTimeOut-fred  Priority: 5
            Thread: CommandTimeOut-freda  Priority: 5
            Thread: CommandTimeOut-raytrace  Priority: 5
            Thread: CommandTimeOut-YYY  Priority: 5
            Thread: DialogueManagerQueryTimeout-YYY  Priority: 5 Not Alive
            Thread: DialogueManagerActiveQuery-YYY  Priority: 5 Not Alive
            Thread: CommandTimeOut-mary  Priority: 5
            Thread: CommandTimeOut-maggie  Priority: 5
            Thread: CommandTimeOut-sally  Priority: 5
            Thread: CommandClient-3438  Priority: 5
            Thread: CommandTimeOut-maggie  Priority: 5
            Thread: DialogueManagerQueryTimeout-maggie  Priority: 5
            Thread: DialogueManagerActiveQuery-maggie  Priority: 5
            Thread: CommandTimeOut-rodney  Priority: 5
            Thread: CommandTimeOut-minnie  Priority: 5
            Thread: CommandTimeOut-mickey  Priority: 5
            Thread: CommandTimeOut-mickey  Priority: 5
            Thread: CommandTimeOut-lilly  Priority: 5
            Thread: CommandTimeOut-harry  Priority: 5
            Thread: CommandTimeOut-david  Priority: 5
            Thread: CommandTimeOut-minnie  Priority: 5
            Thread: CommandClient-1394  Priority: 5
            Thread: CommandTimeOut-turnip  Priority: 5
            Thread: DialogueManagerQueryTimeout-turnip  Priority: 5
            Thread: DialogueManagerActiveQuery-turnip  Priority: 5
            Thread: CommandTimeOut-roger  Priority: 5
            Thread: CommandClient-2285  Priority: 5
            Thread: CommandTimeOut-coss  Priority: 5
            Thread: DialogueManagerQueryTimeout-coss  Priority: 5
            Thread: DialogueManagerActiveQuery-coss  Priority: 5
            Thread: CommandClient-4185  Priority: 5
            Thread: CommandTimeOut-waters  Priority: 5
            Thread: DialogueManagerQueryTimeout-waters  Priority: 5
            Thread: DialogueManagerActiveQuery-waters  Priority: 5
            Thread: CommandClient-42967  Priority: 5
            Thread: CommandTimeOut-camilla  Priority: 5
            Thread: DialogueManagerQueryTimeout-camilla  Priority: 5
            Thread: DialogueManagerActiveQuery-camilla  Priority: 5
            Thread: CommandClient-2868  Priority: 5
            Thread: CommandTimeOut-lilly  Priority: 5
            Thread: DialogueManagerQueryTimeout-lilly  Priority: 5
            Thread: DialogueManagerActiveQuery-lilly  Priority: 5
            Thread: CommandClient-1422  Priority: 5
            Thread: CommandTimeOut-lyn  Priority: 5
            Thread: DialogueManagerQueryTimeout-lyn  Priority: 5
            Thread: DialogueManagerActiveQuery-lyn  Priority: 5
            Thread: CommandTimeOut-raytrace  Priority: 5
            Thread: PsClient-23296  Priority: 5
            Thread: ThreadClient-23303  Priority: 5
        Thread Group: Daemons  Max Priority: 10
                              :
            Thread: stdout reader pid=4170  Priority: 5
            Thread: stderr reader pid=4170  Priority: 5
Connection closed by server.
```

Figure G.25 Output from `ThreadClient` when ***Mentor*** is busy

G.12 Mentor Data Format

This section details the various XML tags and attributes of the format that is used to store the user's interaction session with ***Mentor***. In the following, CDATA is normal Character Data such as 'abcd123', and the attributes will be surrounded by quotes as mandated by XML.

G.12.2 Format tag: `mentor`

The **`mentor`** root tag allows for zero or more lots of **`session`**s.

G.12.3 Format tag: `session`

The **`session`** tag can encapsulate zero or more lots of

`active`	**`timeout`**	**`alertedUser`**	**`response`**
`URLClick`	**`popdown`**	**`popup`**	**`input`**

tags in any order but must have an **`endsession`** tag as the last before the closing </**`session`**>. This represents a single user session. Hence a user's data represents zero or more lots of these sessions where they communicate with the ***Mentor*** System. An example is:

```
<session user="fred" date="Sun Oct 21 20:24:30 PDT 2001" seconds="1003667070945">
<popdown date="Sun Oct 21 20:24:52 PDT 2001" seconds="1003667092127"/>
<popup date="Sun Oct 21 20:24:52 PDT 2001" seconds="1003667092138"/>
<input date="Sun Oct 21 20:24:55 PDT 2001" seconds="1003667095583">
I am the king of the world
</input>
<response responseCode="0" date="Sun Oct 21 20:25:20 PDT 2001" seconds="1003667120684">
What makes you think that you are the king of the world, Fred?
</response>
<endsession date="Sun Oct 21 20:32:43 PDT 2001" seconds="1003667563804"/>
</session>
```

The **`session`** tag has 3 attributes, all of which are required for a valid session:

`user`

CDATA / REQUIRED: specifies the user. This is used even though the file of sessions is named after the user.

Example : raytrace

`date`

CDATA / REQUIRED: The date of this session produced by Java in standard Unix format.

Example : Thu Oct 04 13:58:19 PDT 2001

`seconds`

CDATA / REQUIRED: The date of this session in seconds since the Java epoch.

Example : 1002175099325

G.12.4 Format tag: `input`

This is the single line input by the user. It is assumed to be normal character data. An example is:

```
<input date="Wed Oct 30 09:19:56 PST 2002" seconds="1035944396302">
    c'mon tell me whats in the exam i'm really nice
</input>
<input date="Mon Sep 30 15:44:40 PDT 2002" seconds="1033371880304">
    when is my assignment due?
</input>
```

The **input** tag has 2 attributes, both of which are required for a valid tag:

date

CDATA / REQUIRED: The date of this input.

seconds

CDATA / REQUIRED: The date of this input in seconds since the Java epoch.

G.12.5 Format tag: response

This is the multiline response to the user. It is assumed to be normal character data plus an XHTML subset. Two examples are:

```
 <input date="Sun Oct 21 20:27:00 PDT 2001" seconds="1003667220434">
I'm Mr Sexy Being
</input>
 <response responseCode="0" date="Sun Oct 21 20:27:00 PDT 2001" seconds="1003667220516">
Good evening Sexy
</response>
<response responseCode="0" topicName="Perl_FAQ" date="Sat Sep 28 14:48:38 PDT 2002"
                                  seconds="1033195718444">
<a href="perldiag.html">the perldiag manpage</a> has a complete list of perl's
error messages and warnings, with explanatory text. You can also use the splain
program (distributed with perl) to explain the error messages:
   perl program 2>diag.out
   splain [-v] [-p] diag.out
or change your program to explain the messages for you:
   use diagnostics;
or
   use diagnostics -verbose;
</response>
```

Note that the response must contain strict XHTML **not** just HTML. The GUI is responsible for formatting the output response (à la a Web browser). Note that the response shown here has already had all internal VHML atomic tags such as </**first_name**>, changed into their real character values. The first example has recognised that the user wants to be called "Mr Sexy Being", has changed their first-name to Sexy and has responded with "<**good_time**/> <**first_name**/>". This has used the time to translate the response into "*Good Evening Sexy*".

Note as well that the first example has not specified a "*topicName*" attribute - this response has been taken from an early data-set that did not use that attribute.

The **response** tag has 4 attributes, 3 of which are required for a valid tag:

topicName

CDATA / IMPLIED: The name of the Java Topic class that produced this response.

Example : whoTopic

responseCode

CDATA / REQUIRED: The response code of this response as an integer. Values are :

```
OK       = 0;        END             = 1; NAK                  = 2;
IGNORE   = 3;        UPDATE_FRIENDS  = 4; LAST_TO_BE_ACCEPTED = 5;
UNKNOWNTOPIC = OK+10
```

The UNKNOWNTOPIC value is useful because it indicates that this response is an "*I don't understand you*" response and hence is indicating a failure in the Natural Language Parsing recognition.

Example : 10

date

CDATA / REQUIRED: The date of this response.

seconds

CDATA / REQUIRED: The date of this response in seconds since the Java epoch.

G.12.6 Format tag: `active`

This is the multiline pro-active output to the user. It is not in response to any input from the user but is typically used to prompt the user into some action or response. Note that pro-active questions wait for a lull in the conversation - they do not just 'butt in'. It is assumed to be normal character data plus an XHTML subset. Two examples (with *date* and *seconds* attributes removed) are:

```
<active topicName="UNKNOWN"  … >
Are you interested in this, Fred: Look here for questions and answers on
  <a href="http://www.javaworld.com/columns/jw-qna-index.shtml">Java problems</a>
</active>

<active topicName="spd251UnitTopic" ... >
Have you started the systems programming second assignment yet, Freda?
</active>
<input  ... >
yes
</input>
<response responseCode="0" topicName="spd251UnitTopic" ... >
Good. Hope it goes well.  Freda, need a hand?
</response>
<input ... >
yes
</input>
<response responseCode="0" topicName="spd251UnitTopic" ... >
If you like, I will guide you through a nice learning path that will save you time
in developing your second assignment. The guidance will simply be questions /
suggestions and you can probably just answer yes or no to the questions.
There is no pot of gold at the end of this path unless you work at each step.

This assignment is obviously a little bit of Java plus Regular Expressions plus,
importantly, reading and modifying other people's code.
Let's look at the supplied Java code first. Later we will look
at the Regular Expression stuff and the dialogue simulation.

I assume that you know about the assignment.
The assignment web page is a good place to start - looked at it yet?
</response>
<input ... >
yes
</input>
```

Note that the second example has shown the power of the pro-active response - it encouraged the user in a friendly way to start on the path to assignment completion. Notice that it always asked before giving information about doing the assignment.

Also note that like the **`response`** tag, the pro-active output must contain strict XHTML **not** HTML. Similarly for formatting and the transformation of internal VHML tags into their character values.

The **`active`** tag has 3 attributes, all of which are required for a valid tag:

topicName

CDATA / IMPLIED: The name of the Topic class that produced this pro-active output.

date

CDATA / REQUIRED: The date of this active tag.

seconds

CDATA / REQUIRED: The date of this active tag in seconds since the Java epoch.

G.12.7 Format tag: `timeout`

This is a multiline prompt output to the user. It is not in response to any input from the user but is used to see if the user is still there. If the user does not respond after a number of these prompts, then the session is closed. It is assumed to be normal character data plus maybe an XHTML subset. An example is :

```
<timeout date="Sat Apr 20 15:01:54 PDT 2002" seconds="1019282514357">
knock, knock, knock. Anybody out there?
</timeout>
<timeout date="Sat Apr 20 15:44:58 PDT 2002" seconds="1019285098547">
Has 100% of you gone away?
</timeout>
<timeout date="Sat Apr 20 16:28:57 PDT 2002" seconds="1019291157128">
Fred Bloggs. Paging Fred Bloggs. Is Fred Bloggs still there?
</timeout>
 <endsession date="Sat Apr 20 17:12:57 PDT 2002" seconds="1019286117263"/>
</session>
```

Note that in this example, the user session has been closed after repeated prompts without any response from the user.

Also note that like the **`response`** tag, the timeout output must contain strict XHTML **not** HTML. Similarly for formatting and the transformation of internal VHML tags into their character values.

The **`timeout`** tag has 2 attributes, both of which are required for a valid tag:

`date`

CDATA / REQUIRED: The date of this timeout.

`seconds`

CDATA / REQUIRED: The date of this timeout in seconds since the Java epoch.

G.12.8 Format tag: `popdown`

This atomic tag is used to indicate that either the user has closed the GUI (iconified it) by clicking on the GUI window ‘kill’ button, or by saying "*bye*" or "*go away*", or ***Mentor*** has done this under program control. Two examples follow:

```
<popdown date="Sun Apr 21 18:49:52 PDT 2002" seconds="1019382592891"/>

 <input date="Sun Apr 21 14:44:41 PDT 2002" seconds="1019367881598">
bye
</input>
<response responseCode="1" topicName="bye" ... >
We'll meet again. Don't know where, don't know when,
but I know we'll meet again some foggy day....
</response>
 <popdown date="Sun Apr 21 14:44:43 PDT 2002" seconds="1019367883503"/>
```

The second example has shown that the user has initiated a popdown by saying "*bye*".

The **`popdown`** tag has 2 attributes, both of which are required for a valid tag:

`date`

CDATA / REQUIRED: The date of this popdown.

`seconds`

CDATA / REQUIRED: The date of this popdown in seconds since the Java epoch.

G.12.9 Format tag: popup

This atomic tag is used to indicate that either the user has opened the GUI (de-iconified it) by clicking on the GUI window "de-iconify" button, or ***Mentor*** has done this under program control. Two examples follow:

```
<popup date="Tue Mar 26 09:56:54 PST 2002" seconds="1017107814587"/>
 <input ...>
hi
</input>
<response responseCode="0" topicName="greeting" ...>
Speak Fred, and it will be done!
</response>

 <alertedUser date="Mon Apr 08 14:13:49 PDT 2002" seconds="1018242829435"/>
 <popup date="Mon Apr 08 14:13:54 PDT 2002" seconds="1018242834421"/>
 <active topicName="UNKNOWN" ...>
Did you know, did you know....
The first mudbricks were used during 4000-3500 BC.
</active>
<input ...>
no
</input>
<response responseCode="0" topicName="didYouKnow" ...>
I like funny facts like that.
</response>
<input ...>
tell me more
</input>
```

The first example simply shows the how the user has opened up the GUI, said "hi" to ***Mentor*** and then continued to ask several more questions.

The second example shows how the the system has alerted (see later description) the user that it wants to say something (typically a pro-active output). The user at some later time has acknowledged the alert by popping up the GUI and the pro-active output has been sent to the GUI for displaying. The user has then responded and been engaged in a dialogue.

It is possible for ***Mentor*** to pop up the GUI without alerting the user. This may happen when a serious ‘problem’ exists, or for example, when the system is going down for maintenance, rebooting, etc.

The **popup** tag has 2 attributes, both of which are required for a valid tag:

date

CDATA / REQUIRED: The date of this popup.

seconds

CDATA / REQUIRED: The date of this popup in seconds since the Java epoch.

G.12.10 Format tag: alertedUser

This atomic tag is used to indicate that ***Mentor*** wants to tell the user something. The user is alerted in a GUI dependant way - the user has control over how this may happen. For example, the GUI may be iconified and the user may have set their preferences such that it de-iconifies and raises the window to the front. Or they may have set it such that the system just beeps, or the GUI icon flashes, or both. This gives the user control over the interruptions caused by the ***Mentor*** System.

Two examples follow:

```
<alertedUser date="Wed Apr 24 11:31:35 PDT 2002" seconds="....."/>

<alertedUser date="Mon Apr 08 22:24:59 PDT 2002" seconds="....."/>
<active topicName="jokes" date="Mon Apr 08 22:24:59 PDT 2002" seconds="......">
Do you like jokes?
</active>
```

The first example elicited no action from the user and so was not followed up. The second example shows how the user was alerted, and the GUI was already open, and so a pro-active question was output. Note that pro-active questions wait for a lull in the conversation - they do not just 'butt in'.

The **alertedUser** tag has 2 attributes, both of which are required for a valid tag:

date

CDATA / REQUIRED: The date of this alertedUser.

seconds

CDATA / REQUIRED: The date of this tag in seconds since the Java epoch.

G.12.11 Format tag: URLClick

This atomic tag is used to indicate that the user has clicked on a URL hot link that has been displayed in the GUI. An example follows:

```
<input ...>
Help me with agv
</input>
<response responseCode="0" topicName="cg351UnitTopic" ...>
The GTK home page was a good place to start. The FAQ is also good - have you consulted that?
</response>
<input ...>
no
</input>
<response responseCode="0" topicName="cg351UnitTopic" ...>
It is at <a href="http://www.gtk.org/faq">http://www.gtk.org/faq</a>, Sue
</response>
 <URLClick date="....." seconds="....." href="http://www.gtk.org/faq" />
```

The user has asked for help, has received a response with a URL in it and the user has followed that URL to the Web page indicated.

The **URLClick** tag has 3 attributes, all of which are required for a valid tag:

href

CDATA / REQUIRED: The URL of this URLClick tag.

Example:http://www.mentor.computing.edu.au/documentation/images.html#GDKIMAGE

date

CDATA / REQUIRED: The date of this URLClick.

seconds

CDATA / REQUIRED: The date of this URLClick tag in seconds since the Java epoch.

G.12.12 Format tag: endsession

This atomic tag is used to indicate that the user session has ended because the user has logged off the system and the GUI has closed or been killed. An example follows:

```
<endsession date="Fri Apr 05 15:42:04 PST 2002" seconds="1017992524623"/>
</session>
```

Note that an **endsession** is needed if we want to record the time of the closing of the session. It is not possible for a closing tag to have attributes in XML.

The **endsession** tag is always followed by the </**session**> tag.

The **endsession** tag has 2 attributes, both of which are required for a valid tag:

date

CDATA / REQUIRED: The date of this endsession.

seconds

CDATA / REQUIRED: The date of this endsession in seconds since the Java epoch.

G.12.13 Data Integrity Issues

It should be noted that this format has evolved due to the first informal and three formal case studies. Formative evaluation of the user input data has led to this evolution. This is the reason why some attributes are optional and some early data-sets do not contain instances of some of the tags.

It should also be noted that since the GUI client and the ***Mentor*** server are both written in Java, it is possible for these to die in an untidy way and hence the user files may not be closed cleanly. This is specifically a problem when users kill the client through using the Unix 'kill a process' key. This results in un-closed sessions in their files. Since the output to these files is buffered, it can also cause problems through missing buffered data not being recorded.

In later versions of Java, used after the three formal studies, it was possible to use the `Runtime addShutdownHook` method specifically to catch being 'killed'.

In a similar way, if the server dies or has to be killed / rebooted for some reason, this can also cause open files to lose data. A mechanism exists within the ***Mentor*** System for an authorised user to shut the system down or reboot it in a controlled way. But Java programs do 'crash'. Java servers are especially prone to problems of memory leaks and hanging network connections.

Since it is possible for users to load their own topics to be used in the system, this too can cause problems for the server. Although a Java SecurityManager is installed on the server to stop malicious or incorrect user code, it is possible for a user to simply crash the server, although the mechanism is not obvious.

These issues caused Data Integrity problems which meant that the recorded data had to be hand checked and possibly corrected before being analysed.

G.12.14 Document Type Definition

The following is taken directly from the '*Frequently Asked Questions about the Extensible Markup Language*' edited and maintained by Peter Flynn (peter@silmaril.ie) and available at http://www.ucc.ie:8080/cocoon/xmlfaq [HREF 65].

A Document Type Definition (DTD) is a formal description in XML Declaration Syntax of a particular type of document. It sets out what names are to be used for the different types of element, where they may occur, and how they all fit together. For example, if you want a document type to be able to describe Lists which contain Items, the relevant part of your DTD might contain something like this:

```
<!ELEMENT List (Item)+>
<!ELEMENT Item (#PCDATA)>
```

This defines a list as an element type containing one or more items (that's the plus sign); and it defines items as element types containing just plain text (Parsed Character Data or PCDATA). Validating parsers read the DTD before they read your document so that they can identify where every element type ought to come and how each relates to the other, so that applications which need to know this in advance (most editors, search engines, navigators, databases) can set themselves up correctly. The example above lets you create lists like:

```
<List><Item>Chocolate</Item>
<Item>Music</Item>
<Item>Surfing</Item></List>
```

How the list appears in print or on the screen depends on your stylesheet: you do not normally put anything in the XML to control formatting like you had to do with HTML before stylesheets. This way you can change style easily without ever having to edit the document itself.

A DTD provides applications with advance notice of what names and structures can be used in a particular document type. Using a DTD when editing files means you can be certain that all documents which belong to a particular type will be constructed and named in a consistent and conformant manner. DTDs are less important for processing documents already known to be well-formed, but they are still needed if you want to take advantage of XML's special attribute types like the built-in ID/IDREF cross-reference mechanism, or the use of default attributes.

There are thousands of DTDs already in existence in all kinds of areas (see the SGML/XML Web pages for pointers). Many of them can be downloaded and used freely; or you can write your own (see the question on creating your own DTD [HREF 66]. Existing SGML DTDs need to be converted to XML for use with XML systems: and popular SGML DTDs are becoming available in XML format.

The alternative to a DTD is a Schema [HREF 67], which is written in Instance Syntax and provides much more extensive validation facilities.

G.12.15 Mentor Data Format Document Type Definition

```
<?xml encoding="ISO-8859-1"?>
<!--
#################################################################
# Mentor data Format (DMTL) DTD, version 2.0.                   #
#                                                               #
# Usage:                                                        #
# <!DOCTYPE dialogue SYSTEM                                     #
#   "http://www.mentor.computing.edu.au/DTD/mentor.dtd">        #
#                                                               #
# Author   : Andrew Marriott                                    #
# Copyright: Andrew Marriott 2000-2003                          #
# Date: 17 October 2001                                         #
#                                                               #
# Information about the Mentor System can be found at           #
#        http://www.mentor.computing.edu.au/                    #
#                                                               #
#################################################################
-->
<!-- mentor DTD -->

<!ENTITY % XHTML "a |
                  anchor|
                  img
                        ">
<!ENTITY % id "CDATA">

<!ENTITY % link-type-list "CDATA">

<!ENTITY % character-list "CDATA">

<!ENTITY % uri "CDATA">

<!ENTITY % integer "CDATA">

<!ENTITY % coordinate-list "CDATA">

<!ENTITY % script "CDATA">


<!ENTITY % default-XHTML-attributes
           "accesskey %id; #IMPLIED
            coords %coordinate-list; #IMPLIED
            onblur %script; #IMPLIED
            onfocus %script; #IMPLIED
            shape (default | rect | circle | poly) #IMPLIED
            tabindex %integer; #IMPLIED">
      <!-- The tabindex must be between 0 and 32,767 -->
```

```
<!-- TEXT DESCRIPTION TAGS -->
<!ELEMENT mentor (session)*>

<!ELEMENT session ((active*,timeout*,alertedUser*,
                  popdown*,popup*,
                  input*,response*,
                  URLClick*)*,endsession)>
<!ATTLIST session
      user CDATA #REQUIRED
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT input (#PCDATA)>
<!ATTLIST input
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT response (#PCDATA|%XHTML;)*>
<!ATTLIST response
      responseCode CDATA #REQUIRED
      date CDATA #REQUIRED
      topicName CDATA #IMPLIED
      seconds CDATA #REQUIRED>

<!ELEMENT active (#PCDATA|%XHTML;)*>
<!ATTLIST active
      date CDATA #REQUIRED
      topicName CDATA #IMPLIED
      seconds CDATA #REQUIRED>

<!ELEMENT timeout (#PCDATA|a)*>
<!ATTLIST timeout
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT popdown EMPTY>
<!ATTLIST popdown
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT popup EMPTY>
<!ATTLIST popup
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT endsession EMPTY>
<!ATTLIST endsession
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>
```

```
<!ELEMENT URLClick EMPTY>
<!ATTLIST URLClick
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED
      HREF CDATA #REQUIRED>

<!ELEMENT alertedUser EMPTY>
<!ATTLIST alertedUser
      date CDATA #REQUIRED
      seconds CDATA #REQUIRED>

<!ELEMENT a (#PCDATA)>
<!ATTLIST a %default-XHTML-attributes;
            charset %character-list; #IMPLIED
            href %uri; #IMPLIED
            hreflang NMTOKEN #IMPLIED
            name %id; #IMPLIED
            rel %link-type-list; #IMPLIED
            rev %link-type-list; #IMPLIED
            type NMTOKEN #IMPLIED>

<!ELEMENT anchor (#PCDATA)>
<!ATTLIST anchor %default-XHTML-attributes;
                 charset %character-list; #IMPLIED
                 href %uri; #IMPLIED
                 hreflang NMTOKEN #IMPLIED
                 name %id; #IMPLIED
                 rel %link-type-list; #IMPLIED
                 rev %link-type-list; #IMPLIED
                 type NMTOKEN #IMPLIED>


<!ELEMENT img EMPTY>
<!ATTLIST img %default-XHTML-attributes;
  src         %uri;          #REQUIRED
  alt         CDATA         #REQUIRED
  longdesc    %uri;          #IMPLIED
  height      CDATA       #IMPLIED
  width       CDATA       #IMPLIED
  usemap      %uri;          #IMPLIED
  ismap       (ismap)        #IMPLIED
  >
```

> *Better to light one small candle than to curse the darkness.*
> Chinese Proverb.

H: Example User Topics Showing API for *Mentor* System

This appendix shows a template for the format of a standard user Topic as well as 3 example Topics. These are:

- `myTemplateTopic.java`
- `ItalyTopic.java`
- `cheeseTopic.java`
- `didYouKnowTopic.java`

These Topic examples are used by those wishing to build their own Domain Knowledge. These Topics are dynamically loaded across the network into the ***Mentor*** System by the MentorClient at connection time, so as to extend ***Mentor***'s knowledge.

These listings have been edited and formatted to conform to the structure of the A4 pages.

These can be found on the CDROM in the folder /userTopics.

```
package local.mentor.topics;

import java.lang.*;
import java.io.*;
import java.util.*;

// Needed to be able to do Regular Expression Parsing.
import org.apache.oro.text.regex.*;

import local.raytrace.DialogueManager.*;
import local.raytrace.Topics.*;

// Must extend userTopic or it will not be loaded.
public class myTemplateTopic extends userTopic
    implements Serializable
        , TopicsInterface
{
protected transient String patterns[] =
{
//  First entry is the general area to match
//  The Dialogue Manager looks to see if this area matches this particular
//  Topic. If not then do not look at any further matching. This is for
//  efficiency. All matches must be in lower case as the input line is
//  converted to lower case.
//
//  See http://www.talkingheads.computing.edu.au/documentation/ORO
//  for information on the form of the regular expressions.
//
//  See ItalyTopic.java for an example of how to use this.

   "YOUR ENTRY PATTERN IN HERE - lower case",

//  Now we have any other patterns that you may want to test to find out
//  exactly what was typed.
//  See ItalyTopic.java for an example of how to use this.
//  Remember that some RE need double escaping such as
//  \s and \b.
   "YOUR FIRST PATTERN IN HERE - lower case",
   "YOUR SECOND PATTERN IN HERE - lower case",


// You may like to add an optional help me with type pattern
   HELP_ME_WITH_STRING + "(YOUR HELP_ME_WITH_STRING IN HERE - lower case)",

// You may like to add an optional match anything type pattern
   ".*",         // match everything in a non-specific way
};

   public myTemplateTopic()
   {
       // Tell super class what type of Topic we are. This should be unique.

       super ("myTemplate");

       // And we need to have a one line description of this Topic so Dialogue Manager
       // can let the world know what it knows about.
       briefTopicsHelpString = "myTemplate - knowledge about XXX.";

       // And also a longer description about this Topic. This can be as long as you like and
       // can use the normal Dialogue Manager variables. This is used if the
       // Dialogue Manager is asked for specific help about this Topic.
       topicsHelpString =  "Your description in here.";

       // Now we need to compile these Regular Expressions into proper internal format.
       // This version passes a general keyword pattern as well as the Regular Expression
       // patterns. The keyword pattern is used to match against requests for help on this
       // particular Topic. It is often patterns[0] - ie. the general area to match against
       // but it could be a separate string such as
       //   protected transient String keywordHelpPatternString = ".*italy.*"
       //
       // There is also another version of this called
       //     makeSpecificPatterns(secondaryPatterns) for creating other compiled RE's.
       makePatterns(patterns[0],patterns);
   }
```

MyTemplateTopic.java (part 1/2)

```
    // This is the entry point from the DialogueManager for each Topic. The DM
    // has already checked the general area of this topic via entry[0] of the above patterns.
    // It matched, therefore lets now check exactly what was typed to see if we can recognise it....
    public Response checkTopic(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
    boolean found = false;
    int i;
        response = null;     // Indicate that default is no match....

        // Now check all patterns to see if they match the input string. Note we have already used index 0
        // to get the general area of the match so we normally check from index 1 onwards.
        for (i = 1; i<pattern.length;i++)
        {
            // We can turn on debugging by setting Topics.debug=true in the properties file
            // in the Properties/DM.properties file where the Dialogue Manager is run, or in the classpath.
            if(Debugging(context))
              topicDebug(context,"Checking "+topic_name+":"+context.input+": vs :"+pattern[i].getPattern()+":");
            if(matcher.matches(context.input,pattern[i]))
            {
                found = true;
                if(Debugging(context)) topicDebug(context,"Found!");
                break;
            }
        }
        if(found)
        {
            switch(i)
            {
            case 1:       // XXX
            break;

            case 2:       // XXX
            break;

            default:  // The last pattern entry (if we wanted it) of ".*" which matches everything
                      // So we matched the area but not the input. In this case let the Response class
                      // pick a random response and let it search its history so we suppress similar responses.
            }
        }
        return response;
    }
////////////////////////////////////////////////////////////////////////////////
    public void setTopicMultiModal(DialogueManagerInterface dmi)
    {
        dmi.registerMultiModal(DialogueManagerAdapter.PLAINTEXT,this);
    }
    public Response getTopicsActiveQueryResponse(DialogueManagerInterface dm)
    {
        return (Response)null;
    }
    public void saveTopicState(DialogueManagerServer dms,String basedir)
    {       // do nothing in this case.
    }
    public void topicsActiveQueryInitialise(DialogueManagerInterface dm)
    {       // do nothing in this case.
    }
    public boolean initialiseTopic(DialogueManagerServer dms)
    {
        // do nothing in this case.
        return true;
    }
    public void tidyUp(Context context,Response response)
    {        // do nothing in this case.
    }
    public boolean checkValidState(String topic_string, Vector Prev_Query_State, Context context)
    {    // do nothing in this case.
        return true;
    }
    public Response checkFollowupResponse(Context context, String topic_string, Vector Prev_Query_State)
    {
        return (Response)null;
    }
////////////////////////////////////////////////////////////////////////////////
    public static void main(String[] args)
        throws Exception
    {
        try
        {
        myTemplateTopic topic = new myTemplateTopic();
                topic.testTopic();
        }
        catch (Exception e)
        {
            System.err.println("Your topic could not be instantiated.");
            System.err.println("This is probably because of a bad Regular Expression.");
            throw e;
        }
    }
}
```

MyTemplateTopic.java (part 2/2)

```
package local.mentor.topics;

import java.lang.*;
import java.io.*;
import java.util.*;

// Needed to be able to do Regular Expression Parsing.
import org.apache.oro.text.regex.*;


import local.raytrace.DialogueManager.*;
import local.raytrace.Topics.*;

public class ItalyTopic extends userTopic
    implements Serializable, TopicsInterface
{
private static final boolean debug = true;

protected transient String patterns[] =
{
//  First entry is the general area to match. The Dialogue Manager looks to see if this area matches this
//  particular Topic. If not then do not look at any further matching. This is for efficiency.
//  All matches must be in lower case as the input line is converted to lower case.
//
//  In this case, anything which has the word italy or italian (singular or plural) or the misspelt
//  italien (singular or plural).
//  See http://www.talkingheads.computing.edu.au/documentation/ORO
//  for information on the form of the regular expressions.
//
//  The first example is "any characters, then a word boundary, then either the word italy or italian or
//  italien with optional "s" followed by word boundary, followed by anything.

    ".*\b(italy|italian(s)?|italien(s)?)\b.*",

//  Now we have any other patterns that you may want to test to find out exactly what was typed.
//  Note that this only matches exactly "where is italy" not "where is italy?"
//  nor "where's italy" nor "where is italy!"
//  Also remember that some RE need double escaping such as \s and \b.
    "where\s*is\s*italy",

//  A better pattern for the above question is shown below which matches "where is", "wheres" or
//  "where's" and allows spaces and PUNCTUATION at the end of the string. The uppercase
//  PUNCTUATION_STRING is defined in the super class.
//      In fact, PUNCTUATION as well as some common polite words such as please and
//  thank you are removed as leading and trailing words so we do not need PUNCTUATION_STRING
//  For more information on these strings, look at
//    http://www.mentor.computing.edu.au/mentor/help/reference/TopicStrings.html
    "where(((')?s)|(\s*is))\s*italy", // + PUNCTUATION_STRING,

//  We could match "i am italian" but this is used elsewhere in another Topic
//  to set the language of the greetings, etc. Let us try "who is italian"
    WHOIS_STRING + "\s*(italian|italien)", //+ PUNCTUATION_STRING,

//  Finally in this example, the huge HELP_ME_WITH_STRING
    HELP_ME_WITH_STRING + "((learning|learn)\s*(italien|italian))", // + PUNCTUATION_STRING,

    ".*",           // match everything italian in a non-specific way
};

    public ItalyTopic()
    {
        // Tell super class what type of Topic we are. This should be unique.
        super ("Italy");

        // And we need to have a one line description of this Topic so Dialogue Manager
        // can let the world know what it knows about.
        briefTopicsHelpString = "Italy - knowledge about Italy and its culture.";

        // And also a longer description about this Topic. This can be as long as you like and
        // can use the normal Dialogue Manager variables. This is used if the
        // Dialogue Manager is asked for specific help about this Topic.
        topicsHelpString =  "<getvar name=\"first_name\"/>, you may ask me about Italy, its people,\n" +
                    "its culture, its food, its bad drivers!!\n" +
                    "I will try to help you with my knowledge.\n" ;

        // Now we need to compile these Regular Expressions into proper internal format.
        // This version passes a general keyword pattern as well as the Regular Expression
        // patterns. The keyword pattern is used to match against requests for help on this
        // particular Topic. It is often patterns[0] - ie. the general area to match against
        // but it could be a separate string such as
        //   protected transient String keywordHelpPatternString = ".*italy.*"
        //
        // There is also another version of this called
        //     makeSpecificPatterns(secondaryPatterns) for creating other compiled RE's.
        //
        //  This method creates a compiled regular expression array in the super class called:
        //      protected Pattern pattern[]
        makePatterns(patterns[0],patterns);
    }
```

ItalyTopic.java (part 1/4)

```
// Here we have our responses to the input from the user. We may want to have several so that we can vary the
// response. These are randomly chosen from:
private static final String whereResponses[] =
{
    "Bhh,Italy is in Europe!",
    "Bhh,Italy is in the Mediterranean",
    "Bhh,Italy is where your heart is!",
    "Bhh,Italy is where your stomach is.",
};
private static final String betterWhereResponses[] =
{
    "Rough match: Italy is in Europe!",
    "Rough match: Italy is in the Mediterranean",
    "Rough match: Italy is where your heart is!",
    "Rough match: Italy is where your stomach is.",
};
private static final String a_whoIsResponses[] =
{
    "Fabriccio Ravanelli!",
    "Roberto Pockaj even though his name says otherwise!",
    "Fabio Lavagetto eats like an Italian",
    "Maurizio Costa seems Italian.",
    "Carlo Bonamico is a nice Italian.",
    "Sophia Lauren looks Italian.",
};
private static final String e_whoIsResponses[] =
{
    "Anyone who gets into the 25 yard box and takes a dive!",
    "<getvar name=\"first_name\"/>, it could be you who is Italien",
    "The Pope",
    "Gian Franco Zola.",
    "Anyone born in Italy.",
};
private static final String ehResponses[] =
{
    "I am sorry, <getvar name=\"first_name\"/>, I could not work out what you asked about Italy.",
    "<getvar name=\"first_name\"/>, I did not understand your request about Italy. Sorry.",
    "Non capiche! Italy???",
    "Sorry <getvar name=\"first_name\"/>, I did not understand what you asked about Italy.",
};

    //This is the entry point from the DialogueManager for each Topic. The DM
    // has already checked the general area of this topic via entry[0] of the above patterns.
    // It matched, therefore lets now check exactly what was typed to see if we can recognise it....
    public Response checkTopic(Context context)
    {
        // The following can be used to enable tracing of the code as debug sent back to the MentorClient.
        // It is enabled by setting
            tracing = true;

        topicTrace(context,"checkTopic start:");
        logClientData(context,"checkTopic start:");

    PatternMatcher matcher  = new Perl5Matcher();
    boolean found = false;
    int i;
    String debugString = "";
    String debug_string = "";

        response = null;     // Indicate that default is no match....

        // Now check all patterns to see if they match the input string. Note we have already used index 0
        // to get the general area of the match so we normally check from index 1 onwards.
        for (i = 1; i<pattern.length;i++)
        {
            // We can turn on debugging by setting debug=true above
            if(debug)
            {
                debug_string =  "Checking " + topic_name + ":" + context.input + ": against :" +
                                pattern[i].getPattern() + ":";
                debugString += debug_string + "\n";
            }

            topicTrace(context,"checkTopic " + debug_string + ":");

            if(matcher.matches(context.input,pattern[i]))
            {
                found = true;
                if(debug)
                    debugString += "Found at index " + i +"\n";
                topicTrace(context,"checkTopic : Found a pattern match at index " + i);
                break;
            }
        }
        // We can also send debug information back to the MentorClient by setting
        // debug = true; at the top of this class......

        if(debug)
                sendMessageToUser(context, debugString);
```

ItalyTopic.java (part 2/4)

```
  if(found)
  {
    topicTrace(context,"checkTopic :Switch:" + i);
    switch(i)
    {
    case 1:      // where is italy
        int wchoice = Math.abs(random.nextInt())% whereResponses.length ;// Get a string from the response array.
            response = new Response(this,whereResponses[wchoice]);
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
    break;

    case 2:      // wheres italy // + PUNCTUATION
        int bchoice = Math.abs(random.nextInt())% betterWhereResponses.length ;// Get a string from the array.
            response = new Response(this,betterWhereResponses[bchoice]);
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
    break;

    case 3:      // whois italian //+ PUNCTUATION
            // Lets also use a Match result to find out if they typed italien or italian
        MatchResult result = matcher.getMatch();
        String typed = result.group(1); // Look at the first (xxx) remembered match.
            if(typed == null)
            {
                topicDebug(context, "Got a who is match but with a null MatchResult!"); // log it.
                return null;       // Strange since we did match but just in case..
            }

            if(typed.length() == 0)
            {
                topicDebug(context, "Got a who is match but with a zero length MatchResult!"); // log it.
                return null;       // Strange since we did match but just in case..
            }

            typed = typed.trim();// get rid of extra white space
            if(typed.equals("italian"))// english spelling
            {
            int echoice = Math.abs(random.nextInt())% a_whoIsResponses.length ;// Get a string from the array.
                response = new Response(this,Response.uniqueResponseString(context.dm,a_whoIsResponses));
                response.responseData.response_weight = 0.75F;  // moderate weighting
            }
            else if (typed.equals("italien"))// non-english spelling
            {
            int achoice = Math.abs(random.nextInt())% e_whoIsResponses.length ;// Get a string from the array.
                response = new Response(this,Response.uniqueResponseString(context.dm,e_whoIsResponses));
                response.responseData.response_weight = 0.75F;  // moderate weighting
            }
            else                  // how could this be???
            {
                response = new Response(this,Response.uniqueResponseString(context.dm,ehResponses));
                response.responseData.response_weight = 0.5F;  // low weighting. Something better may match
            }

    break;

    case 4:      // help me learn italian...
                // The following <random>.....</random> chooses a random one of the <li> items
                // specified. Note that this is XML and so all tags must be properly closed
                // and attributes quoted.
            response = new Response(this,
                        "Why not try looking at " +
                        "<random>" +
                        "<li><a href=\"http://www.wuziegames.com/learnitalian.html\">" +
                            "http://www.wuziegames.com/learnitalian.html</a></li>" +
                        "<li><a href=\"http://italian.about.com/library/fare/index.htm?once=true&amp;\">" +
                            "italian.about.com</a></li>" +
                        "<li><a href=\"http://confucius.gnacademy.org:8001/mason/catalog/search.html?" +
                            "p=italian&amp;xyz_max_hits=100\">the GNA academy.</a></li>" +
                        "</random>"
                                );
            response.responseData.response_weight = 0.75F;  // moderate weighting

    break;

    default:  // The last pattern entry (if we wanted it) of ".*" which matches everything
            // So we matched the area but not the input.
            // In this case let the Response class pick a random response and let it search its
            // history so we suppress similar responses.
            response = new Response(this,
                            Response.uniqueResponseString(context.dm,ehResponses)
                            );
            response.responseData.response_weight = 0.5F;  // low weighting. Something better may match

    }
  }

  topicTrace(context,"checkTopic end:" + response.fullResponse());
  return response;
  }
```

ItalyTopic.java (part 3/4)

```
////////////////////////////////////////////////////////////////////////////////

    public void setTopicMultiModal(DialogueManagerInterface dmi)
    {
        dmi.registerMultiModal(DialogueManagerAdapter.PLAINTEXT,this);
    }

    public Response getTopicsActiveQueryResponse(DialogueManagerInterface dm)
    {
        return (Response)null;
    }

    public void saveTopicState(DialogueManagerServer dms,String basedir)
    {
        // do nothing in this case.
    }

    public void topicsActiveQueryInitialise(DialogueManagerInterface dm)
    {
        // do nothing in this case.
    }

    public boolean initialiseTopic(DialogueManagerServer dms)
    {
        // do nothing in this case.
        return true;
    }

    public void tidyUp(Context context,Response response)
    {
        // do nothing in this case.
    }

    public boolean checkValidState(String topic_string, Vector Prev_Query_State,
                                   Context context)
    {
        return true;
    }

    public Response checkFollowupResponse(Context context, String topic_string,
                                   Vector Prev_Query_State)
    {
        return (Response)null;
    }
////////////////////////////////////////////////////////////////////////////////

    public static void main(String[] args)
        throws Exception
    {
        try
        {
        ItalyTopic topic = new ItalyTopic();
                topic.testTopic();
        }
        catch (Exception e)
        {
            System.err.println("Your topic could not be instantiated.");
            System.err.println("This is probably because of a bad Regular Expression.");
            throw e;
        }
    }
}
```

ItalyTopic.java (part 4/4)

```
package local.mentor.topics;

import java.lang.*;
import java.util.*;
import java.io.*;

import org.apache.oro.text.regex.*;

import local.raytrace.Syslog.*;
import local.raytrace.DialogueManager.*;
import local.raytrace.Topics.*;

public class cheeseTopic extends userTopic
     implements Serializable, TopicsInterface
{
private static final boolean debug = true;

protected transient String keywordHelpPatternString = "cheese";
protected transient String patterns[] =
{
//  First entry is the general area to match in this case either cheese or like.
    ".*\b(like|cheese)\b.*",
   "i\s*like\s*cheese",
   "do\s*" + YOU_STRING + "\s*like\s*cheese",
   "do\s*" + YOU_STRING + "\s*like\s*(.*)",
};

    public cheeseTopic()
    {
        super ("cheese");

        briefTopicsHelpString = "The sometime smelly topic of Cheese.";
        topicsHelpString =  "This is the slightly less than famous cheese topic.\n" +
            "It is only an example of using the next state technique in a topics file.\n" +
            "For more information, the gentle reader is asked to visit the Monty Python Cheese Shop site at\n" +
            "<a href=\"http://www.ironworks.com/comedy/python/cheeshop.htm\">" +
            "http://www.ironworks.com/comedy/python/cheeshop.htm</a>.\n" ;

        makePatterns(keywordHelpPatternString,patterns);
        YESNO_pattern = makeSpecificPatterns(YESNOPatterns);
        cheeseophobe_pattern = makeSpecificPatterns(cheeseophobePatterns);
        allCheese_pattern = makeSpecificPatterns(allCheesePatterns);
    }

    public Response checkTopic(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
    boolean found = false;
    int i;

        response = null;
        for (i = 1; i<pattern.length;i++)
        {
            if(matcher.matches(context.input,pattern[i]))
            {
                found = true;
                break;
            }
        }
        if(found)
          switch (i)
          {
          case 1:
            return processCheeseRequest(context);// See end of class.

          case 2:
            return processCheeseTasteRequest(context);

          case 3:
            return processSpecificCheeseTasteRequest(context,matcher);
          }
        return (response);
    }

    public Response checkFollowupResponse(Context context, String topic_string, Vector Prev_Query_State)
    {
    String state_string = null;

    int  index = topic_string.indexOf(':');
    String debugString = "";
        if (index != -1)
            state_string = topic_string.substring(index+1);

            // We can turn on debugging by setting debug=true above
        if(debug)
            debugString +=  "checkFollowupResponse("+topic_name +") state_string is " + state_string;

        if(debug)
                sendMessageToUser(context,debugString);
```

cheeseTopic.java (part 1/6)

```
        if(state_string.equals("checkIfLikeRandomCheese"))
            return checkIfLikeRandomCheese(context);
        else if(state_string.equals("checkWhatTypeOfCheese"))
            return checkWhatTypeOfCheese(context);
        else if(state_string.equals("checkIfLikeCheddar"))
            return checkIfLikeCheddar(context);
        else if(state_string.equals("checkCheeseOPhobe"))
            return checkCheeseOPhobe(context);
        else
            return null;
    }

////////////////////////////////////////////////////////////////////////////////////////////
//  Current state Responses - they said "I like cheese"
////////////////////////////////////////////////////////////////////////////////////////////
// In reality, do you like <a random cheese name>
private static final String doYouLikeRandomCheeseResponses1[] =
{
    "Me also <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
    "Do you like <getvar name=\"subject\"/>, <getvar name=\"first_name\"/>?",
    "Me too, do you like <getvar name=\"subject\"/>?",
    "So do I <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
};

    private Response processCheeseRequest(Context context)
    {
    Response response = null;
    int wchoice = Math.abs(random.nextInt())% doYouLikeRandomCheeseResponses1.length ;// Get string from array.
    int cchoice= Math.abs(random.nextInt())% cheeses.length;
        response = new Response(this,doYouLikeRandomCheeseResponses1[wchoice]);
        response.addToNext_Query_State(topic_name + ":checkIfLikeRandomCheese");
        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        response.putMatch("subject",cheeses[cchoice]);

        return response;
    }

////////////////////////////////////////////////////////////////////////////////////////////
//  Current state Responses - they asked "do you like cheese"
////////////////////////////////////////////////////////////////////////////////////////////
// In reality, do you like <a random cheese name>
private static final String doYouLikeRandomCheeseResponses2[] =
{
    "I do <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
    "Yes. Do you like <getvar name=\"subject\"/>, <getvar name=\"first_name\"/>?",
    "Mm! Do you like <getvar name=\"subject\"/>?",
    "Yes I do <getvar name=\"first_name\"/>. Do you like <getvar name=\"subject\"/>?",
};

    private Response processCheeseTasteRequest(Context context)
    {
    Response response = null;
    int wchoice = Math.abs(random.nextInt())% doYouLikeRandomCheeseResponses2.length ;// Get string from array.
    int cchoice= Math.abs(random.nextInt())% cheeses.length;
        response = new Response(this,doYouLikeRandomCheeseResponses2[wchoice]);
        response.addToNext_Query_State(topic_name + ":checkIfLikeRandomCheese");
        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        response.putMatch("subject",cheeses[cchoice]);

        return response;
    }

////////////////////////////////////////////////////////////////////////////////////////////
//  Current state Responses - they asked "do you like xxx"
////////////////////////////////////////////////////////////////////////////////////////////
    private Response processSpecificCheeseTasteRequest(Context context, PatternMatcher matcher  )
    {
    Response response = null;
        MatchResult result = matcher.getMatch();
        String cheese = result.group(1);
        boolean cheeseQuestion = true;

            if(debug)
                    sendMessageToUser(context,"cheese match is "+ cheese);
            if(cheese == null)
                cheeseQuestion = false;
            else
            {
            int indexOfBracket = PUNCTUATION_STRING.indexOf('[');
                for(int i = indexOfBracket+1; PUNCTUATION_STRING.charAt(i) != ']'; i++)
                    cheese = cheese.replace(PUNCTUATION_STRING.charAt(i),' ');

                cheese = cheese.trim();

                if(cheese.length() == 0)
                    cheeseQuestion = false;
            }
```

cheeseTopic.java (part 2/6)

```
            if(cheeseQuestion)
            {
                if(cheesePlatter == null)
                {
                    cheesePlatter = new Hashtable();
                    for(int i = 0; i < cheeses.length; i++)
                        cheesePlatter.put(cheeses[i],"");
                }

                if(debug)
                    sendMessageToUser(context,"cheese match is "+ cheese);
                if(cheesePlatter.containsKey(cheese))
                {
                    if(debug)
                        sendMessageToUser(context, "Found recognised: " + cheese);
                }
                else
                    return null;
            }

    int wchoice = Math.abs(random.nextInt())% doYouLikeRandomCheeseResponses2.length ;
    int cchoice= Math.abs(random.nextInt())% cheeses.length;
        response = new Response(this,doYouLikeRandomCheeseResponses2[wchoice]);
        response.addToNext_Query_State(topic_name + ":checkIfLikeRandomCheese");
        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        response.putMatch("subject",cheeses[cchoice]);

        return response;
    }
////////////////////////////////////////////////////////////////////////////////////////////
//  checkIfLikeRandomCheese state Responses
////////////////////////////////////////////////////////////////////////////////////////////
// We need to ensure that this and any other patterns we use is compiled in the constructor.
//      YESNO_pattern = makeSpecificPatterns(YESNOPatterns);
// The state processing method is classified in checkFollowupResponse and is called
// based upon the next state string - in this case "checkIfLikeRandomCheese"
protected Pattern YESNO_pattern[] = null;
protected transient final String YESNOPatterns[] =
{
    YES_STRING,     // maybe better is YES_STRING+"\s*"+PUNCTUATION_STRING+"\s*.*"   yes i like cheese
    NO_STRING,
    "(?:" + WHATIS_STRING + "\s*(?:that|taht|(.*))\s*)|" +
    "(?:(?:"+ IVE_STRING + "\s*)?\s*never\s*heard\s*of\s*" +
                "(?:it|(?:(?:that|taht)\s*(?:\s*cheese)?\s*)|(.*)))" + PUNCTUATION_STRING
};
private static final String whatOtherTypesResponses[] =
{
    "What other types of cheese do you like?",
    "What other types do you like <getvar name=\"first_name\"/>?",
    "Me too, what other types?",
    "So do I <getvar name=\"first_name\"/>. What other types?",
};
private static final String iDoNotLikeRandomCheeseResponses[] =
{
    "OK, it is a bit smelly. Do you like cheddar?",
    "It is a bit pongy. A bit like Stinky Bishop! What about cheddar? Do you like that?",
    "Some find it a bit strong. What about English Cheddar?",
    "<getvar name=\"first_name\"/>, it is strong! Do you like cheddar?",
};
private static final String neverHeardOfResponses[] =
{
    "<getvar name=\"subject\"/> is a famous cheese in these parts!",
    "It's one of the most popular cheeses in the world!",
    "It's staggeringly popular in this manor, squire!",
    "It's a number one best seller!",
    "I am sure that the americans make \"Cheesy Puffs\" out of it!",
};

    private Response checkIfLikeRandomCheese(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
        // Now check patterns in YES or NO to see if they match the input string.
        if(matcher.matches(context.input,YESNO_pattern[0]))
        {
            if(debug) sendMessageToUser(context, "Found! YES");
        int wchoice = Math.abs(random.nextInt())% whatOtherTypesResponses.length ;
            response = new Response(this,whatOtherTypesResponses[wchoice]);
            response.addToNext_Query_State(topic_name + ":checkWhatTypeOfCheese");
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        }
        else if(matcher.matches(context.input,YESNO_pattern[1]))
        {
            if(debug) sendMessageToUser(context, "Found! NO");
        int wchoice = Math.abs(random.nextInt())% iDoNotLikeRandomCheeseResponses.length ;
            response = new Response(this,iDoNotLikeRandomCheeseResponses[wchoice]);
            response.addToNext_Query_State(topic_name + ":checkIfLikeCheddar");
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        }
```

cheeseTopic.java (part 3/6)

```
        else if(matcher.matches(context.input,YESNO_pattern[2])) // incredulity!!!!
        {
            if(debug) sendMessageToUser(context, "Found! Never heard of it!");
        MatchResult result = matcher.getMatch();
        String cheese = result.group(1);
        boolean cheeseQuestion = true;

            if(debug) sendMessageToUser(context, "cheese match is "+ cheese);
            if(cheese == null)
                cheeseQuestion = false;
            else
            {
            int indexOfBracket = PUNCTUATION_STRING.indexOf('[');
                for(int i = indexOfBracket+1;
                        PUNCTUATION_STRING.charAt(i) != ']'; i++)
                {
                    cheese.replace(PUNCTUATION_STRING.charAt(i),' ');
                }

                cheese = cheese.trim();
                if(cheese.length() == 0)
                    cheeseQuestion = false;
            }
            if(cheeseQuestion)
            {
                if(cheesePlatter == null)
                {
                    cheesePlatter = new Hashtable();
                    for(int i = 0; i < cheeses.length; i++)
                        cheesePlatter.put(cheeses[i],"");
                }

                if(debug) sendMessageToUser(context, "cheese match is "+ cheese);
                    if(cheesePlatter.containsKey(cheese))
                {
                    if(debug)
                        sendMessageToUser(context, "Found recognised: " + cheese);
                }
                else
                    return null;
            }

        int wchoice = Math.abs(random.nextInt())% neverHeardOfResponses.length ;
            response = new Response(this,neverHeardOfResponses[wchoice]);
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
            response.putMatch("subject",cheese);
        }

        return response;
    }
////////////////////////////////////////////////////////////////////////////////////////////
//  checkWhatTypeOfCheese state Responses
////////////////////////////////////////////////////////////////////////////////////////////
// We need to ensure that this and any other patterns we use is compiled in the constructor.
//      allCheese_pattern = makeSpecificPatterns(allCheesePatterns);
// The state processing method is classified in checkFollowupResponse and is called
// based upon the next state string - in this case "checkWhatTypeOfCheese"
protected Pattern allCheese_pattern[] = null;
protected transient final String allCheesePatterns[] =
{
    "i\s*(?:like|love|luv)\s*(.*)" + PUNCTUATION_STRING,
        "(.*)",
};

private String cheeses[] =
{
    // Ok, Now add your favoutite cheese here.....
    "bel paese", "bleu d'auvergne", "blue vinny", "bres bleu", "brick", "brie", "brie de meaux",
    "brie de pays", "bruson", "caerphilly", "camembert", "cantal", "carrier de lest", "case ness",
    "chaource", "chaumes", "cheddar", "cheshire", "chevre", "chewton", "chewton smoked", "colston basset",
    "compte", "cornish yarg", "cotronese", "cottage cheese", "cream cheese", "crottin de chavignol",
    "czech sheeps milk", "danablu", "danish bimbo", "danish brew", "denhay", "dolce latte", "dolce latté",
    "dolcelatte", "dorset bluveny", "double gloucester", "duddleswell", "dunlop", "edam", "ementhal",
    "emmental", "epoisses de bourgogne", "feta", "flower marie", "fontina", "fribourgeois",
    "fromage a raclette", "fromage frais", "gjetost", "gloucester", "golden cross", "gorgonzola",
    "gouda", "grana", "grana padano", "greek feta", "gruyere", "gruyere de compte", "gruyère", "havarti",
    "hereford hop", "illchester", "japanese sage darby", "jersey blue", "klosterkäse", "lancashire",
    "lancashire", "leicester", "liederkranz", "limburg", "limburger", "lipta", "manchego", "maribo",
    "maroilles", "maroilles", "mimolette", "mont d'or", "monterey jack", "montgomery", "morbier",
    "mozarella", "mozzarella", "muenster", "munster", "neufchatel", "norwegian jarlsburg", "old plawhatch",
    "oxford blue", "paper cramer", "parmesan", "parmigiano", "passelan", "pecorino romano", "pecorino sardo",
    "pol le veq", "pont l'eveque", "port du salut", "port salut", "provolone", "raclette", "red leicester",
    "red windsor", "ricotta", "romano", "roquefort", "sage derby", "saint paulin", "saint-agur",
    "savoy aire", "sbrinz", "schwangenkäse", "scrumpy sussex", "shropshire blue", "single gloucester",
    "smoked austrian", "st loup", "st paulin", "stilton", "stinking bishop", "sussex slipcote",
    "swiss", "taleggio", "taleggio", "tete-de-moine", "tete de moine", "tilsit", "vacherin",
    "venezuelan beaver cheese", "vignotte", "wensleydale", "wensleydale", "white stilton", "yorkshire blue",
};
```

cheeseTopic.java (part 4/6)

```
private static Hashtable cheesePlatter = null;

private static final String knownCheeseResponses[] =
{
    "Ahh! A cheese above all other cheeses!",
    "Yes,<getvar name=\"subject\"/> is a nice cheese.",
    "Some find it to their liking.",
    "<getvar name=\"first_name\"/>, <getvar name=\"subject\"/> is nice!",
};

private static final String unknownCheeseResponses[] =
{
    "I am sorry <getvar name=\"first_name\"/>, I do not know that cheese!",
    "Never heard of that cheese <getvar name=\"first_name\"/>",
    "That cheese is new to me <getvar name=\"first_name\"/>",
    "I have not heard of <getvar name=\"subject\"/> <getvar name=\"first_name\"/>",
    "Well, <getvar name=\"subject\"/> must be a strange cheese. I have never heard of it!",
};

    private Response checkWhatTypeOfCheese(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
    //  In this case we come in with an arbitrary response
    int i;
    boolean found = false;
    Response response = null;

        if(matcher.matches(context.input,allCheese_pattern[0]))
        {
            if(debug) sendMessageToUser(context, "Found! I like xxx");
        }
        else if(matcher.matches(context.input,allCheese_pattern[1]))
        {
            if(debug) sendMessageToUser(context, "Found! xxx");
        }
        else
            return response;

    MatchResult result = matcher.getMatch();
    String cheese = result.group(1);

        if(cheese == null)
            return null;
        cheese = cheese.trim();

        if(cheese.length() == 0)
            return null;

        if(cheesePlatter == null)
        {
            cheesePlatter = new Hashtable();
            for(i = 0; i < cheeses.length; i++)
                cheesePlatter.put(cheeses[i],"");
        }

        if(cheesePlatter.containsKey(cheese))
        {
            if(debug) sendMessageToUser(context, "Found recognised: " + cheese);

        int wchoice = Math.abs(random.nextInt())% knownCheeseResponses.length ;
            response = new Response(this,knownCheeseResponses[wchoice]);
        }
        else
        {
        int wchoice = Math.abs(random.nextInt())% unknownCheeseResponses.length ;
            response = new Response(this,unknownCheeseResponses[wchoice]);
        }

        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        response.putMatch("subject",cheese);

        return response;
    }
////////////////////////////////////////////////////////////////////////////////////////////
//  checkIfLikeCheddar state Responses
////////////////////////////////////////////////////////////////////////////////////////////
private static final String iDoNotLikeCheddarResponses[] =
{
    "Ok, I guess you are not a cheese person. bye for now",
    "Don't like this cheese, dont like cheddar! A Cheese-o-phobe!",
    "I guess you are not a cheese lover. See you.",
    "<getvar name=\"first_name\"/>, cheese is not to everyone's taste. bye.",
};
```

cheeseTopic.java (part 5/6)

```
    private Response checkIfLikeCheddar(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
        if(matcher.matches(context.input,YESNO_pattern[0])) // Now check patterns for YES or NO
        {
            if(debug) sendMessageToUser(context, "Found! YES"); // Re-use the above data and responses.....
        int wchoice = Math.abs(random.nextInt())% whatOtherTypesResponses.length ;
            response = new Response(this,whatOtherTypesResponses[wchoice]);
            response.addToNext_Query_State(topic_name + ":checkWhatTypeOfCheese");
        }
        else if(matcher.matches(context.input,YESNO_pattern[1]))
        {
            if(debug) sendMessageToUser(context, "Found! NO");
        int wchoice = Math.abs(random.nextInt())% iDoNotLikeCheddarResponses.length ;
            response = new Response(this,iDoNotLikeCheddarResponses[wchoice]);
            if (wchoice == 2)    // mentioned Cheese-o-phobe
                response.addToNext_Query_State(topic_name + ":checkCheeseOPhobe");
        }
        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        return response;
    }
////////////////////////////////////////////////////////////////////////////////////////////
//  checkCheeseOPhobe state Responses
protected Pattern cheeseophobe_pattern[] = null;
protected transient final String cheeseophobePatterns[] =
{
        HELP_ME_WITH_STRING + "(cheese(-|\s*)?o(-|\s*)?phobe)" +PUNCTUATION_STRING,
};
private static final String cheeseOPhobeResponses[] =
{
    "A cheese-o-phobe is someone who does not like cheese!",
    "A Cheese o phobe is someone who hates cheese.",
    "A rat likes cheese, a cheeseophobe hates cheese!",
};
    private Response checkCheeseOPhobe(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
        response = null;
        if(matcher.matches(context.input,cheeseophobe_pattern[0]))
        {
            if(debug) sendMessageToUser(context, "Found request for help with cheeseophobe");
        int wchoice = Math.abs(random.nextInt())% cheeseOPhobeResponses.length ;
            response = new Response(this,cheeseOPhobeResponses[wchoice]);
            response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1
        }
        return response;
    }
////////////////////////////////////////////////////////////////////////////////
    public void setTopicMultiModal(DialogueManagerInterface dmi)
    {
        dmi.registerMultiModal(DialogueManagerAdapter.PLAINTEXT,this);
    }
    public Response getTopicsActiveQueryResponse(DialogueManagerInterface dm)
    {
        return (Response)null;
    }
    public void saveTopicState(DialogueManagerServer dms,String basedir)
    {       // do nothing in this case.
    }
    public void topicsActiveQueryInitialise(DialogueManagerInterface dm)
    {       // do nothing in this case.
    }
    public boolean initialiseTopic(DialogueManagerServer dms)
    {       // do nothing in this case.
        return true;
    }
    public void tidyUp(Context context,Response response)
    {        // do nothing in this case.
    }
    public boolean checkValidState(String topic_string, Vector Prev_Query_State, Context context)
    {
        return true;
    }
////////////////////////////////////////////////////////////////////////////////
    public static void main(String[] args)
        throws Exception
    {
        try
        {
        cheeseTopic topic = new cheeseTopic();
                topic.testTopic();
        }
        catch (Exception e)
        {
            System.err.println("Your topic could not be instantiated.");
            System.err.println("This is probably because of a bad Regular Expression.");
            throw e;
        }
    }
}
```

cheeseTopic.java (part 6/6)

```
package local.mentor.topics;

import java.lang.*;
import java.util.*;
import java.io.*;

import org.apache.oro.text.regex.*;

import local.raytrace.Syslog.*;
import local.raytrace.IO.*;
import local.raytrace.Util.*;
import local.raytrace.DialogueManager.*;
import local.raytrace.Topics.*;

public class didYouKnowTopic extends userTopic
    implements Serializable, TopicsInterface
{
private static Vector dyks = null;
private static boolean succesfulLoad = false;
private static String TOPICNAME = "didYouKnow";
private static int requestsToLoad=0;

private static final boolean debug = false;

protected transient String patterns[] =
{
//  First entry is the general area to match
    "A",        // Upper case - never matches....
};

protected transient String keywordHelpPatternString = "(did\s*" + YOU_STRING + "know)";

    public didYouKnowTopic()
        throws FileNotFoundException
    {
        super (TOPICNAME);

        briefTopicsHelpString = "Did You Know - an Active Only Topic that tells you strange things.";
        topicsHelpString =      "This topic periodically tells you something that you \n" +
                    "may not know.\n" + "The facts come from XXXX.\n" ;

        topicsActiveQueryWeight = 1;

        makePatterns(keywordHelpPatternString,patterns);
        tellMeMore_pattern = makeSpecificPatterns(tellMeMorePatterns);

    }

    public Response checkTopic(Context context)
    {
        // We don't have to be active only - we could also match against people asking the Dialogue Manager
        // to tell us something we don't know.. This is left as an exercise for the gentle reader....
        response = null;      // Indicate that default is no match....

        return (response);
    }

    public Response checkFollowupResponse(Context context, String topic_string, Vector Prev_Query_State)
    {
        // We will cater for someone saying "WoW, etc.
    String state_string = null;

    int   index = topic_string.indexOf(':');
        if (index != -1)
            state_string = topic_string.substring(index+1);

        if(debug)
           context.dm.sendMessageToUser("checkFollowupResponse("+topic_name +") state_string is "+state_string);

        if(state_string.equals("wowResponse"))
             return handleWowResponse(context);
        else if(state_string.equals("tellMeMoreResponse"))
             return handleTellMeMoreResponse(context);
        else
             return null;
    }

    public Response getTopicsActiveQueryResponse()
    {
        return null;
    }
    public Response getTopicsActiveQueryResponse(DialogueManagerInterface dm)
    {
    Response response = null;// Don't need this but we might.  e.g our active topic could be as simple as
                    // return new Response(this,"I bet you didn't expect this!");

        loadDidYouKnows(dm);
        return DidUKnow(dm,dyks);
    }
```

didYouKnowTopic.java (part 1/3)

```
    public static Response DidUKnow()
    {
        return null;
    }

    public static Response DidUKnow(DialogueManagerInterface dm, Vector[] dyks, int j)
    {
        if(dyks == null)
            return null;

    int dyk = Math.abs(random.nextInt())% (j );
        return  DidUKnow(dm,dyks[dyk]);
    }

    public static Response DidUKnow(DialogueManagerInterface dm, Vector dyks)
    {
    Response response = null;
        if(dyks == null)
        {
            dm.debug("Could not open did you know file: dyks Vector is null");
            return response;
        }

    int dyk = Math.abs(random.nextInt())% (dyks.size() );
    int dchoice = Math.abs(random.nextInt())% (didYouKnowResponses.length );
        response = new Response(null, didYouKnowResponses[dchoice] + "\n" +
                        (String)dyks.elementAt(dyk));

        response.addToNext_Query_State(TOPICNAME + ":wowResponse");
        response.addToNext_Query_State(TOPICNAME + ":tellMeMoreResponse");
        response.responseData.response_weight = 0.75F;  // moderate weighting. 0<= weight <=1

        return response;
    }


////////////////////////////////////////////////////////////////////////////////////////////
//  Active state Responses - "Did you Know"
////////////////////////////////////////////////////////////////////////////////////////////
private static final String didYouKnowResponses[] =
{
    "Did you know, did you know....",
    "Did you know that:",
    "<getvar name=\"first_name\"/>, did you know that:",
    "Do you know this:",
    "Are you interested in this, <getvar name=\"first_name\"/>:",
};

////////////////////////////////////////////////////////////////////////////////////////////
//  Followup state Responses - "Wow",etc
////////////////////////////////////////////////////////////////////////////////////////////
private static final String wowResponses[] =
{
    "Some people like abstract knowledge <getvar name=\"first_name\"/>!",
    "I find it interesting.",
    "<getvar name=\"first_name\"/>, you can't beat general knowledge or useless facts",
    "You never know when this might come in handy, <getvar name=\"first_name\"/> ",
    "Yeah <getvar name=\"first_name\"/>, but who would have thought....",
    "Trivia intrigues me.....",
    "I like funny facts like that.",
};

    public Response handleWowResponse(Context context)
    {
    Response response = null;
    int wchoice = Math.abs(random.nextInt())% (wowResponses.length );

        // Lets match anything but at a low priority. Lets make it just higher than the unknown topic response.
        // We don't know what they said but lets give a very general reply....
        response = new Response(this,wowResponses[wchoice] );
        response.responseData.response_weight =1.1F*unknownTopic.UNKNOWN_RESPONSE_WEIGHT;
        // Lets also let them say tell me more after wow!!!
        response.addToNext_Query_State(TOPICNAME + ":tellMeMoreResponse");

        return response;
    }

////////////////////////////////////////////////////////////////////////////////////////////
//  Followup state Responses - "tell me more",etc
// We need to ensure that this and any other patterns we use is compiled in the constructor.
//      tellMeMore_pattern = makeSpecificPatterns(tellMeMorePatterns);
// The state processing method is classified in checkFollowupResponse and is called
// based upon the next state string - in this case "tellMeMoreResponse"
protected Pattern tellMeMore_pattern[] = null;
protected transient final String tellMeMorePatterns[] =
    {
        "(" +POLITE_STRING +")?\s*" + MORE_STRING + "\s*(" + POLITE_STRING +")?",
    };
```

didYouKnowTopic.java (part 2/3)

```
    public Response handleTellMeMoreResponse(Context context)
    {
    PatternMatcher matcher  = new Perl5Matcher();
    Response response = null;
        if(matcher.matches(context.input,tellMeMore_pattern[0]))
        {
            if(debug)
                context.dm.sendMessageToUser("Found! Tell me more");

            // Get another trivial fact.
            response = getTopicsActiveQueryResponse(context.dm);
        }

        return response;
    }

////////////////////////////////////////////////////////////////////////////////////////////
//  Auxiliary methods.
////////////////////////////////////////////////////////////////////////////////////////////
    private static void waitForLoadToFinish()
    {
        while (requestsToLoad >0)
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException eie)
            {
                Syslog.debug("DidYouKnow:waitForLoadToFinish forcibly awoken:" + eie);
            }
    }

    private static void loadDidYouKnows(DialogueManagerInterface dm)
    {
        if(requestsToLoad >0)
            waitForLoadToFinish();

        if( succesfulLoad)
            return;

        requestsToLoad++;

    String packageName = DialogueManagerSecurityManager.getPackageName();
    String filename = "didyouknow.gz";
        try
        {
            succesfulLoad = true;
            dyks = FileIO.readZippedVectorSystemResource(packageName,filename);

        }
        catch (Exception e)
        {
            dyks = null;
            succesfulLoad = false;
            Syslog.debug("Did U Know: " + packageName + "/" + filename + ":" + e);
        }
        requestsToLoad--;
    }

////////////////////////////////////////////////////////////////////////////////

    public void setTopicMultiModal(DialogueManagerInterface dmi)
    {
        dmi.registerMultiModal(DialogueManagerAdapter.PLAINTEXT,this);
    }
    public void saveTopicState(DialogueManagerServer dms,String basedir)
    {
        // do nothing in this case.
    }
    public void topicsActiveQueryInitialise(DialogueManagerInterface dm)
    {
        // do nothing in this case.
    }
    public boolean initialiseTopic(DialogueManagerServer dms)
    {
        // do nothing in this case.
        return true;
    }
    public void tidyUp(Context context,Response response)
    {
        // do nothing in this case.
    }
    public boolean checkValidState(String topic_string, Vector Prev_Query_State,
                                   Context context)
    {
        return true;
    }

}
```

didYouKnowTopic.java (part 3/3)

You are where you are today because you stand on somebody's shoulders.
And wherever you are heading, you cannot get there by yourself.
If you stand on the shoulders of others, you have a reciprocal
responsibility to live your life so that others may stand on
your shoulders. It's the quid pro quo of life.
We exist temporarily through what we take,
but we live forever through what we give.

Vernon Jordan

I: Example Extended Topics Showing Multimodal API for *Mentor* System

This appendix shows a template for the format of an extended userModel as well as 3 examples of Topics/classes that can be used to extend the Dialogue Management System. These are:

- `userXXX.java`
- `TVTarget.java`
- `MultiModalStimulus.java`
- `ObjectReceiver.java`

These examples are used by students wishing to extend the Dialogue Manager system to use multimodal input and output. An example of such as system can be seen in Figure I.26 which shows a complex multimodal Dialogue and Interaction Management system created by a user that exploits the extensibilities of the system to cater for input from remote controls as well as directing output commands to devices such as televisions and radios.

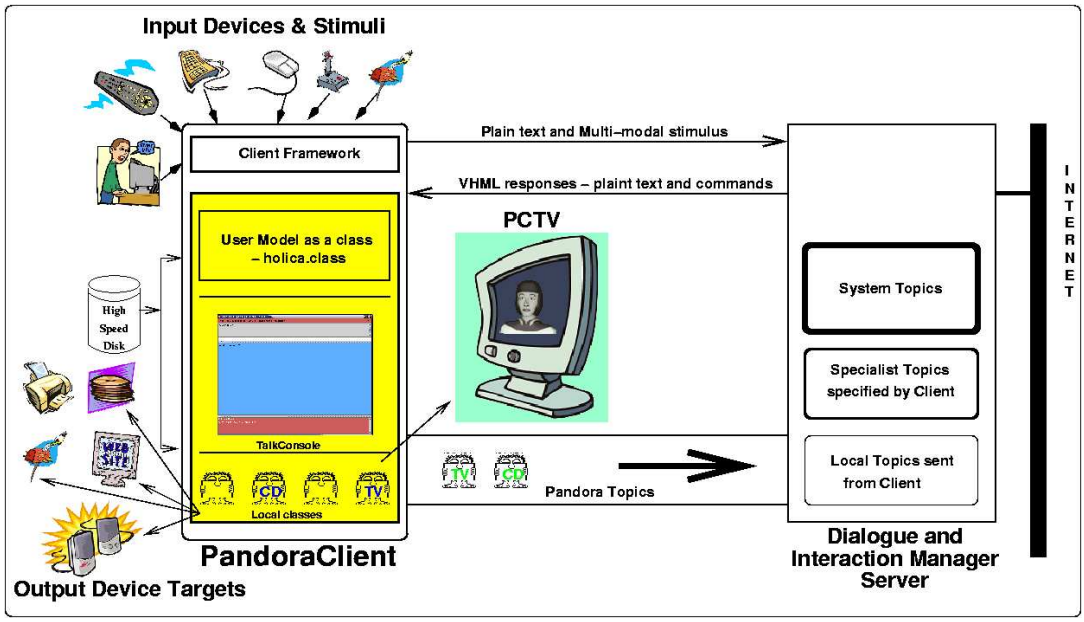


Figure I.26 : The Architecture extended to provide a multimodal DM system.

The listings only show the code segments that are relevant to illustrate the necessary functionality. The listings have been edited and formatted to conform to the structure of the A4 pages.

These Topics can be found on the CDROM in folder /extendedTopics.

```
package local.mentor.userModels;
       :

public class userXXX extends StudentModel
    implements MentorMessageListener
{
/**
 * This field holds the message Identifier that enables communication between the Mentor
 * system and any Listening class. In this case, this class will register itself for listening
 * for messages == commands, data and responses coming back from the Mentor system.
 * This class implements MentorMessageListener so its processCommand will be called when a
 * packet from the Mentor system is received by the MentorClient.
 */
int messageID = -1;

/**
 * This is simply a String that indicates the class == the username of the user
 * executing this userModel. The MentorClient also checks to see if
 * the user has created their own user model by searching for the class
 * file local.mentor.userModels.xxx where xxx is their username.
 */
private static final String  userClass = "userXXX";

/**
 * Holds an instantiation of a GUI class for multimodal input.
 */
private DialogueManagerStimulusInterface multimodal = null;

/**
 * Holds an instantiation of a GUI class for receiving multimodal data.
 */
private DialogueManagerStimulusInterface objectReceiver = null;

   /**
    * Instantiated by the MentorClient if found as local.mentor.userModels.xxx in user's
    * CLASSPATH where xxx is user's username.
    */
   public userXXX()
   {
      System.err.println( "Don't forget to run this as\n" +
         "MentorClient -xslt -vhml\n" +
                     );
   }

   /**
    * Since the userModel follows the Applet strategy, it has an init method that will be
    * called to initialise the userModel.
    * In this case, it simply calls the init("html") method
    * so as to indicate that the system should use an HTML viewing terminal for interaction
    * (if required).
    */
   public void init()
   {
      init("html");
   }

   /**
    * Since the userModel follows the Applet strategy, it has an init method that will
    * be called to initialise the userModel.
    * @param className indicates that the system should use an HTML viewing terminal for
    * interaction and get all its user preferences using this className.
    */
   public void init(String className)
   {
      // Register this userModel as being interested in packets sent from the Mentor System.
      // Any that it does not want to deal with directly, it can pass on to the
      // TalkConsoleInterface (htmlTalkConsole) created below.
      // Typically this class will want to process packets of type incomingPacketString
      // and incomingPacketObject which will hold information that may need to be decoded for
      // support classes such as TVTarget and ObjectReceiver instantiated below.
      messageID= MentorMessageRegister.addMentorMessageListener(this);
                  :
```

userXXX.java (part 1/6)

```
        // Create a normal HTML based TalkConsole for entering and displaying the
        // request-responses to and from the Mentor system.
        // Preferences comes from a file called "htmlMentorConsole"
        tci = new htmlTalkConsole(
                        className + "MentorConsole",
                        "Your window of opportunity to the Mentor System...",
                        client.boss,2,client,1,0, "Mentor",-1, 0,false, false,
                        messageID);
        if (tci == null)
        {
            System.err.println("Whoa baby! Can't create the necessary TalkConsole.\n"+
                "Let's forget about dancing and call a Guru!\nbye");
            System.exit(-1);     // From the client.
        }

    String errmessage0 = "thrown when trying to instantiate\n"+"DialogueManagerStimulusInterface:" ;
    String errmessage1 = "\n" +
            "This probably means that it was not found in your classpath or \n" +
            "the found file with that name is not of class DialogueManagerStimulusInterface\n" ;

    String dir = null;

        // Only do the following if the Mentor System has been contacted - the client is the
        // major framework for the system and handles most of the protocol overhead.
        if(client.clientOK)
            try
            {
                // Now send the MentorMessageRegister MessageID so that returning packets
                // get directed to the correct processor.
                // In this case, we will be the processor but will pass most packets
                // off to the TalkConsole - see the processCommand method below.
                client.sendCommand(client.clientOut,HereIsMyMessageID,
                        new Object[]
                            {
                                new Integer(-1), new Integer(tci.getMessageID())
                            }
                                );

                // Now lets load up the needed Topics - this means that the
                // user class knows what topics must be in the Mentor System.
                // The client side version of TVTarget is instantiated dynamically
                // when a VHML <p target="TV">xxx</p> string is sent from the system.
                dir = "TV";
                tci.setStatusln(userClass + ": sent "+ client.sendUserTopics(dir));

                // Lets load up the local Topic that handles a type of Stimulus.
                // This gets sent across to the Mentor Server.
                // As well, lets instantiate the class that creates the type of
                // stimulus. Since these are the same, they MUST be synchronised.
                // Update the array stimulusClassdirs in this code so that the
                // class can be found by the instantiateStimulusClass method
                dir = "MultiModalStimulus";
                //Send the LocalTopics.jar in ./dir as a local Topic
                tci.setStatusln(userClass + ": sent "+ client.sendUserTopics(dir));

                // Now instantiate a class of this type located somewhere in the current (client)
                // environment. This is specified in the stimulusClassdirs String array which is
                // used to add to the CLASSPATH.
                // NOTE: We now have the same class running on the client and the server,
                // synchronised and yet performing complimentary tasks.
                multimodal = instantiateStimulusClass(dir);

                // Set this classes environment on this side - the client side.
                // In this case, we give it a reference to the client framework
                // as well as the MessageID that the TalkConsole has established. This means
                // we can pass on any messages to the TalkConsole that we intercept and
                // then need to pass on.
                multimodal.setEnvironment( new Object[]
                                        {
                                            client,
                                        }
                                    );
                new Thread((Runnable)multimodal).start();
```

userXXX.java (part 2/6)

```
            // Lets load up the local Topic that expects to send an Object
            // from the Mentor Server to the client.
            // As well, lets instantiate the class that receives that type of
            // Object. Since these are the same, they MUST be synchronised.
            // Update the array stimulusClassdirs in this code so that the
            // class can be found by the instantiateStimulusClass method
            dir = "ObjectReceiver";
            //Send the LocalTopics.jar in ./dir as a local Topic
            tci.setStatusln(userClass + ": sent "+ client.sendUserTopics(dir));

            // Now instantiate a class of this type located somewhere in the current (client)
            // environment. This is specified in the stimulusClassdirs String array which is
            // used to add to the CLASSPATH.
            // NOTE: We now have the same class running on the client and the server,
            // synchronised and yet performing complimentary tasks.
            objectReceiver = instantiateStimulusClass(dir);

            // Set this classes environment on this side - the client side.
            // In this case, we give it a reference to the client framework.
            objectReceiver.setEnvironment( new Object[]
                                        {
                                            client,
                                        }
                                        );
            new Thread((Runnable)objectReceiver).start();
        }
        catch (IOException e)
        {
            client.DisplayErrorMessage(e.getMessage());
        }
        catch (Exception e0)
        {
            client.DisplayErrorMessage("processTarget: Exception " +
                        errmessage0 +  dir + errmessage1 + e0 + "\n" +
                           local.raytrace.Util.Util.stackTraceToString(e0));
        }
        catch (Error er0)
        {
            client.DisplayErrorMessage("processTarget: Error " +
                        errmessage0 + dir + errmessage1 + er0 );
        }
    }

    /**
     *  This method can be used to start up the user model.
     *  In this case it does nothing special since the TalkConsole is running
     *  in its own graphics Thread and all the instantiated classes are Threads.
     *  If any data from the Mentor System comes in, this class will be notified by its
     *  processCommand method below.
     *
     *  In this example, the TalkConsole COULD be started Iconified or
     *  COULD be left invisible - not all usermodels may want to even use a
     *  TalkConsole, or may want one but may want to keep it low profile.
     *  These options are left here but commented out.
     */
    public void start()
    {
        super.start();
//      tci.setState(Terminal.ICONIFIED);
//      tci.setVisible(false);
    }

//  Here is the list of directories that will contain the stimulus classes
//  For example, MultiModalStimulus.class is under MultiModal/, remoteStimulus.class
//  could be there as well or could be in REMOTE, etc.... You choose....
    private static final String[] stimulusClassdirs =
    {
        "MultiModalStimulus",
        "ObjectReceiver",
    };

    public DialogueManagerStimulusInterface instantiateStimulusClass(String stimulusClassName)
        throws Exception
    {
    DialogueManagerStimulusInterface dmsi = null;
            dmsi    = (DialogueManagerStimulusInterface)
                        Client.instantiateObjectClass(stimulusClassName,stimulusClassdirs );
        return dmsi;
    }
```

userXXX.java (part 3/6)

```
    /**
     * Required for the MentorMessageListener Interface
     *
     * @see #processCommand(int,InputStream)
     * @see #getDataObjects
     */
    private Object[] objs = null;

    /**
     * Required by the MentorMessageListener Interface, simply returns any required Objects
     * within the interface to the calling routine.
     *
     * @return the MentorMessageListener specific Objects.
     *
     * @see #processCommand(int,InputStream)
     *
     */
    public Object[] getDataObjects()
    {
        return objs;
    }

    /**
     * This method is the interface for the MentorMessageListener system.
     * When the Mentor system sends across a packet that is directed via a Message ID,
     * the packet is routed to the appropriate Listener.
     * @param command - the packet protocol command type such as incomingPacketString
     * or incomingPacketObject
     * @param is - the InputStream that needs to be read to get the rest of the packet.
     * Only the command protocol type has been read at this point. The Listener must determine
     * what to do with the specified packet protocol type.
     */
    public void processCommand(int command, InputStream is)
    {
    String commandType  = MentorProtocol.commandRequestType(command);
        if(debug)
                System.err.print(commandType + " found in "+userClass + "\n");

        // In this case, we will be the processor but will pass most packets
        // off to the TalkConsole - only interested in incomingPacketString
        // which holds the actual text or targeted text
        // We want this because the text string - a response from the Mentor system -
        // may contain a "<p>" tag with an attribuet that specifies a target other than the
        // default display device (the TalkConsole in this case).
        // If we get an incomingPacketString command, we must decode the packet and
        // see if it means that we must send information to a new Target device.
        //
        if(command == incomingPacketString)
            // Yes, we must process it to see what it contains....
            sendDataToTargets(command,is);
        else
            // No, Just send it to the TalkConsole
            tci.processCommand(command,is);

        if(debug)
                System.err.println("processCommand:finished");
    }


//  Here is the list of directories that will contain the target classes
//  For example, TVTarget.class is under TV/, radioTarget.class could be there as well
//  or could be in RADIO, etc.... You choose.... Note that the name of the class
//  must be <name>Target.class
//
    private static final String[] targetClassdirs =
    {
        "TV"
    };
```

userXXX.java (part 4/6)

```
/**
 * This and the Client methods simply try to break the packet down into its constituent
 * parts.  * It is VHML and so is in a known format. We need to get any
 * <p target="xxx">yyy</p> lines and send them to the appropriate "xxx" target device.
 *
 * Of importance to this exercise is the use below of dmti.processTarget which
 * is where the data trail will end up if a target line is found. At that stage, it has
 * instantiated a Target class appropriate to device type (it uses a strict naming format).
 * It then has to invoke the processTarget(String) method of the instantiated class,
 * passing it the body of the "p" tag ("yyy" above).
 *
 */
public void sendDataToTargets(int command, InputStream is)
{
String commandType  = MentorProtocol.commandRequestType(command);

   ////////////////////////////////////////////////////
   // Let's get the incoming packet from the Mentor System.
String packetString = Client.getIncomingPacketString(command,is);
       if(packetString == null)
       {
              client.DisplayErrorMessage("sendDataToTargets: (" + commandType + "): " +
                  "null String in packet!");
              return;
       }

   ////////////////////////////////////////////////////
   //  Now lets see if we need to process it. Maybe it is just plain
   //  text. If so, squirt it to the TalKConsole.
   //  If not, then we need to see if we need to target some other device.
       packetString = Client.checkForVHML(tci,packetString);
       if(packetString == null)
              return;            //Sent to the tci already.

String[] values = null;
       // Remember that the response from the system may be zero or more lots of
       // high level packets such as <p>. We may have a <p> for the
       // TalkConsole Interface as well as multiple <p>'s for different Targets.
       // For example, we may turn off the radio and turn on the TV in one packet from
       // the Mentor system. These are different Targets.
       while ((values = Client.checkForTarget(packetString)) != null)
       {
           // We now have values as being non-null, therefore we have a packet to process.
           if (values[0] == null)
           {
               client.DisplayErrorMessage("sendDataToTargets: (" + commandType + "): " +
                   "null String in value returned from checkForTarget!");
               return;
           }

           if(values[0].equals("default"))
           {
                 tci.processPacketString(values[1]);
           }
           else
           {
           String[] newValues = new String[values.length-1];
               newValues[0] = values[1];      // null - no error
               newValues[1] = values[2];      // dataString
               newValues[2] = values[3];      // tag
               newValues[3] = values[4];      // body
               for(int i = 4 ; i <values.length-1; i++)
                   newValues[i] = values[i+1];    // attributes

               if(values[0].equals("mark"))
                   processMark(newValues);
               else if(values[0].equals("person"))
                   processPerson(newValues);

               // continued if then else on next page
```

userXXX.java (part 5/6)

```
                // OK - we now have a real <p> maybe with a target
                // if no attributes then send to TalkConsole else
                // get the target class and call its processTarget method.
                else if(values[0].equals("p"))
                {
                    if( newValues.length <=4)
                        tci.processPacketString(newValues[3]);
                    else
                    {
                    DialogueManagerTargetInterface dmti= null;
                        dmti= Client.getDialogueManagerTarget(newValues,targetClassdirs);
                        if(dmti != null)       // null catered for with error message in Client
                            try
                            {
                                dmti.processTarget(newValues[3]);
                            }
                            catch (Exception e)
                            {
                              client.DisplayErrorMessage("processTarget:"+"Exception thrown in "+
                                   "DialogueManagerTargetInterface:processTarget method\n" +
                                      "Target =" + dmti + ", Exception is\n" +
                                      Util.stackTraceToString(e)
                                                );
                              return;
                            }
                        }
                    }
                    else
                    {
                        client.DisplayErrorMessage("sendDataToTargets: (" + commandType + "): " +
                            "Unknown target=" + values[0] + ": returned from checkForTarget!");
                        return;
                    }
                }
                packetString = values[2];
            }
    }

    // What should we do here?  Unknown at the moment.
    public void processMark(String values[])
    {
    String tag = null;
    String attrib = null;
    String dataString = null;
    String body = null;
        dataString=  values[1];
        tag       =  values[2];
        body      =  values[3];
        attrib    =  values[4];

        client.DisplayErrorMessage("processMark: " +
                            "tag=:" + tag +": do not know what to do with it!\n");
        return;
    }

    // What should we do here? Unknown at the moment.
    public void processPerson(String values[])
    {
    String tag = null;
    String attrib = null;
    String dataString = null;
    String body = null;
        dataString=  values[1];
        tag       =  values[2];
        body      =  values[3];
        attrib    =  values[4];

        client.DisplayErrorMessage("processPerson: " +
                            "tag=:" + tag +": do not know what to do with it!\n");
        return;
    }
          :
}
```

userXXX.java (part 6/6)

```
package local.mentor.topics;
      :
      :
/**
 * This class is an example of a dual-use client-server Topic. That is, it is sent
 * across to the Mentor server as well as being instantiated on the client.
 *
 * Note that this class has two distinct sets of modules -
 * 1) those that execute on the server, and
 * 2) those that execute on the client.
 * This is done so that the mechanisms used by this class are consistent for
 * the client-server interaction.

       ////////////////////////                  ////////////////////////
       ////////////////////////     ------>      ////////////////////////
       //                    //                 //                    //
       //  SERVER SIDE METHODS  //     network     // CLIENT SIDE METHODS   //
       //                    //                 //                    //
       ////////////////////////     <------      ////////////////////////
       ////////////////////////                  ////////////////////////

 * The client side can produce multimodal stimulus and send it across to
 * the server either as a string based stimulus or as an object based stimulus.
 * The server processes this stimulus regardless of how it was sent across and
 * produces an appropriate response.
 *
 * This documentation assumes that the reader is familiar with the standard
 * format for a user topic.
 */
public class MultiModalStimulus extends userTopic
   implements Serializable,
      DialogueManagerStimulusInterface,
      Runnable,
      ChangeListener,
      TopicsInterface,
      MentorProtocolInterface,
      ActionListener
{
   /**
    * This constant holds the multimodal Stimulus type.
    * This value is used both by the client side to package up the
    * stimulus to send to the server and by the server side to recognise
    * the stimulus. Since in this example, the code on the client side and the
    * code on the server side is the same class, the stimulus value is consistent.
    */
   private static final String EMOTION_STIMULUS_TYPE = "emotion";

   /**
    * This constant array holds the names of the emotions that are:
    * 1) displayed on the sliders
    * 2) sent across by the client to the server as part of the stimulus
    * 3) recognised by the Regular Expression Pattern matching on the server side.
    */
   private final static String[]  emotionNames =
   {
         "happy", "sad",
         "angry", "surprised",
         "afraid", "disgusted"
   };

   /**
    * This flag indicates whether debug information should be logged and
    * sent across to the client as this Topic is executed.
    */
   private static final boolean debug = false;
```

MultiModalStimulus.java (part 1/4)

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//
// SERVER SIDE METHODS
//
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

protected transient String patterns[] =
{
// First entry is the general area to match
// The Dialogue Manager looks to see if this area matches this particular
// Topic. If not then do not look at any further matching. This is for
// efficiency. All matches must be in lower case as the input line is
// converted to lower case.
//

      ".*",

   emotionNames[0] + "=(.*);" +
   emotionNames[1] + "=(.*);" +
   emotionNames[2] + "=(.*);" +
   emotionNames[3] + "=(.*);" +
   emotionNames[4] + "=(.*);" +
   emotionNames[5] + "=(.*)" , //Note that last punctuation is removed by Mentor

   "emotions",

// Finally in this example, the huge HELP_ME_WITH_STRING
   HELP_ME_WITH_STRING + "(emotion)",

   ".*",       // match everything italian in a non-specific way
};

   ::: Constructor

//This is the entry point from the DialogueManager for each Topic. The DM
// has already checked the general area of this topic via entry[0] of
// the above patterns.
// It matched, therefore lets now check exactly what was typed to see if
// we can recognise it....
public Response checkTopic(Context context)
{
   ::: Code to search for the pattern...

   if(found)
   {
      topicTrace(context,"checkTopic :Switch:" + i);
      switch(i)
      {
      case 1:   // process emotions sent as a String Stimulus
                return processEmotionStringStimulus(matcher,context);
      //break;

      case 2:   // process emotions sent as an Object Stimulus
                return processEmotionObjectStimulus(context);

      ::: Code for default cases, etc.
   return response;
}
```

MultiModalStimulus.java (part 2/4)

```
/**Process the emotions sent across from the client if they came across as a
 * single line of text. Notice that they will have been pattern matched
 * differently but that the input is basically the same.
 */
private Response processEmotionStringStimulus(PatternMatcher matcher, Context context)
{
// We will have 6 values that come across as xxx=value;yyy=value;.....
MatchResult result = matcher.getMatch();
String values[] = new String[6];
   for(int i= 0; i<6 ; i++)   // Lets get each of the values and store it.
   {
   String typed = result.group(i+1);   // Look at the first (xxx) remembered match.
         if(typed == null)
         {
            context.dm.debug("Got an emotion match but with a null MatchResult!");
            return null;      // Strange since we did match but just in case..
         }
         if(typed.length() == 0)
         {
            context.dm.debug("Got an emotion match but with a
                     zero length MatchResult!");
            return null;      // just in case..
         }
         values[i] = typed.trim();      // get rid of extra white space
   }
   // Now lets just join these values back together again to return to the
   // user to show that we have recognised them.
String resultString =   values[0] + "," + values[1] + "," + values[2] + ","
                  + values[3] + "," + values[4] + "," + values[5] + "\n";

Response response = new Response(this, resultString);
   return response;
}

/**
 * Process the emotions sent across from the client if they came across as a
 * DialogueManagerStimulusInput Object. This contains the stimulus type, the
 * pattern to match against as well as the stimulus data values.
 * Notice that they will have been pattern matched differently but that the
 * input is basically the same.
 */
private Response processEmotionObjectStimulus(Context context)
{
String values[] = new String[6];
   context.dm.debug("Got an emotion Object match"); // log it.
   if(context.stimulus_type == EMOTION_STIMULUS_TYPE)
      context.dm.debug("Got an emotion stimulus"); // log it.
   if(context.stimulus_object != null)
      context.dm.debug("Got an emotion object=" + context.stimulus_object);
DialogueManagerStimulusInput dmsi =
      (DialogueManagerStimulusInput)context.stimulus_object;

Response response = new Response(this, (String)dmsi.incomingObject);
   return response;
}

/** Register as being interested in receiving stimulus=EMOTION_STIMULUS_TYPE
 */
public void setTopicMultiModal(DialogueManagerInterface dmi)
{
   dmi.registerMultiModal(EMOTION_STIMULUS_TYPE,this);
}
public boolean initialiseTopic(DialogueManagerServer dms)
{
   setTopicMultiModal(dms);
   return true;
}
```

MultiModalStimulus.java (part 3/4)

```
   ////////////////////////////////////////////////////////////////
   // CLIENT SIDE METHOD ONLY
   // This cannot use the normal DM routines - it has no Context
   // since it executes on the client side
   ////////////////////////////////////////////////////////////////
   protected MentorClient client = null;
      :::: Fields for creating the Slider Demo GUI

   /**This method will be called as part of the DialogueManagerStimulusInterface
    * to set the environment for the instantiated class. Anything can be passed
    * from the usermodel but they must match.... See MentorClient field above.
    */
   public void setEnvironment(Object[] environment)
   {
      client = (MentorClient)environment[0];
   }

   public void run()
   {
      createSliderDemo();
      setVisible(true);
   }

   public void createSliderDemo()
   {
      :::  Create the Slider Demo GUI along with a Frame and ActionListener
   }

   /** This method will send the stimulus to Mentor when the Send Button is pressed
    */
   public void actionPerformed(ActionEvent e)
   {
   String stimulusValues = "";
      if(client.clientOut !=null)
      {
/*       // We could either send across the Stimulus like this as one line.
         // The Mentor system will then accept this and decode it as above.
         // Alternatively and better, we can send the stimulus across as a
         // DialogueManagerStimulusInput Object.
         stimulusValues = "<stimulus type=\"" + EMOTION_STIMULUS_TYPE + "\">";
         for(int i=0; i<emotionNames.length; i++)
         {
            stimulusValues += emotionNames[i]+"="+emotionSliders[i].getValue()+";" ;
         }
         stimulusValues += "</stimulus>\n";
/**/
         // We can send the stimulus across as an Object field within a
         // DialogueManagerStimulusInput Object.
         for(int i=0; i<emotionNames.length; i++)
         {
            stimulusValues += emotionNames[i] + "=" +
               emotionSliders[i].getValue() + ";\n" ;
         }
      DialogueManagerStimulusInput dmsi = new DialogueManagerStimulusInput(
                     EMOTION_STIMULUS_TYPE,  // The stimulus type
                     "emotions",             // The pattern to match against
                     stimulusValues);        // The stimulus value to be used.

         // Send it to the server
         try
         {
//         client.sendCommand(client.clientOut,incomingPacketString,stimulusValues);
           client.sendCommand(client.clientOut,incomingPacketObject,dmsi);
               ::::: end of code
   }
```

MultiModalStimulus.java (part 4/4)

```
package local.mentor.topics;
      :
/**This class is an example of a dual-use client-server Target Topic. That is,
 * it is sent across to the Mentor server as well as being dynamically
 * instantiated on the client when a <p target="TV">xxx</p> packet is sent.
 *
 * Note that this class has two distinct sets of modules -
 * 1) those that execute on the server, and
 * 2) those that execute on the client.
 * This is done so that the mechanisms used by this class are consistent for
 * the client-server interaction.
       ////////////////////////////                    ////////////////////////////
       ////////////////////////////      ------>       ////////////////////////////
       //                          //                  //                          //
       //  SERVER SIDE METHODS     //      network     // CLIENT SIDE METHODS      //
       //                          //                  //                          //
       ////////////////////////////     <------         ////////////////////////////
       ////////////////////////////                    ////////////////////////////
 * The client side can receive commands from the server to adjust a TV card
 * (if fitted).
 * <br>The server sends these commands in response to user requests to turn
 * on/off the TV.
 *
 * This documentation assumes that the reader is familiar with the standard format
 * for a user topic.
 */
public class TVTarget extends userTopic
    implements Serializable,
       TopicsInterface,
       DialogueManagerTargetInterface
{
   /**
    * This flag indicates whether debug information should be logged and sent
    * across to the client as this Topic is executed.
    */
   private static final boolean debug = false;

   // The following constants are Unix commands to control a TV card.
   private static final String DEFAULT_TV_ON_COMMAND = "tvtime&";
   private static final String DEFAULT_TV_OFF_COMMAND = "tvtime-command QUIT&";
   private static String TV_ON_COMMAND =  DEFAULT_TV_ON_COMMAND;
   private static String TV_OFF_COMMAND = DEFAULT_TV_OFF_COMMAND;

   private static String TV_COMMAND_WRAPPER = "CMD";
   private static String TV_COMMAND_WRAPPER_DELIMITER = ":";

   ////////////////////////////////////////////////////////////////////////
   // SERVER SIDE METHODS
   ////////////////////////////////////////////////////////////////////////

   /**
    * This field holds the Reg Expr for the word "TV".
    */
   private static final String TVSTRING = "(?:tv|tele(?:vision)?|telly|it)";

   protected transient String patterns[] =
   {
         ::: the Regular Expressions
      ".*\b" + TVSTRING + "\b.*",

      "(?:(?:turn\s*" + "(?:the\s*)?" + TVSTRING+ "\s*(on|off)))|" +
      "(?:turn\s*" + "(?:(on|off)\s*the\s*" + TVSTRING+ "))",
      HELP_ME_WITH_STRING + "(\b" + TVSTRING + "\b)",
      ".*",       // match everything TV in a non-specific way
   };
```

TVTarget.java (part 1/4)

```
      ::: Constructor

   // Here we have our responses to the input from the user.
   // We may want to have several so that we can vary the response.
   // These are randomly chosen from.
   private static final String ON_Responses[] =
   {
      "I have turned the TV on, <first_name/>",
      "The TV is now on, <first_name/>"
   };

   private static final String OFF_Responses[] =
   {
      "I have turned the TV off, <first_name/>",
      "The TV is now off, <first_name/>"
   };

   /**
    * This method will wrap the given input String in a command format understood
    * by the client.
    *
    * @param input the input String to wrap
    *
    * @return the wrapped input String
    */
   private String wrapCMD(String input)
   {
      return TV_COMMAND_WRAPPER + TV_COMMAND_WRAPPER_DELIMITER +
         input + TV_COMMAND_WRAPPER_DELIMITER;
   }

   /**
    * This method will wrap the given input String in a VHML format that
    * targets the TV on the client.
    *
    * @param input the input String to wrap
    *
    * @return the targetted wrapped input String
    */
   private String wrapTV(String input)
   {
      return "<p target=\"TV\">" + input + "</p>";
   }

   /**
    * This method will wrap the given input String in a VHML format that
    * targets the default output renderer (TalkConsole) on the client.
    *
    * @param input the input String to wrap
    *
    * @return the plain text targetted wrapped input String
    *
    */
   private String wrapPlainText(String input)
   {
      return "<p>" + input + "\n</p>";
   }
```

TVTarget.java (part 2/4)

```
public Response checkTopic(Context context)
{
      ::: Code for the matching.....
   if(found)
   {
      switch(i)
      {
      case 1:   // turn on or off the TV
      case 2:   // turn on or off the TV
         MatchResult result = matcher.getMatch();
         String typed = result.group(1); // First (xxx) remembered match.
            if(typed == null)
            {
               context.dm.debug("Got on/off but with a null MatchResult!");
               return null;      // Strange since we did match but just in case..
            }

            if(typed.length() == 0)
            {
               context.dm.debug("Got on/off but with a 0 length MatchResult!");
               return null;
            }

            typed = typed.trim();      // get rid of extra white space

            // User asked to have the TV turned on. Therefore wrap up a command
            // to turn on the TV with the target being "TV"
            // Note we send two separate <p>'s: one to the user saying OK
            // and one to the TV with the target CMD to turn it on.
            if(typed.equals("on"))   // ON
            {
            int echoice = Math.abs(random.nextInt())% ON_Responses.length ;
               response = new Response(this,
                     wrapPlainText(
                        Response.uniqueResponseString(context.dm,ON_Responses))+
                     wrapTV(wrapCMD(TV_ON_COMMAND)));
            }
            else             // OFF
            // User asked to have the TV turned off. Therefore wrap up a command
            // to turn off the TV with the target being "TV"
            // Note we send two separate <p>'s: one to the user saying OK
            // and one to the TV with the target CMD to turn it off.
            {
            int echoice = Math.abs(random.nextInt())% OFF_Responses.length ;
               response = new Response(this,
                     wrapPlainText(
                        Response.uniqueResponseString(context.dm,OFF_Responses))+
                     wrapTV(wrapCMD(TV_OFF_COMMAND)));
            }

      break;

            ::::::

   return response;
}
```

TVTarget.java (part 3/4)

```
   ////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////
   //
   // CLIENT SIDE METHOD ONLY
   //    This cannot use the normal DM routines - it executes on the client side
   ////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////

   /**
    * This method is executed whenever a packet is received from the Mentor
    * that has a <p target="TV">xxx</p> string in it. The
    * xxx is the argument passed in here as the body of the tag.
    *
    * This method is called because the usermodel has
    * registered an interest in Listening for packets
    * from the Mentor System and has decode the packet to
    * find that it contains <p target="TV">xxx</p>. The
    * usermodel then tries to instantiate a class called
    * TVTarget.class and if successful, caches it and then
    * calls its processTarget method.
    * The processTarget method must exist since this class
    * implements DialogueManagerTargetInterface.

    */
   public void processTarget(String body)
   {
      System.err.println("TVTarget client:processTarget: Got the body string:\n"
               + body + ":");

      ////////////////////////////////////////////////////////
      // First lets isolate the xxx from the various wrappers.
      // At this stage we chould just have the command wrapper.
      //
   int index = body.indexOf(TV_COMMAND_WRAPPER_DELIMITER);
      if(index == -1)
      {
         System.err.println("Cannot find the : delimiter");
         return;
      }

   int lindex = body.lastIndexOf(TV_COMMAND_WRAPPER_DELIMITER);
   String command = body.substring(0,index);
   String arguments = body.substring(index+1, lindex);
      if(debug)
         System.err.println("TVTarget client:processTarget: Got the command="
                  + command + ":" +
                  " with arguments=+" + arguments + "+");

      // OK - if we have a legitimate command then execute it in this case.
      // In general, we could do anything with it.
      if(command.equals(TV_COMMAND_WRAPPER))
            local.raytrace.Util.Util.system(arguments);
   }
}
```

TVTarget.java (part 4/4)

```
package local.mentor.topics;
      :
/**
 * This class is an example of a dual-use client-server Receiver Topic. That is, it
 * is sent across to the Mentor server as well as being instantiated on the client.
 *
 * Note that this class has two distinct sets of modules -
 * 1) those that execute on the server, and
 * 2) those that execute on the client.
 * This is done so that the mechanisms used by this class are consistent for
 * the client-server interaction.
       ////////////////////////////                  ////////////////////////////
       ////////////////////////////     ------>      ////////////////////////////
       //                          //                //                          //
       //  SERVER SIDE METHODS     //     network      // CLIENT SIDE METHODS     //
       //                          //                //                          //
       ////////////////////////////     <------      ////////////////////////////
       ////////////////////////////                  ////////////////////////////
 * The client side can register its interest in listening to Mentor packets and
 * obtain a MessageId which it can send across to the server either as plain text
 * or better, as a special stimulus type, to the same class on the server that can
 * recognise the pattern to record the MessageId. When it later wants to send
 * complex data to this class, it can use the MessageId and the data will be routed
 * to the class.
 *
 * This documentation assumes that the reader is familiar with the standard format
 * for a user topic.
 *
 */
public class ObjectReceiver extends userTopic
   implements Serializable,
      DialogueManagerStimulusInterface,
      Runnable,
      MentorProtocolInterface,
      TopicsInterface,
      MentorMessageListener
{
   /**
    * This flag indicates whether debug information should be logged and sent
    * across to the client as this Topic is executed.
    */
   private static final boolean debug = true;

   /**
    * This constant holds the ObjectReceiver Stimulus type.
    * This value is used both by the client side to package up the stimulus
    * to send to the server and by the server side to recognise the stimulus.
    * Since in this example, the code on the client side and the
    * code on the server side is the same class, the stimulus value is consistent.
    */
   private static final String OBJECT_RECEIVE_STIMULUS_TYPE = "send_to_me";

   /**
    * This constant holds the ObjectReceiver pattern that is used to send the
    * MessageId from the client to the server and to be pattern matched there.
    * Since in this example, the code on the client side and the
    * code on the server side is the same class, the pattern match value is
    * consistent.
    */
   private static final String OBJECT_RECEIVE_INPUT = OBJECT_RECEIVE_STIMULUS_TYPE + ":" +
                                       "here is my id:" ;
```

ObjectReceiver.java (part 1/5)

```
   ////////////////////////////////////////////////////////
   //
   // SERVER SIDE METHODS
   //
   ////////////////////////////////////////////////////////

   // Holds the MentorMessageListener Message Id
   private Integer ObjectReceiverProcess_Message_id = null;

   protected transient String patterns[] =
   {
      ".*(picture|image|" + OBJECT_RECEIVE_INPUT + ").*",

      OBJECT_RECEIVE_INPUT + "(.*):(.*)", // used to get the clients message ID
      "send (picture|image)",       // used for testing that it works.
      ".*",       // match everything about picture
   };

      ::: Constructor, etc

   public Response checkTopic(Context context)
   {
         :: Code for Pattern Matching....
      if(found)
      {
         switch(i)
         {
         case 1:// The client is sending the client classes message id.
               return processMessageId(matcher,context);

         case 2:// The user is asking for a picture to be sent to the client
               return processSendPicture(matcher,context);
            :::
      return response;
   }

   private Response processMessageId(PatternMatcher matcher,Context context)
   {
   MatchResult result = matcher.getMatch();
   String message_id = result.group(1);  // Look at the first (xxx) remembered match.
            ::
      message_id= message_id.trim();      // get rid of extra white space

   String className = result.group(2);   // Look at the second (xxx) remembered match.
            ::
      className= className.trim();     // get rid of extra white space

   int value= 0;
      try
      {
         value = Integer.decode(message_id).intValue();
      }
      catch (NumberFormatException e2)
      {
            ObjectReceiverProcess_Message_id = null;
            return new Response(this, "You sent across a funny integer Message Id:"
                     + message_id);
      }
      ObjectReceiverProcess_Message_id = new Integer(value);

   Response response = new Response(this,"Message id for " + className
                  + " is " + message_id );
// Response response = new Response(this,""); //If we dont want to send any message.
      return response;
   }
```

ObjectReceiver.java (part 2/5)

```
/** Some known random images in the www.mentor.computing.edu.au website....
 */
private static String[] images =
{
   "system.gif",  "logo.gif", "head1_reva.gif",
   "rev_trans_ahair5.gif",    "saila.gif"
};

private Response processSendPicture(PatternMatcher matcher,Context context)
{
DialogueManagerServer dms = context.dm.getDialogueManagerServer();

   if(ObjectReceiverProcess_Message_id == null)
   {
   String tmp = "processSendPicture :ObjectReceiverProcess_Message_id is null";
      dms.DMSErrorLog(tmp);
      return new Response(this,tmp);
   }

int echoice1 = Math.abs(random.nextInt())% images.length ;
byte[] image1 = null;
byte[] image2 = null;
String baseUrl = "http://" + context.dms.getDialogueManagerWWW() + "/images/";

   try
   {
   URL url1 = new URL(baseUrl +images[echoice1] );
   URLConnection connection1 = url1.openConnection();
      dms.authenticateConnection(connection1);
   InputStream is1 = connection1.getInputStream();
      image1= FileIO.getbyteContentsOfStream(is1,connection1.getContentLength());
      is1.close();

   int echoice2 ;
      while( (echoice2 = Math.abs(random.nextInt())% images.length) == echoice1)
                    ;// Get an image within the array.
   URL url2 = new URL(baseUrl + images[echoice2] );
   URLConnection connection2 = url2.openConnection();
      dms.authenticateConnection(connection2);
   InputStream is2 = connection2.getInputStream();
      image2= FileIO.getbyteContentsOfStream(is2,connection2.getContentLength());
      is2.close();
   }
   catch (MalformedURLException e)
   {
      dms.DMSErrorLog("MalformedURLException is :" + e.getMessage() + ":");
      return new Response(this,"MalformedURLException is :" + e.getMessage() + ":");
   }
   catch (IOException e2)
   {
      dms.DMSErrorLog("IOException is :" + e2.getMessage() + ":");
      return new Response(this,"IOException is :" + e2.getMessage()+":");
   }

// Now create an array of byte arrays and send as an Object to Client using the
// Message ID of the client image renderer.
byte[][] obj = new byte[2][];
   obj[0]=image1;
   obj[1]=image2;
   context.dm.sendObjectToUser(ObjectReceiverProcess_Message_id,obj);

// Also send back a response to the user. This goes to the normal TalkConsole.
Response response = new Response(this, "Here are two pictures.");
   return response;
}
```

ObjectReceiver.java (part 3/5)

```
// Register this class as being interested in plain text as well as a simulus
public void setTopicMultiModal(DialogueManagerInterface dmi)
{
   dmi.registerMultiModal(OBJECT_RECEIVE_STIMULUS_TYPE,this);
   dmi.registerMultiModal(DialogueManagerAdapter.PLAINTEXT,this);
}
         :

//////////////////////////////////////////////////////////
//
// CLIENT SIDE METHOD ONLY
//    This cannot use the normal DM routines - it executes on the client side
//////////////////////////////////////////////////////////

/**
 * We need to get a messageID so that we can tell the server side Topic where
 * to send the Object. We then send that over to the server and it will be
 * matched as the appropriate stimulus (see setTopicMultiModal in this class)
 * and with the appropriate pattern ( i.e we send over OBJECT_RECEIVE_INPUT....
 * and we match against that as the first pattern in the REG EXPR patterns
 * on the server).
 *
 * Since we have sent the MessageId, we can now get this Topic to send back
 * an Object that we can process cia the standard MessageListener interface
 * method processCommand and from there on to receiveObjectFromMentor. That
 * method then gets the Object and passes it on to the GUI displayer....
 *
 */
public void run()
{
   messageID= MentorMessageRegister.addMentorMessageListener(this);

String stimulusValues = "";
   if(client.clientOut !=null)
   {
      stimulusValues = "<stimulus type=\""+OBJECT_RECEIVE_STIMULUS_TYPE+"\">";
      stimulusValues += OBJECT_RECEIVE_INPUT + messageID + ":" + topic_name;
      stimulusValues += "</stimulus>\n";
      System.err.println("Message to send is " + stimulusValues);

      // Send it to the server
      try
      {
         client.sendCommand(client.clientOut, incomingPacketString,
                  stimulusValues);
      }
      catch (IOException err)
      {
         client.DisplayErrorMessage("ObjectReceiver:actionPerformed : " +
                  err.getMessage());
         return ;
      }
   }
}

private Object[] objs = null;

public Object[] getDataObjects()
{
   return objs;
}
```

ObjectReceiver.java (part 4/5)

```
   protected MentorClient client = null;
   protected int messageID = -1;

   public void setEnvironment(Object[] environment)
   {
      client = (MentorClient)environment[0];
   }

   /**This is called when a message for this Message Listener arrives from the
    * server. In this case, we are interested in the incomingPacketObject
    * which will have been sent from the server in response to the user asking
    * for a picture to be sent.
    */
   public void processCommand(int command, InputStream is)
   {
   String commandType  = MentorProtocol.commandRequestType(command);

      // In this case, we will be the processor but will pass most packets
      // off to the TalkConsole - only interested in incomingPacketString
      // which holds the actual text or targeted text
      if(command == incomingPacketObject)
            receiveObjectFromMentor(command,is);
      else
         client.DisplayErrorMessage("ObjectReceiver:processCommand:" +
                     "Got an unprocessable command from the Mentor system=" +
                        commandType);

      if(debug)
            System.err.println("ObjectReceiver:processCommand:finished");
   }

   private void receiveObjectFromMentor(int command, InputStream is)
   {
   String commandType  = MentorProtocol.commandRequestType(command);

      try
      {
      Object obj = new ObjectInputStream(is).readObject();

         // OK. Now we have the object which should be two byte arrays of images.
         // Lets give them to the example GUI for displaying.
         buildSplitPane(obj);
      }
      catch (Exception e)
      {
         client.DisplayErrorMessage("ObjectReceiver:receiveObjectFromMentor (" +
            commandType + "): " +  e +
            "\n" + local.raytrace.Util.Util.stackTraceToString(e));
         return;
      }

   }
         :::: GUI Variables.

   private void buildSplitPane(Object images)
   {
   byte[][] bytes = new byte[2][];
      bytes = (byte[][])images;
   byte[] moon_bytes = (byte[])bytes[0];
   byte[] earth_bytes = (byte[])bytes[1];

   ImageIcon ii= null;
      ii= new ImageIcon(earth_bytes);
         :::: Code for displaying the sent images......
}
```

ObjectReceiver.java (part 5/5)

And each small candle
Lights a corner of the dark
Each Small Candle.
Roger Waters

J: User Manual for Topic Network Builder

The User Manual documentation for the Topic Network Builder application can be found on the thesis CDROM in directory:

documentation/topic_builder/

An online version can also be found at:
http://www.mentor.computing.edu.au/phd/documentation/topic_builder [HREF 68]

J.1 Introduction

The Topic Network Builder application creates networks of nodes that model the request-response dialogue between a user and the ***Mentor*** System. This can be done in a user-friendly point and click manner via a Graphical User Interface (Figure J.27). The resulting network can be saved in various formats, one of which allows the network to be converted into a Java Topic file for use within the ***Mentor*** System. The application can also import request-response pairs in various formats, and allows for cutting and pasting of individual nodes and collections of nodes called supernodes.

The following sections briefly describe the features of the application.

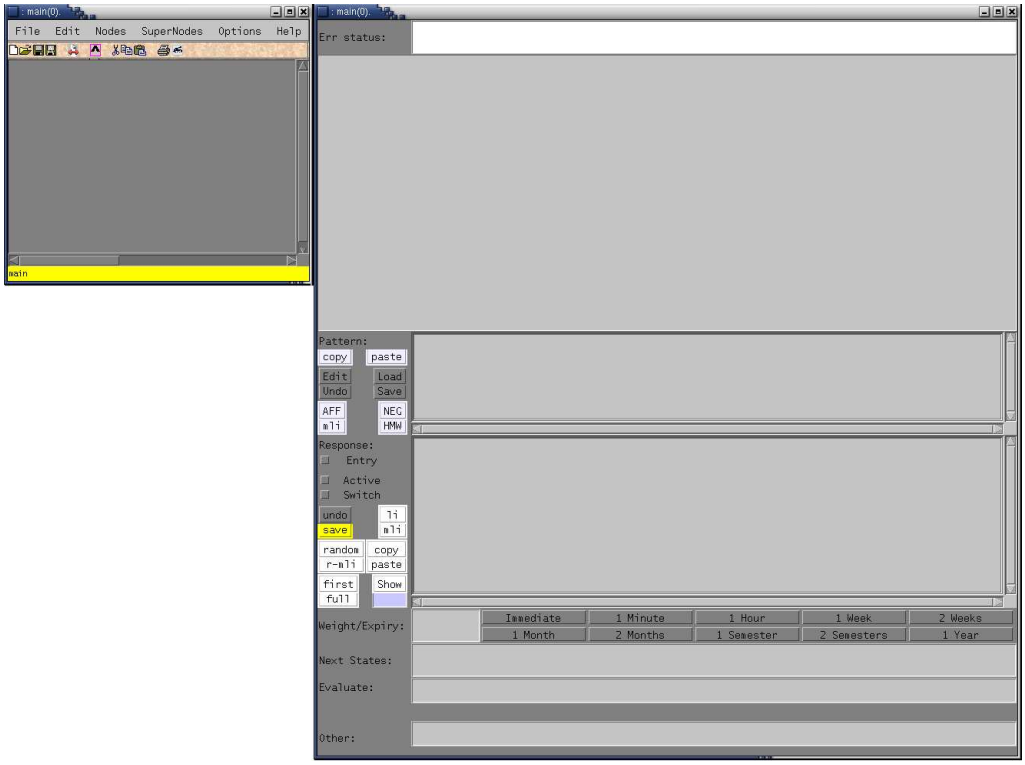


Figure J.27 : Topic Network Builder application

The Topic Network Builder application is composed of two GUI windows - a compose window (Figure J.28) and an information window (Figure J.29). The compose window allows the user to create nodes, supernodes, and edges between nodes in a point and click manner in the construction area underneath the toolbar. The compose window uses a menubar as well as a toolbar for quick easy access to commonly used functions such as saving, exporting, cutting and pasting, traversing the network, printing, etc. The construction area underneath the toolbar can either be plain or have a user defined background, and can be gridded if required.

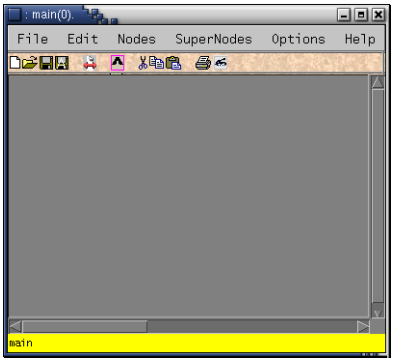


Figure J.28 : Topic Network Builder application: compose window

The information window shows the current list of nodes as well as details of the currently selected node in the compose window. The detailed information includes the RE pattern, the VHML response, the type of node, and any Next State nodes. The information window also allows for evaluation information to be stored in the node as well as expiry information for Directed Learning Path construction. Changes made to nodes in either window reflect in the other.

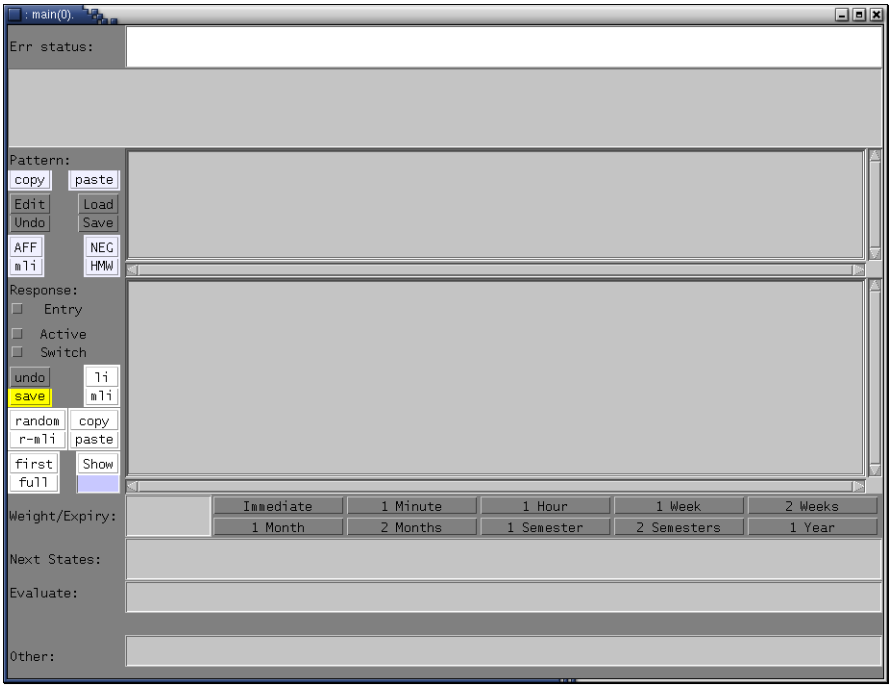


Figure J.29 : Topic Network Builder: information window

Figure J.30 shows the menubar entries. These control File manipulation facilities, node and supernode creation, deletion and searching, as well as the setting of various options.

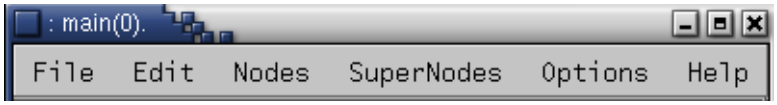


Figure J.30 : Topic Network Builder: Menubar Menu entries

The "File" Menu of Figure J.31 allows various input files to be opened. These represent the different "Save" formats for the application and represent the network that has been constructed. The normal "Save" format is as a Node file which contains the Java data class for the network. The "Topic Entry" is used by support FAQ processing software, and the "Text File" is an experimental XML format.

Similar formats are supported for importing partial networks (Figure J.32). A special "DMTL" format is also supported and represents a complex network format from another research application. The "Q/A" format is a plain text line-by-line question-answer format that can be easily constructed by hand or by the Authoritative Guidance applications.

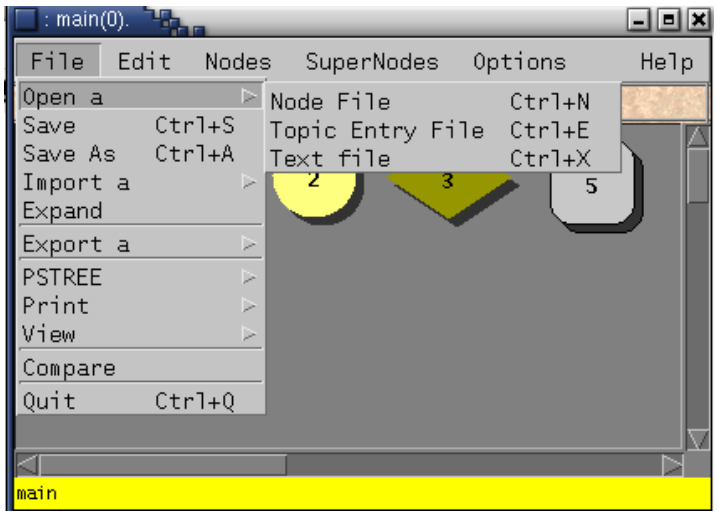


Figure J.31 : Topic Network Builder: File Menu ’Open’ expanded

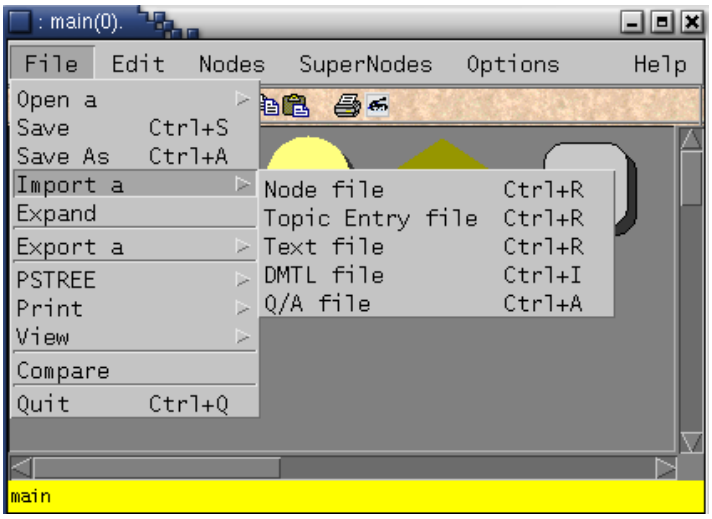


Figure J.32 : Topic Network Builder: File Menu ’Import’ expanded

Similar formats are supported for exporting networks (Figure J.33). A special "GRP" format is also supported and represents the complex network format as a Java data file. It is this file that is post-processed to become the Topic file. The node and edge information is stored in arrays in the constructed Topic (see Figure 4.55 on page 180).

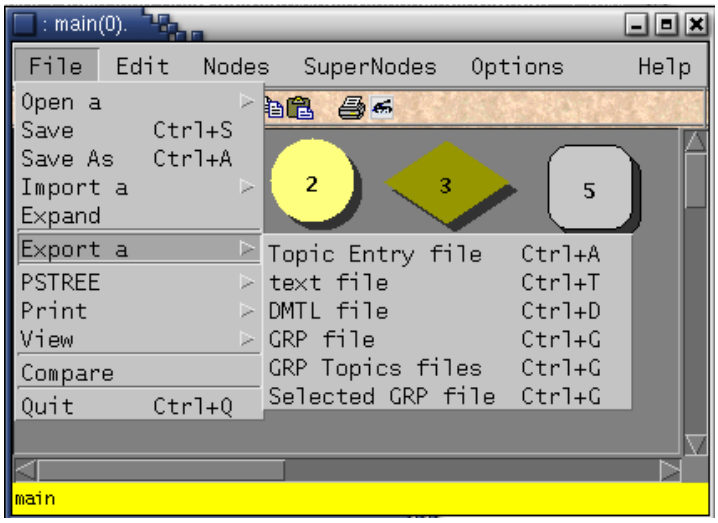


Figure J.33 : Topic Network Builder: File Menu 'Export' expanded

Figure J.34 shows three network visualisation entries. A PostScript tree of the network can be constructed (see Figure J.35) as well as an HTML "View" Table (see Figure J.36). The network can also be printed ("Print"). Note that either the entire network or just parts of it can be visualised.

The HTML "View" of Figure J.36 contains "Next State" hyperlinks to other node entries in the HTML table so that the network can be visualised and analysed more easily.

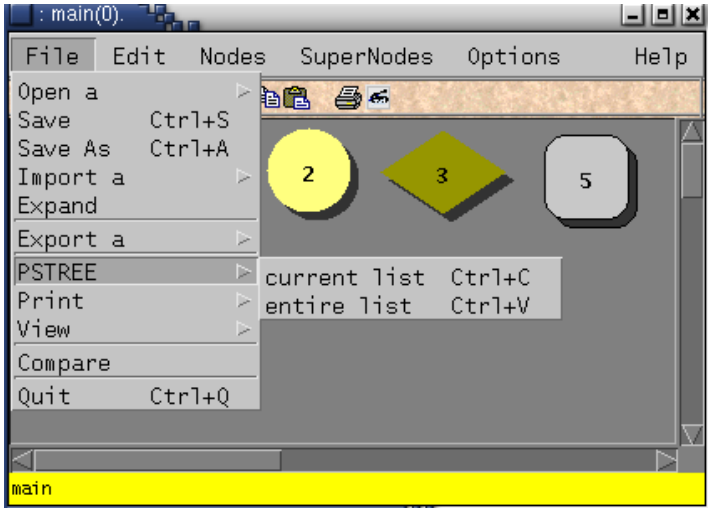


Figure J.34 : Topic Network Builder: File Menu network visualisation entries

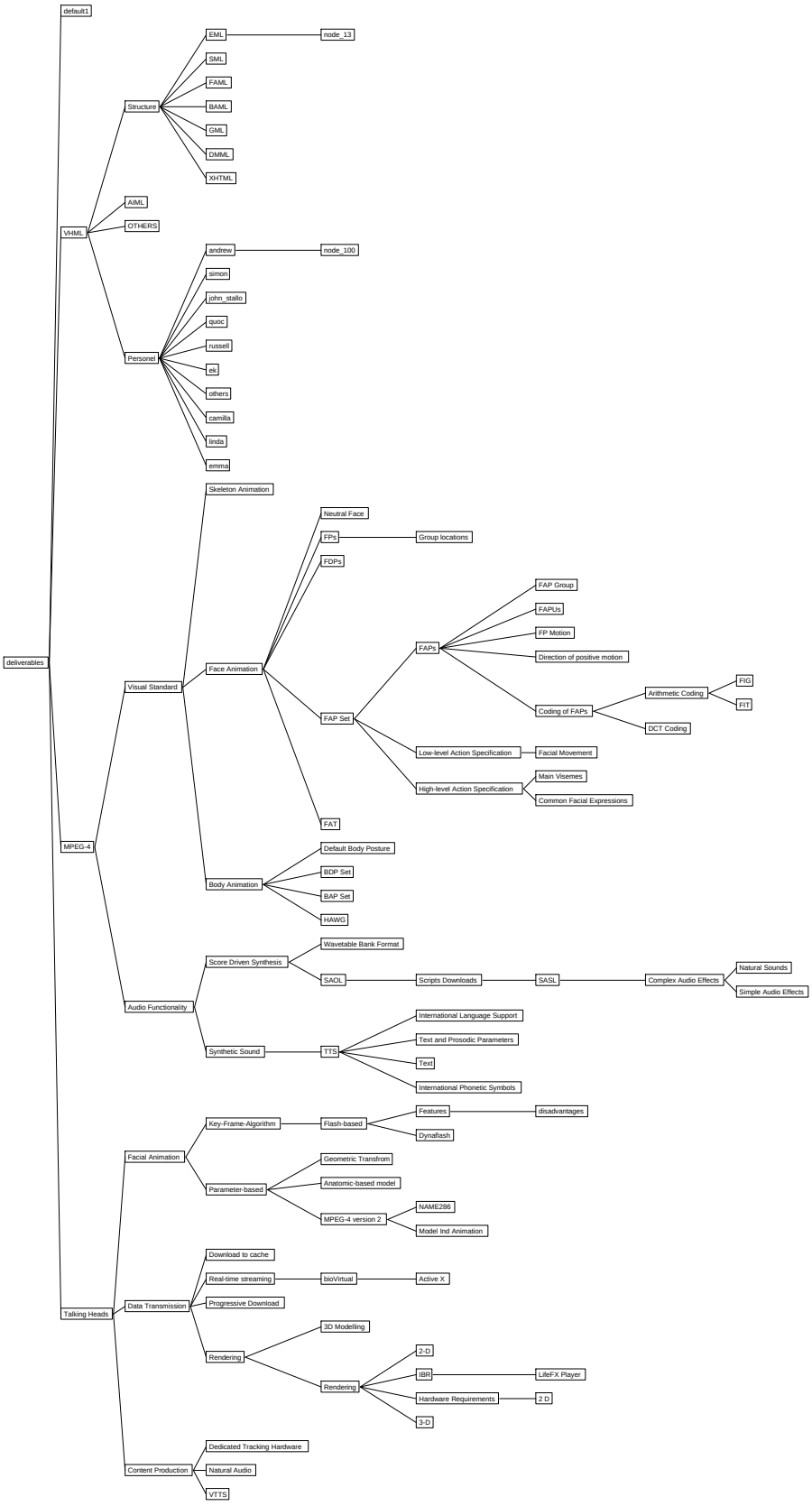


Figure J.35 : Topic Network Builder: example PSTREE output

All nodes in example

	Pattern	Response	Next States	Depth
0	.*	This topic caters for all enquiries about the unit AGV-351 and CG-552. It helps with the assignments as well as general information.	null	0->^
1		<random> <double_first_name/>how is <topic_name/> going<double_first_name/>? <double_first_name/>are you happy about <topic_name/><double_first_name/>? - Is <topic_name/> ok, <first_name/>? - Is <topic_name/> going ok, <first_name/>? - How is <topic_name/> going, <first_name/>? - Are you coping with <topic_name/> <first_name/>? - How goes <topic_name/>? - Are you ok with <topic_name/>? </random>	4,7, 51	1->^
4	(ITSOK|YES)	<random> <double_first_name/>you don't seem to be really confident<double_first_name/> - You seem a bit unsure, <double_first_name/>you don't seem sure<double_first_name/> - hmmm, <double_first_name/>sounds a bit iffy<double_first_name/> - You sure? </random> <double_first_name/>do you need any help<double_first_name/>?	6,10, 16	4->^
From SuperNode: Assessment				
10	(NO|later|(not\s*yet)).?THANKS?.?ITSOK?THANKS?(.*)	Ok. Maybe later.	null	10->^
5	bad|(no\s*good)|lurk|yucky|(no\s*go)|nogo|no	<random> <first_name/>, you need to start soon. - It would be a good idea to start soon. - I suggest that you start soon	null	6->^

Figure J.36 : Topic Network Builder: part of example HTML Table output

Due to the size of the networks, both the HTML Table, and the PostScript output can become unwieldy. The encapsulation provided by supernodes and the ability to print or view selected sub-networks (Figure J.37 and Figure J.38) can alleviate this visualisation problem.

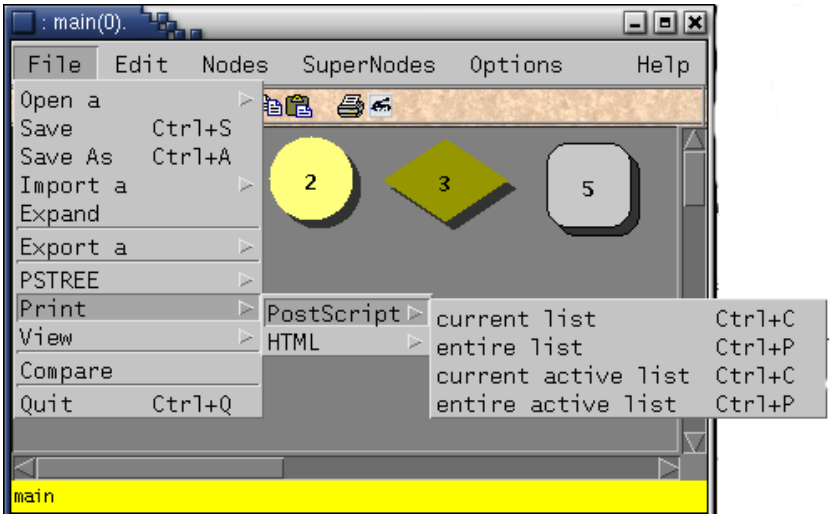


Figure J.37 : Topic Network Builder: print visualisation options

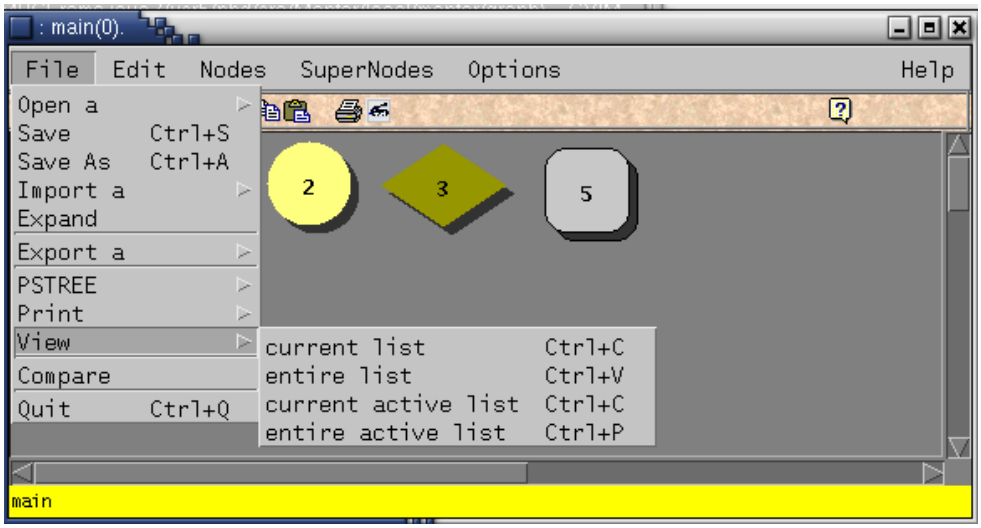


Figure J.38 : Topic Network Builder: view visualisation options

Figure J.39 shows the application's rudimentary search facility, and Figure J.40 shows the popup Dialog GUI that allows the user to enter the plain text search criteria. If found, the node containing the search term is selected in both the compose and the information windows. By repeating the search, subsequent matching nodes can be cycled through.

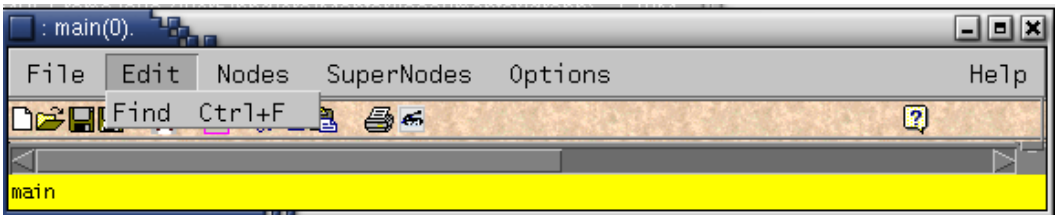


Figure J.39 : Topic Network Builder: Find Menu entry

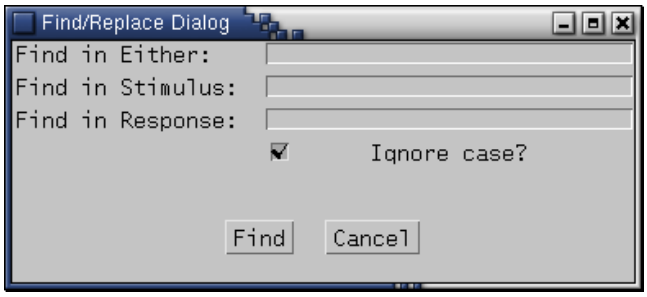


Figure J.40 : Topic Network Builder: Find Dialog

Figure J.41 shows the Node menu entries. These aid in moving around the network ("up a node"), accurate positioning of the selected node ("Node Placement"), and an entry for future work on processing pro-active nodes.

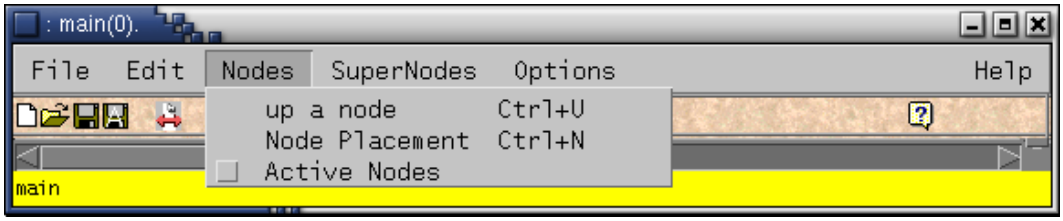


Figure J.41 : Topic Network Builder: Node Menu Entry

Figure J.44 shows the SuperNodes Menu entries. This is a hierarchical list of all the supernodes in the network and, if selected, the chosen supernode will become the current supernode to be displayed in the compose and information windows.

Figure J.42 shows the Options menu entries. These allow the user to set various attributes of the application. These attributes are stored in the saved file. The "HTML Viewing Attributes" brings up the Dialog GUI of Figure J.43, which allows for the specifying of which columns should be present in the HTML Table.

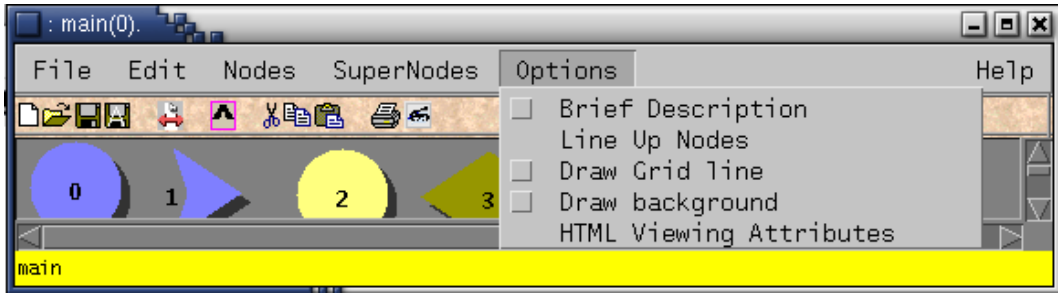


Figure J.42 : Topic Network Builder: Options

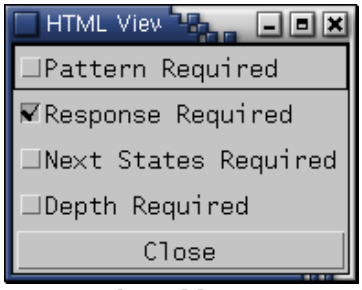


Figure J.43 : Topic Network Builder: HTML Table View Options

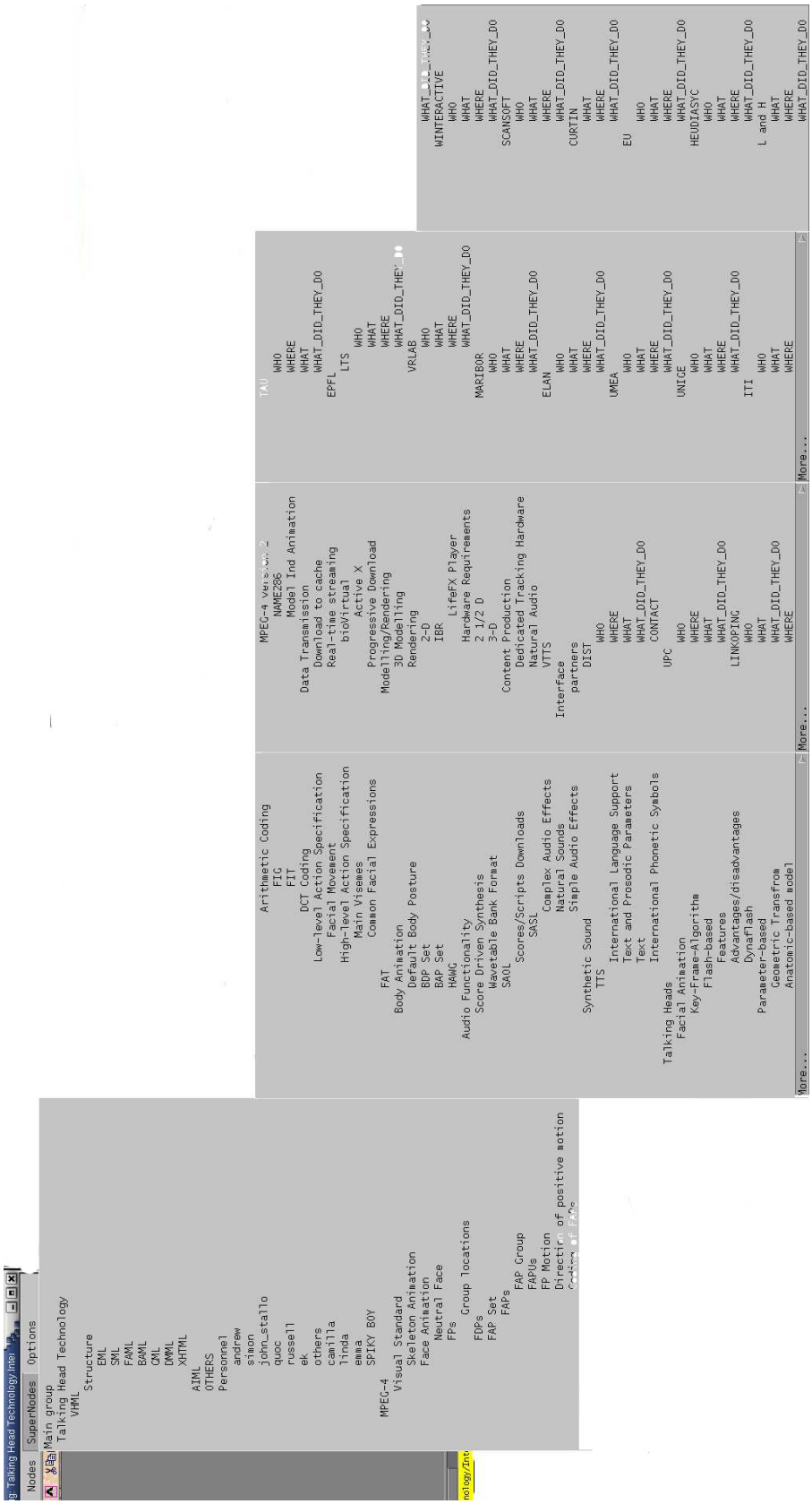


Figure J.44 : Topic Network Builder: SuperNode Menu Entry

Figure J.45 shows the Toolbar entries that provide shortcuts to the functionality available in the menubar. The first group allows for a "New", "Open", "Save" and "Save as" facility for networks. The second group contains a single item that exports the network as a Java data file. The third group moves the point of display up a node level in the network hierarchy. The fourth group allows for cutting and pasting, whilst the last group allows for printing and viewing of the currently selected network nodes.

Figure J.45 : Topic Network Builder: Toolbar entries

Figure J.46 shows a single node in the application. It was created by simply clicking in the compose area with the left mouse button. Nodes are numbered sequentially, and node 0 represents the keyword or key pattern described in Section 4.2.2.1 on page 134.

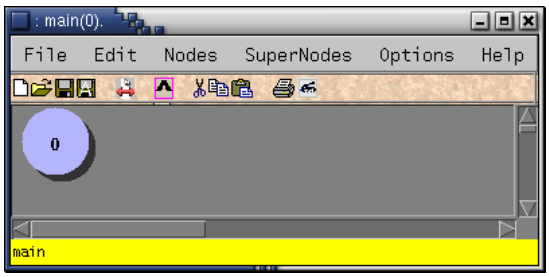


Figure J.46 : Topic Network Builder: Node 0 compose window example

The information window would reflect this newly created node - see Figure J.47. Note that a summary of the node details are displayed, the default RE pattern has been set to '.*', the response has been set to the node label, and the response weight set to 0.7.

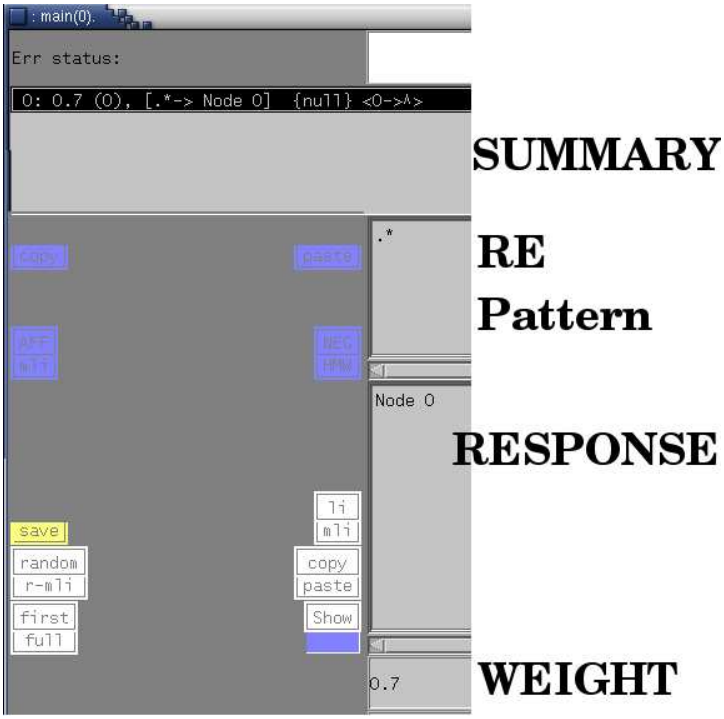


Figure J.47 : Topic Network Builder: Node 0 information window example

Figure J.48 shows the various node types that can be created in the application. The equivalent information window is shown in Figure J.49. The node labelled "1" is an entry node - it will be tested for initial questions but not followup questions. Node "2" is a pro-active node. Node "3" is a switch node and starts the Directed Learning Path as detailed in the Topic Network Builder Section on page 179. Similarly, node "5" is an AFFIRMATIVE node in the DLP. This is used to represent a point on the path that can test the student. Finally, the square node is a supernode with the label "NAME5". This supernode may contain other nodes and supernodes as is seen in the hierarchy of Figure J.44.

A supernode may be created with a left mouse click whilst holding down the Shift key. All other entries are created normally but the type of node is set after creation by the controls shown in Figure J.50.

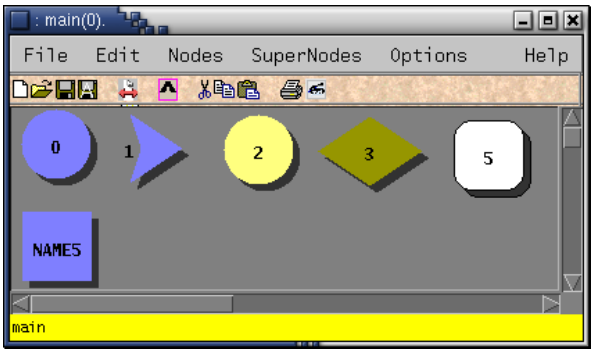


Figure J.48 : Topic Network Builder: example Node types

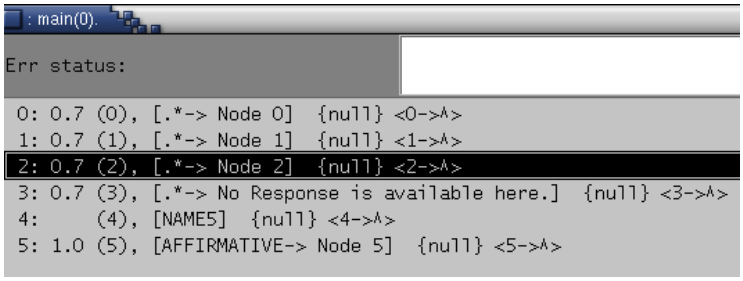


Figure J.49 : Topic Network Builder: example Node information

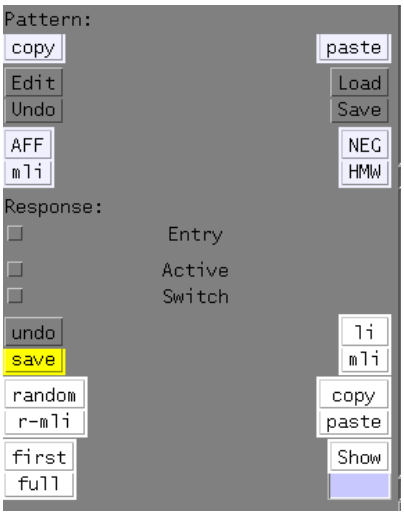


Figure J.50 : Topic Network Builder: Node attributes

Figure J.51 shows the "Help" menu and the two Dialog GUIs that pop up to help in assigning commonly used abbreviations within the Regular Expressions. These two lists are created automatically when the Topic code is updated. An online Help page is also available.

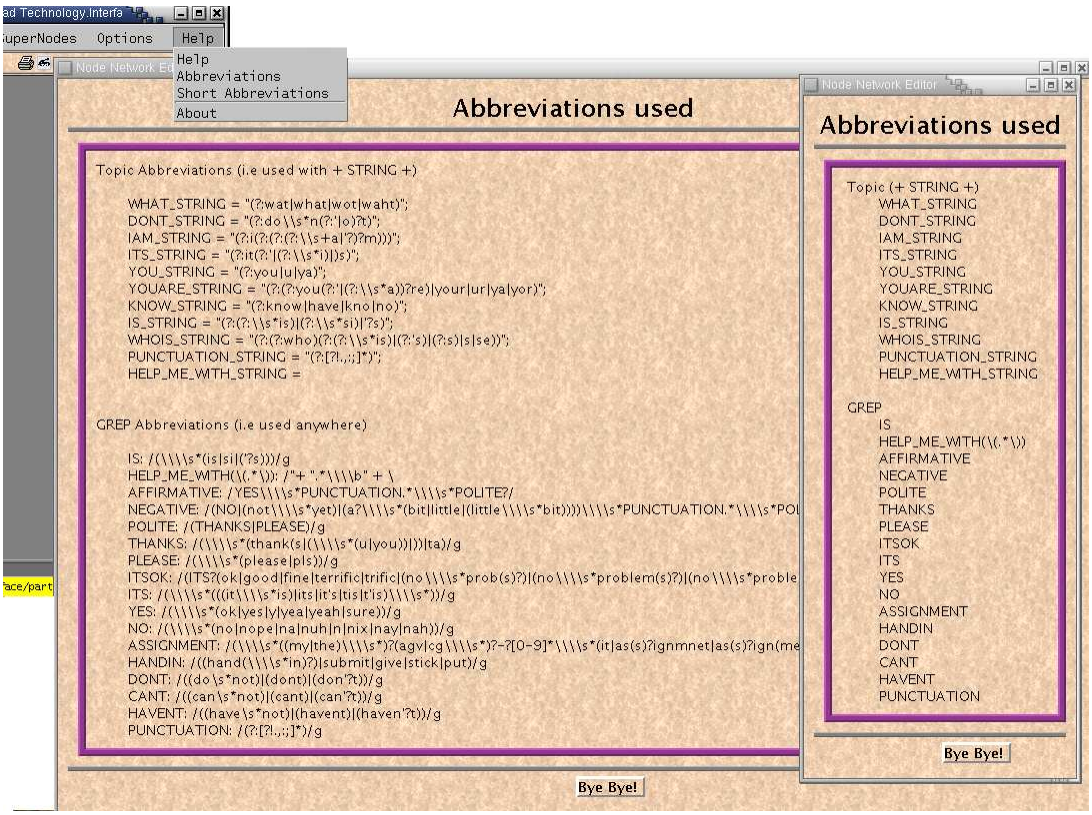


Figure J.51 : Topic Network Builder: Abbreviations

Figure J.52 shows the popup menu available within the compose area. The menu acts on the currently selected nodes and/or supernodes, as appropriate. The entries allow a node to be moved to another hierarchy, to be deleted, to be copied and to be normalised or reassigned the lowest available node number for its label.

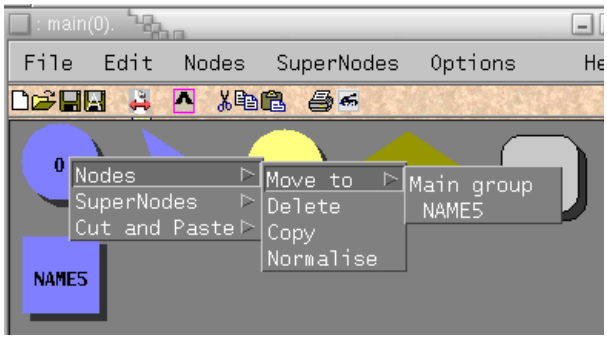


Figure J.52 : Topic Network Builder: Popup entries for processing nodes

Similarly, Figure J.53 shows the popup menu for supernodes. The "Add to" entry allows all selected nodes to be moved into the clicked on supernode. The "Explode" entry simply opens that supernode to be displayed.

The "cut and paste" entry is as expected and allows for the arbitrary copying of nodes and supernodes. Note though that node numbers must be adjusted when this happens since all node numbers must be unique.

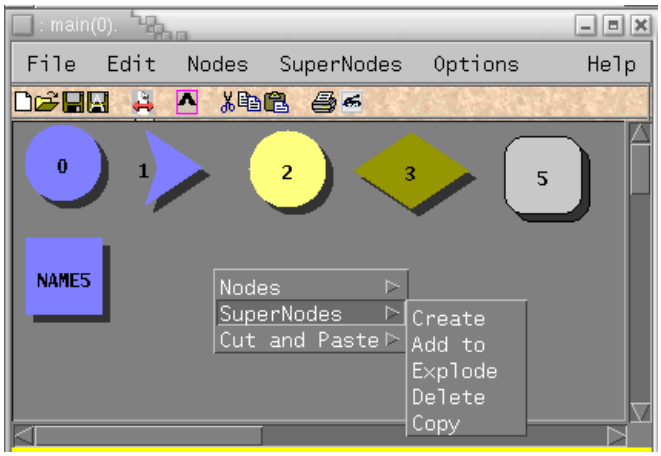


Figure J.53 : Topic Network Builder: Popup entries for processing supernodes

Figure J.54 shows that an edge can be created simply by drawing a line from one node to another. This will set the Next State node of the originating node, and this will be reflected in the information window as well. Note, as can be seen in Figure J.54, the path from one node to the next can go anywhere; it is only the final destination that is important.

To select a node, click on it. To select multiple nodes, click on them with the Shift key held down. To move all currently selected nodes, simply use the middle mouse button. Most operations act on all currently selected nodes.

Double clicking on a supernode opens it up to be displayed. As can be seen in Figure J.55, edges that join to nodes in supernodes at a different level are indicated by an edge to the supernode. Edges that lead to a node not at the same level are indicated by a small edge to a small labelled node. See Figure J.56 and Figure J.57 for a complex example.

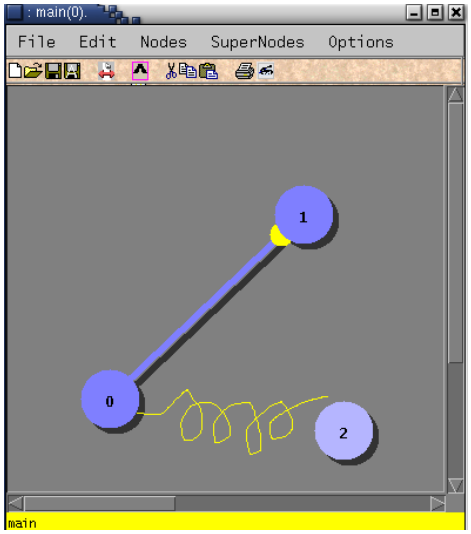


Figure J.54 : Topic Network Builder: Creating edges for Next States

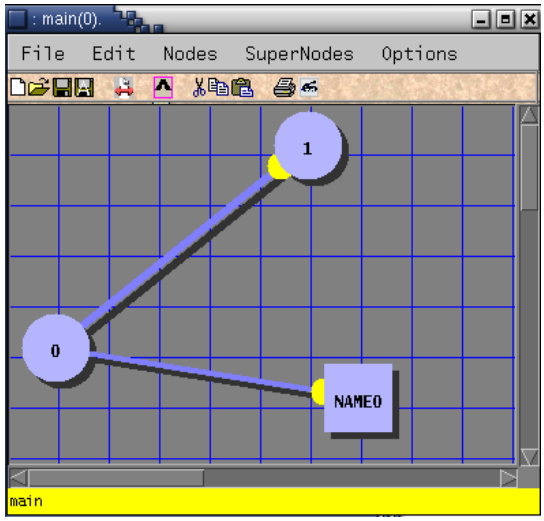


Figure J.55 : Topic Network Builder: Edges for Next States in supernodes

Figures J.56 and J.57 show the integration of the different node types and edges to create an actual Topic network for a unit. Note that the pro-active nodes have next states that lead the user into a dialogue. The main nodes are encapsulated by the Assessment supernode (shown in Figures J.58 and J.59). This in turn holds several supernodes for various aspects of this unit's mentoring help.

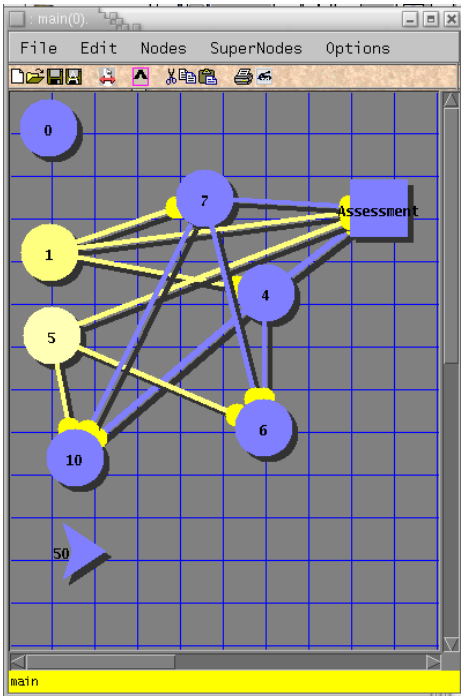


Figure J.56 : Topic Network Builder: Complex network compose window

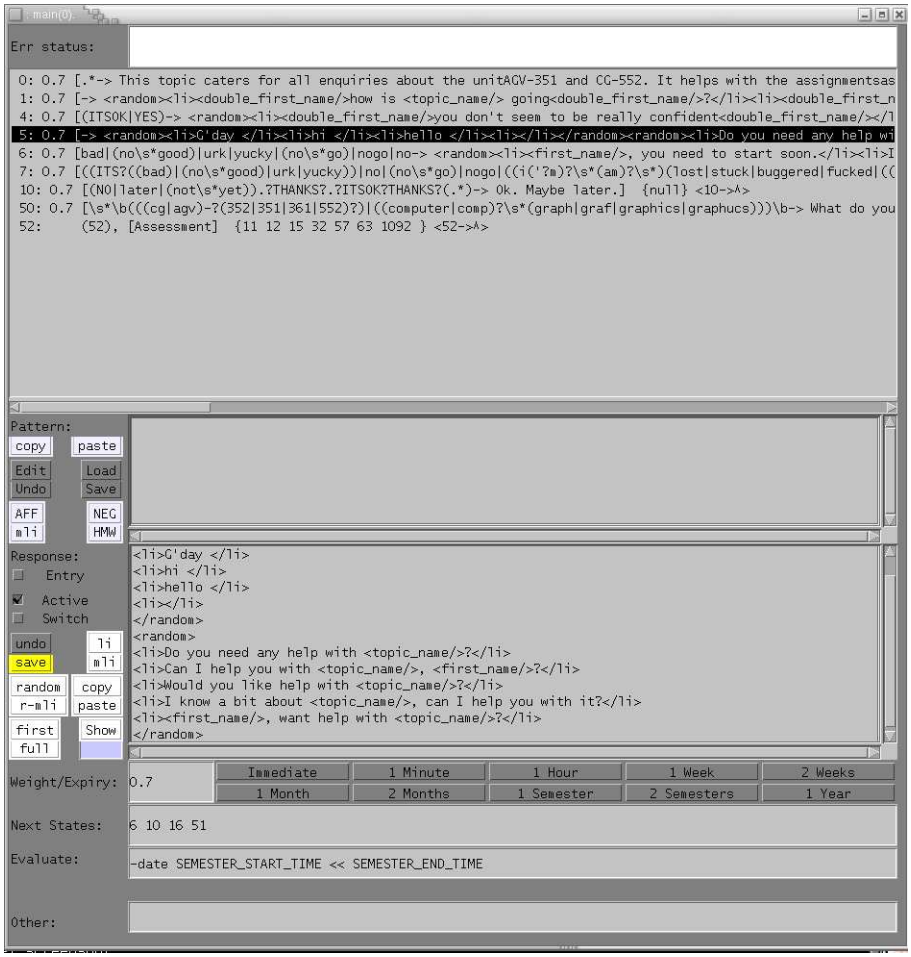


Figure J.57 : Topic Network Builder: Complex network information window

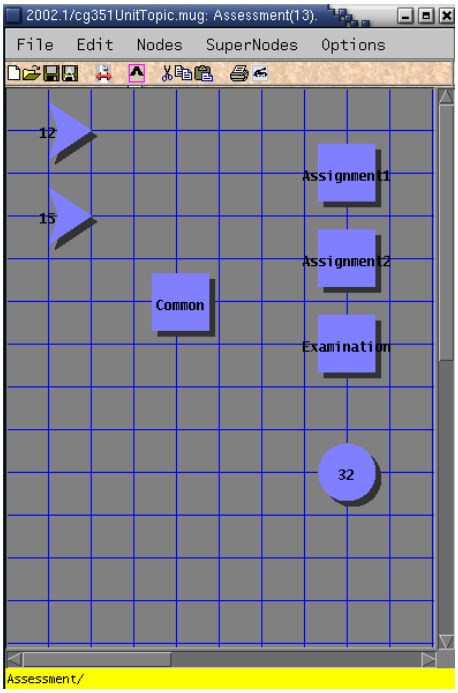


Figure J.58 : Topic Network Builder: Compose window for the 'Assessment' supernode

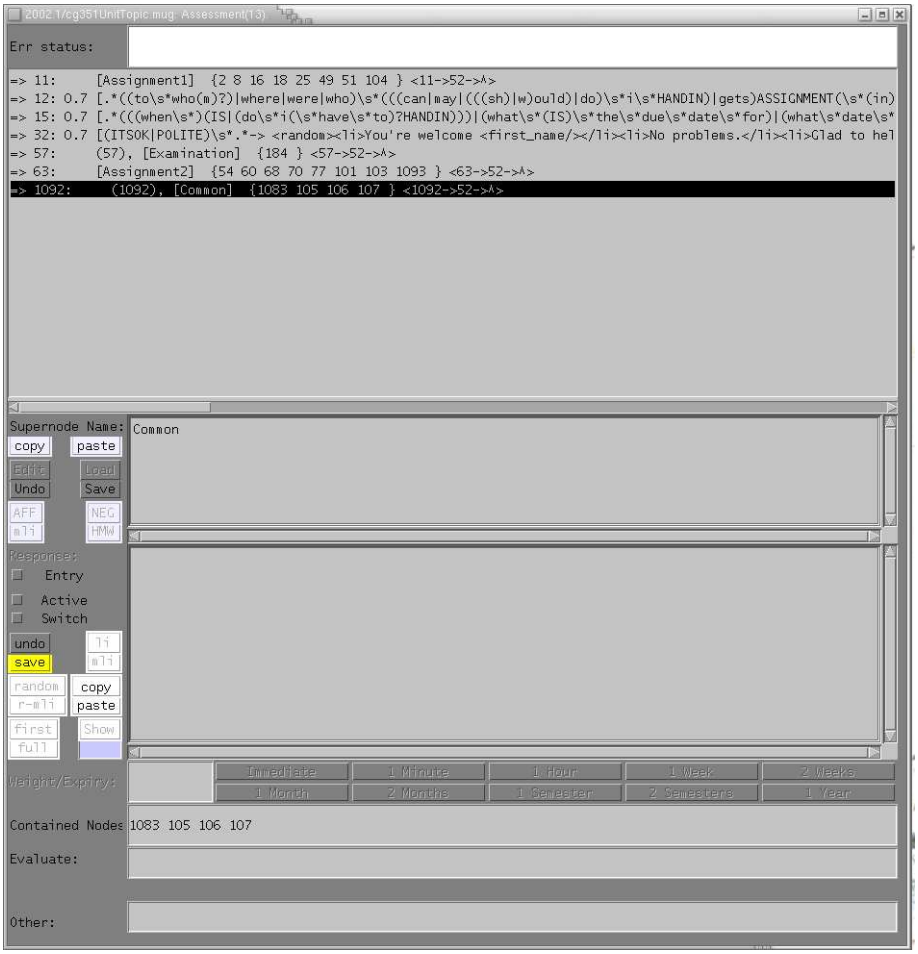


Figure J.59 : Topic Network Builder: Information window for the 'Assessment' supernode

Each of the supernodes in Figures J.58 and J.59 can encapsulate one or more aspects of the learning strategy and question-answer resources for individual parts of a unit. For example, this particular unit had help for both assignments as well as for the examination.

One supernode could hold an "initial question farm" - a series of nodes that are just answers to commonly asked questions (see Figures J.63 and J.64). These may have been mined from an FAQ list, been produced by the Authoritative Guidance support software, or been compiled by hand and imported as a series of question-answer lines (Figures J.60). This farm is easily produced - create a supernode, open it up, and finally, import the appropriate question-answer information.

```
Q:   How do I check that all images exist?
A:   Look at : man access
Q:   How do I use the image template to create the image name?
A:   See : man sprintf
     i.e. sprintf(image_name, image_template, image_number);
Q:   How do I draw pixmap to the window?
A:   gdk_draw_pixmap()
     See : http://developer.gnome.org/doc/API/2.0/gdk/index.html
```

Figure J.60 Example question-answer import data

As with all imports, the Dialog GUI of Figure J.61 is used to set the base node label value as well as the base (x,y) positioning of the created nodes.

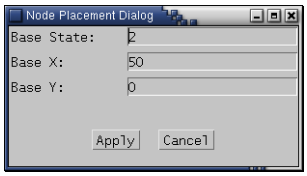


Figure J.61 : Topic Network Builder: Import attributes

The imported questions from a plain text file are minimally optimised into REs. The REs of Figure J.64 have been hand crafted, and this requirement remains the major bottleneck for the ease of use of the system by non-computing people.

Several generic FAQ mining classes have been developed and these can be tailored to produce relatively good "domain specific" Q-A files (Figure J.62).

```
Questions: Is there a Windows version of GTK+? Does GTK run on windows
The FAQ mining application produces the following RE patterns:
0:.*(windows|win[\s-]*95|win[\s-]*nt|windoze) version.*(it|gtk\+?).*
1:.*\b(windows|win[\s-]*95|win[\s-]*nt|windoze)\b.*\bversion\b.*\b
      (it|gtk\+?)\b.*
2:.*does gtk run.*(windows|win[\s-]*95|win[\s-]*nt|windoze).*
3:.*\bgtk\b.*\brun\b.*\b(windows|win[\s-]*95|win[\s-]*nt|windoze)\b.*
```

Figure J.62 Example FAQ mining question-answer import data

Future research should add to the information window functionality so as to allow the inexperienced RE developer to simply type in questions and have an accurate RE generated using set rules as well as domain heuristics.

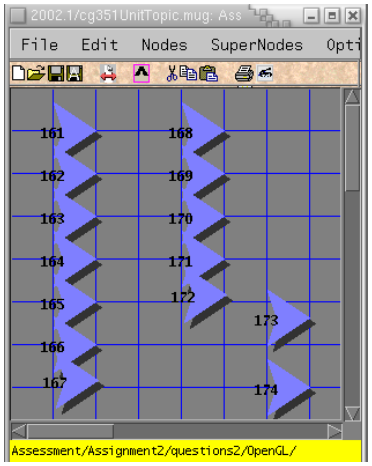


Figure J.63 : Topic Network Builder: Compose window for the ’Initial question farm’

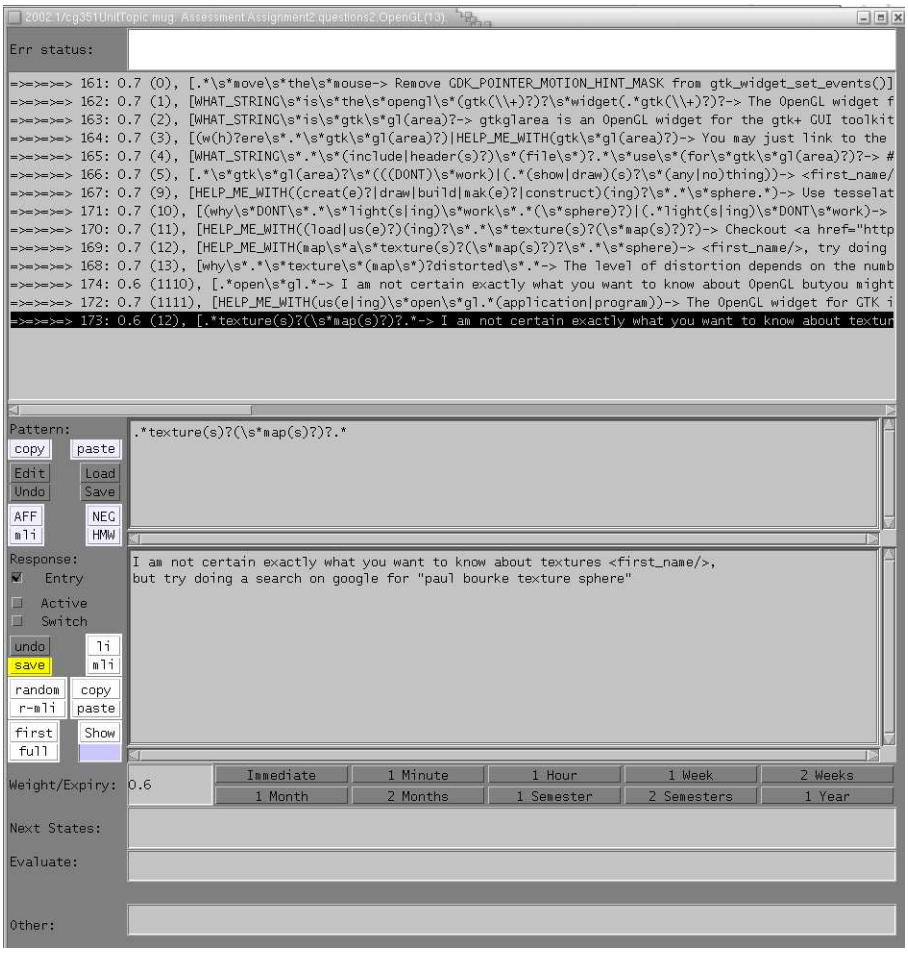


Figure J.64 : Topic Network Builder: Information window for the ’Initial question farm’

Figure J.65 shows a diamond shaped node - this is a general Switch node that allows for the creation of DLPs. In this example, many Entry nodes point to this Switch node to process any Next State requests. This means that the **response** to this request does not come from the Entry node but from some part of the DLP connected to this Switch node.

For example, one of the Entry nodes may match "help me with spd", another may match "I need help on my assignment", and another may match "where do I start with my assignment". The response to this will come from the DLP. Figures J.66 and J.67 show a part of a simple DLP, with the round cornered square nodes being special DLP nodes. See Section 4.2.7.2 on page 184 for detail on how this path is processed.

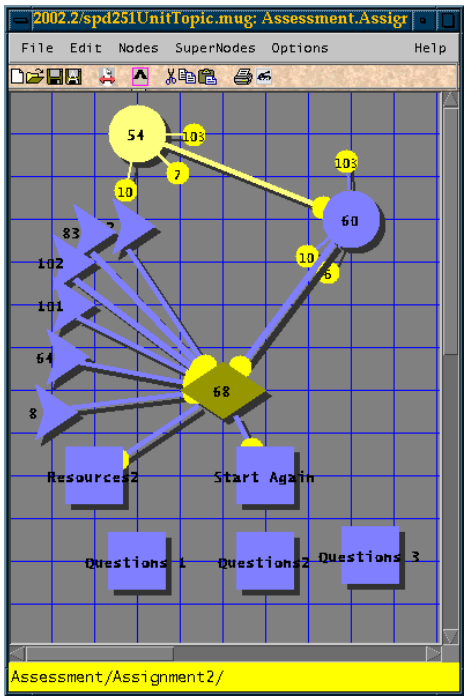


Figure J.65 : Topic Network Builder: Directed Learning Path

A typical DLP scenario is a series of "Have you done this?"- "yes/no" question and answers, but the DLP can be made arbitrarily complex. Figure J.66 shows a DLP, with a complex question-answer "milestone" branching off from node 45.

Once the network has been created, it can be exported as a Java data structure (Figure 4.55 on page 180), and processed via a Unix Makefile to create a standard ***Mentor*** Topic. See page 182 for detail on how this Topic can then process user input.

Complex Topics of over 200 interconnected nodes have been created using the Topic Network Builder application. Although it is not a perfect point and click application for creating Topics, its simplicity at creating nodes of information that respond to user input, and its ability to handle the complexity of Next State node processing, make it a useful application for the production of large scale networks as well as Directed Learning Paths.

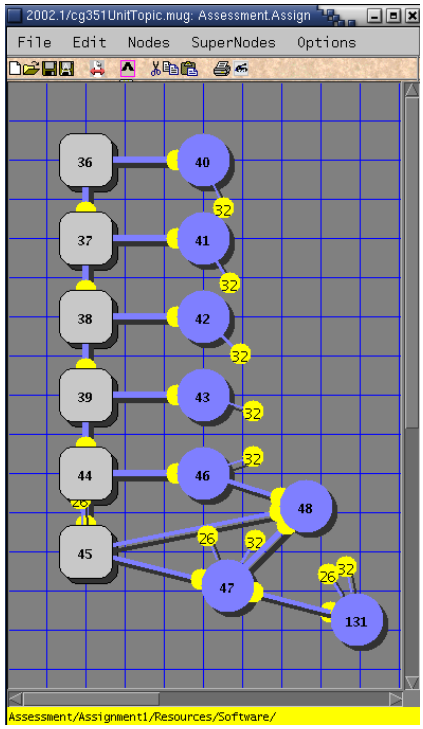


Figure J.66 : Topic Network Builder: Compose window for the ’Directed Learning Path’

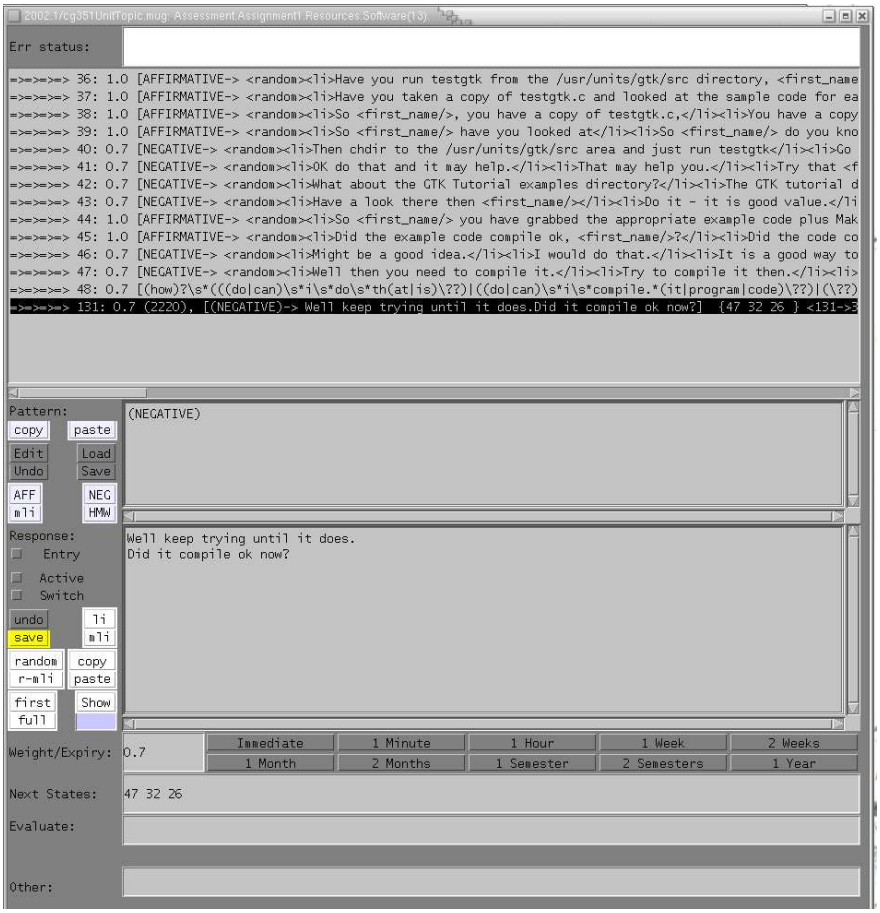


Figure J.67 : Topic Network Builder: Information window for the ’Directed Learning Path’

> *What goes around,*
> *comes around...*
> Anon.

K: CDROM

The CDROM contains the following material and resources:

Description	Path
• The thesis as a PostScript and a PDF file.	thesis/
• The images in the thesis.	thesis/images/
• Online documentation for the ***Mentor*** System	documentation/
• The questionnaires as Word Document files.	questionnaires/
• The SPSS evaluation Raw Data and output results.	evaluation/
• The source Java files for the local.raytrace packages.	src/local/raytrace/
• The source Java files for the local.mentor packages.	src/local/mentor/
• The source Java files for the SADM software.	src/SADM/
• The Javadoc files for the local.raytrace packages.	docs
• The Javadoc files for the local.mentor packages.	docs
• The various DTD files.	DTD/
• The user Topics example files.	userTopics/
• The extended multimodal Topics example files.	extendedTopics/