©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)
© 2008 IEEE.

Conceptual Modeling of Knowledge Based Systems for Digital Ecosystems

Darshan Dillon, Tharam.S. Dillon

Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology
e-mail : {darshan.dillon, tharam.dillon} @cbs.curtin.edu.au

***Abstract*— The agents or entities frequently require intelligence in the form of Knowledge Based Systems(KBS) to support many of their functions. In this Paper we discuss how these KBSs are conceptual are conceptually modeled as a first step towards their development. In particular, we show to effectively model all the different knowledge constructs using an extended definition of an Object. The notation used to express this is UML [Booch 2005].**

INTRODUCTION

Digital Ecosystems frequently require the agents involved to possess intelligence. This intelligence can take several forms, but a widely utilized form is the Knowledge Based System or KBS. Knowledge Based Systems are used to solve problems that normally:

- are solved by skilled human experts, who use their expertise for solution of the problem.
- require human judgment rather than mere calculation, retrieval, collection or display of data. These might precede the exercise of human judgment, but they do not subsume or replace it.
- require experience that is gained over a long period of time. This is normally acquired by the expert through observation and interaction with a myriad of different situations.
- involve a solution approach that cannot be directly formulated into an algorithm or a mathematical model.
- frequently require the manipulation of symbolic rather than numerical information.
- could involve imprecise knowledge that is characterized by uncertainty. This uncertainty could be characterized by probability-based representations or, alternatively, it could be of the subjective kind. Subjective uncertainty could reflect subjective judgments or a lack of precision in the concept.

The knowledge-based system seeks to capture an expert's knowledge for solving a particular class of problem and to represent that knowledge in a form that allows machine implementation. This is sometimes referred to as **knowledge engineering**. This process is, in fact, a particular form of conceptual modeling. In this paper, we will further explore the modeling aspect associated with knowledge-based systems particularly for their use in Digital Ecosystems. It begins by a discussion of the modeling requirements of these systems. In Digital Ecosystems these KBSs will frequently be required to interoperate with databases.

In the Sections that follow, we will introduce the structures that need to be added to the basic object-oriented modeling constructs to permit one to model expert systems using the object-oriented paradigm.

A KBS carries out symbolic processing and consists of the following components:

1. Knowledge using heuristics or rules of thumb of the problem domain. a knowledge base (and a working memory)
2. An inference engine
3. A user interface
4. Interfaces to other databases or systems.

MODELLING THE PROBLEM SOLVER AND THE PROBLEM SPACE

As explained earlier, in a Knowledge Based System one determines the way a domain expert solves problems during the process of knowledge engineering and represents this in an explicit form that is ultimately the basis of a machine implementation. In this case, in addition to modeling the real-world problem space, one also needs to model the manner in which the problem solver tackles the problem. Thus, in an expert system one needs to model:

- the problem space;
- the problem solver.

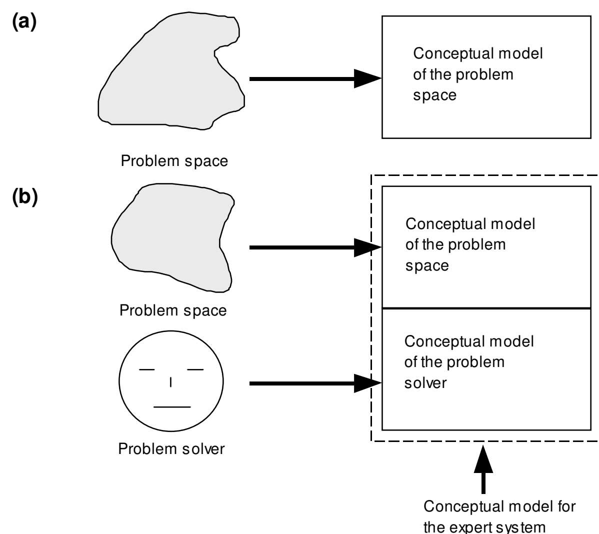

Figure 1 (a) Traditional software application, (b) Knowledge Based System application

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

© 2008 IEEE.

The difference between the conceptual modeling carried out for more traditional software applications and that required for expert or knowledge-based systems is illustrated in Figure 1.

It is important here to emphasize that when considering the model of the problem solver, again one focuses only on aspects of the problem solver related to the specific problem at hand. Indeed, it would not be possible with the current state of the art to model all aspects of a problem solver.

One can distinguish the following types of knowledge that are frequently employed by problem solvers:

- Heuristics or rules of thumb. These are normally related to knowledge that the problem solver has acquired by experience and frequently are empirically determined associations.
- Stereotypes that are used to designate typical examples of some objects or situations.
- Solution hierarchies that employ different levels of looking at the problem. These are frequently associated with the level of detail the problem solver wishes to deal with at one time.
- Procedures that represent explicitly defined solution strategies and algorithms. The solution in this case is defined as a sequence of actions that, if carried out, leads to the required result. This type of knowledge is close to that found in traditional software applications. However, one important difference is that a procedure may only be called by the expert if it is needed or if something changes that requires it.
- Pattern matching a given set of conditions with a situation or the current state to see if the conditions are satisfied.
- Qualitative or quantitative reasoning with a model of the real-world phenomena. This frequently involves obtaining a model of the phenomena and then using it in a qualitative or quantitative simulation.
- Reasoning with primary case material, where it is not possible to reduce the knowledge involved to a simple enough set of heuristics because of the highly context-dependent nature of the knowledge. This has recently been modellled using soft computing approaches [Pal, Dillon and Yueng 2001] .

The important thing is that when one wishes to represent all these different kinds of knowledge that the problem solver uses, it may be necessary to provide additional modeling constructs to those discussed in UML. [Booch, 2005]

A KBS system frequently involves an open world assumption rather than a closed world assumption. The closed world assumption is often associated with traditional software and database systems. There are some fundamental differences between the modeling required for traditional software applications and for knowledge-based systems.. Hence additional constructs are necessary, in particular to assist with modeling the problem solver's knowledge and the imprecise and uncertain knowledge, and to permit the construction of open world models.

3. MODELING HEURISTICS WITH PRODUCTION RULES, STATES AND OBJECTS

One can effectively model heuristic knowledge through production rules and facts. Heuristics can be effectively represented by production rules which are condition-action pairs are of the form:

IF P THEN Q

The condition or premise P is matched against the working memory and if it is found to be true, the action Q is carried out. The premise P may be composed of other simpler premises P1, ..., Pn. In early versions (Buchanan & Shortliffe 1984) of these production rules, the premises P1, ..., Pn were joined together by the connective AND. However, most Knowledge Based System tools now allow for the premises to be joined together with the connectives AND or OR and negated by NOT. Facts are assertions about the state of the system and are used to fire the rules. We note that attributes of objects are used to characterize the state of the system can be used as storage for the facts or state of the system and then to use them to trigger rules. These *objects* or *entities* each have an entity name, attribute names and attribute values. They do not include any methods or messages. The objects or entities, with their associated attribute names and attribute values, are then used as the premises in the left-hand side or condition part of the rule.
Thus, in these cases the rule would have the syntax:

```
IF      entity A, attribute A1 = value V1
AND     entity B, attribute A2 = value V2
THEN    carry out (action Act1)
```

If the entity is the same in all the premises of a particular rule, it can simply be written as:

```
IF      entity A, attribute A1 = value V1
AND     attribute A3 = value V3
THEN    carry out (action Act2)
```

This form of an *object* or *entity* within a rule is, of course, much more limited than that used in object-oriented systems. However, the idea of having (object, attribute, value) triples as premises of rules can be used in modeling and we will use it in our extension of the object-oriented paradigm to include knowledge bases.Thus, the rules used in our extension of the object-oriented paradigm have the form:

```
IF P1
$ P2
$ P3
.. ..
$ Pn
THEN carry out (action A)
```

where Pi has the syntax:
Pi = (object name i, attribute name Ai, value Vi)
and $ denotes the connectives:
AND, OR, AND NOT and OR NOT

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)
© 2008 IEEE.

As the number of rules in the knowledge base becomes very large, the cognitive on the knowledge engineer is unacceptable and an additional structuring facility must be provided such as grouping of thematically consistent rules or rules with a similar purpose into smaller chunks known as a **rule set** or **rule state**. A rule state is analogous to a function module. Each rule state has an agenda and a control mechanism to fire the rules within the rule state or to call another rule state. Each rule state can be called dependently or independently, and in sequence or at random.

The next extension is to associate a particular rule state with a particular object. The rules in this rule state will be largely restricted to using the attribute names and values of its associated object. This approach allows one to achieve some degree of encapsulation. In practice though, one frequently needs to access the attributes of at least one other object to obtain the desired goal state. However, some degree of encapsulation has the advantage of making the modeling clearer and making future maintenance and perhaps re-use easier. The symbols used for a rule state and rules are given in Figure 2. Note that the rule state (<<rule-state>>) is a stereotype in UML [Booch, 2005] which represents the collection of rules which reason using the facts in a class it is attached to. It contains information such as the name of the rule state as a whole. A stereotyped class for rule (<<rule>>) is bound to the rule state in order to reflect the fact that rules belong to a specific rule state and are part of it. In our case the rule state is known as Rule_State_A and the class whose instances represent specific rules is known as Rule_State_A_Rule. An instance of the rule class indicates a specific rule, known as Rule_A1. Thus, if a specific rule state has 10 rules then there will be one instance of the rule state class and 10 instances of the rule class.

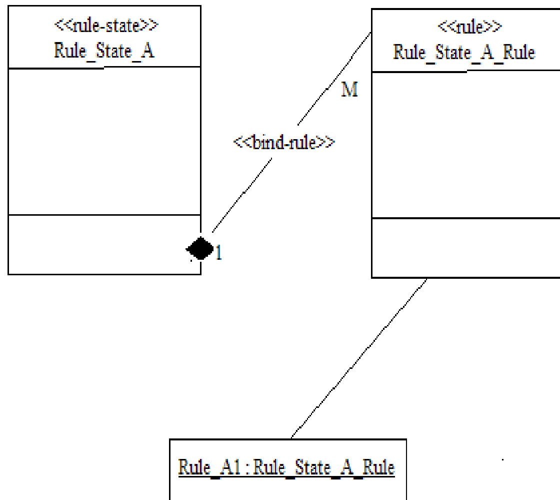

Figure 2 Symbols for a rule state and rules in an Object-Property diagram

An illustration of the rule states attached to particular objects is given in Figures 3.

We first begin by examining the stereotypes defined in Figure 3. Note that there are 2 stereotypes which relate to relationships. These include <<bind-rule-state>> which binds the <<rule-state>> to the class that is using it to reason. Note that the <<rule>> stereotyped class contains attributes which specify the rule name and also the content. The internal structure of each rule will be represented in a string that will go in this attribute. A class such as CAR_SEAT_COVER (we are using the car rental example) may have a subset of the total rule base used to select a car seat cover. For instance, there may be a total of a 1000 rules in the system, but only 10 of them that relate to choosing a car seat cover. Rather than going through the entire rule base every time we try and choose a car seat cover, we isolate and restrict our attension to only the rules that focus on the relevant goal. This is done by putting those 10 rules into a rule state and attaching it to the CAR_SEAT_COVER class. A stereotype <<bind-rule-state>> is built on the association relationship which will bind the rule state to the method section of the CAR_SEAT_COVER class. Similiarly there is a stereotype built on the composition relationship (<<bind-rule>>) which will bind the class that represents individual rules to the rule state as a whole.

The above discussion applies equally to the CAR class, rather than the CAR_SEAT_COVER.

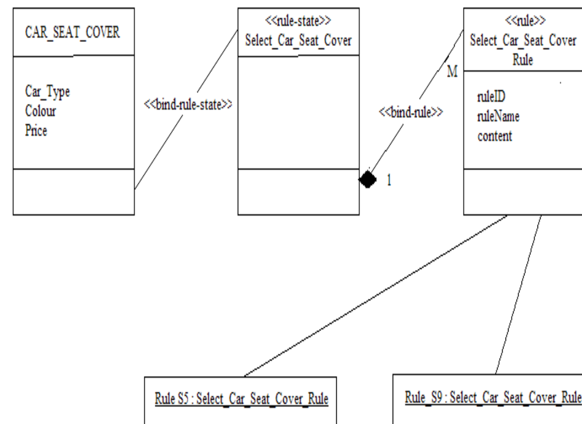

Figure 3 Symbols for a rule state and rules in an Object-Property diagram

It is important to notice in the above extension of the object-oriented paradigm to include rules, that the objects correspond to the conceptual model of the real world while the attached rules and rule states correspond to the model of the problem solver's heuristic knowledge.

REPRESENTING STEREOTYPES AND HIERARCHICAL KNOWLEDGE USING FRAMES

A representation that uses only production rules is suitable for purely heuristic-oriented applications. However, not many applications are solely of this nature. There are stereotyped items in the real world that participate in the problem-solving process. Frames would be a more suitable representation for these stereotyped items (Minsky 1975). A frame is a prototypical structure that contains the features of an object, item or event. It has a frame name and a number of slots and fillers. The slots are analogous to field names in records or attribute names in objects, while the fillers represent the values.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)
© 2008 IEEE.

Facets are used as the means of attaching this control information to a slot (Walters & Nielsen 1988). Classically, we can distinguish between the following types of facets (Winston 1984):

1. value facets
2. if-needed facets
3. default facets

Value facets provide control of actual values, On the other hand, it may be necessary to pass control to a function or procedure to return a value for a particular slot. This can be achieved using the **if-needed facet**. Thus the price of the rental could be computed by a function or procedure. Note that such functions or procedures would only be activated to carry out a computation when the value for the particular slot is needed rather than using a procedure call at a particular point in a program. This sort of function or procedure is called a *daemon*. Thus, when an access to the slot is made to fetch information the daemon is activated. These daemons are discussed in greater detail in Section 6.2. **Default facets** provide a most likely value that is used in situations where an actual value is not given.

In addition to providing a representation of a stereotype, frames permit one to structure objects and events into a hierarchy using classes. Therefore, frames also provide an effective means of representing knowledge associated with classification of objects and events. The reader might have noticed some similarities between frames and objects. The similarities and the differences between frames and objects are discussed in Section 6.1.

HYBRID RULE AND FRAME-BASED SYSTEMS

Rule-based systems essentially combine heuristics, facts and relationships in a uniform structured knowledge base. In many cases there is considerable leverage to be gained when heuristics are combined with the stereotyped elements. This leads us to another category of Knowledge Based Systems, namely *hybrid rule and frame-based* systems. The knowledge base of this category consists of rules and frames. Rules are used to represent heuristics and frames are used to represent the stereotyped items.

The rules considered here, however, have an extended type, namely pattern-matching rules which are written as IFpm-THEN, in addition to the normal IF-THEN rules. In this paper, we distinguish the pattern-matching rules with the notation IFpm-THEN from IF-THEN rules for clarity, although this distinction may not be present in the particular Knowledge Based System tool used. A pattern-matching rule, IFpm-THEN, matches the frames with conditions specified in the IFpm part of the rule.

The rules in such hybrid systems use the slot values of the frames to set up the premises or conditions part of the rule. The pattern-matching rules also carry out a match across several classes and perhaps even across different hierarchies. An important manner in which rules and frames are used together is to associate a particular rule state to an if-needed (or if-changed) slot. When the slot is accessed, the associated set of rules in the rule state is activated to carry out the reasoning required. Note that during this reasoning only the specified set of rules in that particular rule state will be considered by the inference engine rather than all the rules in the system.

This ability to associate rule states with slots in a particular frame allows one to structure a reasoning hierarchy. The frames in the reasoning hierarchy are structured so that those at the highest level deal with the more general conclusions, with these conclusions being specialized as one comes down the hierarchy.

In certain circumstances, it is desirable to fire a pattern-matching rule only once even though it matches more than one object.

EXTENDING THE OBJECT MODEL FOR HYBRID RULE AND FRAME-BASED EXPERT OR KNOWLEDGE-BASED SYSTEMS

Rules and objects can be integrated to provide extensions to the object model to include IF-THEN rules and rule states. These extensions permit one to incorporate aspects of heuristic knowledge into an object-oriented model. In this section we will consider additional extensions to incorporate in the object-oriented conceptual model the modeling power available in hybrid rule and frame-based systems described in the last section. In order to do this, we begin with a comparison of frames and objects.

Rules and objects can be integrated to provide extensions to the object model to include IF-THEN rules and rule states. These extensions permit one to incorporate aspects of heuristic knowledge into an object-oriented model. In this section we will consider additional extensions to incorporate in the object-oriented conceptual model the modeling power available in hybrid rule and frame-based systems described in the last section. In order to do this, we begin with a comparison of frames and objects.

Frame	*Object*
• Frame	• **Class**
• instance	• **instance**
• slots	• **instance attributes or class attributes**
• passive	• **active**
• inheritance	• **inheritance**
• defaults	• **defaults**
• **daemons**	
• **rule attachment**	
• **reasoning with a pattern-matching rule**	
	• **constraints**
	• **object identity**
	• **encapsulation of methods**
	• **polymorphism**
	• **message passing**

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

© 2008 IEEE.

Figure 4 Differences between frames and objects

6.1 FRAMES AND OBJECTS

There are some important similarities and differences between the concepts of frames and objects.

The similarities between an object and a frame are as follows:

1. Frames and objects each have an identifier or name.
2. One can have the notion of class and instance with both objects and frames.
3. Frames have slots with slot names and these can take on a value. Similarly, objects have attribute names and attribute values.
4. Both object and frame structures permit single and multiple inheritance.
5. Both frames and objects permit one to attach procedures. However, the manner and type of attachment are different and this is discussed among the differences below.

Important differences between frames and objects are as follows:

1. Frames possess daemons that are activated each time a slot is accessed, updated or deleted. Objects do not possess a similar form.
2. Objects use messages for activating a method. A frame lacks the message-passing facility and the methods of an object. Although a frame has daemons that are activated when a slot is accessed, it is more of a passive structure as distinct from the active object that responds to messages received from other objects.
3. A frame permits one to attach IF-THEN rules and rule states to slots. This allows one to model heuristic knowledge.
4. An object has the notion of encapsulation of attributes and methods within the object. Frames do not provide an effective mechanism for this.
5. Pattern-matching rules, which require the direct access of information in more than one frame, can be used without complication in frames. These extend the heuristic modeling capability to permit reasoning about structures and across structures. No pattern-matching rules are available in the objects discussed to date.
6. Objects can easily incorporate complex constraints. This could be more difficult with frames, even though facets and constraint specifying capabilities have been provided with some Knowledge Based System tools.
7. Objects have the facility of polymorphism, whereas frames do not have this property.

The differences between frames and objects are summarized in Figure 4.

6.2 EXTENDING THE OBJECT CONCEPT TO OBJECTFS

To distinguish the new notion from an object as defined earlier, we use the stereotype <<*objectf*>> to indicate that it has the characteristics of an object as defined in pure object-oriented languages, as well as the characteristics of a frame. The characteristics of an objectf are in bold type in Figure 4.

To facilitate this discussion, we draw attention to the fact that slots are often used to refer to attribute names in the Knowledge Based Systems area. We adopt this convention as, in addition to attribute values and attribute defaults, it is possible to attach a procedure or rule set to such a slot. This gives the slots an ability to have a dynamic component rather than purely static components.

The extensions that need to be made to the object-oriented model include:

1. *For objectfs, permitting the attachment of daemons to a slot in an object*

Daemons are functions that are attached to the slots of a objectf. They are activated when the associated slot is being updated, accessed or deleted. A daemon is somewhat similar to a method in that it consists of procedural code but it is automatically activated without the need for a message to be passed. To activate daemons, one can access the relevant slots of the objectfs in two ways:

(a) directly, as shown in Figure 5;
(b) indirectly, such that a message is sent to a method of a objectf to access the slot of interest, as shown in Figure 6.

These daemons are useful for determining derived values, monitoring a situation and carrying out a calculation only when something changes. Together with the methods

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

© 2008 IEEE.

in objectfs, these daemons allow modeling of procedural knowledge.

As far as the stereotypes are concerned in Figure 5, there are several. Firstly, note that we have stereotyped a class to represent an objectf by using the <<objectf>> stereotype. Each objectf will have many slots, and slots are part of the objectf. We bind each slot to it's parent <<objectf>> by the <<bind-slot>> stereotype. We have defined a stereotype to represent each slot, named <<slot>>. It is important to note that if a particular objectf has, say, ten slots there will still be only one <<slot>> class. There will be ten instances of this class. The <<slot>> class has four stereotyped attributes. They include a field that represents the name of the slot (<<name-field>>) and three types of slots that represent the different types of facets that were discussed earlier. It is necessary to have all three facets of a slot represented concurrently since each slot may have more than one facet used at a particular time. In this example, the objectf is called A, and it has a slot named AA with a daemon named D used to calculate the value of the slot. The <<if-needed>> stereotype is of particular interest since it may have either the name of a daemon or a rule state attached to calculate the value which goes in the value facet.

Figure 6 has the same layout as Figure 5 with the exception of an additional method in the <<objectf>> class. Every other stereotype has the same meaning.

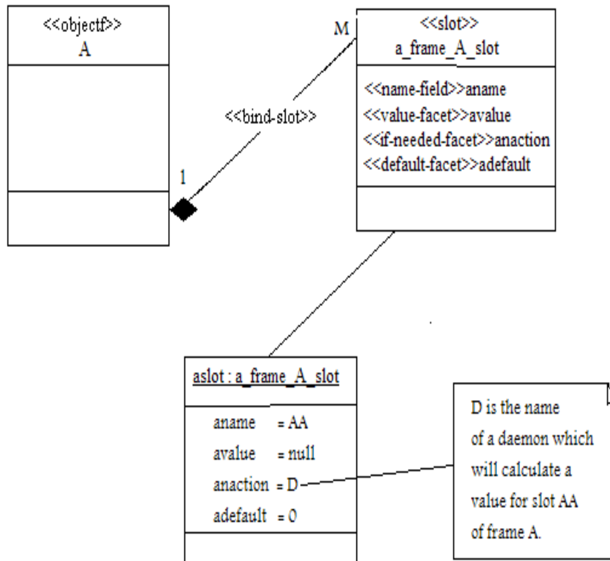

Figure 5 Change in or access to slot AA causes activation of the daemon

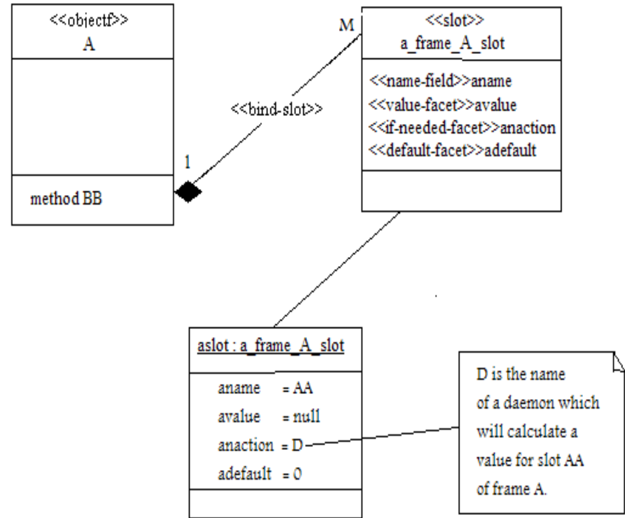

Figure 6 Message to method BB of objectf A leading to indirect activation of the daemon

2. *For objectfs, permitting the attachment of rule states to an object*

The production rules should have the form that the premises of the rules consist of a conjunction or disjunction of triples (objectf, attribute = value). Such a rule state could be attached to a slot and activated when the slot is accessed, much in the manner of a daemon, as shown in Figure 7. Alternatively, it could be called from within a method or function, as shown in Figure 8. Some Knowledge Based Systems tools support this call from within a method.

Stereotypes for Figure 7 have the same meaning as for Figures 5 & 6 except the use of the <<if-needed-facet>> is slightly different. Previously the name of a daemon was used as the value. If we choose to use a rule state rather than a procedure to calculate the value for slot AA, we simply put the name of the rule state as the <<if-needed-facet>> value rather than the procedure name. The note attached to the class reflects this slightly different use of the stereotyped attribute <<if-needed facet>>.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)
© 2008 IEEE.

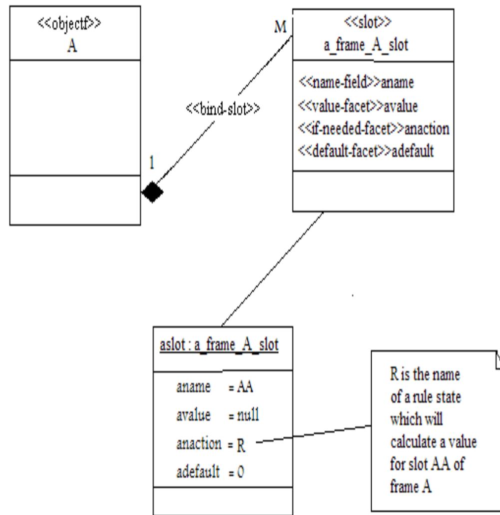

Figure 7 Rule state attached to a slot and activated by accessing the slot

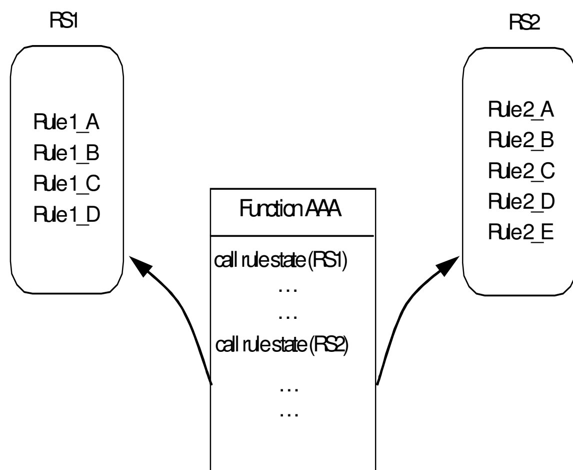

Figure 8 Rule state called from within a function

These rule states permit the modeling of heuristic knowledge

1. *Allowing for pattern-matching rules to be added to the system*

These rules have premises that involve the attribute values from more than one objectf or more than one hierarchy of objectfs. Such a rule for the Rent-a-Car example is shown in Figure 9.

```
IFpm    ?aCar of CAR
WITH    ?aCar.Size = Small
        ?aCar.Status = Not Rented
AND     ?Customer.Status = Eligible
THEN    set (?aCar.Status = Rented)
AND     add (customer name and car
        registration number to rental
        list)
```

Figure 9 Pattern-matching rules for the Rent-a-Car example

For the purposes of modeling, pattern-matching rules could be put into a pattern-matching rule state for the system. However, an object-oriented pattern-matching rule state may contain a mixture of ordinary IF-THEN rules as well as pattern-matching rules, provided that it has at least one of the latter. The symbols for a pattern-matching rule state and a pattern-matching rule are shown in Figure 14.36. As far as stereotypes are concerned we define a different stereotype for a pattern matching rule state. It is named as <<pm-rule-state>> where 'pm' is short for pattern matching. Note that we still use the <<rule>> stereotype to represent both rules that are pattern matching and normal. Usually we include a stereotyped attribute <<rule-type>> in the class stereotyped <<rule>>. The reason for doing it this way is that we propose that the rule itself be included in a field stereotyped as <<content>> in <<rule>>. So whether a rule is pattern-matching or normal it will be included in this field. The inference engine that examines instances of rules can look at the <<rule-type>> field to decide how to interpret the content of the rule. It is important to note that even though a pattern-matching rule has the 'IF' part of itself relying on facts from more than one class, it is still attached to a rule state that belongs to one class in particular. For example, the rule in Figure 14.35 has references (in the 'IF' part of the rule) to both CAR and CUSTOMER. Nevertheless, a CAR is rented by a CUSTOMER so CAR is the subject in the relationship between the two classes. Hence the rule will be included in a rule state attached to the class CAR, rather than CUSTOMER. This additional analysis will have to be done by the modeler in the case of all pattern-matching rule states. Bear in mind that a pattern-matching rule state will contain some normal rules which only rely on a single class, so this will simplify the process of deciding which class to assign the rule state too.

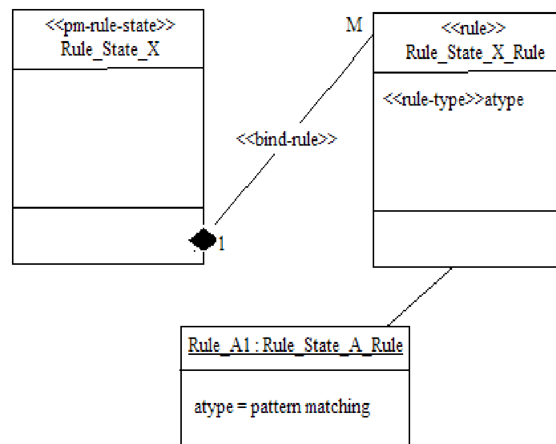

Figure 10 Symbols for pattern-matching rules and rule states

2. *Allowing for inheritance to be stopped at a particular class level*

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.

© 2008 IEEE.

In object-oriented systems, inheritance implies that all properties pertaining to a superclass are inherited by the subclasses down the hierarchy chain, unless they are overridden. Despite the feature of overriding of the properties by a subclass, there are situations where properties of a superclass are not needed by all the subclasses or instances of the class itself. These properties could exist for a particular purpose for the superclass.
Inheritance can be stopped along the hierarchy chain. The prevention of inheritance of attributes can be specified at two levels, the class and instance levels.

3.Permitting the specification of reasoning hierarchies through the use of objectfs and associated rule states or procedures.

The objects in this case would capture the features of the reasoning associated
with a particular level of inference, including the attributes, rules and procedures used at this level. It should be noted that this organization of objectfs into reasoning hierarchies is somewhat different from the hierarchy associated with a physical object. These reasoning hierarchies capture aspects of the problem solver's knowledge, whereas the others are modeling classifications, generalizations and abstractions of the problem space. This, however, could limit the objectfs associated with a reasoning hierarchy to a particular application.

The type of expert or knowledge-based system that requires qualitative or quantitative reasoning does not pose any special new problems for object-oriented modeling, as the model required can easily be encapsulated in a single object or a collection of several objects. The challenge is to pick the right model during analysis and ensure that the reasoning approach written was suitably devolved to the right objectfs.

In systems that contain both model-based and rule-based reasoning, the so-called second generation systems, the rule-based components could be grouped into rule states that are attached to objectfs, and the model-based components could be expressed as a set of objects. Communication between them, including the passing of control asynchronously from one type of reasoning system to the other, can be uniformly performed using messages.
If the qualitative or quantitative model consists of several objects, it is important that it be isolated into its own subject layer in the representation. If the model is still too complex, one could nest subject layers within this isolated layer to control the complexity of the representation.

RECAPITULATION

In this paper, conceptual modeling appropriate to an expert or knowledge-based system was considered. Specifically, the extensions necessary to the object-oriented paradigm were described.

REFERENCES

Booch, G. & Rumbaugh, J & Jacobson, I. 2005 “Unified Modeling Language User Guide”, 2nd Edition The Addison-Wesley Object Technology Series

Buchanan, B.G. & Shortliffe, E.H., eds 1984, *Rule-Based Expert Systems*, Addison-Wesley, Reading, Massachusetts

Cooke, N.M. & McDonald, J.E. 1986, "A formal methodology for acquiring and representing expert knowledge", *Proc. of the IEEE*, vol. 74, no. 10, October, pp. 1422-30 *

Dillon T.S. and Tan P.L. ”Conceptual Modelling of Object Oriented Systems” Prentice Hall 1993 *

Dillon T.S. ,Chang E., Rahayu W., Darshan Dillon 2008“ Conceptual Modelling and Design of Object Oriented and Component Based Systems “ In Print *

Duda, R.O., Hart, P.E., Nilsson, N.J. & Sutherland, G.L. 1978, "Semantic network representations in rule-based inference systems", in *Pattern Directed Inference Systems,* eds D.A. Waterman & F. Hayes-Roth, Academic Press, New York *

Duda, R.O., Gaschnig, J.G. & Hart, P.E. 1979, "Model design in the PROSPECTOR consultation system for mineral exploration", in *Expert Systems in the Micro-Electronic Age*, ed. D. Michie, Edinburgh University Press, Edinburgh, Scotland *

Gevarter, W.B. 1987, "The nature and evaluation of commercial expert system building tools", *IEEE Computer*, May, pp. 24-41 *

Johnson, P.E. 1983, "What kind of expert should a system be?", *J. Med. Phil.*, vol. 8, pp. 77-97 *

Liu, N.K. & Dillon, T.S. 1987, "Detection of consistency and completeness in expert systems using Numerical Petri Nets", *Proc. Australian Artificial Intelligence Congress,* Sydney, Australia, November, pp. 170-85 *

Liu, N.K. & Dillon, T. 1991, "An approach towards the verification of expert systems using Numerical Petri Nets", *International Journal of Intelligent Systems,* vol. 6, no. 3, June, pp. 255-76 *

Luger, G.F. & Stubblefield, W.A. 1989, *Artificial Intelligence and the Design of Expert Systems*, Benjamin-Cummings, Redwood City, California *

Minsky, M. 1975, "A framework for representing knowledge", in *The Psychology of Computer Vision,* ed. P.H. Winston, McGraw-Hill, New York

Sell, P.S. 1985, *Expert Systems - A Practical Introduction,* Macmillan, Basingstoke, Hampshire
Walters, J. & Nielsen, N.R. 1988, *Crafting Knowledge-Based Systems*, John Wiley & Sons, New York

Winston, P.H. 1984, *Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts

Zadeh, L.A. 1988, "Fuzzy Logic", *IEEE Computer*, vol. 21, no. 4, April, pp.83-93

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on October 7, 2009 at 21:44 from IEEE Xplore. Restrictions apply.