

**School of Information Systems**

**A Framework for Active Software Engineering  
Ontology**

**Udsanee Pakdeetrakulwong**

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

**February 2017**

# Declaration

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.

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

Udsanee Pakdeetrakulwong

Udsanee Pakdeetrakulwong

20 February 2017

# Abstract

Software agent and multi-agent systems have attracted considerable attention and become active research areas in recent years. Furthermore, the advent of the Semantic Web technology has provided the underlying infrastructure that allows software agents to process data and perform sophisticated tasks on behalf of users. Consequently, the agent-based technology has become much more practical and the number of emerging real-world applications has increased, spanning a wide range of domains.

In this thesis, the multi-agent approach is utilised to address the issues associated with the passive structure of ontologies in terms of knowledge assimilation and knowledge dissemination. This research has developed a state-of-the-art framework for active ontology based on a multi-agent system. In this thesis, the Software Engineering Ontology is integrated with a multi-agent system to provide active support to software development teams to effectively manage and share software engineering knowledge and software project information when they are collaboratively working on software development projects. The framework makes three main contributions by offering: i) automated knowledge capture of software project information, ii) effective management of knowledge captured in the Software Engineering Ontology, and iii) active platforms for multi-site distributed software development environments. The framework has been realised through the prototype system as proof-of-concept experiments. The prototypes are evaluated based on existing case studies found in the literature. The evaluation is undertaken to assess the effectiveness and efficiency of the framework in assisting collaborative team members to manage and share software engineering knowledge relevant to various activities throughout the software development life cycle.

# Acknowledgements

I would like to express my deepest gratitude and appreciation to my supervisor, Dr. Ponnice Clark. I thank her for her consistent support and encouragement and for helping me to grow as a researcher. She dedicated much time to my thesis and always provided valuable feedback. I would not have been able to accomplish all that I did without her suggestions, ideas, and motivation. I would also like to thank my co-supervisor, Dr. Wendy Hui, for her guidance and encouragement. In addition, my appreciation extends to the other member of my thesis committee, Dr. Vidyasagar Potdar, for his kind support.

This thesis would not have been possible without the support and encouragement of my family. A special thanks to my wonderful husband, Suksawat Sae-Lim, who has always been beside me and has supported me in many ways throughout the journey. I also thank my lovely daughter, Natkrita Pakdeetrakulwong, who inspires me every day. Furthermore, many thanks to my lovely sisters for their ongoing love and caring.

In addition, I am grateful to the Royal Thai Government Scholarship Program and the Australia Awards–Endeavour Scholarships and Fellowships Program for the financial support provided for pursuing the PhD degree.

I also would like to express my sincere thanks to Ms. Bruna Pomella for proofreading this thesis. Finally, I would like to thank all my friends particularly Dang and Thoa's family, Koy and Mike Watkins, Emanuel, Ilham, Sara, Ernita, Novita, Yuni, Vy, Imran, Ariful, Nahid, Omid, Azadeh, Hassan, Rohini, Borak, Shahadat, and other fellow PhD candidates at Enterprise Unit 4 Technology Park with whom I have shared a memorable and enjoyable time. You have all contributed in some way to my development as a researcher and academic.

# **Dedication**

I dedicate this thesis to the loving memory of my late parents, Chokchai Pakdeetrakulwong and Ponnapa Pakdeetrakulwong.

# List of Publications

The following papers have been published either based on or related to the research presented in this thesis:

## Referred Journal Articles

Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, Waralak V. Siricharoen, and Naveed Khan. 2016. "An Ontology-Based Multi-Agent System for Active Software Engineering Ontology." *Mobile Networks and Applications* 21 (1): 65-88. Springer US. doi: 10.1007/s11036-016-0684-x. **(2015 Impact factor: 1.538; 2010 ERA-B)**

Pakdeetrakulwong, Udsanee, and Pornpit Wongthongtham. 2013. "State of the Art of a Multi-Agent Based Recommender System for Active Software Engineering Ontology." *International Journal of Digital Information and Wireless Communications (IJDIWC)* 3 (4): 363-376.

## Book Chapter

Wongthongtham, Pornpit, Udsanee Pakdeetrakulwong, and Syed Hassan Marzooq. 2017. "Ontology Annotation for Software Engineering Project Management in Multisite Distributed Software Development Environments." In *Software Project Management for Distributed Computing: Life-Cycle Methods for Developing Scalable and Reliable Tools*, ed. Zaigham Mahmood, 315-343. Cham: Springer International Publishing.

## Referred Conference Articles

Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, Suksawat Sae-Lim, and Hassan Marzooq Naqvi. 2016. "SEOMAS: An Ontology-Based Multi-Agent Approach for Capturing Semantics of Software Project Information." In *Proceedings of the 4th International Conference on Enterprise Systems*,

Melbourne, Victoria, Australia, 2-3 November 2016. 110-121. IEEE. doi: 10.1109/ES.2016.21. **(26% full paper acceptance rate)**

Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, and Naveed Khan. 2015. "An Ontology-Based Multi-Agent System to Support Requirements Traceability in Multi-Site Software Development Environment." In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, Volume II, Adelaide, South Australia, Australia, 28 September 2015 – 1 October 2015*. 96-100. 2811700: ACM. doi: 10.1145/2811681.2811700.

Pakdeetrakulwong, Udsanee, Wongthongtham Pornpit, and Waralak V. Siricharoen. 2014. "Recommendation Systems for Software Engineering: A Survey from Software Development Life Cycle Phase Perspective." In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for, London, United Kingdom, 8-10 December 2014*. 137-142. IEEE. doi: 10.1109/ICITST.2014.7038793.

Pakdeetrakulwong, Udsanee, and Pornpit Wongthongtham. 2015. "Use and Design of Ontology-Based Multi-Agent System for Multi-Site Software Development Environment." In *The 3rd Annual Conference on Engineering and Information Technology, Osaka, Japan, 22-24 March 2015*. 538-544.

Pakdeetrakulwong, Udsanee, and Pornpit Wongthongtham. 2013. "Towards Active Software Engineering Ontology." In *The International Conference on E-Technologies and Business on the Web (EBW2013), Bangkok, Thailand, 7-9 May 2013*. 206-211. The Society of Digital Information and Wireless Communication.

# Table of Contents

Declaration .....	ii
Abstract .....	iii
Acknowledgements .....	iv
Dedication .....	v
List of Publications .....	vi
Table of Contents .....	viii
List of Figures .....	xvi
List of Tables .....	xix
List of Abbreviations .....	xx
Chapter 1    Introduction .....	1
1.1        Introduction .....	1
1.2        Background and Signification of the Research Problem.....	3
1.2.1    Software Engineering Ontology and Its Passive Structure Problem.....	3
1.2.2    Problem in Multi-site Software Development Environment .....	5
1.3        Motivations of Research .....	8
1.3.1    Time Consuming and Expensive Approaches of Capturing Knowledge .....	9
1.3.2    Ineffectiveness to Obtain Knowledge Captured in the Ontology .....	10
1.3.3    Inefficient Communication.....	10
1.3.4    Lack of Real-time Awareness for Coordination Needed to Manage Work Dependencies .....	11
1.4        The Concerns that Need to be Addressed by a Framework for Active Software Engineering Ontology .....	12
1.4.1    Automated Knowledge Capture of Software Project Information.....	12
1.4.2    Effective Management of Knowledge Captured in the Software Engineering Ontology .....	13
1.4.3    Active Platforms for Multi-site Software Development Environments .....	14
1.5        Objectives of the Thesis .....	16
1.6        Thesis Structure .....	17
1.7        Conclusion.....	20
1.8        References .....	20
Chapter 2    Literature Review .....	26
2.1        Introduction .....	26
2.2        Ontology-based semantic annotation .....	27
2.2.1    Evaluation of Semantic Annotation of Software Project Information .....	29



2.3	Ontology-based Multi-agent Systems .....	30
2.3.1	Evaluation of Ontology-based Multi-agent Systems .....	38
2.4	Assistive Systems for Software Engineering.....	39
2.4.1	Requirement Gathering and Analysis .....	39
2.4.2	Software design .....	41
2.4.3	Software Implementation and Maintenance .....	42
2.4.4	Software Testing.....	45
2.4.5	Evaluation of Assistive Platforms in Software Engineering .....	48
2.5	Critical Evaluation of Existing Approaches: an Integrated View.....	49
2.5.1	Lack of effective approach to automate knowledge capture of software project information .....	50
2.5.2	Lack of Effective Management of Knowledge Captured in the Ontology.....	51
2.5.3	Lack of Active Platforms Available for Multi-site Software Development Environments .....	52
2.6	Conclusion.....	54
2.7	References .....	54
Chapter 3	Problem Definition .....	64
3.1	Introduction .....	64
3.2	Preliminary Concepts for Active Software Engineering Ontology.....	65
3.3	Passive Software Engineering Ontology Problems .....	66
3.3.1	Manually Capturing Software Project Information .....	66
3.3.2	Lack of Effective Management of the Software Engineering Ontology Instantiations .....	68
3.3.2.1	Knowledge Access and manipulation.....	68
3.3.2.2	Timely Awareness .....	68
3.3.3	Availability of Active Platforms for Multi-site Software Development Environments.....	69
3.3.3.1	Knowledge Management .....	70
3.3.3.2	Communication .....	70
3.3.3.3	Coordination.....	71
3.4	Underlying Research Issues .....	72
3.4.1	Research Issue 1: Automated Knowledge Capture of Software Project Information.....	72
3.4.2	Research Issue 2: Software Engineering Ontology Instantiations Management .....	73
3.4.3	Research Issue 3: Active Platforms for Multi-site Software Development Environments .....	74
3.5	Research Methodology.....	76
3.5.1	Overview of Design Science Research Paradigm.....	77
3.5.2	Choice of Design Science Research Framework .....	80

3.6	Conclusion.....	86
3.7	References .....	86
Chapter 4	Ontology-based Multi-agent Approach Solution Proposal .....	90
4.1	Introduction .....	90
4.2	Solution Requirements .....	91
4.2.1	Requirement 1: Requirement of Automated Knowledge Capture of Software Project Information .....	91
4.2.2	Requirement 2: Requirement of Software Engineering Ontology Instantiations Management.....	92
4.2.3	Requirement 3: Requirement of Active Platforms for Multi-site Software Development Environments .....	93
4.2.4	Requirement 4: Requirement of Framework Evaluation.....	93
4.3	Agent-Based Technology .....	94
4.3.1	Software Agent .....	94
4.3.2	Multi-Agent System.....	96
4.3.3	The integration of ontology and multi-agent systems .....	97
4.4	Ontology-Based Multi-Agent Systems Solution Proposal for a Framework for Active Software Engineering Ontology .....	99
4.4.1	Ontology-Based Multi-Agent Systems as a Solution for Automated Knowledge Capture of Software Project Information.....	100
4.4.2	Ontology-Based Multi-Agent System as a Solution for Software Engineering Ontology Instantiations Management.....	101
4.4.3	Ontology-based Multi-agent Systems as a Solution for Active Platforms for Multi-site Software Development Environments .....	102
4.5	Conclusion.....	104
4.6	References .....	104
Chapter 5	Conceptual Framework .....	108
5.1	Introduction .....	108
5.2	Overview of Existing Agent-Oriented Software Engineering Methodologies .....	109
5.2.1	GAIA .....	109
5.2.2	Agent Unified Modelling Language (AUML) .....	109
5.2.3	Multi-agent Systems Engineering (MaSE) and Organisation-based Multi-agent System Engineering (O-MaSE) .....	110
5.2.4	MAS-CommonKADS .....	110
5.2.5	MESSAGE .....	111
5.2.6	Process for Agent Societies Specification and Implementation (PASSI) .....	111
5.2.7	PROMETHEUS.....	112
5.2.8	The Agent-Oriented Development Methodology (ADEM) .....	112
5.3	Development Approaches .....	113

5.3.1	Macro-perspective .....	113
5.3.2	Micro-perspective .....	114
5.4	Active Software Engineering Ontology through a Multi-Agent System Engineering .....	115
5.4.1	System Requirements.....	115
5.4.2	Roles and Agent Types .....	116
5.4.3	Architecture Modelling .....	120
5.4.4	Structural design of the agent society .....	123
5.4.5	Agent Interoperations.....	124
5.5	Conclusion.....	127
5.6	References .....	127
Chapter 6	Ontology-based Multi-agent Approach for Capturing Software Project Information.....	131
6.1	Introduction .....	131
6.2	Ontology-based Multi-agent System to Capture Software Project Information.....	132
6.3	User Agent.....	134
6.3.1	Structure .....	134
6.3.2	Behaviours.....	136
6.3.2.1	Overview.....	136
6.3.2.2	ACL message generation behaviour .....	137
6.3.2.3	Output generation behaviour.....	138
6.3.3	Interactions .....	139
6.4	VersionControl Agent .....	140
6.4.1	Structure.....	140
6.4.2	Behaviours.....	141
6.4.2.1	Overview.....	141
6.4.2.2	VersionControlManager Behaviour.....	142
6.4.3	Interactions .....	143
6.5	Annotation Agent.....	144
6.5.1	Structure.....	144
6.5.2	Behaviours.....	146
6.5.2.1	Overview.....	146
6.5.2.2	Semantic Annotation behaviours.....	147
6.5.2.2.1	IdentifySourceCodeKeyConcepts behaviours .....	148
6.5.2.2.2	AnnotateSourceCode behaviours .....	149
6.5.3	Interactions .....	151
6.6	Ontology Agent .....	152
6.6.1	Structure.....	153

6.6.2	Behaviours.....	154
6.6.2.1	Overview.....	154
6.6.2.2	Ontology Population behaviour.....	154
6.6.3	Interactions .....	156
6.7	Implementation.....	157
6.8	Results .....	162
6.9	Practical uses .....	170
6.10	Discussion .....	179
6.11	Conclusion.....	181
6.12	References .....	181
Chapter 7	Ontology-based Multi-agent Approach for Software Engineering Ontology Instantiations Management .....	184
7.1	Introduction .....	184
7.2	Ontology-based Multi-agent Approach for Software Engineering Ontology Instantiations Management .....	185
7.3	User Agent.....	186
7.3.1	Structure.....	186
7.3.2	Behaviours.....	187
7.3.2.1	ACL message generation behaviour.....	188
7.3.2.2	Output Generation Behaviour .....	188
7.3.3	Interactions .....	189
7.4	Ontology Agent .....	191
7.4.1	Structure.....	191
7.4.2	Behaviours.....	192
7.4.2.1	Overview.....	192
7.4.2.2	Instance Knowledge Management Behaviours .....	193
7.4.2.2.1	QueryKnowledge behaviour .....	194
7.4.2.2.2	AddInstanceKnowledge behaviour .....	195
7.4.2.2.3	ModifyInstanceKnowledge behaviour .....	195
7.4.2.2.4	DeleteInstanceKnowledge behaviour.....	196
7.4.3	Interactions .....	198
7.5	Recommender Agent .....	200
7.5.1	Structure.....	200
7.5.2	Behaviours.....	201
7.5.2.1	Overview.....	201
7.5.2.2	Recommendation Management Behaviours.....	202
7.5.2.2.1	GenerateChangeImpactRecommendation behaviour.....	203
7.5.2.2.2	GenerateChangeImpactNotification behaviour.....	204
7.5.2.2.3	ManageMonitoring behaviour .....	204

7.5.3	Interactions .....	206
7.6	Practical Uses of the SEOMAS approach to Support Requirement Traceability.....	208
7.6.1	Requirements Interdependencies Modelling .....	208
7.6.2	Agent Capabilities.....	210
7.6.2.1	Manipulating a requirement captured in the Software Engineering Ontology .....	211
7.6.2.2	Recommend change impact on related requirements .....	211
7.6.2.3	Recommend change impact on other related software artefacts .....	212
7.6.2.4	Notify relevant team members about the impact of the requirement change.....	213
7.6.2.5	Generate traceability matrix.....	213
7.6.3	Case Study.....	213
7.6.3.1	Scenario 1 - Querying instance knowledge.....	214
7.6.3.2	Scenario 2 - Modifying instance knowledge.....	215
7.6.3.3	Scenario 3 - Adding new instance knowledge .....	218
7.7	Discussion .....	220
7.8	Conclusion.....	222
7.9	References .....	223
Chapter 8	Active Platforms for Multi-site Software Development Environments .....	226
8.1	Introduction .....	226
8.2	Platforms Framework.....	227
8.3	Knowledge Capture Platform .....	231
8.4	Query Platform .....	235
8.5	Monitoring Platform .....	237
8.6	Manipulation Platform .....	239
8.7	Practical Uses .....	241
8.7.1	Problem Analysis.....	241
8.7.2	Platform Uses .....	242
8.8	Discussion .....	245
8.8.1	Software Engineering Knowledge Management .....	246
8.8.1.1	Knowledge Capture .....	246
8.8.1.2	Knowledge Search.....	246
8.8.1.3	Knowledge Dissemination .....	247
8.8.1.4	Knowledge Maintenance .....	247
8.8.2	Knowledge Sharing and Reuse.....	248
8.8.3	Communication .....	248
8.8.4	Coordination.....	249
8.9	Conclusion.....	250

8.10	References .....	250
Chapter 9	Evaluation of the Framework for Active Software Engineering Ontology .....	252
9.1	Introduction .....	252
9.2	Framework Requirements .....	253
9.2.1	Automated Knowledge Capture of Software Project Information.....	253
9.2.2	Software Engineering Ontology Instantiations Management.....	254
9.2.3	Active Platforms for Multi-site Software Development Environments .....	255
9.3	Prototype Systems Evaluation .....	255
9.3.1	Evaluation of Automated Knowledge Capture of Software Project Information.....	256
9.3.2	Evaluation of Software Engineering Ontology Instantiations Management .....	263
9.3.3	Evaluation of Active Platforms for Multi-site Software Development Environments.....	270
9.4	Discussion of Results .....	275
9.4.1	Automated Knowledge Capture of Software Project Information.....	275
9.4.2	Effective Management of Software Engineering Ontology Instantiations.....	276
9.4.3	Active Platforms for Multi-site Software Development Environments .....	277
9.5	Conclusion.....	278
9.6	References .....	279
Chapter 10	Recapitulation and Future Work.....	281
10.1	Introduction .....	281
10.2	Recapitulation.....	282
10.3	Contribution of the Thesis .....	285
10.3.1	Contribution 1: Current State-of-the-art Research.....	285
10.3.2	Contribution 2: Conceptual Framework.....	286
10.3.3	Contribution 3: A Systematic Approach to Capture Semantics of Software Project Information .....	286
10.3.4	Contribution 4: A Systematic Approach to Manage Software Engineering Ontology Instantiations.....	288
10.3.5	Contribution 5: Active Platforms for Multi-site Software Development Environments.....	289
10.3.6	Contribution 6: Prototype Implementation and Evaluation.....	290
10.4	Future work .....	291
10.4.1	Future Work Focusing on the Framework Enhancement.....	291
10.4.2	Future Work on the Intelligent System Enhancement .....	293
10.5	Conclusion.....	295
10.6	References .....	295

Appendix A	Additional Information of the Case Study of an Online Shopping Software Development.....	298
------------	---	-----

# List of Figures

Figure 1-1: Overview of knowledge representation in the Software Engineering Ontology (Wongthongtham et al. 2009) .....	4
Figure 1-2: A framework of active Software Engineering Ontology.....	16
Figure 3-1: Design Science Research Methodology Process Model (Peffer et al. 2007, 54).....	81
Figure 5-1: Macro- and micro-perspective of AOSE (Zimmermann 2006).....	113
Figure 5-2: The active Software Engineering Ontology through a multi-agent system architecture .....	121
Figure 5-3: AUML class diagram of the macro-perspective of an agent society .....	123
Figure 5-4: Overall interactions among the agents .....	126
Figure 6-1: Micro-perspective of each agent type described in AUML models.....	132
Figure 6-2: AUML class diagram at implementation level of a user agent.....	136
Figure 6-3: Overview of user agent behaviours.....	137
Figure 6-4: The GenerateACLMessage behaviour details .....	138
Figure 6-5: Details of GenerateOutput behaviour.....	139
Figure 6-6: Agent interaction of a user agent .....	140
Figure 6-7: AUML class diagram at implementation level of the versioncontrol agent .....	141
Figure 6-8: Overview of the versioncontrol agent behaviours .....	142
Figure 6-9: Activities of ImportProjectInformation behaviour.....	143
Figure 6-10: Agent interaction of the versioncontrol agent.....	144
Figure 6-11 AUML class diagram at implementation level of the annotation agent .....	146
Figure 6-12: Behaviour overview diagram of the annotation agent.....	147
Figure 6-13: Details of IdentifySourceCodeKeyConcepts behaviour.....	148
Figure 6-14: Details of AnnotateSourceCode behaviour .....	151
Figure 6-15: Agent interactions of the annotation agent .....	152
Figure 6-16 AUML class diagram at implementation level of the ontology agent.....	153
Figure 6-17: Behaviour overview diagram of the ontology agent for the ontology population.....	154
Figure 6-18: Details of PopulateInstance behaviour .....	155
Figure 6-19: Agent interactions of the ontology agent.....	156
Figure 6-20: The automated knowledge capture by the SEOMAS approach.....	157
Figure 6-21: The content reference model in JADE (Caire and Cabanillas 2010) .....	159
Figure 6-22: An ACL message requesting to import a Java source code file into the version control repository.....	160
Figure 6-23: A Java source code file imported into the version control repository .....	161
Figure 6-24: The interactions among agents captured by a sniffer agent.....	161



Figure 6-25: Parsed Source code to be annotated with the Software .....	162
Figure 6-26: A Java source code being annotated with concepts and their relationships defined in the Software Engineering Ontology .....	164
Figure 6-27: Output of annotation from the BankAccount.java source code .....	166
Figure 6-28: Populated instances and their relations presented in Protégé tool.....	167
Figure 6-29: BankAccount Java class being interlinked with a metadata term to its relevant entity in DBpedia dataset .....	167
Figure 6-30: Information about Java class file from DBpedia .....	168
Figure 6-31: OntoGraf presentation of the BankAccount class instance.....	169
Figure 6-32: Protégé reasoner's logs for consistency checking of new populated instances .....	170
Figure 6-33: Vehicle registration class diagram in a multi-site development environment.....	172
Figure 6-34: Agents in the SEOMAS platform .....	174
Figure 6-35: Existing bug reported to the Car class.....	175
Figure 6-36: Recommend an expert or a potential fixer for a new bug report.....	176
Figure 6-37: The potential fixer's name attached in the bug report.....	176
Figure 6-38: A message to notify a potential bug fixer or an expert about a new bug.....	176
Figure 6-39: Related information about the Car class.....	178
Figure 6-40: A message to the bug reporter giving a notice of the bug status.....	179
Figure 7-1: AUML class diagram at implementation level of a user agent.....	187
Figure 7-2: The GenerateACLMessage behaviour detail.....	188
Figure 7-3: The GenerateACLMessage behaviour detail.....	189
Figure 7-4: Agent interactions of a user agent.....	190
Figure 7-5: AUML class diagram at implementation level of the ontology agent.....	192
Figure 7-6: Behaviour overview diagram of the ontology agent .....	193
Figure 7-7: Activities of QueryKnowledge behaviour .....	194
Figure 7-8: Activities of AddInstanceKnowledge behaviour .....	195
Figure 7-9: Activities of ModifyInstanceKnowledge behaviour .....	196
Figure 7-10: Activities of DeleteInstanceKnowledge behaviour.....	197
Figure 7-11: Interactions among agent types of the ontology agent .....	199
Figure 7-12: AUML class diagram at implementation level of the recommender agent .....	201
Figure 7-13: Behaviour overview diagram of the recommender agent.....	202
Figure 7-14: Activities of GenerateChangeImpactRecommendation behaviour .....	203
Figure 7-15: Activities of GenerateChangeImpactNotification behaviour.....	204
Figure 7-16: Activities of ManageMonitoring behaviour .....	205
Figure 7-17: Interactions among agent types of the recommender agent.....	207
Figure 7-18: Types of potentially affected requirements .....	211
Figure 7-19: New R is added to existing requirement R <sub>2</sub> .....	212

Figure 7-20: Types of potential affected artefacts .....	213
Figure 7-21: Agents in the SEOMAS platform .....	214
Figure 7-22: Excerpt of a traceability matrix.....	215
Figure 7-23: An ACL message requesting to modify the Requirement FR01.....	216
Figure 7-24: Recommendation of potentially affected artefacts.....	217
Figure 7-25: A message to confirm the modification of the requirement FR01 .....	217
Figure 7-26: Messages to notify the authors of potentially affected artefacts .....	218
Figure 7-27: A new added instantiation of a Requirements class .....	220
Figure 8-1: Software Engineering knowledge management and sharing infrastructure .....	227
Figure 8-2: Four levels of Software Engineering Ontology-based multi-agent system knowledge management and knowledge sharing platforms .....	229
Figure 8-3: A flow of the processes when utilising the platforms .....	231
Figure 8-4: Alex user agent requests to import a Java source code file into the version control repository .....	232
Figure 8-5: Result of annotation from the Employee Java source code .....	234
Figure 8-6: Excerpt of querying test report of the requirement FR03.....	235
Figure 8-7: Use cases related to a use case UC9.....	236
Figure 8-8: Result of use cases and test cases required to be tested with a use case UC9 .....	236
Figure 8-9: Message to give notice of requirement testing coverage.....	239
Figure 8-10: Recommendation of potentially affected artefacts.....	240
Figure 8-11: Messages to notify the authors of potentially affected artefacts .....	240
Figure 8-12: Use case and test case fragment of traceability (Leffingwell and Widrig 2000, 354) .....	242
Figure 8-13: Notification about missing requirement information .....	243
Figure 8-14: Excerpt of querying traceability matrix with missing use cases and test cases .....	244
Figure 8-15: Excerpt of querying traceability matrix after adding missing use cases and test cases .....	244
Figure 8-16: Message to notify a tester regarding FR06 is updated .....	245
Figure 9-1: Use case and test case fragment of traceability .....	270

# List of Tables

Table 2-1: Review of some existing ontology-based multi-agent systems.....	34
Table 2-2: Summary of reviewed assistive systems according to software development activities .....	45
Table 3-1: Set of activities of DSRM process mapped with the thesis chapters.....	85
Table 4-1: Differences between traditional software applications and software agents (adapted from Turban, Sharda and Delen 2010).....	95
Table 5-1: The mapping of system requirements to agents' roles and their associated agent types.....	118
Table 6-1: Content slots for semantic annotation request .....	160
Table 7-1: Dahlstedt's Interdependency model .....	209
Table 7-2: Change impact rules adapted from (Göknil, Kurtev and van den Berg 2008).....	210
Table 9-1: Bug resolution process described in (Wongthongtham, Dillon and Chang 2011) .....	257
Table 9-2: Bug resolution process with supporting from the SEOMAS approach.....	258
Table 9-3: Requirement change process described in (Lai and Ali 2013, 46-51).....	264
Table 9-4: Requirement change process with support from the SEOMAS approach.....	265
Table 9-5: Scenarios for investigating missing project information (adapted from Leffingwell and Widrig 2000, 354-356) .....	270
Table 9-6: Proactively monitoring of project information by the SEOMAS platforms .....	271

# List of Abbreviations

ACL	Agent Communication Language
ADEM	Agent-Oriented Development Methodology
AOSE	Agent-Oriented Software Engineering
AST	Abstract Syntax Tree
AUML	Agent Unified Modelling Language
DC	Dublin Core
FOAF	Friend-of-a-Friend
FIPA	Foundation for Intelligent Physical Agents
JADE	Java Agent Development Framework
KQML	Knowledge Query and Manipulation Language
MAS	Multi-agent System
MaSE	Multi-agent Systems Engineering
O-MaSE	Organisation-based Multi-agent Systems Engineering
OWL	Web Ontology Language
PASSI	Process for Agent Societies Specification and Implementation
RDF	Resource Description Framework
SEOMAS	Software Engineering Ontology through a Multi-Agent System

SE Ontology	Software Engineering Ontology
SIOC	Semantically-Interlinked Online Communities
SKOS	Simple Knowledge Organisation System
UML	Unified Modelling Language

# Chapter 1 Introduction

## 1.1 Introduction

The evolution of Web technologies began with Web 1.0 which is considered as the traditional document-centric Web (Sheth and Thirunarayan 2013). Then it moved to Web 2.0, focusing on user-generated contents or community-oriented information gathering. However, the substantial amount of data and unstructured content that are generated make it difficult for users to efficiently search web contents. Therefore, Web 3.0, also known as Semantic Web, was developed to alleviate this problem. Berners-Lee (1999) stated that the real power of the Semantic Web is realised when the Web content is understandable and processable by computer. The underlying structure of the Semantic Web is that data are given structure and well-defined meaning so that they can enable software agents to understand contents on the Web and process the information to carry out sophisticated tasks for users (Chu and Yang 2012). A software agent is a computer system situated in some environment and has the ability to perform autonomous actions in order to achieve its desired objective (Wooldridge 2009). With the Semantic Web technology, agents can be more effective and efficient in discovering and retrieving of knowledge because information is semantically annotated and can be understood by them. In other words, agents can act as autonomous software entities that can assist their users through the automation of tasks such as information discovery, information integration, and services monitoring with minimum human involvement.

The agent-based technology has become much more practical and has attracted considerable attention in recent years. Although an agent can work as a stand-alone entity to perform a particular task on behalf of a user, many of the agent-based applications are operated in environments that contain multiple agents collaboratively working together as a group, otherwise known as a multi-agent system. Multi-agent systems offer various advantages compared with a single agent, such as reliability and robustness, modularity, scalability, adaptability, concurrency,

parallelism, and dynamism (Elamy 2005). They are employed in several real-world applications, spanning a wide range of domains such as e-learning, healthcare, web-services, supply chain management, etc.

In this thesis, a multi-agent approach is utilised to address the issues associated with the passive structure of ontologies in terms of knowledge assimilation and knowledge dissemination. This research has developed a state-of-the-art framework for active ontology based on a multi-agent system. In this thesis, the Software Engineering Ontology, the ontology designed for multi-site distributed software development, is integrated with a multi-agent system to provide active support to team members by giving them access to software engineering knowledge when they are working in software development projects. The framework makes three main contributions by offering: i) automated knowledge capture of software project information, ii) effective management of knowledge captured in the Software Engineering Ontology, and iii) active platforms for multi-site software development environments. The framework has been realised through the prototype system as proof-of-concept experiments. The prototypes are evaluated based on existing case studies found in the literature. The evaluation is undertaken to assess the effectiveness and efficiency of the framework in assisting collaborative team members to manage and share software engineering knowledge throughout various software development activities in the software life cycle.

To this end, the next section introduces and describes the Software Engineering Ontology and its passive structure, and provides an overview of multi-site distributed software development. It then explains the need for a framework to make the ontology active, and the concerns that need to be addressed by the proposed framework. The research objectives are stated and the chapter concludes with an outline of the thesis structure.

## **1.2 Background and Signification of the Research Problem**

In this section, the Software Engineering Ontology is introduced. This is followed by an explanation of the issues associated with its passive structure. The section concludes with a brief overview of a multi-site software development environment including its benefits and challenges.

### **1.2.1 Software Engineering Ontology and Its Passive Structure Problem**

The Software Engineering Ontology (SE Ontology) (Wongthongtham et al. 2009; Wongthongtham et al. 2005) was first developed to facilitate efficient collaboration among software development team members who are geographically distributed. It is considered to be an effective means of clarifying concepts and project information and enabling knowledge-sharing among team members. It represents software engineering knowledge and concepts, software development methodologies, software tools and techniques. The software engineering knowledge captured in the Software Engineering Ontology comes from two sources, namely, the Software Engineering Body of Knowledge (SWEBOK), an international standard describing generally accepted knowledge about software engineering (Abran et al. 2004), and the software engineering textbook of Ian Sommerville (Sommerville 2004).

The Software Engineering Ontology comprises two sub-ontologies: the generic ontology and the application-specific ontology. The generic ontology contains concepts and relationships annotating the whole set of software engineering concepts (e.g., the vocabulary, the semantic interconnections, and logic for the software development) which are captured as domain knowledge. The application-specific ontology is an explicit specification of software engineering for a particular software development project, and is defined as sub-domain knowledge. For instance, if a software development project is implemented using object-oriented paradigm, the concepts of object-oriented development (e.g., use cases, class diagrams) are included in the application-specific ontology. However, the concept of a data flow diagram may not be necessarily included (Wongthongtham et al. 2009).



Additionally, in each project, software project information including the actual project data, project agreement, and project understanding which is specifically for a particular project need, is captured as instance knowledge. They are related to each other according to the specific relations between the concepts. Put differently, once the Software Engineering Ontology has been developed, it is populated with software project information as the ontology instances (or instances of concepts). The population process is generally achieved by mapping software project information to the concepts described in the ontology. When the mappings have been completed, software project information is conceptualised and is in a semantically rich as well as machine-readable form.

The Software Engineering Ontology can facilitate common understanding and consistent communication among distributed software project teams by allowing them to access shared software engineering knowledge and to query the semantically-linked project information. Figure 1-1 provides an overview of knowledge representation in the Software Engineering Ontology.

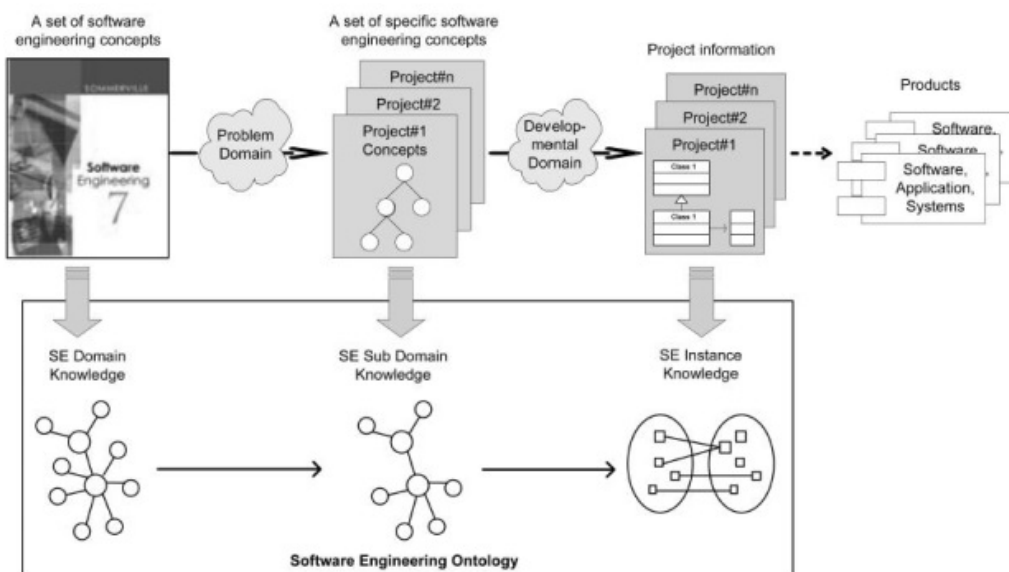


Figure 1-1: Overview of knowledge representation in the Software Engineering Ontology (Wongthongtham et al. 2009)

However, the Software Engineering Ontology has the same passive structure as that of other ontologies. After the ontology has been developed and used in the ontology deployment phase, its passive structure gives rise to two main challenges regarding knowledge assimilation and knowledge dissemination. Knowledge

assimilation is related to the process of capturing and representing the domain-specific knowledge in a formal conceptual model, while knowledge dissemination is the process of delivering the knowledge to different types of applications or communicating the knowledge to users (Forbes 2013).

In terms of knowledge assimilation, because of the passive structure of the Software Engineering Ontology, a software team member manually extracts software engineering knowledge from the software project information and maps it to the concepts defined in the Software Engineering Ontology. This manual approach to capturing knowledge requires a great deal of time and effort on the user's part.

In terms of knowledge dissemination, ontology users are required to explicitly request the information that they need. Moreover, given the passive structure of the ontology, the users need to know exactly the concepts and relationships to which they are referring. Otherwise, they may not be able to obtain or manipulate the knowledge captured in the ontology. However, users sometimes have some knowledge of an issue, rather than knowing precisely which concepts and relationships are defined in the ontology. As a result, they may not be able to obtain the knowledge that they require.

It is to be noted that in this thesis, the focus is on the application-specific ontology including the software project information captured in it as instance knowledge. Changes of domain knowledge in the generic ontology which introduces new concepts and changes in the conceptualisation are out of the scope of this thesis.

### **1.2.2 Problem in Multi-site Software Development Environment**

The Software Engineering Ontology has been developed to assist team members who are working in multi-site software development environments. However, with its passive structure, it has several shortcomings. Hence, the need for improvement has provided the motivation for this research. Accordingly, in this section, a brief overview is given of a multi-site software development environment including its benefits and challenges.

Multi-site distributed software development projects take place in an

environment where project teams are dispersed across multiple sites. This type of software development setting is also known as global software development. A multi-site software development project offers several strategic and economic benefits (Jiménez, Piattini and Vizcaíno 2009; Ågerfalk et al. 2008). For example, in many countries such as China, India, Vietnam, and Philippines, a large labour pool of competent IT professionals is available at reasonably low wages. Therefore, this can help to decrease software development costs. Furthermore, a ‘follow-the-sun’ development model in the global software development context provides a virtual 24-hour workday to maximise productivity and reduce development time. This is done by one team, at the end of the workday, passing its tasks to another team at a different development site in a different time zone. Moreover, this provides access to local opportunities such as the market and customers. These advantages have made multi-site distributed software development an attractive approach for software companies, becoming a prevalent trend in recent years (Jain and Suman 2015).

Nonetheless, reports in the literature indicate that not all global software development projects benefit from working in a distributed environment. The International Conference on Global Software Engineering (ICGSE) 2015 industry panel stated that 20-25% of outsourcing contracts fail within the first two years and 50% fail within five years (Ebert, Kuhrmann and Prikladnicki 2016; Ebert 2011). Software industries should be concerned about the challenges of working across multiple sites as it may require additional effort to overcome potential problems. The challenges arising from geographical, temporal, and socio-cultural distances in multi-site distributed software development are summarised below.

- Lack of effective and efficient communication

In traditional co-located software development, informal communication plays a significant role in coordination activities (Holmstrom et al. 2006). However, because of geographical distances and time-zone differences, communication among distributed teams is usually in electronic form and asynchronous mode with limited opportunities for informal communication or face-to-face communication (Noll, Beecham and Richardson 2011). These factors can reduce the frequency and richness of communication among remote teams. Moreover, there may be ambiguity in

the written communications that can lead to misunderstanding or misinterpretation.

- Misinterpretation or misunderstanding resulting from the diversity of languages and cultures

Development teams are often comprised of members from different nations. The diversity of languages, cultures, and education backgrounds are major factors that have a significant effect on the way in which development teams interpret a certain situation and respond to it. Moreover, different levels of language proficiency can result in misunderstandings or difficulty in following a discussion (Philip, Schwabe and Ewusi-Mensah 2009).

- Lack of effective and efficient coordination

Coordination in software development refers to the act or action of orchestrating each development task in order to contribute to the overall objective of a software project (Lanubile 2009). Coordination problems arise as a result of insufficient communication due to geographical distance. When team members are geographically dispersed, they have little knowledge of what other members at different sites are doing (Boden 2011). Therefore, they may not be aware of what and when they are required to coordinate or to manage work dependencies. Coordination issues also make it difficult to keep track of the evolution of software due to changes.

In the literature, several studies have found that lack of coordination among team members can create unwanted side effects such as prolonged task resolution time, increase in software defects, and duplication of tasks (Cataldo and Herbsleb 2013; Souza and Redmiles 2008). These issues can affect software productivity or the quality of the final product. For example, when there is a change in requirements, this should be made known to relevant team members in a timely manner so that they can be aware of the change and take appropriate action to communicate and coordinate with others.

- Lack of shared understanding and knowledge sharing

Software development is a collaborative and knowledge-intensive activity. One of the main critical factors contributing to its success is shared understanding and effective knowledge sharing (Zahedi, Shahin and Ali Babar 2016). Unfortunately, geographical distance means physical separation, temporal distance, and language and cultural barriers that impede the shared understanding and knowledge sharing. Therefore, in order to address these issues, the implicit knowledge residing in each individual team member must be made explicit and accessible to other project members who require it.

### **1.3 Motivations of Research**

Although the Software Engineering Ontology has been developed to clarify software engineering concepts and software project information, and to enable knowledge sharing among distributed team members, its passive structure has raised several concerns that have motivated this research. These concerns are stated as follows.

- The current approaches for capturing knowledge are time consuming and expensive.
- The process of obtaining knowledge captured in the ontology is ineffective.
- Communication is inefficient.
- There is a lack of real-time awareness when coordination is needed to manage software dependencies.

### **1.3.1 Time Consuming and Expensive Approaches of Capturing Knowledge**

Software development generates a large amount of software project information, e.g., project data, project agreement, project understanding, etc. Software artefacts are also part of project data produced throughout the software life development cycle. These artefacts include: software requirement specifications from the requirement gathering phase, UML diagrams from the software design phase, source code from the implementation phase, and test plans from the testing phase, etc. However, this software project information is in syntactic form so the structures are not conducive to an understanding of the semantics, and thus may create ambiguities (e.g. misunderstanding or misinterpretation) (Panagiotou, Paraskevopoulos and Mentzas 2011). This issue is particularly significant in a multi-site software development context where project members are geographically dispersed and they have less face-to-face contact. In addition, information related to the software project is distributed across various software repositories and the interconnections among these software repositories are typically not explicit (Iqbal et al. 2009). In other words, there is no link among software repositories; therefore, the relevant software project information is not easily and readily accessible.

The Software Engineering Ontology was developed to define common sharable software engineering knowledge and to enable knowledge integration among relevant software artefacts. Project team members can use it to capture software development knowledge by mapping software project information to the concepts defined in the Software Engineering Ontology. Once the software project information has been mapped, it is conceptualised and semantically linked so that it can be used to clarify any ambiguity in communication and to enable knowledge integration and sharing among software project development teams. However, because of its passive structure, one of the main challenges is that in order to capture software project information, project team members mostly do it manually. There is a lack of an effective approach that can help them to automate the knowledge-capturing task. Given the considerable volume of software project information produced, the knowledge-capturing approach that relies primarily on software teams' manual processing is not practical because it is time-consuming, laborious, tedious, and error-prone. Moreover, because the manual approach requires a great amount of effort from team members, it could discourage them from sharing their knowledge

with others.

### **1.3.2 Ineffectiveness to Obtain Knowledge Captured in the Ontology**

The Software Engineering Ontology is a comprehensive ontology covering all the aspect of software engineering. It consists of a large set of software engineering concepts, their relations and their constraints. Software project information is also captured in the ontology to allow team members to query the semantically-linked project development information to facilitate their tasks. In order to obtain the knowledge captured in the ontology, project team members are required to explicitly request and specify the concepts and relations to which they are referring; otherwise, they might not be able to obtain the knowledge needed. However, it could happen that a user might not be aware of certain relevant knowledge that exists in the ontology (e.g., new knowledge that has just recently been captured, a new member who has just joined the project team). Hence, some useful knowledge may be overlooked. In other words, given the passive structure of the Software Engineering Ontology, there is a lack of an effective approach that can proactively deliver useful and relevant knowledge to project team members.

### **1.3.3 Inefficient Communication**

In a collaborative software development environment, project teams require effective and efficient communication in order to coordinate their work. This is particularly important in a multi-site distributed software development setting in order to overcome the barriers imposed by long distance and different time zones (Alqhtani and Qureshi 2014). Even though the Software Engineering Ontology can facilitate effective communication among team members in terms of providing shared understanding of software development information to overcome the issues regarding misinterpretations, misunderstandings, and miscommunications, some aspects of efficient communication are still missing and need to be improved. The communication is not efficient in the sense that information cannot be targeted or directed to every team member who needs to know about it in a timely manner. When an issue arises, a team member mostly depends on the traditional means of

communication (e.g., phone calls, emails, online chats, etc.) which are not very conducive to semantic understanding. Effective and efficient communication is considered critical for the success of a software project (Purna Sudhakar 2012; Lind and Culler 2013). If the communication is not effective and efficient enough to enable software team members to be kept well-informed of the project's progress, there is a higher probability of challenges resulting from different levels of anticipation.

#### **1.3.4 Lack of Real-time Awareness for Coordination Needed to Manage Work Dependencies**

The development of a large-scale software project is complex and requires the whole software system to be decomposed into smaller modules. Because the software modules need to interact with each other, this creates work dependencies, and thus produces the need for coordination among software development teams (Oliva and Gerosa 2012). Coordination become more complex as the degree of distribution of team members increases and it can lead to a lack of team awareness. If project team members are not aware of coordination needs in order to manage work dependencies in a timely manner, software quality and software development productivity might be affected, resulting in duplication of tasks, software defects, and additional effort to rectify the problems (Cataldo and Herbsleb 2013). Although several existing methods and tools have been proposed to help project team members to be aware of when coordination is needed in order to manage work dependencies, they do not provide real-time awareness to support efficient coordination (Blincoe, Valetto and Damian 2015). Real-time awareness is critical because the awareness will be valuable if it occurs when the information is useful.

Several software development issues arise from a lack of real-time awareness of the need for coordination. Software development projects involve various work dependencies and linkages which need information about others' activities and their coordination (Cataldo et. al. 2009). These dependencies result in critical challenges if they are not managed properly in particular during software maintenance and software evolution. Changes are inevitable and can occur at any stage in the software development life cycle. There are different types of changes such as changes in



users' requirements, changes in the system's environment, or ongoing maintenance to correct failures. With every type of change, the overall quality, schedule, and cost of a software project are affected if the change is not well-managed. For example, when there is a change in a software artefact, it generally impacts on other artefacts. Although several change impact analysis techniques (e.g., Gupta, Tripathi and Kuswaha 2015; Shahid and Ibrahim 2016) are applied or the Software Engineering Ontology is in use, these can assist project teams only to a certain extent. They do not provide real-time awareness of the change and need for coordination to other team members. As a consequence, inconsistencies among software artefacts can result because relevant members might not be aware of the change and do not coordinate to notify others of the change within appropriate time constraints. This may lead to software defects that compromise the quality of a software system (Pete and Balasubramaniam 2015).

## **1.4 The Concerns that Need to be Addressed by a Framework for Active Software Engineering Ontology**

In this section, there are three main concerns that need to be addressed by a framework for active Software Engineering Ontology. They are identified as follows.

- Automated Knowledge Capture of Software Project Information
- Effective Management of Knowledge Captured in the Software Engineering Ontology
- Active Platforms for Multi-site Distributed Software Development Environments

### **1.4.1 Automated Knowledge Capture of Software Project Information**

Various types of software project information that are produced throughout the software development life cycle describe different levels of abstraction and

perspectives of a software system. However, they are in syntactic format that does not facilitate the understanding of the concepts or meaning (Panagiotou and Mentzas 2009). The syntactic representation of software project information produces several issues such as ambiguities, difficulty in data integration, limitation of information retrieval, etc. These problems are more significant in a multi-site software development environment where project team members are dispersed across several locations and face-to-face communication (e.g., formal or information meeting) is limited. As a result, software project information should be transformed into semantic representation in order to alleviate the aforementioned issues. Some existing approaches have been introduced to capture the semantics of a software project (e.g., Qiang, Ming and Zhiguang 2008; Zygmotiotis, Dranidis and Kourtesis 2009). However, many of them are based on manual approaches or require effort from project team members to carry out additional steps in the knowledge capturing process because they are not integrated in a software development process. The manual capturing of knowledge of software project information is time-consuming, labour-intensive, tedious and error-prone task. In order to tackle these issues, *there is the need for a systematic approach that can automatically capture knowledge of software project information and that is seamlessly integrated in a software development process*. Once this information has been captured and conceptualised, it can be semantically interlinked with other relevant information. It can then be used to clarify any ambiguity in communication and to enable knowledge sharing among team members. This knowledge is also in machine-readable format which means that it can be understood by software agents. As a result, the agents can make use of this knowledge to assist project teams with their software development activities such as managing project issues, monitoring software project status, suggesting solutions or experts.

#### **1.4.2 Effective Management of Knowledge Captured in the Software Engineering Ontology**

Once software project information has been captured in the ontology repository, this knowledge has to be managed effectively. The management of this knowledge includes various operations, namely, retrieving, adding, modifying, and deleting. In order to obtain the knowledge captured in the Software Engineering

Ontology, project team members need to explicitly request it and know exactly the concepts and relationships to which they are referring. However, it is often the case that a person who utilises the ontology may try to resolve an issue but he/she cannot translate it into the exact concepts and relations contained in the ontology. Furthermore, because of the considerable amount of knowledge captured in the ontology, it is possible that software teams might not be aware of the existence of certain knowledge. This could lead to valuable knowledge being overlooked.

Furthermore, during software development, software changes are inevitable in all stages of a software project (Basri et al. 2016). Software project information is always evolving, often making it difficult to manage dependencies that exist between software artefacts (e.g., maintaining consistencies). Therefore, it is important to have an effective management of knowledge captured in the ontology to reflect the software change that can support real-time awareness of the need for coordination. Relevant team members should be informed about the change and its impact in a timely manner so that they can coordinate to accommodate the change within a proper time frame.

Accordingly, there is a need to *develop a systematic approach that can provide active support to help software teams obtain and manipulate software engineering knowledge captured in the Software Engineering Ontology effectively*. This active support is able to deliver knowledge that is potentially useful to software teams even without receiving an explicit request. Moreover, it can assist team members to maintain real-time awareness for effective and efficient coordination in order to manage work dependencies.

### **1.4.3 Active Platforms for Multi-site Software Development Environments**

Software development is a knowledge- and collaborative-intensive process the success of which mostly depends on project team members effectively managing and sharing software engineering knowledge through efficient and timely collaboration and interaction. This is particularly critical in a multi-site software development setting where a high degree of collaboration and knowledge sharing is required (Shiva et al. 2009). Software development activities are interconnected, and

team members need support with software engineering knowledge throughout various phases of a software life cycle. As a result, *there is a need to have platforms for multi-site software development environments that can support remote team members to effectively manage and share software engineering knowledge when they are engaged in software development activities throughout the software life cycle.* The active platforms are intended to assist remote teams to collaborate through effective and efficient communication and coordination in order to minimise the challenges related to physical and temporal distance. The active platforms mentioned in this thesis are similar to the cutting-edge inventions such as Amazon.com or Youtube.com. They do not only make information available to users, and passively rely on them to pull the information needed; these websites also proactively deliver useful information to their users. Likewise, the platforms are intended to provide software engineering knowledge to project team members in a flexible manner either by information-delivery push or information-delivery pull mode. In other words, the platforms can reactively provide knowledge in response to a user's request or proactively deliver knowledge that can assist team members with their tasks.

In summary, Figure 1-2 describes the framework for active Software Engineering Ontology. They are intended to cover both knowledge assimilation and knowledge dissemination phases. The active support can assist software team members in terms of automating knowledge capturing process that is transparently integrated into daily software development activities (e.g., integrated with the process of importing software artefacts into version control repositories). If the knowledge captured in the ontology is manipulated to reflect a software change, the active support proactively provides real-time awareness of coordination needs, or other useful information that is relevant to team members' tasks (e.g., change impact analysis, expert identification). Furthermore, the active support includes proactive monitoring of software project information to identify any possibility of encountering deviation before an actual issue occurs. Notifications are provided to corresponding team members on a push-based delivery mode without receiving an explicit request.

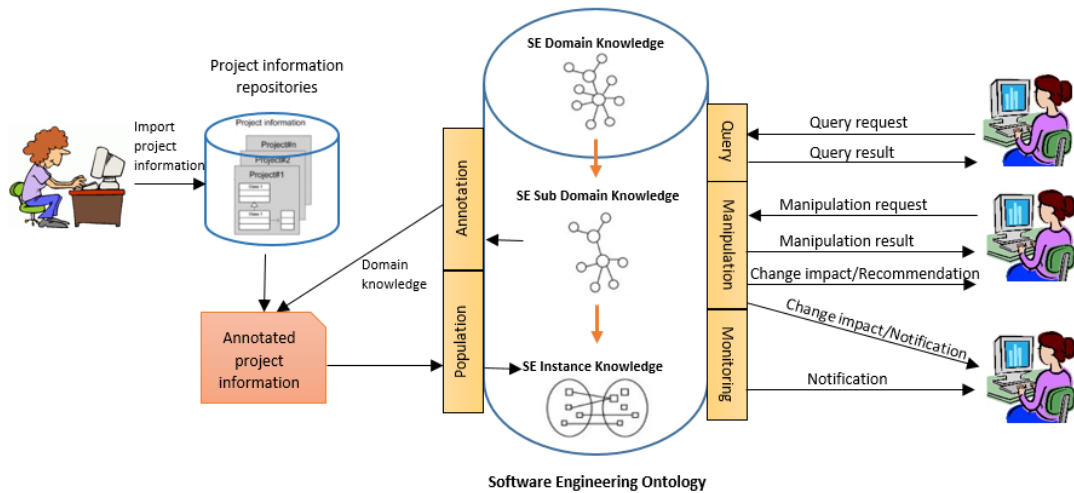


Figure 1-2: A framework of active Software Engineering Ontology

## 1.5 Objectives of the Thesis

In the previous sections, the motivations and the concerns that need to be addressed for a framework of active Software Engineering Ontology are outlined. Three major concerns have been identified and are the basis for the research objectives of this thesis. The primary objective of this thesis is to *develop a framework for active Software Engineering Ontology that can be used to provide active support to assist software development team members with software engineering knowledge when they are working on software development projects.* The research objective can be segmented into the following sub-objectives:

Sub-objective 1: To develop a framework for active Software Engineering Ontology specifically focusing on the ontology deployment phase. The framework is intended to address challenges resulted from the passive structure of the Software Engineering Ontology in regard to knowledge assimilation and knowledge dissemination.

Sub-objective 2: To develop an approach to automate knowledge capture of software project information that is seamlessly integrated into the software development process.

Sub-objective 3: To develop an approach to access and manage software engineering knowledge captured in the Software Engineering Ontology effectively.

Sub-objective 4: To develop active platforms for multi-site software development environments that can assist remote project team members to manage and share software engineering knowledge throughout the software development life cycle.

Sub-objective 5: To evaluate the effectiveness and efficiency of the proposed framework and platforms based on existing case studies found in the literature through the prototype system used as proof-of-concept experiments.

## **1.6 Thesis Structure**

The thesis is structured as follows.

### **Chapter 1: Introduction (current chapter)**

This chapter introduces issues arising from the passive structure of the Software Engineering Ontology. It discusses the motivations for this study and the concerns that need to be addressed by a framework for active Software Engineering Ontology. Furthermore, the objective and sub-objectives of this research are stated and the structure of this thesis is briefly described.

### **Chapter 2: Literature Review**

This chapter provides the survey of the literature related to ontology-based semantic annotation, ontology-based multi-agent systems and assistive systems in software engineering. The literature related to each area is reviewed and evaluated. At the end of the chapter, a critical evaluation using an integrated view is discussed.

### **Chapter 3: Problem Definition**

The first section of this chapter presents the key concepts and definitions that are used in this thesis. An overview of the problems is provided and the underlying

research issues related to these problems are discussed. At the end of the chapter, a summary of research approaches is given. A design science research methodology is chosen as the preferred option to address the research issues and for the development of the proposed solution.

#### **Chapter 4: Ontology-based Multi-agent Approach Solution Proposal**

This chapter begins with an overview of the key solution requirements for the framework solution development. The scientific approach based on ontology-based multi-agent systems is investigated and used as a solution for the identified research issues.

#### **Chapter 5: Conceptual Framework for active Software Engineering Ontology**

This chapter presents a brief overview of well-known existing agent-oriented software engineering methodologies. The integration of the macro-perspective and micro-perspective of agent-oriented software engineering as well as the AUML methodology are used to implement the framework solution. Then, the analysis and design phases of the macro-perspective are discussed in detail to derive the overall solution including the structure of an agent society and its dynamic interactions. They result in the agent specifications for each agent type. These specifications are refined in Chapter 6 and Chapter 7 according to each solution proposal.

#### **Chapter 6: Ontology-based Multi-agent Approach for Capturing Software Project Information**

This chapter provides a detailed description of the ontology-based multi-agent approach for capturing software project information. The approach consists of two main processes: semantic annotation and ontology population. The agents that are involved in these processes are discussed in detail in respect to structure features, behaviours, and inter-agent interactions. The practical use of this approach as a means of assisting collaborative project team members to address software development issues is provided and discussed.

## **Chapter 7: Ontology-Based Multi-agent Approach for Software Engineering Ontology Instantiations Management**

This chapter presents a detailed description of the ontology-based multi-agent approach to manage Software Engineering Ontology instantiations. The agents involved in managing the knowledge captured in the Software Engineering Ontology are discussed in detail in respect to structure features, behaviours, and inter-agent interactions. The practical use of the approach as a means of assisting project team members to coordinate their work is provided and discussed.

## **Chapter 8: Active Platforms for Multi-site Software Development Environment**

This chapter presents active platforms for multi-site software development environments that are intended to assist distributed project teams to effectively manage and share software engineering knowledge throughout the various phases of the software development life cycle. Four platforms, namely, semantic annotation, knowledge query, monitoring, and knowledge manipulation, are demonstrated along with their practical uses. The chapter is concluded with a discussion of the ways that the platforms tackle the issues identified in Chapter 3.

## **Chapter 9: Evaluation of the Framework for Active Software Engineering Ontology**

This chapter demonstrates prototypes as proof-of-concept experiments. Several scenario experiments based on existing case studies found in the literature are carried out to evaluate the effectiveness and efficiency of the framework. The chapter is concluded with a discussion of the evaluation results in an integrated view according to the framework solution requirements.

## **Chapter 10: Recapitulation and Future Work**

This chapter concludes the thesis by summarising the research that has been carried out and the contributions that it has made to this field of study. Moreover, it offers several suggestions for future research that may extend the proposed



framework developed in this thesis.

## 1.7 Conclusion

In this chapter, the passive structure of the Software Engineering Ontology and its issues are discussed. These have led to the motivations for this study. Several major concerns that need to be addressed by a framework for active Software Engineering Ontology are highlighted. Finally, the thesis objectives are stated and the thesis structure is described.

In the next chapter, the literature related to existing approaches is reviewed in order to provide a comprehensive background and a synopsis of the relevant literature. In addition, it will be evaluated in terms of the concerns related to passive Software Engineering Ontology in order to identify the gaps that this research can address.

## 1.8 References

Abran, Alain, JW Moore, P Bourque, R Dupuis, and LL Tripp. 2004. "Software Engineering Body of Knowledge." *IEEE Computer Society, Angela Burgess*.

Ågerfalk, Pär J., Brian Fitzgerald, Helena Holmström Olsson, and Eoin Ó Conchúir. 2008. "Benefits of global software development: The known and unknown." In *Making Globally Distributed Software Development a Success Story: International Conference on Software Process, ICSP 2008 Leipzig, Germany, May 10-11, 2008 Proceedings*, eds Qing Wang, Dietmar Pfahl and David M. Raffo, 1-9. Berlin, Heidelberg: Springer Berlin Heidelberg.

Alqhtani, Masha'el Saeed, and M Rizwan Jameel Qureshi. 2014. "A proposal to improve communication between distributed development teams." *International Journal of Intelligent Systems and Applications* 6 (12): 34-39. doi: <http://dx.doi.org/10.5815/ijisa.2014.12.05>.

- Basri, Sufyan, Nazri Kama, Faizura Haneem, and Saiful Adli Ismail. 2016. "Predicting Effort for Requirement Changes During Software Development." In *Proceedings of the Seventh Symposium on Information and Communication Technology*, Ho Chi Minh City, Viet Nam, 380-387. 3011096: ACM. doi: 10.1145/3011077.3011096.
- Berners-Lee, Tim. 1999. "Weaving the Web: The Past, Present and Future of the World Wide Web by Its Inventor (with M. Fischetti)." *London: Orion Business Books*.
- Blincoe, Kelly, Giuseppe Valetto, and Daniela Damian. 2015. "Facilitating coordination between software developers: A study and techniques for timely and efficient recommendations." *Software Engineering, IEEE Transactions on* 41 (10): 969-985.
- Boden, Alexander. 2011. "Coordination and Learning in Global Software Development: Articulation Work in Distributed Cooperation of Small Companies.", University of Siegen 2011.
- Cataldo, M., A. Mockus, J. A. Roberts, and J. D. Herbsleb. 2009. "Software Dependencies, Work Dependencies, and Their Impact on Failures." *IEEE Transactions on Software Engineering* 35 (6): 864-878. doi: 10.1109/TSE.2009.42.
- Cataldo, M., and J. D. Herbsleb. 2013. "Coordination breakdowns and their impact on development productivity and software failures." *IEEE Transactions on Software Engineering* 39 (3): 343-360. doi: 10.1109/TSE.2012.32.
- Chu, Hai-Cheng, and Szu-Wei Yang. 2012. "Innovative Semantic Web services for next generation academic electronic library via web 3.0 via distributed artificial intelligence." In *Intelligent Information and Database Systems*, 118-124. Springer Berlin Heidelberg.
- Ebert, Christof. 2011. "The dark side: Challenges." In *Global Software and IT*, 19-25. John Wiley & Sons, Inc.

- Ebert, Christof, Marco Kuhrmann, and Rafael Prikladnicki. 2016. "Global software engineering: An industry perspective." *IEEE Software* 33 (1): 105-108. doi: 10.1109/ms.2016.27.
- Elamy, A.H. 2005. "Perspectives in agent-based technology." *AgentLinkNews* 18: 19-22.
- Forbes, David E. 2013. "A Framework for Assistive Communications Technology in Cross-Cultural Healthcare." School of Information Systems, Curtin Business School, Curtin University.
- Gupta, Avinash, Aprna Tripathi, and Dharmendra Singh Kuswaha. 2015. "Use case based approach to analyze software change impact and its regression test effort estimation." In *Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International Conference on Communication and Computer Engineering*, eds Hamzah Asyrani Sulaiman, Mohd Azlishah Othman, Mohd Fairuz Iskandar Othman, Yahaya Abd Rahim and Naim Che Pee, 1057-1067. Cham: Springer International Publishing.
- Holmstrom, H., E. O. Conchuir, P. J. Agerfalk, and B. Fitzgerald. 2006. "Global software development challenges: A case study on temporal, geographical and socio-cultural distance" *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, doi: 10.1109/ICGSE.2006.261210.
- Iqbal, Aftab, Oana Ureche, Michael Hausenblas, and Giovanni Tummarello. 2009. "Ld2sd: Linked Data Driven Software Development" The 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009), Boston, USA.
- Jain, Ritu, and Ugrasen Suman. 2015. "A systematic literature review on global software development life cycle." *SIGSOFT Software Engineering Notes* 40 (2): 1-14. doi: 10.1145/2735399.2735408.
- Jiménez, Miguel, Mario Piattini, and Aurora Vizcaíno. 2009. "Challenges and improvements in distributed software development: a systematic review." *Advances in Software Engineering* 2009: 1-16. doi: 10.1155/2009/710971.

- Lanubile, Filippo. 2009. "Collaboration in distributed software development." In *Software Engineering: International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, eds Andrea De Lucia and Filomena Ferrucci, 174-193. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Lind, Mary R, and Evetta Culler. 2013. "Information technology project performance: The impact of critical success factors." *Perspectives and Techniques for Improving Information Technology Project Management*: 39.
- Noll, John, Sarah Beecham, and Ita Richardson. 2011. "Global software development and collaboration: barriers and solutions." *ACM Inroads* 1 (3): 66-78. doi: 10.1145/1835428.1835445.
- Oliva, G. A., and M. A. Gerosa. 2012. "A method for the identification of logical dependencies" *2012 IEEE Seventh International Conference on Global Software Engineering Workshops*, doi: 10.1109/ICGSEW.2012.19.
- Panagiotou, Dimitris, and Gregoris Mentzas. 2009. "A Knowledge Workbench for Software Development" The 5th International Conference in the I-Semant.css series on Semantic Technologies (I-SEMANTICS), Graz, Austria.
- Panagiotou, Dimitris, Fotis Paraskevopoulos, and Gregoris Mentzas. 2011. "Knowledge-Based Interaction in Software Development." *Intelligent Decision Technologies* 5 (2): 163-175.
- Pete, I., and D. Balasubramaniam. 2015. "Handling the Differential Evolution of Software Artefacts: A Framework for Consistency Management" *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, doi: 10.1109/SANER.2015.7081889.
- Philip, Tom, Gerhard Schwabe, and Kweku Ewusi-Mensah. 2009. "Critical issues of offshore software development project failures." In *The 13th International Conference on Information Systems, Phoenix, USA*.
- Purna Sudhakar, Goparaju. 2012. "A model of critical success factors for software projects." *Journal of Enterprise Information Management* 25 (6): 537-558. doi: doi:10.1108/17410391211272829.

- Qiang, Lu, Chen Ming, and Wang Zhiguang. 2008. "A semantic annotation based software knowledges sharing space" *Network and Parallel Computing*, 2008. NPC 2008. IFIP International Conference on, doi: 10.1109/npc.2008.55.
- Shahid, M., and S. Ibrahim. 2016. "Change impact analysis with a software traceability approach to support software maintenance" *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, doi: 10.1109/IBCAST.2016.7429908.
- Sheth, A., and K. Thirunarayan. 2013. *Semantics Empowered Web 3.0: Managing Enterprise, Social, Sensor, and Cloud-based Data and Services for Advanced Applications*: Morgan & Claypool.
- Shiva, Sajjan G., Sarah B. Lee, Lubna A. Shala, and Chris B. Simmons. 2009. "Knowledge management in global software development." *International Journal of Distributed Sensor Networks* 5 (1): 6-6. doi: 10.1080/15501320802498513.
- Sommerville, I. 2004. *Software Engineering*. 7th ed: Pearson Education Limited.
- Souza, Cleidson R. B. de, and David F. Redmiles. 2008. "An empirical study of software developers' management of dependencies and changes." In *Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany*, 241-250. 1368122: ACM. doi: 10.1145/1368088.1368122.
- Wongthongtham, P., E. Chang, T.S. Dillon, and I. Sommerville. 2009. "Development of a software engineering ontology for multi-site software development." *IEEE Transactions on Knowledge and Data Engineering* 21 (8): 1205-1217. doi: 10.1109/TKDE.2008.209.
- Wongthongtham, Pornpit, Elizabeth Chang, Chan Cheah, and Tharam S Dillon. 2005. "Software engineering sub-ontology for specific software development." In *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA, Maryland, USA, April 7, 2005*. 27-33. IEEE. doi: 10.1109/SEW.2005.4.
- Wooldridge, Michael. 2009. *An Introduction to Multiagent Systems*: John Wiley & Sons.

Zahedi, Mansooreh, Mojtaba Shahin, and Muhammad Ali Babar. 2016. "A systematic review of knowledge sharing challenges and practices in global software development." *International Journal of Information Management* 36 (6, Part A): 995-1019. doi: <http://dx.doi.org/10.1016/j.ijinfomgt.2016.06.007>.

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.

# Chapter 2 Literature Review

## 2.1 Introduction

As mentioned in the previous chapter, the aim of this thesis is to assist software development team members with software engineering knowledge when they are working on software development projects. Generally, when an ontology has been created and evaluated from the development stage, it is then transited to the deployment stage which enables the interaction between the ontology and the application system (Jain, Malik and Lathar 2010). There are two main phases/stages associated with the ontology deployment stage, namely, knowledge assimilation and knowledge dissemination (Forbes 2013). Knowledge assimilation is related to the process of capturing and representing the domain-specific knowledge in a formal conceptual model, while knowledge dissemination is related to the process of delivering the knowledge to different types of applications or communicating the knowledge to users. In order to make the Software Engineering Ontology active, the focus will be on both the knowledge assimilation and knowledge dissemination phases.

In this chapter, a comprehensive survey is conducted of various existing approaches relevant to this research in order to provide a sufficiently broad background and pertinent literature synopsis. They are grouped under three categories: ontology-based semantic annotation, ontology-based multi-agent systems, and assistive systems in software engineering. The structure of this chapter is as follows:

- In Section 2.2, a review of the ontology-based semantic annotation approaches is presented, followed by an evaluation of ontology-based semantic annotation approaches for capturing software project information.
- In Section 2.3, a review of the ontology-based multi-agent systems is presented, followed by an evaluation of these systems.

- In Section 2.4, a review of the assistive systems in software engineering according to the software development activities in the software life cycle is presented, followed by an evaluation of these systems.
- The chapter concludes with Section 2.5 which contains a critical evaluation of the existing approaches from an integrated perspective which includes the ontology-based semantic annotation for capturing software project information, the ontology-based multi-agent systems, and the assistive systems in software engineering.

## **2.2 Ontology-based semantic annotation**

Knowledge assimilation is the process of capturing and representing the domain-specific knowledge in a formal conceptual model (Forbes 2013). Schwotzer and Berlin (2008) see it as a process whereby new knowledge is captured and incorporated into the knowledge base. When large amounts of knowledge need to be captured, an important point is the assimilation of extracted knowledge by means of systematic approaches that do not require great amount of human effort. The captured knowledge can be conceptually represented using the ontological model. In the literature, several studies have proposed the use of ontology-based semantic annotation, or semantic annotation for short, to express a formal representation of the resource's content by connecting it to concepts defined in an ontology. Ontology is a major part of an application domain description and is used as a means of identifying semantically-related annotations. Semantic annotation is deployed to generate intelligent content and provide a wide range of benefits to content-oriented intelligent applications (Yang 2006). Kiyavitskaya (2006) states that semantic annotation has been widely used in several applications and in different areas such as personalisation, text summarisation and question answering, information filtering, and intelligent knowledge management. In the Semantic Web, semantic annotation tools are used to annotate web documents, enabling human and machine to understand web contents. Amardeilh (2009) points out that another advantage of semantic annotation is that it can be used to supplement an ontology populating task



which is a process that enriches the knowledge base with new instances of the concepts, attributes and relations defined by the ontology model. In this work, the semantic annotation process is a step performed to extract relevant information related to the concerned domain from a set of documents and then map it with the concepts defined in the domain ontology. The aim is to obtain new instances before populating them to a knowledge base constrained by the ontology.

In the software development domain, the semantic annotation process is used to tackle problems regarding inappropriate, incomplete, and inconsistent syntactic descriptions of software development artefact properties and qualities. It also helps to enable automated knowledge acquisition tasks (Graubmann and Roshchin 2006). The semantic annotation provides the opportunity to transform software development artefacts so that they are conceptually organised and can be semantically linked. It helps to facilitate the integration of data from multiple software artefacts produced during the software development process (e.g. requirement specification, design documents, source code). Qiang, Ming, and Zhiguang (2008) implement a semantic annotation-based software knowledge-sharing space to improve the level of knowledge sharing and facilitate collaborative work among project members. Ontologies are used to create a link between software artefact contents and the abstract knowledge in the space. However, the annotation process is done manually by team members. Zygkotiatis, Dranidis, and Kourtesis (2009) propose a manual approach to semantically annotate Java source code using domain ontologies for the purpose of software reuse. This approach makes use of the standard annotation facility equipped with the release of Java 5.0 to add metadata to source code elements. In (Arantes and Falbo 2010), the authors discuss the use of semantic annotations in requirements document templates to support the management and evolution of requirements. The semi-automatic annotation process is based on the conceptualisation captured in the defined software requirement ontology. In (Panagiotou and Mentzas 2011b), the authors propose KnowBench, a semantic-based knowledge management system to assist developers to reuse code or knowledge about solving problems that had been previously addressed in the organisation. The source code is captured by means of both manual and semi-automatic annotation. Damjanovic, Amardeilh, and Bontcheva (2009) introduce an automatic approach to enhance semantic access to software artefacts (e.g., software document, source code)

using the semantic annotation process. This approach is based on the text analysis technique. The authors utilise the PROTON KM ontology<sup>1</sup> to interlink documents based on the identified key concepts. Taglialatela and Taglino (2012) propose an approach to enrich the semantic description of source code by semantically annotating it with a common domain ontology. The goal is to develop a semantic-based search and retrieval of software artefacts in order to facilitate software reuse. The annotation mechanism is based on the analysis of the source code comments which are added by a developer. The annotation process is automatic. However, the quality of the annotation result depends on the quality of the code comments.

Tichy, Köerner, and Landhäußer (2010) propose an approach to automatically create software models from natural language texts with semantic annotation. In (Graubmann and Roshchin 2006), the authors present a concept whereby automated software composition is supported by semantic modelling and making use of the annotation process and semantic extensions through knowledge-based techniques.

### **2.2.1 Evaluation of Semantic Annotation of Software Project Information**

In the software engineering domain, a significant amount of literature contains proposals for semantically annotating software project-related information. A number of works have contributed to source code semantic annotation. However, most of the reviewed approaches are based on manual and semi-automatic annotation. The manual approaches (e.g., Qiang, Ming, and Zhiguang 2008; Zygkostiots, Dranidis, and Kourtesis 2009) are considered inappropriate because they are tedious, time consuming, and error-prone, especially when a large volume of software artefacts is generated within a project. The semi-automatic annotation approaches (e.g., Arantes and Falbo 2010; Panagiotou and Mentzas 2011b) can be a good solution; however, they still require human intervention at some annotation level.

Some works have proposed the automatic approach (e.g., Graubmann and Roshchin 2006; Damljanovic, Amardeilh, and Bontcheva 2009; Tichy, Köerner, and

---

<sup>1</sup> <http://proton.semanticweb.org/2005/04/protonkm>

Landhäußer 2010; Taglialatela and Taglino 2012). However, most of them are based on text analysis techniques so that they are applicable only to textual artefacts (e.g., software requirement specification, software documents); they are not suitable for the semantic annotation of certain types of artefacts such as source code. In addition, most of the reviewed works regarding semantic annotation approaches in the software engineering domain focus only on semantic annotation which is intended to create semantic descriptions of software resources. Fewer works have paid attention to populating the ontology which is the task of adding new instances of concepts to the ontology (Petasis et al. 2011). The new instances could be derived from the semantic annotation.

Source code is considered as the main, centrally located artefact and is critical in software development; therefore, the need to capture its semantics in order to facilitate remote communication, coordination and knowledge sharing is obvious. Hence, given the volume of source code that needs to be dealt with, it is imperative to have a systematic approach for automating semantic annotation and ontology population tasks in order to ease the burden of manual tasks. This approach should be automated, or should require minimum human effort.

### **2.3 Ontology-based Multi-agent Systems**

From the literature, it is evident that considerable efforts have been put into the integration of ontologies and multi-agent systems, also known as ‘ontology-based multi-agent’ approaches in order to disseminate the knowledge captured in ontologies. Furthermore, some researchers have mentioned them as a means of facilitating knowledge assimilation by capturing and incorporating the knowledge into the ontology knowledge base. These works encompass various domains including software engineering, health, and education, to name a few. In the software engineering domain, a series of researches related to ontology-based multi-agent systems to support software development activities have been undertaken.

MAEST (Maamri and Sahnoun 2007) is a multi-agent system that is intended to assist testers during the testing process. An ontology for software testing is

developed to model several aspects related to testing software systems such as testing activities, testing methods, software artefacts, information about the environment in which testing is conducted, available resources, and the requirements of the test results. The agents use this information as a means of sharing knowledge and facilitating consistent communications.

In (Palacio et al. 2009), the authors propose an ontology-based multi-agent system to provide support for remote collaboration in multi-site distributed software development environments. In this work, agents are structured into two agencies, namely, user agency and the project agency to create Collaborative Working Spheres (CSW) for software developers to obtain information related to other remote team members' activities. A shared component ontology is created and used by the agents to facilitate consistent communication between the agents in different agencies.

Lee and Wang (2009) introduce an ontology-based computational intelligent multi-agent for Capability Maturity Model Integration (CMMI) assessment. This system consists of three main agents interacting with one another to achieve the goal of effectively summarising the evaluation reports of the software engineering process in regard to CMMI assessment. The CMMI ontology is developed specifically based on the fundamental knowledge of the Process and Product Quality Assurance (PPQA) process area of CMMI. The software agents make use of the defined concepts in this ontology to extract key sentences from the evaluated reports in order to enable the relevant team members to comprehend it easily and quickly.

The integration of multi-agent systems and Software Product Lines (SPL) is addressed in (Nunes et al. 2011). It provides a solution for producing higher quality software at lower development costs and less time-to-market by taking advantage of agent technologies. The ontology is used to model the Multi-agent System Product Lines (MAS-PLs) domain. The agents use this ontology to facilitate inter-agent communication.

The authors of (Monte-Alto et al. 2012) and (Teixeira and Huzita 2014) propose a context processing mechanism called ContextP-GSD (Context Processing on Global Software Development) that utilises contextual information to assist users during the software development process. This mechanism applies agent-based

technology to process contextual information and support human resource allocation.

The OntoDiSEN ontology (Chaves et al. 2011) is developed to represent context information in a global software development environment. The software agents use this ontology for context information retrieval and reasoning. In addition, the authors claim that the proposed ontology agent can manipulate the ontology instance knowledge such as updating contextual information or inserting new inferred action and facts. However, no details are provided to show how the ontology agent can perform these tasks.

In (Hadzic et al. 2009b), the authors offer a case study of an ontology-based multi-agent system in which collaborative agents are interacting and mediating with the Software Engineering Ontology to support multi-site software development teams. This thesis is the extension and the realisation of this work.

For the health domain, Hadzic et al. (2009a) propose a framework to unify the multi-agent approach with the human disease ontology in order to create an intelligent information retrieval system for human disease. The proposed ontology represents the knowledge regarding human diseases. The agents make use of this ontology for information retrieval and information analysis and to facilitate consistent communications among agents and knowledge reasoning.

Wang et al. (2010) introduce an ontology-based multi-agent system for intelligent healthcare applications to assist users to evaluate diets. The ontologies have been developed to represent personal profiles and food models. Agents use these ontologies to analyse appropriate diet information based on a user profile.

Li and Mackaness (2015) develop a system that is based on a multi-agent architecture to support decision-making for epidemic management. The system is intended to enhance the performance of information retrieval in a dynamic decision-making environment. Inexperienced personnel can use this system to locate online data and to process services for spatio-temporal analysis of a specified environmental epidemic. Ontologies for dataset and service semantics are used to describe general concepts of GIS web service and epidemiology data management, while lightweight ontologies for simple spatial and temporal reasoning are used to add spatial and temporal semantics to the geospatial data. The agents utilise these ontologies to

enable automated semantic service discovery and composition.

In educational domain, Oriche, Chekry, and Khaldi (2013) propose a semantic annotation system based on three main agents to manage the semantic annotation of educational resources. These agents utilise the domain ontology to assign domain knowledge to learning objects. Once these resources have been annotated, they are conceptualised and organised well so that they can be delivered to the users on demand according to their profiles and needs.

Dolia (2010) presents an ontology-based multi-agent system to provide useful information regarding academic institutions such as course information, course registration and scheduling. The Academic Institute Ontology is developed to define concepts and relationships that exist in university teaching environments. The agents make use of this ontology to facilitate their understanding for consistent communication and to provide responses to various types of queries.

In (García-Sánchez et al. 2008) and (García-Sánchez et al. 2009), the authors propose an ontology-based multi-agent framework to automatically discover, compose, invoke and monitor web services. Several kinds of ontologies, namely, application and domain ontology, agent local knowledge ontology, negotiation ontology, and semantic web services ontologies are utilised in this framework. In these works, the agents make use of these ontologies to automatically discover, compose, and invoke the available web services, and to facilitate consistent agent communication. The researchers evaluated the proposed framework by applying it to the e-commerce and biology domains.

In (Parhi, Pattanayak and Patra 2015), the authors develop an ontology-based multi-agent system to discover appropriate cloud services as requested by consumers. The system consists of three agents collaboratively working to provide dynamic searching for a cloud service. The Cloud Service Ontology is developed to represent cloud service description. The agents use this ontology for reasoning about the services and for information retrieval.

In addition to the abovementioned works, ontology-based multi-agent approaches have been used extensively in other domains. For example, Yang, Lo, and Steele (2007) introduce an ontology-based multi-agent system for the

accommodation services industry to support the online accommodation market. The domain ontology is used to facilitate agent communication and collaboration as well as the development of an ontology-based data transformation mechanism for data structure translation.

Ying, Ray, and Lewis (2013) introduce MOMA, a framework for creating ontology-based multi-agent systems, and incorporated an experiment in financial application development. MOMA consists of two main development phases: ontology development and agent development. However, the researchers focus only on the development of ontology and the use of the ontology to drive the implementation of the agent application. The agent development part is treated as a black box, but no details are provided regarding the design of the agent’s application. The agents make use of the ontology to facilitate consistent inter-agent communication and coordination.

Iribarne et al. (2014) propose an ontological web trading agent approach for environmental information retrieval. This work attempted to address the complexity of information retrieval in the information system to support environmental management. The ontologies used in this system are intended for information retrieval and to facilitate agent communication.

Table 2-1 provides a summary of the aforementioned ontology-based multi-agent systems.

Table 2-1: Review of some existing ontology-based multi-agent systems

Application Domain	Source	Objectives of ontology-based multi-agent systems	Purpose of agent’s use of ontology
Software Engineering	(Maamri and Sahnoun 2007)	Provide assistance to software testers by automating the process of test.	- Represent domain knowledge about software testing - Facilitate agent communication
	(Palacio et al. 2009)	Assist software development team to identify or create	-Facilitate consistent communication

Application Domain	Source	Objectives of ontology-based multi-agent systems	Purpose of agent's use of ontology
		opportunities for remote collaboration establishment	between the agents in different agencies.
	(Hadzic et al. 2009b)	Provide support for multi-site software development teams as a communication framework	<ul style="list-style-type: none"> <li>- Represent software engineering domain knowledge</li> <li>- Information retrieval</li> <li>- Facilitate agent communication</li> </ul>
	(Lee and Wang 2009)	Summarise the evaluation reports of the software engineering process in regard to CMMI assessment	<ul style="list-style-type: none"> <li>- Use defined concepts to extract the key sentences from the evaluated reports</li> <li>- Support reasoning of the term relation</li> </ul>
	(Nunes et al. 2011)	Provide a solution for producing higher quality software at lower development costs and less time-to-market	- Facilitate inter-agent communication
	(Monte-Alto et al. 2012)	Process contextual information and support human resource allocation	<ul style="list-style-type: none"> <li>- Contextual information retrieval</li> <li>- Knowledge reasoning</li> </ul>
	(Teixeira and Huzita 2014)	Support human resource allocation in globally distributed software projects.	<ul style="list-style-type: none"> <li>- Information retrieval</li> <li>- Knowledge reasoning</li> <li>- Knowledge</li> </ul>



Application Domain	Source	Objectives of ontology-based multi-agent systems	Purpose of agent's use of ontology
			manipulation
Health	(Hadzic et al. 2009a)	Intelligent and dynamic information retrieval of human disease information	<ul style="list-style-type: none"> <li>- Represent medical domain knowledge regarding human diseases</li> <li>- Information retrieval and analysis</li> <li>- Facilitate agent communication</li> <li>- Knowledge reasoning</li> </ul>
	(Wang et al. 2010)	Evaluate the health of diets	<ul style="list-style-type: none"> <li>- Represent personal profile and food model</li> <li>- Information analysis</li> </ul>
	(García-Sánchez et al. 2008)	Dynamically retrieve biological information	<ul style="list-style-type: none"> <li>- Facilitate agent communication and coordination</li> <li>- Information retrieval</li> </ul>
	(Li and Mackaness 2015)	Enhance the performance of Epidemiology information retrieval in a dynamic decision-making environment	<ul style="list-style-type: none"> <li>- Information retrieval</li> <li>- Spatial and temporal reasoning</li> </ul>
Education	(Dolia 2010)	Provide useful information for users in academic institutes	<ul style="list-style-type: none"> <li>- Facilitate the interactions among different agents</li> <li>- Information retrieval</li> </ul>
	(Oriche, Chekry)	Automate the semantic annotation of educational	<ul style="list-style-type: none"> <li>- Assign domain knowledge to</li> </ul>

Application Domain	Source	Objectives of ontology-based multi-agent systems	Purpose of agent's use of ontology
	and Khaldi 2013)	resources	educational resources
E-commerce	(Yang, Lo and Steele 2007)	Support communication, interaction, and management among different parties engaged in the accommodation e-market	<ul style="list-style-type: none"> <li>- Facilitate agent communication</li> <li>- Describe agent services</li> </ul>
	(García-Sánchez et al. 2009)	Facilitate the selection of the provider whose proposal best matches the users' preferences	<ul style="list-style-type: none"> <li>- Facilitate agent communication and coordination</li> <li>- Information retrieval</li> </ul>
Finance	(Ying, Ray and Lewis 2013)	Automate some market analysis tasks	<ul style="list-style-type: none"> <li>- Represent financial domain knowledge</li> <li>- Facilitate agent's communication and collaboration</li> </ul>
Environment	(Iribarne et al. 2014)	Address the complexity of information retrieval in the information system supporting environment management	<ul style="list-style-type: none"> <li>- Information retrieval</li> <li>- Facilitate agent communication</li> </ul>
Cloud service	(Parhi, Pattanayak and Patra 2015)	Discover appropriate cloud services as requested by consumers	<ul style="list-style-type: none"> <li>- Represent cloud service description</li> <li>- Reasoning</li> <li>- Information retrieval</li> </ul>

### 2.3.1 Evaluation of Ontology-based Multi-agent Systems

Although there is substantial literature on ontology-based multi-agent systems, the existing approaches have two shortcomings that this thesis intends to address, namely, the ontology-based multi-agent system for manipulating ontology instances, and the ontology-based multi-agent system that can provide support covering various activities in the software development life cycle.

First, in the literature, most of the ontology-based multi-agent systems focus on facilitating the dissemination of knowledge captured in the ontology. However, very little attention has been paid to utilising the ontology-based multi-agent approach for assimilating knowledge captured in the ontology, i.e., the ontology instantiation manipulation. The purposes for which the software agents make use of the ontology can be categorised as follows:

- 1) representing application and domain knowledge (e.g., Maamri and Sahnoun 2007; Hadzic et al. 2009a; Hadzic et al. 2009b; Lee and Wang 2009; Wang et al. 2010; Ying, Ray and Lewis 2013; Parhi, Pattanayak and Patra 2015);

- 2) locating and retrieving the information (e.g., García-Sánchez et al. 2008; García-Sánchez et al. 2009; Hadzic et al. 2009a; Hadzic et al. 2009b; Dolia 2010; Wang et al. 2010; Monte-Alto et al. 2012, Teixeira and Huzita 2014; Iribarne et al. 2014; Li and Mackaness 2015);

- 3) reasoning the knowledge (e.g., Monte-Alto et al. 2012; Teixeira and Huzita 2014; Hadzic et al. 2009a; Li and Mackaness 2015; Parhi, Pattanayak and Patra 2015);

- 4) facilitating agents' communication (e.g., Maamri and Sahnoun 2007; Yang, Lo and Steele 2007; García-Sánchez et al. 2008; Hadzic et al. 2009a; Hadzic et al. 2009b; Palacio et al. 2009; García-Sánchez et al. 2009; Dolia 2010; Nunes et al. 2011; Ying, Ray and Lewis 2013; Iribarne et al. 2014); and

- 5) facilitating semantic annotation of resources (e.g., Oriche, Chekry and Khaldi 2013).

Although some research (e.g., Monte-Alto et al. 2012; Teixeira and Huzita 2014) mentions the utilising of software agents to manipulate the ontology instantiations, no details or supporting information are provided to explain how the agents work on the ontology manipulation task. Because software agents are able to read and reason published knowledge with the guidance of the ontology (Hadzic et al. 2009b), it would be a challenge to utilise the ontology-based multi-agent approach for assimilating knowledge in order to manage the evolution of ontology instantiations.

Second, over recent years, the deployment of ontology-based multi-agent systems for effectively disseminating software development knowledge to support software team members has become more prevalent. Nevertheless, many of the works are specific in that they address only a particular task or a certain issue. Thus, it would be a challenge to investigate the use of the ontology-based multi-agent approach to provide useful support for software development team that can cover several tasks spanning the software life cycle.

## **2.4 Assistive Systems for Software Engineering**

In the literature, several researchers have proposed assistive systems to help team members and stakeholders to obtain useful project-related information during the software development process. They can be categorised according to software development activities as follows.

### **2.4.1 Requirement Gathering and Analysis**

This activity is considered one of the most critical activities during the software development life cycle because problems of requirement-related issues can have a great impact and even cause the failure of the software project. It includes the tasks of eliciting, analysing, and specifying the functional and behavioural properties of a software intensive system (Castro-Herrera et al. 2009). The majority of the reviewed assistive systems for this activity focus on the requirement engineering

process. Mobasher and Cleland-Huang (2011) highlight three areas where assistive systems such as recommendation systems can support requirement engineering tasks. These are:

- identifying potential stakeholders for a given project;
- generating possible user requirements or features; and
- providing useful information for decision making about requirement-related issues.

Numerous proposals for assistive systems for the online requirement elicitation process through the use of online tools such as wikis or forums are found in (Castro-Herrera, Cleland-Huang and Mobasher 2009a; Castro-Herrera et al. 2009; Castro-Herrera and Cleland-Huang 2009; Castro-Herrera, Cleland-Huang and Mobasher 2009b). In (Castro-Herrera, Cleland-Huang and Mobasher 2009a), the authors develop an assistive system for requirements elicitation in large-scale software projects. The system uses data-mining techniques and a collaborative recommendation approach to build a system that can support collaborations with stakeholders who are involved in software requirement elicitation, by placing stakeholders in appropriate discussion forums. In (Castro-Herrera et al. 2009), two techniques to enhance stakeholder profiles are employed to improve the performance of the assistive system for online requirements elicitations. In (Castro-Herrera and Cleland-Huang 2009), the approach that utilises machine learning techniques for identifying potential stakeholders to place to the relevant forums is discussed. Furthermore, in (Castro-Herrera, Cleland-Huang and Mobasher 2009b), the authors improve the quality of the system in order to support the dynamically evolving online forums when there are new posts and new users by focusing on two variations of the standard KNN algorithm: Binary Profiles, and Inclusion of Knowing Data.

Other research works that support requirement elicitation in large scale software projects with a focus on stakeholder analysis are StakeNet, StakeRare, and StakeSource. StakeNet (Lim, Quercia and Finkelstein 2010) uses social networks to identify and analyse the stakeholders. It asks stakeholders to recommend other stakeholders and then builds a social network from their recommendations. Finally, it prioritises stakeholders using social network measures. StakeRare (Soo Ling and

Finkelstein 2012) stands for Stakeholder and Recommender-assisted method for requirements elicitation. It extends StakeNet by providing additional features for prioritizing the requirements using stakeholders' ratings weighted by their project influence. StakeSource (Lim et al. 2013) is a web-based tool that automates the StakeNet approach for stakeholder analysis. It uses Web 2.0 technologies such as crowdsourcing and social networking to identify and prioritise stakeholders.

INTELLIREQ (Felfernig et al. 2012) is a group decision environment designed to support the decision-making process in requirement negotiation. It suggests which requirements should be implemented within the scope of the small-sized software projects. By applying group recommendation technologies, INTELLIREQ can improve the usability and the quality of decision-making support in requirements engineering environments.

Unlike the aforementioned systems that focus on requirement elicitation, in (Dumitru et al. 2011), the authors develop an assistive system that models and identifies product features during the domain analysis process. This system employs association rule mining to identify the relationship between product features and then generates a feature recommendation.

#### **2.4.2 Software design**

Many of the assistive applications developed for this activity focus mainly on helping designers to find or make a decision about the design pattern that is the most appropriate for a given problem. Guéhéneuc and Mustapha (2007) propose an assistive system to support work on design pattern. The system is based on analysing the textual descriptions of design patterns and then extracting important words. These words are then compared with the key words chosen by the user. Because this approach is based on the similarity of those key words, it might not exactly match what a user desires. The other limitation is that users cannot query using natural language. They can make choices only from those provided by the system.

Designer Pattern Recommender (DPR) (Palma et al. 2012) is an assistive system that suggests appropriate reusable design patterns for a software designer. It is based on a simple Goal-Question-Metric (GQM) model for the interactivity. It

employs a weighting scheme and ranking for selecting a pattern. Suresh et al. (2011) develop a design pattern assistive system to assist developers to identify the right design pattern for their given situation. Although they claim that the search facility can be extended to search any category of software design patterns because of the same underlying schema, the current system can support only the search for Gang of Four (GoF) patterns.

The abovementioned systems are based on natural language techniques; therefore, they are subject to ambiguous interpretation by different users. Liu et al. (2014) propose an automated approach for service-oriented architecture (SOA) design patterns advisement. A lightweight ontology is constructed and used to provide a formal description and organisation structure of SOA design patterns. The system obtains the user's requirement in the form of question and answer in order to avoid the complexities associated with the use of natural language. It then identifies appropriate design patterns based on the user's answer to the proposed question and the sorting choice of property value in Constraint Program (CP).

### **2.4.3 Software Implementation and Maintenance**

The majority of existing assistive systems have been developed to assist software development teams with software implementation and maintenance. They can help developers with a wide range of programming tasks such as suggesting source code, identifying related artefacts, and resolving bug issues. Examples of these are given below.

DebugAdvisor (Ashok et al. 2009) is proposed as a search tool for debugging that provides all contextual information related to a bug issue. Developers can search bug reports from multiple software repositories with a single query. The system returns a bug description ranked list that matches the query and then uses it to identify the related artefacts such as experts, source code and functions from the generated relationship graph. McMillan, Poshyvanyk, and Grechanik (2010) introduce an approach to identify source code examples by matching key words in queries to the documentations of Application Programming Interface (API) calls rather than source code by using text-based Information Retrieval. They consider that

the documentation may contain terminology that is closer to user queries than is the source code. However, this approach utilises neither the developer's contextual information nor user profiles to supplement the representation.

Fishtail (Sawadsky and Murphy 2011) is a plugin tool for the Eclipse IDE which is intended to automatically identify source code examples from the web that are relevant to a developer's current task. Task context is captured to obtain key words when the developer interacts with the related artefacts. It then automatically queries the web using those key words and identifies relevant pages for the user. However, because Fishtail does not consider the history of visited web pages that a developer has previously found to be useful, it cannot identify appropriate pages with high accuracy. Cordeiro, Antunes, and Gomes (2012) propose a context-based recommendation to support problem solving in software development. They develop a client/server tool to integrate recommendation of question/answering web resources in the developer's work environment to provide automatic assistance when the exception errors occur. The content on stack overflow which is a question/answering website for software development issues has been used and processed for information extraction, representation and indexing to generate the knowledge base.

In contrast to the above mentioned assistive systems, Dhruv (Ankolekar et al. 2006), Switch! (Maalej and Sahm 2010), and KnowBench (Panagiotou and Mentzas 2011a) utilise the Semantic Web technologies which are ontologies to support knowledge representation of software project-related information. Dhruv (Ankolekar et al. 2006) is intended to assist developers with problem-solving activities in the open source software community. Ontologies are used to describe the structure of the project and interaction within the community, and to provide a basis for determining how artefacts are related. They can enable developers to identify related software artefacts and relevant bug information during the bug resolution process. Switch! (Maalej and Sahm 2010) is a context-aware artefact recommendation and switching tool that can assist software developers to switch artefacts based on their task semantics and interaction history. The TeamWeaver Ontologies are used to describe the task semantic model, artefacts and their relationship with each other. Instead of just analysing artefacts stored in the repository, Switch! uses interaction data to generate more precise recommendations. KnowBench (Panagiotou and Mentzas



2011a) is an ontology-based knowledge management system that helps software developers to manage error handling and to reuse software components. It is integrated into the Eclipse IDE in order to capture the knowledge generated during the software development process as soon as it is generated.

Some assistive systems are intended to help developers to locate relevant experts, thereby saving time during the software development process. Examples of these systems are as follows.

SmallBlue (Ching-Yung et al. 2008), also known as IBM Atlas, provides a relevance-ranked list of experts from the social network connection by associating their names with topics extracted from emails and instant messages. An Artificial Intelligence algorithm is applied to infer users' expertise and their social network. Rather than relying only on search algorithms or techniques to obtain the required information, SmallBlue utilises the organisational and social contexts of developers to make the suggestion more reliable. Ensemble (Xiang et al. 2008) is an assistive application that helps software team members to communicate about their current work by identifying relevant people to contact when there is an update on particular artefacts. Codebook (Begel, Yit Phang and Zimmermann 2010) is a social network web service that links developers and their work artefacts and maintains connections with other software team members. Conscius (Moraes et al. 2010) is an assistive system that locates a source code expert for a given software project by using communication history (archived mail threads), source code, documentation and software configuration management change history. A mining algorithm has been used to relate the emails to the documentation or source code. Steinmacher, Wiese, and Gerosa (2012) propose a system that helps newcomers to discover the expert who has the skill matching the selected issue to mentor a particular technical task regarding technical and social aspects. They use historical information from source code repositories, mail lists threads and issue tracker comments to determine the social score; the workspace context from user interaction with the IDE is used to produce the developers' technical score and to collect developers' recent activities for calculating their current interest score.

#### 2.4.4 Software Testing

Quality assurance is one of the most important processes for achieving software product quality. Assistive systems can be applied to assist team members to manage various activities related to software testing. Examples of recent works are given below.

Kpodjedo et al. (2008) propose a system that focuses on identifying critical classes that deserve to get more attention because they are frequently subject to change and have an impact on other classes. Miranda, Aranha, and Iyoda (2012) develop a assistive system for allocating test cases to testers. The system is integrated in the Eclipse Integrated Development Environment. It offers two main benefits to team members involved in testing. First, it helps test managers to allocate test cases faster. Second, it can provide useful information to a new test manager regarding test cases and tester details as well as the history of previous allocations. Li and Zhang (2012) introduce a platform for software test case reuse which is based on the ontology representation and the knowledge management model. It is intended to assist test engineers to retrieve and reuse existing test cases effectively.

A summary of reviewed assistive systems according to activities in the software development life cycle is presented in Table 2-2.

Table 2-2: Summary of reviewed assistive systems according to software development activities

Activities	Applications/ Authors	Objective	Semantic knowledge representation	Information delivery mode
Requirement gathering and analysis	(Castro-Herrera, Cleland-Huang and Mobasher 2009a	Facilitate the placement of stakeholders into related discussion forums	No	Push
	(Castro-Herrera et al. 2009)	Enhance the placement of stakeholders into related discussion forums for collaborative work to generate requirements	No	Push
	(Castro-Herrera and Cleland-Huang 2009)	Identify potential stakeholders for a given topic	No	Not specifically mentioned

<b>Activities</b>	<b>Applications/ Authors</b>	<b>Objective</b>	<b>Semantic knowledge representation</b>	<b>Information delivery mode</b>
	(Castro-Herrera, Cleland-Huang and Mobasher 2009b)	Enhance the forum recommender to handle the challenges of dynamically nature of online forums which constantly evolve according to new posts and new users	No	Not specifically mentioned
	StakeNet (Lim, Quercia and Finkelstein 2010)	Identify and prioritise stakeholders and their roles	No	Pull
	StakeRare (Soo Ling and Finkelstein 2012)	Identify and prioritise requirements for requirement elicitation	No	Pull
	(Dumitru et al. 2011)	Identify product features during the domain analysis process	No	Push
	INTELLIREQ (Felfernig et al. 2012)	Recommend requirements that should be implemented according to software project scope	No	Pull
	StakeSource (Lim et al. 2013)	Identify and prioritise stakeholders and their roles	No	Not specifically mentioned
Software Design	DPR (Guéhéneuc and Mustapha 2007)	Help designers to find or decide which pattern to use for a particular design problem	No	Pull
	(Suresh et al. 2011)	Find a suitable pattern to users	No	Pull
	(Liu et al. 2014)	Assist users to select appropriate SOA design patterns	Yes Lightweight Ontology	Pull
Software Implementation and	Dhruv (Ankolekar et al. 2006)	Support open source software communities regarding bug resolution with the Semantic Web	Yes Heavyweight Ontology	Push

<b>Activities</b>	<b>Applications/ Authors</b>	<b>Objective</b>	<b>Semantic knowledge representation</b>	<b>Information delivery mode</b>
Maintenance	Ensemble (Xiang et al. 2008)	Identify relevant people to contact when artefacts get updated	No	Push
	SmallBlue (Ching-Yung et al. 2008)	Discover expert from the social network	No	Pull
	DebugAdvisor (Ashok et al. 2009)	Provide a search tool for debugging	No	Pull
	Codebook (Begel, Yit Phang and Zimmermann 2010)	Help developers to discover and maintain connections to others	No	Pull
	Conscious (Moraes et al. 2010)	Locate expert on a given software project	No	Pull
	McMillan, Poshyvanyk, and Grechanik (2010)	Suggest source code by making use of API call documentation	No	Pull
	Switch! (Maalej and Sahm 2010)	Facilitate software developers in switching artefacts based on their task semantics and interaction history.	Yes Lightweight Ontology	Pull
	KnowBench (Panagiotou and Mentzas 2011a)	Assist developers to manage error handling and to reuse software component	Yes Lightweight Ontology	Pull
	Fishtail (Sawadsky and Murphy 2011)	Support source code writing	No	Push
	Cordeiro, Antunes, and Gomes (2012)	Help developers access to question/answering web resources when code fails with an exception	No	Pull
	(Steinmacher, Wiese and Gerosa 2012)	Support newcomers to find relevant experts	No	Pull
Software Testing	(Kpodjedo et al. 2008)	Identify critical classes that need special attention	No	Pull

Activities	Applications/ Authors	Objective	Semantic knowledge representation	Information delivery mode
	(Miranda, Aranha and Iyoda 2012)	Help to allocate test cases to testers	No	Pull
	(Li and Zhang 2012)	Assist test engineer to retrieve and reuse existing test cases	Yes Heavyweight Ontology	Pull

#### 2.4.5 Evaluation of Assistive Platforms in Software Engineering

A large number of assistive systems have been developed to assist team members with software engineering knowledge when they are working on various software development activities. However, the assistive systems reviewed above have three limitations: they apply to only a specific software development activity; they mostly use typical knowledge representation and syntactic matching techniques; and generally, the user has to initiate a request for a specific piece of information.

First, all the systems described above have been developed to support software teams by providing useful software project information when they are working on a software development project. However, these systems are task-specific and most of them focus on software implementation and maintenance activities. Although project team members can benefit from individual systems which address separate software development activities, the need for integrated tools/services to support software development activities across the software life cycle is also important (Sengupta, Chandra and Sinha 2006).

Second, most of the reviewed assistive systems use typical knowledge representation and syntactic matching techniques which could produce ambiguity in keyword-based queries. One word may have several meanings. For example, the word 'Java' refers to either a programming language or an island. Hence, assistive systems could use the Semantic Web and ontologies to tackle this problem. Ontologies not only facilitate knowledge access and sharing, but also enable semantic query and semantic matching to improve obtained results. Some of the aforementioned systems (e.g., Liu et al. 2014; Maalej and Sahm 2010; Panagiotou

and Mentzas 2011a) have made use of ontologies for semantic knowledge representation; however, these are lightweight rather than heavyweight ontologies. A lightweight ontology includes only a hierarchy of concepts and a hierarchy of relations. On the other hand, a heavyweight ontology is enriched with axioms that can be used to infer the semantic interpretation of concepts and relations (Fürst and Trichet 2006). Thus, if the assistive systems make use of a heavyweight ontology such as the Software Engineering Ontology, which is a comprehensive ontology covering all the aspect of software engineering, they would be able to provide greater support for information access by searching only for relevant information, thereby improving the query retrieval result.

Finally, there are two approaches for delivering information to users: push and pull. An information push is an approach whereby the systems deliver knowledge or useful information without the user having to explicitly request it. This is opposite to an information pull approach that requires users to query the knowledge or initiate their requests so that the systems can provide the knowledge. Most of the reviewed assistive systems deliver the knowledge based on the information pull approach. This approach has the advantage that it does not cause an information overload problem to users because they can request the knowledge as needed. Nonetheless, they may not be aware of the existence of the knowledge and therefore may miss information that could be useful for their work. Therefore, effective assistive systems should be able to proactively deliver relevant information to their users based on an appropriate context by considering what to deliver and when to push it to users.

## **2.5 Critical Evaluation of Existing Approaches: an Integrated View**

In this section, the existing systems and approaches carried out in the literature are discussed evaluated, and the main issues that need to be addressed for devising a framework that enables an active Software Engineering Ontology, are identified. This section provides an overview of all the issues.

The main shortcomings of the existing systems and approaches pertain to three areas.

- Lack of effective approach to automate knowledge capture of software project information
- Lack of effective management of knowledge captured in the ontology
- Lack of active platforms available for multi-site software development environments

### **2.5.1 Lack of effective approach to automate knowledge capture of software project information**

As discussed in section 2.2, in the literature regarding software engineering domain, several works have been conducted on knowledge assimilation by capturing software project information via semantic annotation. Some of them are still based on the manual semantic annotation approaches (e.g., Qiang, Ming and Zhiguang 2008; Zygmoustis, Dranidis and Kourtesis 2009). The manual approaches have major shortcomings: they are tedious, time-consuming, prone-to-error, and require a great deal of human effort. Several works have attempted to overcome these issues by taking a semi-automatically semantic annotation approach (e.g., Arantes and Falbo 2010; Panagiotou and Mentzas 2011b). However, they still require additional human intervention. Recently, the focus has shifted toward the automated approaches to capture the semantics of software project information which are more efficient and require either minimal or no effort from software team members. However, most of the reviewed approaches (e.g., Graubmann and Roshchin 2006; Damjanovic, Amardeilh and Bontcheva 2009; Tagliatela and Taglino 2012) are based on the analysis of text. Therefore, they are appropriate for software artefacts that contain text descriptions such as software documents, software requirement specification. However, they are not suitable for capturing knowledge of certain types of artefacts such as source code.

In addition, the outputs from the capturing process of those approaches are mostly in RDF (Resource Description Framework) and RDFS (Resource Description

Framework Schema). RDF is particularly aimed at describing the semantics of information in a machine-understandable and machine-processable form. RDFS extends RDF with schema vocabulary such as Class, subclassOf, Property, domain, range. Fewer studies have been conducted to populate the captured knowledge into the ontology repository in OWL (Web Ontology Language) which was developed as an extension of RDF and RDFS. Even though RDF and RDFS is useful for describing resources with simple semantics containing objects and their relations, it has certain limitations. For example, it does not provide transitive, inverse or symmetrical properties, which OWL can do. Because OWL is very expressive and the relation between classes can be formally defined based on description logics, capturing knowledge and storing it in OWL is more advantageous. It allows properties of software resources to be described and inferred from a knowledge base.

As a result, there needs to be a specific area of research that focuses on knowledge assimilation by automatically capturing semantics of software project information and storing the captured knowledge in the ontology repository for subsequent use. This requires a systematic approach to assign software engineering domain concepts to the software project information, and to populate the captured knowledge in the ontology knowledge base. In Chapter 3, the need for such a systematic approach is explained. In Chapter 4, the proposed solution is presented and discussed in detail.

### **2.5.2 Lack of Effective Management of Knowledge Captured in the Ontology**

Changes are inevitable and can occur in any stage of a software project. There are different types of changes including changes in users' requirements, changes in the system's environment, or ongoing maintenance to correct failures. Once the software development project information has been captured in the Software Engineering Ontology, it continuously evolves throughout its lifetime. Any change to software project information can lead to the inconsistency of knowledge captured in the Software Engineering Ontology. If such a change is not managed appropriately, it could prevent the use of knowledge. During a multi-site distributed software development project, remote team members need current and accurate information about the people and the artefacts that might be affected by the change



made by team members at different sites. Therefore, real-time awareness is important and has become a critical factor that can help software teams to properly manage the change and its impact on software evolution.

In the literature, several researchers have proposed the use of agent-based technology together with ontologies for the purpose of knowledge assimilation and knowledge dissemination. Most of the reviewed ontology-based multi-agent systems make use of ontologies to support software agents in the following tasks:

- 1) representing application and domain knowledge
- 2) locating and retrieving the information
- 3) reasoning the knowledge
- 4) facilitating agent's communication and interoperability
- 5) facilitating semantic annotation

Even though some works (Monte-Alto et al. 2012; Teixeira and Huzita 2014) have claimed to manage the evolution of knowledge captured in the ontology by software agents, to the best of our knowledge, none of them explicitly addresses how the agents can manipulate this knowledge. The explanation is only at an abstract level. As mentioned earlier, knowledge in a software development project constantly evolves; therefore, there is the need for an approach that can effectively manage the evolution of knowledge captured in the Software Engineering Ontology. The lack of an effective approach to manage captured knowledge is discussed in Chapter 3. The solution is proposed in Chapter 4.

### **2.5.3 Lack of Active Platforms Available for Multi-site Software Development Environments**

Software development is considered as a knowledge-intensive, complex and collaborative activity. The quality of a software product largely depends on the quality of the software process which is the result of the activities conducted throughout the software development process. An effective software process is associated with people, tools, and procedures working as an integrated whole (Paulk 2002). Therefore, much research attention has been given to the development of software applications or tools that can assist project team members to perform their tasks effectively. Even though project teams can benefit from tools specifically

designed for individual software development activities, the assistive integrated tools that can be used for related activities can be of more benefit. Ossher, Harrison, and Tarr (2000) point out that the identification of the requirement for integrated support for software development activities throughout the various phases of software development life cycle represents the genesis of software engineering environments. They define the software engineering environments (SEEs) as “*the integrated collections of software applications that facilitate software engineering activities across the software life cycle*”.

As discussed in section 2.4.5, in the existing literature, all of the reviewed assistive systems are intended to provide knowledge support to development teams only for particular software development activities or a specific phase in the software development life cycle. In particular, they focus on software implementation and maintenance tasks. However, software development activities and their artefacts are interconnected. The work or a change in one activity may have an effect on the work in other activities. Therefore, software development team members need to be supported in their various activities. This is particularly so in a multi-site software development environment where team members are geographically dispersed, and inadequate communication and coordination are the main factors that can hinder the success of a software project. It is important to have assistive platforms for multi-site software development environments that can assist remote team members to work collaboratively throughout the various phases in a software development life cycle (Sengupta, Chandra and Sinha 2006).

Furthermore, many of the reviewed systems do not operate in a proactive manner. This means that they mostly rely on certain efforts of team members for knowledge acquisition. Frequently, team members may not be aware of the existence of the useful knowledge due to the large amount of information or because they are new members who have just joined the project. The lack of active platforms that can provide effective knowledge management and knowledge sharing in order to deliver the right information to the right people at the right time is an issue that still needs to be addressed in this research.

The lack of active platforms for multi-site software development environments is discussed in Chapter 3, and the solution is presented in Chapter 4.

## 2.6 Conclusion

This chapter has reviewed the current state-of-the-art in ontology-based semantic annotation, ontology-based multi-agent systems, and assistive systems for software engineering. The relevant literature is discussed to provide the necessary background and context to address the identified gaps related to making the Software Engineering Ontology active. The reviewed literature evidently indicates that substantial progress has been made to support project team members when they are working on software development project. However, the existing systems and approaches still have shortcomings in terms of (1) the lack of an effective approach to automate knowledge capture of software project information; (2) the lack of effective management of knowledge captured in the ontology; and (3) the lack of active platforms available for multi-site software development environments.

Based on this review, in the next chapter, the key concepts, problem definition and research issues pertaining to the framework for making the Software Engineering Ontology active, are identified.

## 2.7 References

- Amardeilh, Florence. 2009. "Semantic annotation and ontology population." In *Semantic Web engineering in the Knowledge Society*, 135-160. IGI Global.
- Ankolekar, A., K. Sycara, J. Herbsleb, R. Kraut, and C. Welty. 2006. "Supporting online problem-solving communities with the Semantic Web." In *Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland*, 575-584. ACM. doi: 10.1145/1135777.1135862.
- Arantes, L. d. O., and R. d. A. Falbo. 2010. "An infrastructure for managing semantic documents" *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, Vitoria, Brazil*, doi: 10.1109/EDOCW.2010.17.

- Ashok, B., Joseph Joy, Hongkang Liang, Sriram K. Rajamani, Gopal Srinivasa, and Vipindeep Vangala. 2009. "DebugAdvisor: a recommender system for debugging." In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Amsterdam, The Netherlands*, 373-382. 1595766: ACM. doi: 10.1145/1595696.1595766.
- Begel, A., Khoo Yit Phang, and T. Zimmermann. 2010. "Codebook: discovering and exploiting relationships in software repositories" *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, doi: 10.1145/1806799.1806821.
- Castro-Herrera, C., and J. Cleland-Huang. 2009. "A machine learning approach for identifying expert stakeholders" *Managing Requirements Knowledge (MARK), 2009 Second International Workshop on*, doi: 10.1109/mark.2009.1.
- Castro-Herrera, C., J. Cleland-Huang, and B. Mobasher. 2009a. "Enhancing stakeholder profiles to improve recommendations in online requirements elicitation" *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, doi: 10.1109/re.2009.20.
- Castro-Herrera, Carlos, Jane Cleland-Huang, and Bamshad Mobasher. 2009b. "A recommender system for dynamically evolving online forums." In *Proceedings of the third ACM Conference on Recommender systems, New York, New York, USA*, 213-216. 1639751: ACM. doi: 10.1145/1639714.1639751.
- Castro-Herrera, Carlos, Chuan Duan, Jane Cleland-Huang, and Bamshad Mobasher. 2009. "A recommender system for requirements elicitation in large-scale software projects." In *Proceedings of the 2009 ACM Symposium on Applied Computing, Honolulu, Hawaii*, 1419-1426. 1529601: ACM. doi: 10.1145/1529282.1529601.
- Chaves, A.P., I. Steinmacher, L. Leal, G. Camila, E.H.M. Huzita, and A.B. Biasão. 2011. "OntoDiSEnv1: An ontology to support global software development." *CLEI Electronic Journal* 14 (2): 2-2.

- Ching-Yung, Lin, K. Ehrlich, V. Griffiths-Fisher, and C. Desforges. 2008. "SmallBlue: people mining for expertise search." *MultiMedia, IEEE* 15 (1): 78-84. doi: 10.1109/mmul.2008.17.
- Cordeiro, J., B. Antunes, and P. Gomes. 2012. "Context-based recommendation to support problem solving in software development" *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, doi: 10.1109/rsse.2012.6233418.
- Damljanovic, Danica, Florence Amardeilh, and Kalina Bontcheva. 2009. "CA manager framework: creating customised workflows for ontology population and semantic annotation." In *Proceedings of the Fifth International Conference on Knowledge Capture, Redondo Beach, California, USA*, 177-178. 1597770: ACM. doi: 10.1145/1597735.1597770.
- Dolia, Prashant M. 2010. "Integrating ontologies into multi-agent systems engineering (MaSE) for university teaching environment." *Journal of Emerging Technologies in Web Intelligence* 2 (1): 42-47.
- Dumitru, Horatiu, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. 2011. "On-demand feature recommendations derived from mining public product descriptions." In *Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu, HI, USA*, 181-190. 1985819: ACM. doi: 10.1145/1985793.1985819.
- Felfernig, Alexander, Christoph Zehentner, Gerald Ninaus, Harald Grabner, Walid Maalej, Dennis Pagano, Leopold Weninger, and Florian Reinfrank. 2012. "Group decision support for requirements negotiation." In *Advances in User Modeling*, eds Liliana Ardissono and Tsvi Kuflik, 105-116. Springer Berlin Heidelberg.
- Forbes, David E. 2013. "A Framework for Assistive Communications Technology in Cross-Cultural Healthcare." School of Information Systems, Curtin Business School, Curtin University.

Fürst, Frédéric, and Francky Trichet. 2006. "Heavyweight ontology engineering." In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops: OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWWS, and SeBGIS 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I*, eds Robert Meersman, Zahir Tari and Pilar Herrero, 38-39. Berlin, Heidelberg: Springer Berlin Heidelberg.

García-Sánchez, Francisco, Jesualdo Tomás Fernández-Breis, Rafael Valencia-García, Juan Miguel Gómez, and Rodrigo Martínez-Béjar. 2008. "Combining semantic web technologies with multi-agent systems for integrated access to biological resources." *Journal of Biomedical Informatics* 41 (5): 848-859. doi: <http://dx.doi.org/10.1016/j.jbi.2008.05.007>.

García-Sánchez, Francisco, Rafael Valencia-García, Rodrigo Martínez-Béjar, and Jesualdo T. Fernández-Breis. 2009. "An ontology, intelligent agent-based framework for the provision of Semantic Web services." *Expert Systems with Applications* 36 (2, Part 2): 3167-3187. doi: <http://dx.doi.org/10.1016/j.eswa.2008.01.037>.

Graubmann, P., and M. Roshchin. 2006. "Semantic annotation of software components." In *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, Aug. 29 2006-Sept. 1 2006. 46-53. doi: 10.1109/euromicro.2006.54.

Guéhéneuc, Yann-Gaël, and Rabih Mustapha. 2007. "A simple recommender system for design patterns." *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*.

Hadzic, Maja, Pornpit Wongthongtham, Tharam Dillon, and Elizabeth Chang. 2009a. "Case Study I: Ontology-Based Multi-Agent System for Human Disease Studies." In *Ontology-Based Multi-Agent Systems*, 179-216. Springer Berlin Heidelberg.

- . 2009b. "Case study II: Ontology-based multi-agent system for software engineering studies." In *Ontology-Based Multi-Agent Systems*, 217-270. Springer.
- Iribarne, Luis, Nicolás Padilla, Rosa Ayala, José A Asensio, and Javier Criado. 2014. "OntoTrader: An ontological web trading agent approach for environmental information retrieval." *The Scientific World Journal* 2014: 25. doi: 10.1155/2014/560296.
- Jain, Vishal, Sanjay Kr. Malik, and Pankaj Lathar. 2010. "Ontology: Development, deployment and merging aspects in Semantic Web: An overview." *National Journal by IMS Noida*: 23-27.
- Kiyavitskaya, Nadzeya. 2006. "Tool Support for Semantic Annotation." International Doctorate School in Information and Communication Technologies DIT - University of Trento
- Kpodjedo, Segla, Filippo Ricca, Philippe Galinier, and Giuliano Antoniol. 2008. "Not all classes are created equal: toward a recommendation system for focusing testing." In *Proceedings of the 2008 International Workshop on Recommendation Systems for Software Engineering, Atlanta, Georgia*, 6-10. 1454250: ACM. doi: 10.1145/1454247.1454250.
- Lee, Chang-Shing, and Mei-Hui Wang. 2009. "Ontology-based computational intelligent multi-agent and its application to CMMI assessment." *Applied Intelligence* 30 (3): 203-219. doi: 10.1007/s10489-007-0071-1.
- Li, Sen, and William A Mackaness. 2015. "A multi-agent-based, semantic-driven system for decision support in epidemic management." *Health Informatics Journal* 21 (3): 195-208. doi: 10.1177/1460458213517704.
- Li, X., and W. Zhang. 2012. "Ontology-based testing platform for reusing" *2012 Sixth International Conference on Internet Computing for Science and Engineering*, doi: 10.1109/ICICSE.2012.18.
- Lim, S. L., D. Damian, F. Ishikawa, and A. Finkelstein. 2013. "Using web 2.0 for stakeholder analysis: StakeSource and its application in ten industrial

- projects." In *Managing Requirements Knowledge*, eds Walid Maalej and Anil Kumar Thurimella, 221-242. Springer Berlin Heidelberg.
- Lim, Soo Ling, Daniele Quercia, and Anthony Finkelstein. 2010. "StakeNet: using social networks to analyse the stakeholders of large-scale software projects." In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, Cape Town, South Africa*, 295-304. 1806844: ACM. doi: 10.1145/1806799.1806844.
- Liu, Lei, Pinxin Miao, Luka Pavlic, Marjan Hericko, and Rui Zhang. 2014. "An ontology-based advisement approach for SOA design patterns." In *The 8th International Conference on Knowledge Management in Organizations*, eds Lorna Uden, Leon S. L. Wang, Juan Manuel Corchado Rodríguez, Hsin-Chang Yang and I. Hsien Ting, 73-84. Springer Netherlands.
- Maalej, Walid, and Alexander Sahn. 2010. "Assisting engineers in switching artifacts by using task semantic and interaction history." In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, Cape Town, South Africa*, 59-63. 1808935: ACM. doi: 10.1145/1808920.1808935.
- Maamri, Ramdane, and Zaidi Sahnoun. 2007. "MAEST: multi-agent environment for software testing." *Journal of Computer Science* 3 (4): 249-258.
- McMillan, Collin, Denys Poshyvanyk, and Mark Grechanik. 2010. "Recommending source code examples via API call usages and documentation." In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, Cape Town, South Africa*, 21-25. 1808925: ACM. doi: 10.1145/1808920.1808925.
- Miranda, Breno Alexandro Ferreira de, Eduardo Henrique da Silva Aranha, and Juliano Manabu Iyoda. 2012. "Recommender systems for manual testing: deciding how to assign tests in a test team." In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Lund, Sweden*, 201-210. 2372289: ACM. doi: 10.1145/2372251.2372289.



- Mobasher, Bamshad, and Jane Cleland-Huang. 2011. "Recommender systems in requirements engineering." *AI Magazine* 32 (3): 81-89.
- Monte-Alto, Helio, Alberto Biasão, Lucas Teixeira, and Elisa Huzita. 2012. "Multi-agent applications in a context-aware global software development environment distributed computing and artificial intelligence." 265-272. Springer Berlin / Heidelberg.
- Moraes, Alan, Eduardo Silva, Cleyton da Trindade, Yuri Barbosa, and Silvio Meira. 2010. "Recommending experts using communication history." In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, Cape Town, South Africa*, 41-45. 1808929: ACM. doi: 10.1145/1808920.1808929.
- Nunes, Ingrid, Carlos J. P. Lucena, Uirá Kulesza, and Camila Nunes. 2011. "On the development of multi-agent systems product lines: A domain engineering process." In *Agent-Oriented Software Engineering X*, 125-139. Springer Berlin Heidelberg.
- Oriche, Aziz, Abderrahman Chekry, and Mohamed Khaldi. 2013. "Intelligent agents for the semantic annotation of educational resources." *International Journal of Soft Computing and Engineering (IJSCE)* 3 (5): 2231-2307.
- Ossher, Harold, William Harrison, and Peri Tarr. 2000. "Software engineering tools and environments: A roadmap." In *Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland*, 261-277. 336569: ACM. doi: 10.1145/336512.336569.
- Palacio, R. R., A. L. Moran, V. M. Gonzalez, and A. Vizcaino. 2009. "Providing Support for Starting Collaboration in Distributed Software Development: A Multi-agent Approach" *Computer Science and Information Engineering, 2009 WRI World Congress on*, doi: 10.1109/CSIE.2009.723.
- Palma, F., H. Farzin, Y. Gueheneuc, and N. Moha. 2012. "Recommendation system for design patterns in software development: An DPR overview" *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, doi: 10.1109/rsse.2012.6233399.

- Panagiotou, Dimitris , and Gregoris Mentzas. 2011a. "Leveraging software reuse with knowledge management in software development." *International Journal of Software Engineering and Knowledge Engineering* 21 (05): 693-723. doi: doi:10.1142/S0218194011005414.
- Panagiotou, Dimitris, and Gregoris Mentzas. 2011b. "Leveraging software reuse with knowledge management in software development." *International Journal of Software Engineering and Knowledge Engineering* 21 (05): 693-723.
- Parhi, Manoranjan, Binod Kumar Pattanayak, and Manas Ranjan Patra. 2015. "A multi-agent-based framework for cloud service description and discovery using ontology." In *Intelligent Computing, Communication and Devices: Proceedings of ICCD 2014, Volume 1*, eds C. Lakhmi Jain, Srikanta Patnaik and Nikhil Ichalkaranje, 337-348. New Delhi: Springer India.
- Paulk, Mark. 2002. "Capability maturity model for software." In *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc.
- Petasis, Georgios, Vangelis Karkaletsis, Georgios Paliouras, Anastasia Krithara, and Elias Zavitsanos. 2011. "Ontology population and enrichment: State of the art." In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution: Bridging the Semantic Gap*, eds Georgios Paliouras, Constantine D. Spyropoulos and George Tsatsaronis, 134-166. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Qiang, Lu, Chen Ming, and Wang Zhiguang. 2008. "A semantic annotation based software knowledges sharing space" *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on*, doi: 10.1109/npc.2008.55.
- Sawadsky, Nicholas, and Gail C. Murphy. 2011. "Fishtail: from task context to source code examples." In *Proceedings of the 1st Workshop on Developing Tools as Plug-ins, Waikiki, Honolulu, HI, USA*, 48-51. 1984722: ACM. doi: 10.1145/1984708.1984722.
- Schwotzer, Thomas, and FHTW Berlin. 2008. "Building context aware P2P systems with the shark framework." In *Fourth International Conference on Topic Maps Research and Applications, Leipzig, Germany*, 157-168.

- Sengupta, Bikram, Satish Chandra, and Vibha Sinha. 2006. "A research agenda for distributed software development." In *Proceedings of the 28th International Conference on Software Engineering, Shanghai, China*, 731-740. 1134402: ACM. doi: 10.1145/1134285.1134402.
- Soo Ling, Lim, and A. Finkelstein. 2012. "StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation." *Software Engineering, IEEE Transactions on* 38 (3): 707-735. doi: 10.1109/tse.2011.36.
- Steinmacher, I., I. S. Wiese, and M. A. Gerosa. 2012. "Recommending mentors to software project newcomers" *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, doi: 10.1109/rsse.2012.6233413.
- Suresh, SS, MM Naidu, S Asha Kiran, and Pune Tathawade. 2011. "Design pattern recommendation system: a methodology, data model and algorithms." In *International Conference on Computational Techniques and Artificial Intelligence (ICCTAI'2011), Pattaya, Thailand 7-8 October 2011*.
- Taglialatela, Andrea, and Francesco Taglino. 2012. "A semantics-based approach to software reuse" *The Fifth Interop-Vlab.It Workshop on Complexity of Systems, Complexity of Interoperability in conjunction with itAIS 2012., Rome, Italy*,
- Teixeira, Lucas O., and Elisa H. M. Huzita. 2014. "DiSEN-AlocaHR: A multi-agent mechanism for human resources allocation in a distributed software development environment." In *Distributed Computing and Artificial Intelligence, 11th International Conference*, eds Sigeru Omatu, Hugues Bersini, M. Juan Corchado, Sara Rodríguez, Paweł Pawlewski and Edgardo Bucciarelli, 227-234. Cham: Springer International Publishing.
- Tichy, Walter F., Sven J. Köerner, and Mathias Landhäußer. 2010. "Creating software models with semantic annotation." In *Proceedings of the Third Workshop on Exploiting Semantic Annotations in Information Retrieval, Toronto, ON, Canada*, 17-18. 1871973: ACM. doi: 10.1145/1871962.1871973.

- Wang, Mei-Hui, Chang-Shing Lee, Kuang-Liang Hsieh, Chin-Yuan Hsu, Giovanni Acampora, and Chong-Ching Chang. 2010. "Ontology-based multi-agents for intelligent healthcare applications." *Journal of Ambient Intelligence and Humanized Computing* 1 (2): 111-131. doi: 10.1007/s12652-010-0011-5.
- Xiang, P. F., A. T. T. Ying, P. Cheng, Y. B. Dang, K. Ehrlich, M. E. Helander, P. M. Matchen et al. 2008. "Ensemble: a recommendation tool for promoting communication in software teams." In *Proceedings of the 2008 International Workshop on Recommendation Systems for Software Engineering, Atlanta, Georgia*, 1-1. 1454259: ACM. doi: 10.1145/1454247.1454259.
- Yang, K, AC Lo, and RJ Steele. 2007. "An Ontology-based Multi-Agent System for the Accommodation Industry." In *The Thirteenth Australasian World Wide Web Conference, AusWeb07, New South Wales, Australia*, 30 June - 4 July 2007. 193-205.  
<http://ausweb.scu.edu.au/aw07/papers/refereed/yang/paper.html>.
- Yang, Kun. 2006. "A Conceptual Framework for Semantic Web-based E-Commerce." Département d'informatique et de génie logiciel, Université Laval.
- Ying, Wier, Pradeep Ray, and Lundy Lewis. 2013. "A methodology for creating ontology-based multi-agent systems with an experiment in financial application development." In *System Sciences (HICSS), 2013 46th Hawaii International Conference on, Wailea, Maui, Hawaii, USA*, January 7-10, 2013. 3397-3406. IEEE.
- Zygkostiots, Zinon, Dimitris Dranidis, and Dimitrios Kourtesis. 2009. "Semantic annotation, publication, and discovery of Java software components: an integrated approach" *The 2nd Workshop on Artificial Intelligence Techniques in Software Engineering (AISEW 2009), Thessaloniki, Greece*: CEUR-WS.org.

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.

# Chapter 3 Problem Definition

## 3.1 Introduction

In Chapter 1, the passive structure of the Software Engineering Ontology has been briefly discussed, and this has provided the motivation for this research. It then highlights the need for a systematic approach to make the Software Engineering Ontology active and explains the concerns associated with the development of a framework for active Software Engineering Ontology. Chapter 2 surveys the literature to provide the necessary background and reviews the existing works to identify gaps in researchers' past attempts to make the Software Engineering Ontology active. As discussed in Chapter 2, even though substantial studies have proposed various approaches to the assimilation and dissemination of knowledge captured in ontologies, they still have shortcomings. First, the approaches to capture software project information are still ineffective in terms of automating the capturing process and instantiating the ontology knowledge base. Second, once the software project information is captured in the ontology, the management of software engineering knowledge is not yet effective. Several approaches in the software engineering domain have been proposed to integrate the agent-based technology with ontologies, also known as the ontology-based multi-agent approach, to facilitate knowledge dissemination. However, these works cover only a specific software development activity or address a particular software development issue. Additionally, most of these works focus only on utilising software agents to access and disseminate knowledge captured in ontologies, but not on using agents to manipulate the knowledge. A software agent has the ability to understand the knowledge defined in ontologies and the knowledge base because it is in machine-readable and processable form. Therefore, it would be a challenge to use the agent to manage the evolution of this knowledge. Furthermore, software development is a knowledge-intensive activity that requires software teams to obtain useful knowledge to facilitate their daily work and for timely decision making. This is particularly critical in multi-site software development settings where project teams are dispersed across multiple sites. Therefore, there is a need for assistive platforms that can

actively help remote team members to manage and share software engineering knowledge in order to enable effective collaborative work during software development activities throughout the life cycle.

In order to address these shortcomings, in this chapter, the problems arising from the passive structure of the Software Engineering Ontology, that are the focus of this thesis, are discussed. These problems will lead to the research issues that will be tackled in order to solve them. The chapter is then concluded with the research methodology and research framework for the development of a systematic solution.

## **3.2 Preliminary Concepts for Active Software Engineering Ontology**

In this section, the definitions of the key concepts used in this thesis are given as follows.

### **Active Software Engineering Ontology**

Definition: Active Software Engineering Ontology refers to the Software Engineering Ontology which is equipped with the active support that can be used to proactively facilitate and assist its users with software engineering knowledge when they are working on a multi-site software development project.

### **Software Project Information**

Definition: Software project information refers to project data (e.g., software documentation, requirements, UML diagrams, source code, bug reports, test cases), project agreement, and project understandings that are produced within a software development project.

### **Software Engineering Domain Knowledge**

Definition: Software engineering domain knowledge or domain knowledge for short is defined as a set of software engineering concepts.

### **Software Engineering Instance Knowledge**

Definition: Software engineering instance knowledge or instance knowledge

for short is defined as software project information that is captured according to the software engineering domain knowledge. The instance knowledge is also known as instantiation.

### **Semantic Annotation**

Definition: Semantic annotation is a process used to capture software engineering knowledge from the software project information based on the concepts described in the Software Engineering Ontology.

### **Ontology Population**

Definition: Ontology population is a process whereby new instances resulting from the semantic annotation process are added in order to enrich the ontology knowledge base.

## **3.3 Passive Software Engineering Ontology Problems**

The nature of existing ontologies including the Software Engineering Ontology is passive, which results in two main challenges regarding knowledge assimilation and knowledge dissemination. The key problems related to these two challenges are identified as follows.

### **3.3.1 Manually Capturing Software Project Information**

*Definition: Manually capturing software project information, in the context of this thesis, refers to the conventional knowledge assimilation approach to manually extract software engineering knowledge from the software project information and map it as instance knowledge to the concepts defined in the Software Engineering Ontology.*

A software development project produces a large volume of software project information. However, this is in syntactic form so their structures are not conducive to an understanding of the semantics, and therefore may create ambiguities (e.g. incorrect or different interpretations). This problem is particularly significant in a

multi-site distributed software development context where project members are geographically dispersed. The ambiguity problem cannot be easily resolved through direct or face-to-face communication in a formal or informal meeting. Furthermore, in this type of setting, information related to the software project is scattered across various, unlinked software repositories. This results in two main challenges. First, this software project information is not readily accessible because of its dispersal in several distributed software repositories. Second, there is a lack of integration among relevant software artefacts. The Software Engineering Ontology has been developed to define common sharable software engineering knowledge and to enable knowledge integration in a multi-site software development environment. Project team members can transform software project information to the concepts defined in the Software Engineering Ontology as instance knowledge. Once the software project information is transformed, it is conceptualised and semantically linked so that it can be used to dispel any ambiguity in remote communication and to enable knowledge sharing among distributed software project development teams.

However, given the huge volume of software project information produced within a software project, the manual approach that relies primarily on software teams' processing, is not practical. Because human resources are sparse, an extensive manual capture of large software project information can be an extremely time-consuming, laborious, tedious, and error-prone task. As a result, such challenges could discourage team members from sharing their knowledge with their colleagues. Some existing approaches are proposed to capture software project information by means of the semantic annotation process. However, several of them are based on manual or semi-automatic approaches which still require additional intervention from project members. Some automatic semantic annotation approaches are introduced but they mostly rely on text analysis techniques for knowledge extraction which may not be suitable for certain types of software artefacts (e.g., source code). Moreover, substantial research efforts have been made to enrich software project information metadata by means of semantic annotation process in order to improve its comprehension and its search ability. However, fewer works have been concerned with populating the annotated resources as the ontology instantiations which can provide a better reasoning capability to derive new knowledge not explicitly defined in the ontology.



### **3.3.2 Lack of Effective Management of the Software Engineering Ontology Instantiations**

*Definition: Lack of effective management of the Software Engineering Ontology instantiations, in the context of this thesis, refers to the complication of obtaining software project information captured in the ontology. It also refers to a lack of effective management of the impact associated with the instance knowledge manipulation which reflects the evolution of software project information.*

The key concepts here are those of knowledge access and manipulation as well as timely awareness which are further defined below.

#### **3.3.2.1 Knowledge Access and manipulation**

*Definition: Knowledge access and manipulation, in the context of this thesis, refers to the ability to obtain or manipulate software project information captured in the Software Engineering Ontology.*

Software engineering knowledge and software project information are captured and organised according to concepts and relations specified by the Software Engineering Ontology. To obtain or manipulate this knowledge, project team members need to know exactly the concepts and relationships to which they are referring. However, it is often the case that a person who utilises the ontology may try to resolve an issue, but he/she cannot translate it into the exact concepts and relations formed in the ontology. As a consequence, the use of the Software Engineering Ontology alone is not an effective means of resolving the issue.

Furthermore, due to the large amount of knowledge captured in the ontology, it could be possible that software teams are not aware of the existence of certain knowledge in the ontology, or even if they are, they might not be able to find it effectively. Therefore, there is the possibility that potentially useful knowledge will be ignored.

#### **3.3.2.2 Timely Awareness**

*Definition: Timely awareness, in the context of this thesis, refers to having knowledge about the current state of other team members' work and achieving the*

*coordination necessary to manage work dependencies during the software development process.*

In a multi-site software development environment, project team members are geographically distributed. Physical and temporal distances make it difficult to maintain group awareness. The Software Engineering Ontology can assist dispersed teams to overcome the issue concerning limited awareness of others' work through the instance knowledge which explicitly specifies the current status of the project. However, team members have to retrieve such information by themselves. If they do not realise that such information is available in the ontology, the issues regarding awareness and remote coordination still remain. An example that can be used to explain this scenario is the management of software evolution. Software project information is subject to continuous evolution according to the changes that can occur over time. In a multi-site distributed software project, this software evolution presents a challenge in terms of maintaining consistency among software development artefacts. Even though the Software Engineering Ontology is utilised in this project, with its passive structure, it does not make relevant team members aware of the change. If dispersed team members are not promptly made aware of what other people at different sites are doing, such as making a change, inconsistencies related to artefacts and remote coordination can occur. With respect to this example, in order to be of most benefit, awareness must be timely enough to allow project team members to fully understand what is going on at the other sites and react to them at the appropriate time (Tekinerdogan et al. 2012). Accordingly, timely awareness achieved by means of just-in-time knowledge about coordination needs is important, particularly in a multi-site distributed software development setting.

### **3.3.3 Availability of Active Platforms for Multi-site Software Development Environments**

*Definition: Active platforms for multi-site software development environments, in the context of this thesis, refers to the availability and suitability of current software development tools and technologies that can assist distributed software project teams to effectively manage and share software engineering knowledge when they are engaged in software development activities throughout the*

*software development life cycle.*

The key concepts of active platforms for multi-site distributed software development environments are knowledge management, communication and coordination.

### **3.3.3.1 Knowledge Management**

*Definition: Knowledge management, in the context of this thesis, refers to the ability to manage and share software project information contained in the Software Engineering Ontology through a series of stages ranging from capture, search and dissemination, through to maintenance.*

Software development is a knowledge- and collaborative-intensive process, the success of which depends on the effectiveness with which software engineering knowledge is managed and shared among project team members (Kavitha and Ahmed 2011). This is particularly critical in a multi-site distributed software development context where the collaboration is affected by physical and temporal distance. The passive structure of software Engineering Ontology imposes limitations, such as the need for manual knowledge capture and passive knowledge distribution, on effective knowledge management for knowledge sharing. Furthermore, following our review of works, presented in Chapter 2, it is evident that there is a lack of ready-to-use assistive platform or tool that can actively assist remote team members to manage and share software engineering knowledge in order to facilitate collaborative work that covers various software development activities throughout the software life cycle.

### **3.3.3.2 Communication**

*Definition: Communication, in the context of this thesis, refers to the effective and efficient exchange of information among software development teams. Communication is considered effective if it is clear to the receivers. Efficient communication is specifically targeted and timely to the receivers so that they can use it to facilitate their work or to make decisions at the appropriate time.*

Communication in multi-site distributed software development settings

should be effective and efficient in order to overcome the barriers that are imposed by long distance and different time-zones (Alqhtani and Qureshi 2014). The Software Engineering Ontology defines common shareable software engineering knowledge. It is used as a solution for knowledge representation in order to reduce miscommunication, misunderstanding, and misinterpretation of issues in multi-site software development environments. However, because of its passive structure, certain crucial elements of efficient communication are still missing: i) the communication provided by the ontology is not efficient in the sense that it cannot target the relevant people, and ii) the communication provided by the ontology needs to be timely which means that it should be available and just-in-time for project team members to facilitate their work or to make appropriate decisions.

### **3.3.3.3 Coordination**

*Definition: Coordination, in the context of this thesis, refers to the ability to manage dependencies among tasks and task holders or to maintain the consistency of software products.*

In a software development project, coordination needs arise due to dependencies among tasks and software artefacts. Coordination become more complex as the degree of distribution of team members increases and it can lead to a lack of team awareness. However, because of decreased communication, remote software teams might not, within an appropriate time frame, obtain information on what other teams at different sites are doing; thus, they may not be aware of work dependencies that can cause coordination problems. Although the Software Engineering Ontology can be used to facilitate remote coordination by making project tasks explicit, because of its passive structure, team members have to rely on their own efforts to meet their coordination needs. Existing methods and tools also have limitations in terms of providing timely information to ensure efficient coordination among software development teams (Blincoe, Valetto and Damian 2015). For example, in (Sengupta, Chandra and Sinha 2006), the authors point out that existing requirements management tools do not provide adequate support for remote coordination and collaboration. When a requirement is changed, the information is not propagated to relevant project teams in a timely and proactive manner to enable them to be aware of coordination needs. Consequently, a

discrepancy occurs regarding distant teams' understanding of the project, subsequently leading to software quality or productivity problems.

### **3.4 Underlying Research Issues**

In the previous section, the problems associated with the passive structure of the Software Engineering Ontology have been identified. In this section, the underlying research issues that need to be addressed in order to solve the aforementioned problems, or in other words, to make the ontology active are discussed. These four issues are related to:

- automated knowledge capture of software project information
- Software Engineering Ontology instantiations management
- active platforms for multi-site software development environments
- evaluation for prototyping proof-of-concept

In the next section, these research issues are clearly defined and explained in detail.

#### **3.4.1 Research Issue 1: Automated Knowledge Capture of Software Project Information**

*Automated knowledge capture of software project information, in the context of the framework for active software Engineering Ontology, is defined as automatically capturing software engineering knowledge from software project information based on the concepts defined in the Software Engineering Ontology and then populating it in the ontology knowledge base. The purpose is to enable knowledge sharing by making the knowledge available to other project team members. In software development project, large amounts of information are produced and stored in various locations. This poses two main challenges. First, this software project information is not readily accessible because of its dispersed nature.*

Second, there is a lack of integration among relevant software artefacts. In the literature, some existing research attempted to address these issues by capturing software project information and structuring it in conceptualised form. However, most of the proposals are based on manual or semi-automatic approaches. Thus, considerable effort is still required from software team members when undertaking this work. The lack of an effective approach to capture software engineering knowledge has raised the research issue for the automated knowledge capture of software project information. An effective approach is needed that can automatically capture software project information and populate it in the ontology. Once this information is captured and organised well, relevant concepts can be interlinked. Consequently, related software project information will not appear in isolation, but will be included within a large group of related information that is easily and readily accessible. Moreover, the software agents will be able to understand and make use of this knowledge. They can provide useful information to distributed project teams in order to dispel any ambiguity resulting from remote or inadequate communication, to address major software development issues, and to facilitate effective and efficient coordination.

### **3.4.2 Research Issue 2: Software Engineering Ontology Instantiations Management**

*Software Engineering Ontology instantiations management, in the context of the framework for active Software Engineering Ontology, is defined as accessing software engineering knowledge and manipulating instance knowledge.*

Software Engineering Ontology instantiations management are defined below.

- Knowledge retrieval

*Knowledge retrieval refers to the ability to identify and extract instance knowledge captured in the ontology repository.* The process of knowledge retrieval includes query, search, and proactive monitoring of software project information in order to identify a possible deviation or disruptive event before an actual issue arises. A proactive notification is

provided to corresponding team members in a push-based delivery mode.

- Instance knowledge manipulation

*Instance knowledge manipulation refers to the ability to add, modify, and delete instance knowledge and then identify the potential impact of the change made to the instantiations based on the relationship defined in the Software Engineering Ontology.* Notifications are pushed to relevant team members to make them aware of the change. In a multi-site software development environment, communication and coordination are critical challenges because of physical and temporal distances. Team awareness with respect to the change made by other members at different sites is important. Therefore, proactive and timely knowledge delivery to avoid any confusion and integration risks can help to reduce the distance barrier.

### **3.4.3 Research Issue 3: Active Platforms for Multi-site Software Development Environments**

*Active platforms for multi-site software development environments, in the context of the framework for active Software Engineering Ontology, refer to platforms used to operate the application programs that assist collaborative software teams to manage and share software engineering knowledge throughout the software development life cycle in a multi-site software development environment.*

Software development is considered as a knowledge-intensive, complex and collaborative activity. The quality of a software product largely depends on the quality of the software process which is the result of the activities conducted throughout the software development process. In addition, it involves the integration of knowledge from multiple sources which is constantly evolving according to the changing needs of customers and business environments. Therefore, software development knowledge should be formalised, stored, distributed, and easily shared among project team members. A knowledge management approach can be used to support these activities and improve the development process so that better software quality and productivity can be achieved. Additionally, an effective knowledge management approach can also help to address the challenges faced in multi-site

distributed software development regarding remote communication and coordination. These challenges could be major causes of project delay or failure. In a distributed setting, project team members frequently work on tasks in parallel. Technical dependencies between software development tasks mean that team members must have efficient coordination. Timely and efficient awareness of coordination needs is important and critical in a globally distributed project (Blincoe, Valetto and Damian 2015).

Software development is a collaborative activity in which team members interact with each other. Most software development tasks require collaboration between team members who are probably not physically present at the same location. In addition, coordination in dispersed teams becomes more difficult as problems arise from remote communication and a lack of group awareness. Remote coordination requires more people to participate, resulting in delays. When there is a change, it may involve several people from multiple sites and increases the time needed for development tasks. Several mechanisms (e.g. project reviews, conference calls, progress reports) are required to minimise task dependencies in a multi-site distributed software development environment. Therefore, collaborative tools must support the software development process in order to allow monitoring activities and managing of dependencies, notifications and implementation of corrective measures (Jiménez, Piattini and Vizcaíno 2009).

Hence, there is a need for active platforms for multi-site software development environments that can enable effective knowledge management and sharing in order to facilitate remote collaboration among distributed project members. In this research, effective knowledge management comprises the following activities:

- Knowledge capture

Knowledge capture refers to the ability to automate the process of transforming software project information knowledge into a conceptualised and semantically rich form according to the concepts defined in the Software Engineering Ontology, and then instantiating it into the Software Engineering Ontology knowledge base as new instances.



- Knowledge search

Knowledge search refers to the process of accessing and retrieving knowledge captured in the Software Engineering Ontology. This activity is initiated by a user.

- Knowledge dissemination

Knowledge dissemination is different from knowledge search in the sense that it is initiated by the system and does not require a user to explicitly make a request (Natali and Falbo 2002). In other words, it refers to the proactive delivery of software engineering knowledge and project information to software team members who may need it.

- Knowledge maintenance

Knowledge maintenance is the process of adding, modifying, or deleting particular knowledge instances.

Remote software teams can utilise such platforms to support their collaborative work throughout the various phases of the software development life cycle in order to minimise the challenges related to temporal and physical distances. The platforms are intended to improve the effectiveness and efficiency of communication and coordination, and to provide useful information to address software development issues and to support decision making.

### **3.5 Research Methodology**

In this section, the research methodology that this research will follow to ensure that the framework development is based on a quality scientific method, is described. Design science research is chosen as a research paradigm as it is the most appropriate approach for investigating problems in the domain of Information Systems research (Hevner et al. 2004). Hevner's design science research guidelines (Hevner et al. 2004) and Peffers's design science research methodology process model (Peffers et al. 2007) are incorporated to present a complete research

methodology that addresses the research issues and guides the framework development.

### **3.5.1 Overview of Design Science Research Paradigm**

Over the years, many researchers in the Information Systems research community have adopted design science research approach and have acknowledged the value of design science as an information systems research paradigm (Peffer et al. 2007; Gregor and Hevner 2013; Walls, Widmeyer and El Sawy 1992). Hevner and Chatterjee (2010) point out that the design science research paradigm is very relevant to Information Systems research and it supports a pragmatic research paradigm that focuses on the innovation of artefacts to resolve real-world problems. Hevner et al. (2004) propose a set of guidelines for conducting and evaluating good design science research. In this section, each guideline is addressed in terms of what is proposed and how it is implemented in this research.

#### **Guideline 1: Design as an Artefact**

*“Design-science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation”* (Hevner et al. 2004, 83).

The artefacts of this research include the conceptual framework and architecture designed for active Software Engineering Ontology (proposed in Chapter 5), the frameworks for assimilating and disseminating knowledge captured in the Software Engineering Ontology (proposed in Chapter 6 and Chapter 7), and the active platforms to facilitate collaborative work in a multi-site software development environment (proposed in Chapter 8).

#### **Guideline 2: Problem Relevance**

*“The objective of design-science research is to develop technology-based solutions to important and relevant business problems”* (Hevner et al. 2004, 83).

This guideline addresses the need for problem relevance. In this thesis, the problems associated with the passive structure of the Software Engineering Ontology are addressed in Section 3.3. In brief, the challenges arising from remote

communication and coordination in multi-site software development project have been formerly addressed by the use of the Software Engineering Ontology. However, because of its passive structure, the assimilation and dissemination of knowledge still requires a great amount of effort from software teams. As a result, what is needed is active support that can help team members to effectively manage and share software engineering knowledge captured in the ontology in order to facilitate remote collaborative work through effective and efficient communication and coordination.

### **Guideline 3: Design Evaluation**

*“The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods”* (Hevner et al. 2004, 83).

This guideline emphasises that the evaluation of the designed artefacts is an essential component of the research process. Venable, Pries-Heje, and Baskerville (2016) consider the evaluation of the design artefacts and theories to be a key activity in the Design Science Research approach. They point out that it can ensure the rigour of the research to achieve research objectives, and provides feedback for further improvement.

In this research, the proposed framework and platforms are evaluated through the prototype system as proof-of-concept experiments. A framework for evaluation in design science research addressed by Venable, Pries-Heje, and Baskerville (2012) is adopted. Experiments based on case studies in the literature are conducted to evaluate the effectiveness and efficiency of the proposed framework through the implemented prototype which is presented in Chapter 9.

### **Guideline 4: Research Contributions**

*“Effective design-science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies”* (Hevner et al. 2004, 83).

This guideline highlights the need for a science research project to make several clear contributions. In this research, a framework for active Software Engineering Ontology is intended to provide active support to assist collaborative software development teams to effectively access, manage and share software

engineering knowledge as well as project information to enable effective and efficient communication and coordination among teams. Furthermore, the proposed framework and platforms could serve as a reference model for the development of similar systems in other domains.

### **Guideline 5: Research Rigor**

*“Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact”* (Hevner et al. 2004, 83).

The design and construction of the research framework for active Software Engineering Ontology is presented in Chapter 5. They are based on rigorous methodologies for agent-oriented software engineering. In Chapter 6 and Chapter 7, the frameworks for solution implementation and outcomes are described in detail to demonstrate the applicability in a problem domain. To evaluate the research framework, a prototype system is implemented and evaluated through several existing case studies found in the literature according to the framework requirements in Chapter 9.

### **Guideline 6: Design as a Search Process**

*“The search for an effective artefact requires utilising available means to reach desired ends while satisfying laws in the problem environment”* (Hevner et al. 2004, 83).

Hevner et al. (2004) state that design science is an iterative process used to search for the best or most effective solution for realistic Information Systems problems. Each step requires the search process in order to identify appropriate means to reach the desired ends. In Chapter 2, the extensive literature review related to this research is presented. The review shows the state of the art ontology-based semantic annotation, ontology-based multi-agent systems, and assistive systems in software engineering in order to identify the gaps that this research can address. These gaps and the problems associated with the passive structure of the Software Engineering Ontology are combined, leading to an identification of the research issues in Chapter 3 and the proposed solution requirements in Chapter 4. In Chapter 5, the existing agent-oriented software engineering methodologies are reviewed and

the most appropriate methodologies are selected to implement the framework.

### **Guideline 7: Communication of Research**

*“Design science research must be presented effectively both to technology-oriented as well as management-oriented audiences”* (Hevner et al. 2004, 83).

This guideline addresses the importance of disseminating new knowledge and communicating this to audiences who have different perspectives and information needs. In Chapters 5-9 of this thesis, technology-oriented audiences are provided with sufficient details regarding how the research frameworks are designed, implemented, and evaluated. Moreover, in these chapters, management-oriented audiences are given information regarding practical applications to demonstrate how the framework can help them manage software development information effectively within a multi-site distributed software development setting so that they can decide whether or not the proposed framework should be adopted given their specific organisation context.

#### **3.5.2 Choice of Design Science Research Framework**

Peppers et al. (2007) propose a design science research framework (DSRM) for the development and presentation of design science research in Information Systems. They provided a process model and a mental model to carry out and present the design science research. Their design science methodology process model involves six activities as shown in Figure 3-1.

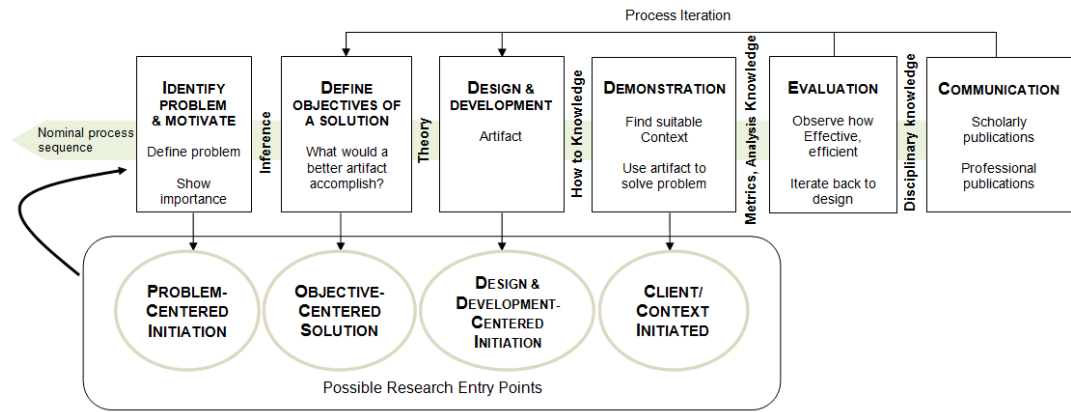


Figure 3-1: Design Science Research Methodology Process Model (Peppers et al. 2007, 54)

There are four research entry points depending on the nature of the project. If the research is a problem-centred initiation the idea for which stemmed from the observation of the problem or from a suggested future research direction, the entry point is activity 1 (identify problem and motivate). The entry point for an objective-centred solution for an industry project is activity 2 (Objective-centred solution). A design and development-centred initiation could be from an existing artefact used to solve a different problem or it might be considered as a similar idea. This kind of research will start with activity 3 (Design and development). Lastly, a client/context-initiated solution could be from observing a practical solution such as a real world project. The entry point will be activity 4 (Demonstration).

In this research, the entry point is “Problem-centred initiation” because the idea comes from the observation of the problem. The lack of active support to assimilate and disseminate knowledge captured in the Software Engineering Ontology could hinder the use of the ontology. Therefore, there is the need for a framework that makes the Software Engineering Ontology active. Hence, the entry point for this research is activity 1 (Identify problem and motivate) and six other activities follow as explained below.

### Activity 1: Identify problem and motivate

The Software Engineering Ontology was developed to clarify the software engineering concepts and project information as well as to enable knowledge sharing

among project team members who are geographically located across multiple software development sites (Wongthongtham et al. 2009). However, the current Software Engineering Ontology has a passive structure in regard to knowledge assimilation and knowledge dissemination. In order to capture a large amount of software project information into a conceptualised and semantically rich form, a great amount of effort is required from software teams to manually transform this information into knowledge captured in the Software Engineering Ontology. As a consequence, the manual approach could discourage them from sharing their knowledge with other team members. Its passive structure also raises the need to know exactly the concepts and relationships to which software team member are referring in the ontology. Otherwise, they may not be able to access or manipulate the knowledge required. Therefore, what is needed is active support that can help software teams to effectively manage and share knowledge captured in the Software Engineering Ontology. The ultimate goal is to enable software project teams to maintain collaborative work through effective and efficient communication and coordination.

### **Activity 2: Define objectives of a solution**

The primary objective of this thesis is to *develop a framework for active Software Engineering Ontology that can provide active support in order to assist software development team members with software engineering knowledge when they are working on software development projects*. The research objective can be segmented into the following sub-objectives:

Sub-objective 1: To develop a framework for active Software Engineering Ontology specifically focusing on the ontology deployment phase. The framework is intended to address challenges resulted from the passive structure of the Software Engineering Ontology in regard to knowledge assimilation and knowledge dissemination. This is addressed in Chapter 5.

Sub-objective 2: To develop an approach to automate knowledge capture of software project information that is seamlessly integrated into the software development process. The proposed approach is elaborated and discussed in Chapter 6.

Sub-objective 3: To develop an approach to access and manage software engineering knowledge captured in the Software Engineering Ontology effectively. The proposed approach is elaborated and discussed in Chapter 7.

Sub-objective 4: To develop active platforms for multi-site software development environments that can support remote project team members to manage and share software engineering knowledge throughout various phases of the software life cycle. The development of these platforms is presented in Chapter 8.

Sub-objective 5: To evaluate the effectiveness and efficiency of the proposed framework and platforms based on existing case studies found in the literature through the prototype system used as proof-of-concept experiments. The evaluation is presented in Chapter 9.

### **Activity 3: Design and development**

During the design and development process, the objectives of the proposed solutions are transformed to determine the artefact's functionality and its architecture. In this thesis, the design of the active Software Engineering Ontology conceptual framework and its architecture is described in Chapter 5, while the design and development of the frameworks and platforms corresponding to the research sub-objectives are presented in Chapters 6-8.

### **Activity 4: Demonstration**

To demonstrate the feasibility of the proposed framework and platforms to solve one or more instances of the problem, the working prototype system is designed and implemented. The prototype system is used to demonstrate the feasibility of the framework and platforms to facilitate collaborative software development teams through effective and efficient communication and coordination. The demonstrations are presented in Chapters 6-8.

### **Activity 5: Evaluation**

This activity observes and measures how well the proposed framework can provide solutions for the research issues. The framework requirements are compared with the results from the prototype demonstration. Three quantitative parameters,



namely, time to complete the task, number of team members involved in the task, and number of team members' actions, are used to measure the efficiency of the framework and platforms in assisting software team members to work collaboratively on software development projects. The evaluation of the proposed framework is presented in Chapter 9.

### **Activity 6: Communication**

Various parts of this research have been presented at several international conferences and have been published in the conference proceedings and in peer-reviewed journals throughout the research process as follows.

- The research motivation and the preliminary conceptual framework for the active Software Engineering Ontology were presented at *The International Conference on E-Technologies and Business on the Web (EBW2013)* and published in the conference proceedings (*Pakdeetrakulwong and Wongthongtham 2013b*). An extended version of this paper was submitted to the *International Journal of Digital Information and Wireless Communications (IJDIWC)* (*Pakdeetrakulwong and Wongthongtham 2013a*).
- A survey of existing systems/tools related to assistive systems for software engineering was presented in *The 9<sup>th</sup> International Conference for Internet Technology and Secured Transactions (ICITST 2014)* and published in the conference proceedings (*Pakdeetrakulwong, Wongthongtham and Siricharoen 2014*).
- The design of the conceptual framework focusing on overall system architecture and inter-agent interactions was presented at *The 3rd Annual Conference on Engineering and Information Technology* and published in the conference proceedings (*Pakdeetrakulwong and Wongthongtham 2015*).
- The design and implementation of the framework for the Software Engineering Ontology instantiations management were presented at *The 24th Australasian Software Engineering Conference (ASWEC 2015)*, and

published in the conference proceedings (Pakdeetrakulwong, Wongthongtham and Khan 2015).

- The extended version of the above paper was published in *The Journal of Mobile Network and Application*. The design, implementation, and evaluation of the framework and platforms for the Software Engineering Ontology instantiations management were presented. The prototype demonstrated the feasibility of using the framework and platforms to support requirements traceability tasks within a multi-site software development project. The result was discussed and compared with existing works in the literature (Pakdeetrakulwong, Wongthongtham, Siricharoen, et al. 2016).
- The design, implementation, and evaluation of the framework and platforms for capturing software project information were presented at *The 4th International Conference on Enterprise Systems* and published in the conference proceedings (Pakdeetrakulwong, Wongthongtham, Sae-Lim, et al. 2016).

This thesis is also the main means of communication and is intended for academic audiences. Table 3-1 shows the thesis chapters that are mapped with the set of activities of the design science research methodology process model.

Table 3-1: Set of activities of DSRM process mapped with the thesis chapters

Activity	Thesis Chapters
1. Identify problem and motivate	Chapters 1-3
2. Define objectives of a solution	Chapter 1
3. Design and development	Chapters 5-8
4. Demonstration	Chapters 6-8
5. Evaluation	Chapter 9
6. Communication	Two journal articles, one book chapter, five conference papers, and one thesis (this thesis)

### 3.6 Conclusion

In this chapter, the key terminologies used in this chapter and throughout the thesis are defined. The problems facing the passive structure of the Software Engineering Ontology are also addressed. They comprise the manual capture of software project information, the lack of effective management of the ontology instantiations, and the current lack of active platforms to support multi-site software project teams. Then the research issues which need to be addressed as a basis for the new framework in order to produce the solution to these problems are proposed. Finally, a summary of research approaches is given. A design science research methodology is chosen as the preferred option to address these research issues and for the development of the proposed solution.

In the next chapter, the key requirements of any solution development are described. The conceptual solution to the issues addressed in this chapter is presented. An ontology-based multi-agent approach is proposed as the conceptual solution to develop the framework to make the Software Engineering Ontology active. The reason for the choice of the research conceptual solution is also given.

### 3.7 References

- Alqhtani, Masha'el Saeed, and M Rizwan Jameel Qureshi. 2014. "A proposal to improve communication between distributed development teams." *International Journal of Intelligent Systems and Applications* 6 (12): 34-39. doi: <http://dx.doi.org/10.5815/ijisa.2014.12.05>.
- Blincoe, Kelly, Giuseppe Valetto, and Daniela Damian. 2015. "Facilitating coordination between software developers: A study and techniques for timely and efficient recommendations." *Software Engineering, IEEE Transactions on* 41 (10): 969-985.
- Gregor, Shirley, and Alan R Hevner. 2013. "Positioning and presenting design science research for maximum impact." *MIS Quarterly* 37 (2): 337-355.

- Hevner, Alan, and Samir Chatterjee. 2010. "Design science research in Information Systems." In *Design Research in Information Systems: Theory and Practice*, 9-22. Boston, MA: Springer US.
- Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. "Design science in Information Systems research." *MIS Quarterly* 28 (1): 75-105.
- Jiménez, Miguel, Mario Piattini, and Aurora Vizcaíno. 2009. "Challenges and improvements in distributed software development: a systematic review." *Advances in Software Engineering* 2009: 3.
- Kavitha, R. K., and M. S. Irfan Ahmed. 2011. "A knowledge management framework for agile software development teams" *2011 International Conference on Process Automation, Control and Computing*, doi: 10.1109/PACC.2011.5978877.
- Natali, Ana Candida Cruz, and RA Falbo. 2002. "Knowledge management in software engineering environments" *Proceedings of the XVI Brazilian Symposium on Software Engineering (SBES'2002)*,
- Pakdeetrakulwong, U., P. Wongthongtham, and W. V. Siricharoen. 2014. "Recommendation systems for software engineering: A survey from software development life cycle phase perspective." In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*, 8-10 Dec. 2014. 137-142. IEEE. doi: 10.1109/ICITST.2014.7038793.
- Pakdeetrakulwong, Udsanee, and Pornpit Wongthongtham. 2013a. "State of the art of a multi-agent based recommender system for active software engineering ontology." *International Journal of Digital Information and Wireless Communications (IJDIWC)* 3 (4): 363-376.
- . 2013b. "Towards active software engineering ontology" *The International Conference on E-Technologies and Business on the Web (EBW2013)*, Bangkok, Thailand: The Society of Digital Information and Wireless Communication.

- . 2015. "Use and design of ontology-based multi-agent system for multi-site software development environment" *The 3rd Annual Conference on Engineering and Information Technology, Osaka, Japan: ACEAIT*.
- Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, and Naveed Khan. 2015. "An ontology-based multi-agent system to support requirements traceability in multi-site software development environment" *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, Volume II, Adelaide, SA, Australia*, 2811700: ACM. doi: 10.1145/2811681.2811700.
- Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, Suksawat Sae-Lim, and Hassan Marzooq Naqvi. 2016. "SEOMAS: An ontology-based multi-agent approach for capturing semantics of software project information" *The 4th International Conference on Enterprise Systems, Melbourne, Victoria, Australia: IEEE Computer Society*.
- Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, Waralak V. Siricharoen, and Naveed Khan. 2016. "An ontology based multi-agent system for active software engineering ontology." *Mobile Networks and Applications* 21 (1): 65-88. doi: 10.1007/s11036-016-0684-x.
- Peppers, Ken, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. "A design science research methodology for Information systems research." *Journal of Management Information Systems* 24 (3): 45-77.
- Sengupta, Bikram, Satish Chandra, and Vibha Sinha. 2006. "A research agenda for distributed software development." In *Proceedings of the 28th International Conference on Software Engineering, Shanghai, China*, 731-740. 1134402: ACM. doi: 10.1145/1134285.1134402.
- Tekinerdogan, Bedir, Semih Cetin, Muhammad Ali Babar, Patricia Lago, and Juho Mki. 2012. "Architecting in global software engineering." *SIGSOFT Software Engineering Notes* 37 (1): 1-7. doi: 10.1145/2088883.2088900.
- Venable, John, Jan Pries-Heje, and Richard Baskerville. 2012. "A comprehensive framework for evaluation in design science research." In *Design Science*

*Research in Information Systems. Advances in Theory and Practice*, 423-438.  
Springer.

———. 2016. "FEDS: a framework for evaluation in design science research."  
*European Journal of Information Systems* 25 (1): 77-89. doi:  
10.1057/ejis.2014.36.

Walls, Joseph G, George R Widmeyer, and Omar A El Sawy. 1992. "Building an  
information system design theory for vigilant EIS." *Information systems  
research* 3 (1): 36-59.

Wongthongtham, P., E. Chang, T.S. Dillon, and I. Sommerville. 2009. "Development  
of a software engineering ontology for multi-site software development."  
*IEEE Transactions on Knowledge and Data Engineering* 21 (8): 1205-1217.  
doi: 10.1109/TKDE.2008.209.

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.

# **Chapter 4   Ontology-based Multi-agent Approach Solution Proposal**

## **4.1   Introduction**

In the previous chapter, the problems facing the passive Software Engineering Ontology and their underlying research issues were identified. The research issues comprise: i) automated knowledge capture of software project information; ii) Software Engineering Ontology instantiations management; and iii) active platforms for multi-site software development environments.

This chapter addresses the identified issues and suggests solution proposals for each of the identified research issues. First, an overview of the three key requirements of the solution development is presented. Then the current available technologies that are able to satisfy the solution requirements are reviewed. The agent-based technology is considered to be able to provide active components according to their features, namely, autonomy, reactivity, pro-activeness, and social ability. It is also found that the multi-agent system consisting of multiple agents that act in an environment to achieve a common goal is a promising technology for the realisation of the distributed collaborative systems. Therefore, a careful study is carried out to examine the feasibility of integrating the multi-agent system and the ontology also known as the ontology-based multi-agent approach as a basis for a possible solution for a framework of active Software Engineering Ontology.

## 4.2 Solution Requirements

The aim of this research is to develop a framework for active software Engineering Ontology using a design science research approach as discussed in Chapter 3. Three key research issues are identified and any new solution for active Software Engineering Ontology should address and provide a solution for these key issues. In this section, four solution requirements for the proposed framework are given as follows.

- Automated Knowledge Capture of Software Project Information
- Software Engineering Ontology Instantiations Management
- Active Platforms for Multi-site Software Development Environments
- Framework Evaluation

### 4.2.1 Requirement 1: Requirement of Automated Knowledge Capture of Software Project Information

In order to develop a framework for making the Software Engineering Ontology active, the first requirement is the assimilation of software development knowledge into the ontology knowledge base during the development process with very minimal additional effort from team members.

Because software project information is generally in syntactic form which can create ambiguity issues during the remote communication, the information needs to be transformed into a conceptually formalised and organised form in order to facilitate the sharing of understanding and the semantic linkage with other relevant resources. The Software Engineering Ontology can be utilised to provide the software engineering domain knowledge for the semantic annotation process. However, due to the large volume of software project information generated, the capturing process needs to be automated and transparently integrated into the software development process in order to avoid unnecessary complexity and to minimise the workload of software team members, thereby encouraging them to share the knowledge with others. Once the software project information is semantically annotated and populated as the ontology instantiations, it can be subsequently used to clarify any ambiguity resulting from remote communication



and to enhance the team's coordination. Moreover, this knowledge is in machine understandable form, enabling it to be understood by the software agents. They can make use of this knowledge to support distributed project teams to manage project issues or to suggest solutions and provide expertise to address issues that are raised.

#### **4.2.2 Requirement 2: Requirement of Software Engineering Ontology Instantiations Management**

Once the software project information has been captured and conceptualised in the Software Engineering Ontology as instantiations, the knowledge needs to be managed. The management of the instantiations includes retrieving knowledge, adding new instantiations, and modifying or deleting existing instantiations.

In order to provide access to knowledge, the knowledge retrieved from the Software Engineering Ontology can be delivered to team members by means of information-pull or information-push mode. For the information-pull mode, project teams initiate the search or query the semantic linked project information captured in the ontology explicitly. Nonetheless, it might be the case that they are not aware of the existence of certain knowledge shared in the ontology or even if they are, they might not be able to search for it effectively. Therefore, the management of Software Engineering Ontology instantiations can provide active support by delivering information that is potentially useful without requiring users to explicitly make a request (information-push). This information could be obtained by processing the existing knowledge in the ontology.

Because software development knowledge is always evolving as a result of fluctuating requirements and technology advances, it is critical that the instantiations management deal with the instance knowledge manipulation (i.e., add, delete, modify) to reflect the software project evolution. Moreover, because of the geographical and temporal distances in a multi-site software development setting, when the instantiations are manipulated, the Software Engineering Ontology instantiations management should include proactive features to support remote communication in order to maintain team members' awareness and coordination. For example, relevant distributed team members who are dispersed geographically should promptly be informed about any changes made at different sites that possibly

affect their workspace.

#### **4.2.3 Requirement 3: Requirement of Active Platforms for Multi-site Software Development Environments**

Active platforms for multi-site software development environments are required as main forms to enable effective knowledge assimilation and knowledge dissemination for the proposed active Software Engineering Ontology framework. They are intended to enable the communication and coordination in a multi-site software development project to be done in a more efficient and productive manner. The platforms are required to capture knowledge generated during the software development process and populate it in the ontology repository in order to enable knowledge sharing and reuse. Even though each software project is different, sharing similar knowledge could be useful for team members to expedite their tasks. Regarding knowledge dissemination, the platforms can offer both passive and active means of communication since the user can either query the required information, or the platforms can provide knowledge that is likely to be relevant to the users' tasks. The platforms are also equipped with a proactive monitoring and notification service in order to actively inform team members about useful information or alert them to any event that could change the project's as-planned schedule or affect the project performance. The platforms play significant roles in supporting software development activities across the software development life cycle in order to reduce the physical and temporal distance barriers faced by remote team members.

#### **4.2.4 Requirement 4: Requirement of Framework Evaluation**

Automated knowledge capture of software project information enables software project information to be automatically captured into the ontology knowledge base. Software Engineering Knowledge instantiations management enables the captured knowledge to be accessed and manipulated more effectively. Active platforms are used to facilitate effective software engineering knowledge management and sharing to enable effective and efficient communication and coordination among software development teams within a multi-site software development project. Integrating all ideas, processing them into practice, and

evaluating them by means of a working prototype system as a proof-of-concept are required for the active Software Engineering Ontology framework. The prototype systems will be used to demonstrate the realisation and feasibility of the proposed framework as well as to check whether the framework fulfils its purpose and solves the key issues associated with passive Software Engineering Ontology.

## 4.3 Agent-Based Technology

### 4.3.1 Software Agent

The agent-based technology has attracted considerable attention and become active research areas in recent years. In addition, the advent of the Semantic Web technology has provided the underlying infrastructure that allows software agents to process data and perform sophisticated tasks on behalf of users. Regarding the term “agent”, the following definition is widely accepted:

*“An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its delegated objectives.”* (Wooldridge 2009, 21)

Accordingly, the key properties of an agent are as follows (Wooldridge and Jennings 1995; Jennings 2000)

- **Autonomy:** agents encapsulate some state and make decisions on what to do based on this state without the direct intervention of humans or others.
- **Reactivity:** agents are situated in an environment and are able to perceive this environment through their sensors. Then, through effectors, they respond in a timely fashion to changes that occur in their environment.
- **Pro-activeness:** agents do not simply act in response to their environment. They are able to exhibit goal-directed behaviour by taking the initiative.
- **Social ability:** agents are able to cooperate with humans and other agents in

order to achieve their design objectives.

Software agents can be differentiated from traditional software applications in terms of certain characteristics. The differences between traditional software applications and software agents are presented in Table 4-1 which is adapted from (Turban, Sharda and Delen 2010).

Table 4-1: Differences between traditional software applications and software agents (adapted from Turban, Sharda and Delen 2010)

<b>Characteristics</b>	<b>Traditional software applications</b>	<b>Software agents</b>
<b>Nature</b>	Static	Dynamic
<b>Autonomy</b>	Follow instructions	Be able to perform tasks without direct control, or at least with the minimum of human intervention
<b>Manipulation</b>	User initiates every action	Sense the environment and react autonomously
<b>Interactivity</b>	Non-interactivity	Can interact with other agents, humans, or software programs
<b>Temporal continuity</b>	Terminate when process is complete	Continue to run over time (persistent)
<b>Concurrency</b>	Generate process in one dedicated server with limited processing power	Dispatch simultaneously to accomplish several parts of a task in parallel
<b>Mobility</b>	Stay in one place	Be able to travel from one machine to another

From Table 4-1, it is clear that the software agents are different from traditional software applications. Moreover, compared with the object-oriented paradigm, the agent technology can be considered as a descendant that improves the nature of passive objects with the notion of autonomous actors (Braubach et al. 2005). In contrast to simple objects with methods that can be invoked by other objects, an agent communicates with other agents by means of message-passing. In

addition, it can act proactively to accomplish its individual goal. Agents can work as stand-alone entities to perform particular tasks on behalf of a user. However, many agent applications are based on environments that contain multiple agents collaboratively working together as a group. This is also known as a multi-agent system.

### **4.3.2 Multi-Agent System**

Even though an individual agent can perform a task on behalf of a single user, its capacity is limited by its knowledge and resources. Thus, agents are usually implemented in a multi-agent context. A multi-agent system (MAS) consists of multiple agents acting in an environment to achieve a common goal or their individual goals (Ye, Zhang and Vasilakos 2016). There is an increasing interest in MAS research because of its significant advantages including its ability to solve problems that may be too large for a single agent. MAS allows a complex task to be decomposed into sub-tasks, each of which is then assigned to an individual agent to undertake independently, but which can be supported by a knowledge base. They have distributed architectures which control distribution by utilising the mechanisms of cooperation and coordination.

MAS have various advantages over a single agent, such as reliability and robustness, modularity, scalability, adaptability, concurrency, parallelism, and dynamism (Elamy 2005). When a system is implemented based on MAS architecture, it is easy to add a new functionality or to modify an existing functionality. Within MAS, the functionality is created by calling the service that a particular agent offers. Therefore, in order to add a new functionality, a new agent responsible for a new service can be added into a system. In order to modify or improve the functionality of the system, the existing agent can be modified or substituted with a new one. In this case, a system is loosely coupled which means that it is easy to extend, remove, and modify without breaking down the system. In addition, MAS can make the system more fault-tolerant by replacing an agent that has crashed with a new agent that can be launched on the fly as a substitute for a failing agent (Terje and Marius 2015).

MAS are suitable for applications that require distributed and concurrent processing capabilities. They are employed in the applications in several domains such as supply chain management (Rady 2011; Zimmermann 2006; Ngan and Kanagasabai 2013), web-services (Mohamed and Makhlouf 2014; García-Sánchez et al. 2009), healthcare (Dolgui et al. 2015; Shakshuki and Reid 2015; Isern, Sánchez and Moreno 2010), e-learning (Terje and Marius 2015), etc. When a group of individual agents constitutes MAS, it is crucial to have a mechanism that can control such a group. Communication is a key for MAS to exhibit social behaviour (e.g., share information, coordinate their tasks). Individual agents in MAS interact with one another by exchanging messages using a specific Agent Communication Language (ACL). The purpose of ACL is to enable agents to convey messages to one another with meaningful statements (Vaniya, Lad and Bhavsar 2011). Most ACLs are based on the speech-act theory. Speech acts are expressed by means of standard key words also known as communicative acts or performatives (e.g., request, inform, confirm, and propose). They are used to inform the intention of the communication from the sender to the receiver. The agent's message consists of various parameters such as sender, receiver, content language, ontology, and the actual content. Examples of well-known ACL languages are KQML (Knowledge Query and Manipulation Language) and FIPA-ACL (Foundations for Intelligent Physical Agents-Agents Communication Language) proposed by FIPA (The Foundation for Intelligent Physical Agents 2015). FIPA is the relevant standardisation body that promotes agent-based technology and the interoperability of its standards with other technologies.

### **4.3.3 The integration of ontology and multi-agent systems**

Ontologies play an important role in enabling knowledge representation, knowledge management, and knowledge sharing. Many applications benefit greatly from making use of ontologies as a means of achieving semantic interoperability among heterogeneous and distributed systems. They are considered as one of the key enablers for the emerging Semantic Web by making the Web content accessible to humans and computers (Li, Wu and Yang 2005). Ontologies are in a machine-understandable and processable format, thereby enabling the software agents to understand the contents autonomously. Therefore, the integration of ontologies and

multi-agent systems, also known as the ontology-based multi-agent approach, allows software applications to benefit from both technologies. For instance, ontologies can assist with data retrieval, while the agents can act as autonomous software entities that can interact with the environment and with other agents (Garanina, Sidorova and Bodin 2013).

In recent years, the ontology-based multi-agent approach has attracted considerable interest in research to support various works operated in distributed and dynamic environments. As presented and discussed in Chapter 2 regarding the state-of-the-art ontology-based multi-agent systems, the majority of research has focused on the use of ontology to facilitate agents' communication, represent domain knowledge and help to locate and retrieve information, and reasoning the knowledge.

- Facilitating agents' communication

In a multi-agent system, each agent usually cooperates with other agents to achieve a common goal; therefore, it needs the ability to communicate and interact with other agents by exchanging messages. The agent communication languages such as KQML and FIPA-ACL specify the syntax of the exchange messages but not the semantics of the messages. In this case, ontology can be additionally supplied in the messages to formalise the semantics of the exchanged message in a format that is understandable by agents in order to facilitate consistent communication and interoperability.

- Representing domain knowledge and helping to locate and retrieve information

Ontology can be used to describe domain knowledge and information content which is pertinent to that domain. With the use of ontologies in MAS, domain knowledge does not need to be embedded within the agents. Therefore, it creates an opportunity to share and reuse the domain knowledge and also has the potential to reuse the MAS infrastructure for other applications. Moreover, software agents have the ability to read and understand knowledge captured in ontologies. Therefore, they are able to locate and retrieve the information requested by their user.

- Reasoning the knowledge

The use of ontologies coupled with MAS can support knowledge representation and reasoning capabilities of software applications that are developed by deploying the MAS approach. The integration of ontologies in MAS can lead to the creation of logic rules that can be applied by a semantic reasoner to infer new knowledge not explicitly defined in ontologies (Freitas et al. 2015).

The benefits of both technologies can be had by integrating ontology and MAS. Ontology is used for knowledge representation, knowledge integration, knowledge sharing and reuse. The features of the software agent and MAS, such as autonomy, reactivity, pro-activeness, social ability, adaptability and dynamism, provide a potential solution for applications that are complex, dynamic and distributed. Therefore, they can be deployed in the application if only one of the approaches cannot satisfactorily resolve the problem. As the ontology and agent-based technology address different aspects of the same problem, they complement each other. Therefore, the ontology-based multi-agent system has been chosen as the preferred scientific approach for this research as explained in the next section.

#### **4.4 Ontology-Based Multi-Agent Systems Solution Proposal for a Framework for Active Software Engineering Ontology**

In Chapter 3, three research issues have been identified: i) automated knowledge capture of software project information, ii) Software Engineering Ontology instantiations management, and iii) active platforms for multi-site software development environments. They lead to the solution requirements for each research issue stated in Section 4.2. In the previous section, definitions of software agent, multi-agent system are given. The integration of ontology and multi-agent system and its benefit are also described. In this section, the features of the ontology-based multi-agent approach are explained to show their appropriateness for each solution as an underlying architecture for the framework of active Software Engineering Ontology.

Over the years, agent-based technology has become a promising solution to



support working processes in a distributed environment (Hammouch, Medromi and Sayouti 2015). Its concepts and features such as autonomy, reactivity, proactivity, and sociability of a software agent, can assist remote collaborative teams who are dispersed across boundaries, and can also be applied to data, expertise, and resources that are scattered. In the software engineering domain, due to the globalisation of software development and for a number of business reasons, several software companies have adopted a multi-site software development approach that enables project team members to work across multiple sites. While on the one hand, a globally dispersed project offers several advantages, on the other hand, it creates additional challenges in regard to communication, coordination and information sharing. The Software Engineering Ontology is an underlying knowledge representation of software engineering knowledge that enables all team members working on a multi-site software development project to have a common understanding. However, it does not possess a degree of autonomy, and nor can it adapt dynamically to any change such as the capturing of new software project information, proactively delivering useful information without the user's explicit request, or alerting team members to an unusual event that might change the project plan. Put differently, it still heavily relies on the user's effort to manage such situations although the Software Engineering Ontology is in use. As a means of addressing this limitation, agent-based technology can be integrated with the ontology to provide the autonomous and flexible features. Consequently, the integration of the Software Engineering Ontology and agent-based technology or the ontology-based multi-agent approach is regarded as the best means of addressing the solution requirements for a framework of active Software Engineering Ontology as described in the following section.

#### **4.4.1 Ontology-Based Multi-Agent Systems as a Solution for Automated Knowledge Capture of Software Project Information**

To reiterate the definition given in Chapter 3, *“Automated knowledge capture of software project information, in the context of the framework for active software Engineering Ontology, is defined as automatically capturing software engineering knowledge from software project information based on the concepts defined in the Software Engineering Ontology and then populating it in the ontology knowledge*

*base.*” Because software development-related information generated within a software project is in syntactic form, its structure is not conducive to an understanding of the semantics, and therefore may create ambiguities (e.g. incorrect or different interpretations). Moreover, due to the great amount of this information, it is not practical to manually annotate it. The ontology-based multi-agent approach is able to automatically capture the semantics of software project information and populate it in the ontology repository by means of the semantic annotation process and ontology population. Software Engineering Ontology is deployed by a software agent to provide software engineering domain knowledge to software development artefacts. The agent annotates software project information according to the corresponding concepts and then generates new instances which are subsequently populated into the ontology repository. The aforementioned processes can be done by software agents with minimum human intervention. In addition, the utilisation of agents can speed up the process because they are able to act in parallel. To sum up, an ontology-based multi-agent approach will encourage team members to share their knowledge by offering automated and transparent support to semantically capture software project information when they are working on software development process.

#### **4.4.2 Ontology-Based Multi-Agent System as a Solution for Software Engineering Ontology Instantiations Management**

In Chapter 3, this definition was given: “*Software Engineering Ontology instantiations management, in the context of the framework for active Software Engineering Ontology, is defined as accessing software engineering knowledge and manipulating instance knowledge.*” Software engineering knowledge and software development project information are presented in the Software Engineering Ontology in machine-understandable and machine-processable form. Thus, a software agent is able to access and process this knowledge with the guidance of the ontology. As a result, the ontology-based multi-agent approach will not only respond to users’ requests to retrieve the knowledge required (information pull), but also offer the mechanisms to disseminate proactively the knowledge that is potentially useful to them (information push). The agents can do this by inferring the knowledge based on the semantic relations defined in the ontology.

As a consequence, the ontology-based multi-agent approach to access and manipulate Software Engineering Ontology instance knowledge can assist team members by facilitating some time-consuming tasks such as searching for relevant information to address project development issues, locating experts, analysing the impact of a change in software artefacts as well as propagating the change and its impact to relevant team members, proactively monitoring particular software project information to detect potential disruptive event or deviation. In this case, even though software teams may not be aware of the existence of the knowledge or they might not be able to search for it effectively, the ontology-based multi-agent system approach can help them to locate the information and can deliver it to them. The proactive knowledge delivery can mitigate the arduous task of having to explicitly search for useful knowledge; in other words, it can deliver the knowledge at the right time to the right people who need it.

#### **4.4.3 Ontology-based Multi-agent Systems as a Solution for Active Platforms for Multi-site Software Development Environments**

According to the definition given in Chapter 3, “*Active platforms for multi-site software development environments, in the context of the framework for active Software Engineering Ontology, refer to frameworks to operate the application programs that assist collaborative software teams to manage and share software engineering knowledge across software life cycle in a multi-site software development environment.*” In a co-located software development environment, informal communication is a key factor in knowledge sharing and collaborative work. However, in a multi-site software development setting, physical and temporal distances create communication gaps which could impede the sharing of knowledge and team collaboration. Therefore, there is the need for a good development infrastructure that includes active platforms for effective knowledge management to simplify the process of capturing, searching, and distributing software engineering knowledge. This will enable effective knowledge sharing and team collaboration in a distributed setting. The Software Engineering Ontology-based multi-agent approach can fulfil these requirements for the reasons given below.

First, the Software Engineering Ontology, which is a comprehensive

ontology covering all the aspects of software engineering in the software development life cycle, can contribute to knowledge management by representing the shared software development knowledge relevant to a software project. Because the knowledge captured in the ontology is in machine processable form that software agents can read and process it. Therefore, the ontology-based multi-agent approach can offer the platforms that are equipped with the underlying knowledge representation and active components to manage the knowledge. These active platforms include the knowledge capture platform, the knowledge query platform, the knowledge monitoring platform, and the knowledge maintenance platform. The knowledge capture platform is used to automate the process of formalising and conceptualising the knowledge generated during the software development process. The knowledge query platform is used to retrieve shared software project information. The knowledge monitoring platform monitors particular software project information in order to anticipate any deviation or disruptive event and then notify corresponding team members before an issue actually arises. Finally, the knowledge maintenance platform is used to maintain software development knowledge captured in the Software Engineering Ontology. These active platforms can provide useful knowledge to distributed team members in a flexible manner by means of either the push or pull mode of information delivery. Thus, these platforms can reactively provide knowledge based on user requests or proactively deliver knowledge that the team members may need for their specific tasks.

Second, in a multi-site software development environment, software team members do not reside at the same place, but are dispersed in different locations. Two main key factors that can contribute to the success of this development setting are effective and efficient communication and coordination (Khan and Khan 2014; Jimnez et al. 2009). Communication is considered effective if it is targeted and clear to the receivers. It is considered efficient if the communication is clear and timely. Effective and efficient coordination refers to the ability to improve group awareness to manage task dependencies and artefact dependencies in a timely and proactive manner, e.g., early enough to drive their decision making. The ontology-based multi-agent approach can respond to these needs. With the social ability, multiple agents communicate and collaborate with each other in order to achieve goals. They can engage in dialogues with other software agents in order to facilitate effective and

efficient communication with the relevant team members. With the integration of the Software Engineering Ontology and the agent's features of reactivity and proactivity, remote communication can be clear (without ambiguity), targeted, and timely. These features can improve timely awareness within distributed teams and can lead to effective and efficient coordination. At the appropriate time, team members can be made aware of when they should coordinate in order to respond to others' activities carried out at different sites.

## 4.5 Conclusion

This chapter begins with an overview of the four key solution requirements for the framework of active Software Engineering Ontology solution development. Definitions and explanations of a software agent, the differences between agent and traditional software applications, multi-agent system, and the integration of ontology and multi-agent systems, are given to justify the feasibility of using the ontology-based multi-agent approach as the underlying framework of this study. Then explanations of how the ontology-based multi-agent system approach can be utilised to serve each solution requirement, namely, automated knowledge capture of software project information, Software Engineering Ontology instantiations management, and active platforms for multi-site software development environments, are given. In the next chapter, the development approach for an active Software Engineering Ontology framework will be discussed in detail.

## 4.6 References

Braubach, Lars, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. 2005. "Deployment of distributed multi-agent systems." In *Engineering Societies in the Agents World V: 5th International Workshop, ESAW 2004, Toulouse, France, October 20-22, 2004. Revised Selected and Invited Papers*, eds Marie-Pierre Gleizes, Andrea Omicini and Franco Zambonelli, 261-276. Berlin, Heidelberg: Springer Berlin Heidelberg.

- Dolgui, Alexandre, Jurek Sasiadek, Marek Zaremba, N. Benhajji, D. Roy, and D. Anciaux. 2015. "15th IFAC Symposium on Information Control Problems in Manufacturing Patient-centered multi agent system for health care." *IFAC-PapersOnLine* 48 (3): 710-714. doi: <http://dx.doi.org/10.1016/j.ifacol.2015.06.166>.
- Elamy, A.H. 2005. "Perspectives in agent-based technology." *AgentLinkNews* 18: 19-22.
- The Foundation for Intelligent Physical Agents. 2015. Accessed January 5, 2016, <http://www.fipa.org>.
- Freitas, Artur, Alison R Panisson, Lucas Hilgert, Felipe Meneguzzi, Renata Vieira, and Rafael H Bordini. 2015. "Integrating ontologies with multi-agent systems through CArtAgO artifacts" *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Singapore: IEEE.
- Garanina, Natalia Olegovna, Elena A Sidorova, and Evgeny V Bodin. 2013. "A multi-agent approach to unstructured data analysis based on domain-specific ontology" *The 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland*: Citeseer.
- García-Sánchez, Francisco, Rafael Valencia-García, Rodrigo Martínez-Béjar, and Jesualdo T. Fernández-Breis. 2009. "An ontology, intelligent agent-based framework for the provision of Semantic Web services." *Expert Systems with Applications* 36 (2, Part 2): 3167-3187. doi: <http://dx.doi.org/10.1016/j.eswa.2008.01.037>.
- Hammouch, H., H. Medromi, and A. Sayouti. 2015. "Toward an intelligent system for project management based on the multi agents systems" *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, doi: 10.1109/SITA.2015.7358412.
- Isern, David, David Sánchez, and Antonio Moreno. 2010. "Agents applied in health care: A review." *International Journal of Medical Informatics* 79 (3): 145-166. doi: <http://dx.doi.org/10.1016/j.ijmedinf.2010.01.003>.

- Jennings, Nicholas R. 2000. "On agent-based software engineering." *Artificial Intelligence* 117 (2): 277-296. doi: 10.1016/s0004-3702(99)00107-1.
- Jimnez, Miguel, Mario Piattini, Aurora Vizca, and no. 2009. "Challenges and improvements in distributed software development: a systematic review." *Advances in Software Engineering* 2009: 1-16. doi: 10.1155/2009/710971.
- Khan, Rafiq Ahmad, and Siffat Ullah Khan. 2014. "Communication and coordination challenges in offshore software outsourcing relationships: A systematic literature review protocol." *Gomal University Journal of Research* 30 (1): 9-17.
- Li, Li, Baolin Wu, and Yun Yang. 2005. "Agent-based ontology integration for ontology-based applications." In *Proceedings of the 2005 Australasian Ontology Workshop - Volume 58, Sydney, Australia*, 53-59. 1151943: Australian Computer Society, Inc.
- Mohamed, Gharzouli, and Derdour Makhoul. 2014. "To implement an open-MAS architecture for Semantic Web services discovery: what kind of P2P protocol do we need?" *International Journal of Agent Technologies and Systems (IJATS)* 3 (6): 58-71. doi: 10.4018/ijats.2014070103.
- Ngan, Le Duy, and Rajaraman Kanagasabai. 2013. "Semantic Web service discovery: State-of-the-art and research challenges." *Personal and Ubiquitous Computing* 17 (8): 1741-1752. doi: 10.1007/s00779-012-0609-z.
- Rady, Hussein A. 2011. "Multi-agent system for negotiation in a collaborative supply chain management." *International Journal of Video & Image Processing and Network Security IJVIPNS-IJENS* 11 (5).
- Shakshuki, Elhadi, and Malcolm Reid. 2015. "Multi-agent system applications in healthcare: Current technology and future roadmap." *Procedia Computer Science* 52: 252-261.
- Terje, Kristensen, and Dyngeland Marius. 2015. "Design and development of a multi-agent e-learning system." *International Journal of Agent Technologies and Systems (IJATS)* 7 (2): 19-74. doi: 10.4018/IJATS.2015040102.

- Turban, Efraim, Ramesh Sharda, and Dursun Delen. 2010. *Decision Support and Business Intelligence Systems*: Prentice Hall Press.
- Vaniya, Sandip, Bhavesh Lad, and Shreyansh Bhavsar. 2011. "A survey on agent communication languages." In *International Conference on Innovation, Management and Service, Singapore*, 16-18 September 2011. IACSIT.
- Wooldridge, Michael. 2009. *An Introduction to Multiagent Systems*: John Wiley & Sons.
- Wooldridge, Michael, and Nicholas R Jennings. 1995. "Intelligent agents: Theory and practice." *The Knowledge Engineering Review* 10 (02): 115-152.
- Ye, D., M. Zhang, and A. V. Vasilakos. 2016. "A survey of self-organization mechanisms in multiagent systems." *IEEE Transactions on Systems, Man, and Cybernetics: Systems* PP (99): 1-21. doi: 10.1109/TSMC.2015.2504350.
- Zimmermann, Roland. 2006. *Agent-based Supply Network Event Management, Whitestein Series in Software Agent Technologies*. Switzerland: Birkhäuser Verlag.

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.



# Chapter 5 Conceptual Framework

## 5.1 Introduction

In the previous chapter, the four key solution requirements for the framework of active Software Engineering Ontology solution development were identified. This was followed by the justification of an ontology-based multi-agent approach as a solution to the corresponding research issues and each solution requirements, namely, automated knowledge capture of software project information, Software Engineering Ontology instantiations management, and active platforms for multi-site software development environments.

This chapter begins with a brief overview of several existing agent-oriented software engineering methodologies in order to justify the appropriate methodology to be used for the development of the proposed framework. In this thesis, the basic software engineering processes such as analysis, design, implementation and evaluation are used and integrated with the Agent Unified Modelling Language (AUML) in order to provide the complete development processes of the framework of active Software Engineering Ontology. The framework development is divided into the macro-perspective and the micro-perspective as proposed in (Zimmermann 2006). The first one involves the design of an agent society as a whole while each individual agent type is regarded as a black-box. The latter one refers to the detailed design of each agent type. In this chapter, only the macro-perspective is considered and discussed in order to create the structure of an agent society and its inter-agent interactions. The micro-perspective will be discussed subsequently in Chapters 6 and 7.

## **5.2 Overview of Existing Agent-Oriented Software Engineering Methodologies**

A number of methodologies have been proposed that offer well-defined guidelines and systematic approaches for the development of a multi-agent system. These methodologies differ in their intended purposes, concepts, approaches, processes, and modelling notations. Several examples of these methodologies are presented with brief descriptions below.

### **5.2.1 GAIA**

GAIA was one of the first methodologies intended to facilitate the development of an agent-based system (Wooldridge, Jennings and Kinny 2000; Wooldridge and Ciancarini 2001). It consists of two main phases, namely, the analysis phase and design phase. The results of these two phases are a set of models that are used to implement the multi-agent system. In the analysis phase, the agent's roles and its associated protocols are analysed in order to produce the roles model and interactions model. In the design phase, different agent types are derived from the identified roles in order to create the agent model. The services or functions of the agents are also described to create the services model. Finally, the communication links between different agent types are defined as the acquaintance model.

### **5.2.2 Agent Unified Modelling Language (AUML)**

Agent Unified Modelling Language (AUML) has been used as a standard representation by FIPA in order to describe agent communication and protocols (Huguet and Odell 2005; Huguet, Odell and Bauer 2004). The main purpose of AUML is to give developers a notation to analyse, design, and implement an agent-based system. It is based on the UML object-oriented modelling representation and it is extended to represent agents, their behaviour, and interactions among them. Therefore, the developers who are familiar with the UML diagrams can easily understand and use the AUML diagrams to capture the agent concepts and its interactions.

### **5.2.3 Multi-agent Systems Engineering (MaSE) and Organisation-based Multi-agent System Engineering (O-MaSE)**

Multi-agent Systems Engineering (MaSE) has been developed to facilitate the entire development process of a multi-agent system from problem description to realisation (DeLoach 2004; Wood and DeLoach 2001). It comprises three main phases, namely, analysis, design, and implementation. MaSE applies a number of standard UML object-oriented modelling notations to describe the agent types in a system and their interactions to other agents. It also includes an architecture-independent detailed definition of the internal design of an agent. MaSE is designed to develop general-purpose multi-agent systems and provides several advantages. However, it needed some improvement, such as the inclusion of a mechanism to model agent interactions within the environment. Thus, MaSE has now been replaced by the Organisation-based Multi-agent Systems Engineering (O-MaSE).

The purpose of O-MaSE methodology is to assist developers to create customised agent-oriented software engineering processes. A metamodel, a set of method fragments, and a set of method construction guidelines are the three main components of the O-MaSE methodology. The metamodel identifies a set of key components that are required for the analysis, design, and implementation of a multi-agent system. The method identifies the separate tasks that need to be executed to create a set of work products such as code, models, or a set of activities and their actors. The method construction guidelines show how the method fragments are related to each other (DeLoach 2014).

### **5.2.4 MAS-CommonKADS**

MAS-CommonKADS originated from the extension of CommonKADS, a well-known knowledge engineering methodology, and object-oriented methodology for the development of a multi-agent system (Carlos and Mercedes 2005). The MAS-CommonKADS software development life cycle comprises five main phases: conceptualisation, analysis, design, implementation and testing, and operation. In the

conceptualisation phase, the agent properties are identified. The analysis phase involves an analysis of the system from different perspectives. The outputs from the analysis phase are refined in the design phase. The system is coded and tested during the development and testing phase. Finally, during the operation phase, the system is operated and maintained.

### **5.2.5 MESSAGE**

MESSAGE is an agent-oriented software engineering methodology that includes the analysis and design phases of the multi-agent systems development (Francisco, Jorge and Philippe 2005). It extends the basic UML concepts with new agent-relevant concepts and notations as well as enriches with the Rational Unified Process (RUP) model for the analysis and design of the system. The objective of the analysis phase is to produce a system specification or an analysis model to define the problem to be resolved. During the design phase, the artefacts produced in the analysis model are mapped into software entities (e.g., classes, objects, operation signatures, interfaces, etc.) for the implementation.

### **5.2.6 Process for Agent Societies Specification and Implementation (PASSI)**

Process for Agent Societies Specification and Implementation (PASSI) is a methodology for the development of a multi-agent system that combines design models and concepts from object-oriented software engineering and artificial intelligence techniques (Massimo 2005). PASSI is made up of five models, namely, system requirements model, agent society model, agent implementation model, code model, and deployment model. The system requirements model is related to the analysis of the system with respect to agency and its purpose. The agent society model involves the interactions and dependencies among agents of the society. The agent implementation model defines development detail of the system architecture with regard to classes, attributes, and methods. The code model describes a representation of the system at the code level. Finally, the deployment model represents the distribution of sub-systems across hardware processing units and their migration between processing units.

### **5.2.7 PROMETHEUS**

Prometheus is a methodology that allows developers to analyse, design, and implement a multi-agent system by covering all software engineering processes (Gomez, Isern and Moreno 2007; Padgham and Winikoff 2003). Prometheus has been specifically designed to facilitate the development of the belief-desire-intention (BDI) agents. There are three main phases as follows.

- System specification - This phase focuses on the identification of the functionalities of the systems.
- Architectural design - This phase is to identify which agents should exist in the system and then assign their functionalities including events, interactions, and shared data objects.

Detailed design - This phase focuses on the design of the internal structure of individual agents and how they complete their tasks within a system.

### **5.2.8 The Agent-Oriented Development Methodology (ADEM)**

The Agent-Oriented Development Methodology (ADEM) is a development methodology that focuses on modelling aspects of agent-based systems utilising the Agent-Modelling Language (AML) as a modelling language (Sturm and Shehory 2014). ADEM comprises method fragments, techniques, artefacts, and guidelines to support the creation of multi-agent system models. It principally addresses the business modelling, requirements, and analysis and design workflows. Even though ADEM does not cover the whole software development process and defines only the multi-agent system-specific parts of the process, it extends RUP to define other parts in order to complete the methodology.

## 5.3 Development Approaches

In order to develop the framework for active Software Engineering Ontology, a suitable software engineering methodology is required. In this thesis, the AUML has been chosen as an appropriate methodology for the development process. AUML is an extension of UML modelling representation for agents and has been selected as the standard de facto used by FIPA to capture agent concepts and their interactions. However, the AUML focuses mainly on the design of multi-agent systems by using and extending the widely-used UML notations. The methodology does not cover the complete process of software development life cycle. Therefore, the basic software engineering processes (i.e., analysis, design, implementation, and evaluation) (Zimmermann 2006; Nienaber 2008) are additionally integrated with the AUML in this thesis in order to complete a full life-cycle specification of the system development.

According to Zimmermann (2006), the fundamental software engineering steps are generally applied to the development of agent-based systems. There are two main categories of these systems, namely, macro-perspective and micro-perspective (Figure 5-1).

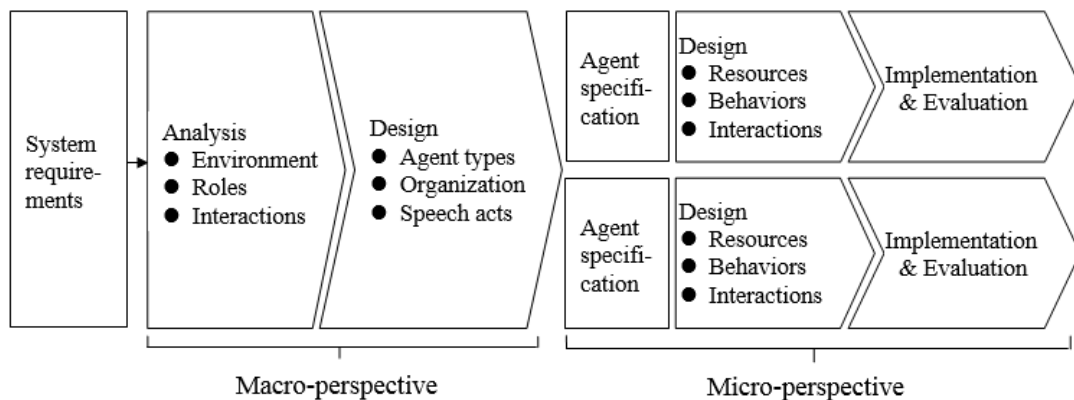


Figure 5-1: Macro- and micro-perspective of AOSE (Zimmermann 2006)

### 5.3.1 Macro-perspective

The macro-perspective (Zimmermann 2006) focuses on the analysis and design phases of an agent society as a whole, while an individual agent is regarded as

a black-box. The requirements of a multi-agent system are analysed and the overall solution is designed which includes the structure of the agent society and the interactions and dependencies among different agent types. These specifications are subsequently refined in the micro-perspective in the next chapters. There are two main phases in the macro-perspective, namely, the analysis phase and the design phase.

Within the analysis phase, the environment, roles, and interactions are considered according to the system requirements. Their details are given below.

- The environment of a multi-agent system is analysed. For example, the domain and the problem that a system is intended to provide a solution or the objectives of a system.
- The roles to fulfil main functions of a multi-agent system are defined.
- Interactions between functions are modelled to define the cooperation.

At the end of the system analysis phase, the outputs of this phase are used in the design phase in order to model the holistic model of the system. The defined roles are realised as agent types. The structural dependencies between them are also identified to form an agent society. Then the interactions in the analysis phase are transformed into speech act communications among different agent types.

### **5.3.2 Micro-perspective**

The outputs of the design phase of the macro-perspective become the specifications for each agent type. These specifications are refined into resources, behaviours, and interactions during the design phase of the micro-perspective as follows (Bauer and Müller 2004; Zimmermann 2006).

- *Resources* – These comprise the information that an agent can use to perform activities, namely, its internal knowledge assets, its goals, and external resources (e.g., rules, databases, ontologies).
- *Behaviours* – They describe how an agent performs activities that utilise its

available resources to satisfy its goal.

- *Interactions* – They describe how an agent interacts with other agents in order to exchange information according to the agent's behaviours.

In the next section, the development of the active Software Engineering Ontology through a multi-agent system in accordance with the macro-perspective is discussed in detail.

## **5.4 Active Software Engineering Ontology through a Multi-Agent System Engineering**

In this section, the macro-perspective of the active Software Engineering Ontology through a multi-agent system (SEOMAS) framework is discussed. First, the system requirements according to the previously defined research issues are analysed. These requirements lead to the identification of corresponding roles and agent types in the framework. Finally, the structure of an agent society is defined along with its dynamic interactions among different agent types.

### **5.4.1 System Requirements**

The proposed SEOMAS framework is intended to provide active support to assist software development team members with software engineering knowledge when they are working on software development projects. The requirements of the system are listed below.

- Knowledge capture of software project information
  - Annotating new imported software project information imported into the version control repository with the appropriate concepts defined in the Software Engineering Ontology.
  - Populating new instance knowledge identified from the semantic annotation process.



- Software Engineering Ontology instantiations management
  - Querying instance knowledge.
  - Manipulating instance knowledge (i.e., adding, modifying, deleting).
  - Recommending the impact of instance knowledge manipulation according to software evolution by considering the change made and how it affects other artefacts and team members.
  
- Advanced services in addition to the instantiations management
  - Providing useful and relevant knowledge (e.g. a potential bug fixer or expert, information about software components and their dependencies).
  - Proactively monitoring software project information to detect the probability of deviations or disruptive events in anticipation of encountering situations that may threaten the success of the software project.
  - Generating real-time notification messages to enable effective management of the awareness of remote software development teams.

#### **5.4.2 Roles and Agent Types**

The aforementioned system requirements are mapped and assigned various roles as following.

- VersionControlManager

This role is to manage the version control repository. In this research, the focus is on the import of source code artefacts into the version control repository, leading to the activation of the semantic annotation process.

- SemanticAnnotator

This role is to automate the semantic annotation process. The source code which is considered as the main artefact centrally located and critical in software development is assigned with software engineering domain knowledge concepts in order to clarify any ambiguity in distant communication.

- InstanceKnowledgeManager

This role is to access and manage the instance knowledge. It involves querying, adding, modifying, or deleting the instance knowledge. The addition of new instance knowledge also includes the population of the ontology with new instances identified by the *SemanticAnnotator* role.

- Recommender

This role is to generate recommendation regarding the impact of a change made to the instantiation. It also involves the monitoring of certain instantiations in order to proactively identify any possible deviation or disruptive events before they actually occur. In addition, it manages the messages that are sent to notify other relevant agents.

- ACLMessageGenerator

This role translates the request from a user into an ACL message and sends it to a corresponding agent.

- OutputGenerator

This role transforms the content of an ACL message into meaningful output and delivers it to the user.

- DomainKnowledgeManipulationRequester and DomainKnowledgeManager

These two roles are related to the Software Engineering Ontology evolution which is about the manipulation of domain knowledge of the Software Engineering Ontology. The DomainKnowledgeManipulation-Requester role maintains the output from the Software Engineering Social Network system (SESN) in an ontology evolution repository and notifies the ontology agent to update the Software Engineering Ontology domain knowledge. More information about SESN can be found in (Aseeri 2011) and (Kasisopha 2013). The DomainKnowledgeManager is responsible for updating the Software Engineering Ontology domain knowledge.

All the abovementioned system requirements and roles derive the specific agent types according to the cluster of roles as shown in Table 5-1.

Table 5-1: The mapping of system requirements to agents' roles and their associated agent types

<b>System requirements</b>	<b>Roles</b>	<b>Cluster of roles</b>	<b>Agent types</b>
Manage the version control repository	VersionControlManager	Version control management	VersionControl agent
Capture software engineering knowledge from the software project information based on the concepts described in the Software Engineering Ontology	SemanticAnnotator	Semantic annotation	Annotation agent
Querying domain knowledge.	DomainKnowledgeManager	Knowledge management	Ontology agent
Manipulating domain knowledge			
Querying instance knowledge.	InstanceKnowledgeManager		
Manipulating instance knowledge			
Recommending the impact of instance knowledge manipulation	Recommender	Recommendation and notification management	Recommender agent
Providing useful and relevant knowledge			
Proactively monitoring software project information			
Generating real-time notification messages			

<b>System requirements</b>	<b>Roles</b>	<b>Cluster of roles</b>	<b>Agent types</b>
Maintaining the output from the Software Engineering Social Network system (SESN) in an ontology evolution repository	DomainKnowledge-ManipulationRequester	Domain knowledge manipulation proposal	Evolution agent <sup>1</sup>
Mediating between a user and the system	ACLMessageGenerator	Communication management	User agent
	OutputGenerator		

---

<sup>1</sup> The evolution agent is not included because its functionality is beyond the scope of this thesis. See section 5.4.3 for more detail.

### 5.4.3 Architecture Modelling

The proposed architecture of the SEOMAS framework is illustrated in Figure 5-2. It comprises six agent types with brief descriptions of their roles as follows.

*User agent (UA)* is a mediator between a user and the system. A user employs his/her user agent to perform tasks on his/her behalf. Its roles are *ACLMessageGenerator* and *OutputGenerator*.

*VersionControl agent (VA)* is responsible for managing the version control repository. In this research, this agent focuses on the import of new source code file(s) into the version control repository. Its role is *VersionControlManager*.

*Annotation agent (AA)* is responsible for annotating software project information that is imported into the version control repository. Its role is *SemanticAnnotator*.

*Ontology agent (OA)* is responsible for accessing and manipulating the Software Engineering Ontology domain and instance knowledge. It also manages the ontology population according to the semantic annotation process. Its roles are *DomainKnowledgeManager* and *InstanceKnowledgeManager*.

*Recommender agent (RA)* is responsible for generating recommendations and notification (e.g., change impact analysis, expert identification). Its role is *Recommender*.

*Evolution agent (EA)* is responsible for facilitating the Software Engineering Ontology evolution. Its role is *DomainKnowledgeManipulationRequester*.

It is to be noted that the Software Engineering Ontology domain knowledge evolution is beyond the scope of this thesis. In addition, currently there are many research efforts in the ontology evolution field such as those of (De Leenheer 2009; Zablith et al. 2015; Palma et al. 2012). Therefore, the role *DomainKnowledgeManipulationRequester* of the evolution agent and the role *DomainKnowledgeManager* of the ontology agent are not pertinent to this research. In this research, the

focus is on the management of the evolution of instance knowledge only. The architecture of active Software Engineering Ontology through a multi-agent system is shown in Figure 5-2. The evolution agent and its interaction with the ontology agent are represented in grey to indicate that they are not implemented.

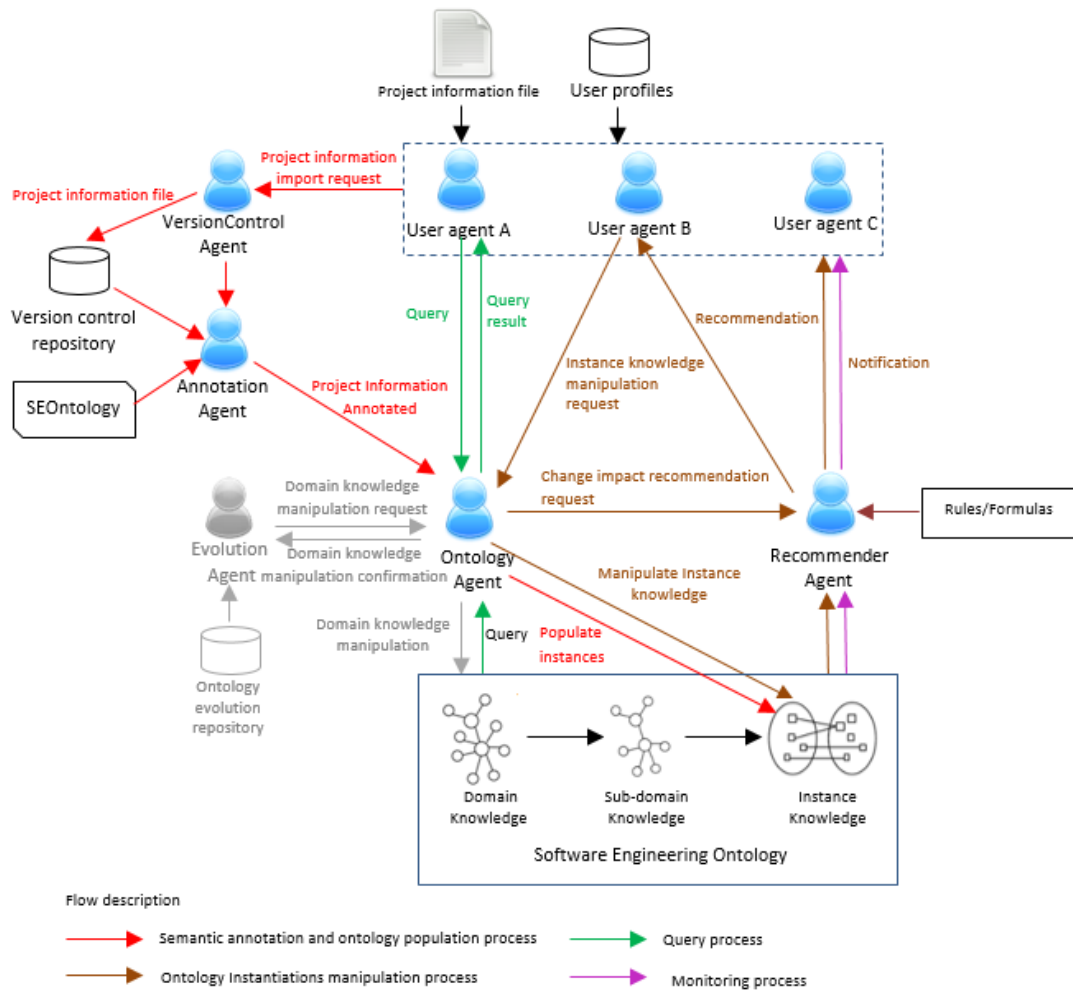


Figure 5-2: The active Software Engineering Ontology through a multi-agent system architecture

In Figure 5-2, a user agent is initialised when each user logs in to the system. It will check with the user profile in order to manage the access level allocation according to the user's role in the software project such as project manager, team leader, requirement engineer, analyst, developer and tester. This ensures that each user has the level of access enabling him/her to perform operations that are his/her responsibility. There are four main processes: semantic annotation and ontology population, query, ontology instantiations manipulation, and monitoring. They are differentiated by the colour of the arrows. The explanation of these processes is as

follows.

The semantic annotation and ontology population processes are represented by the red arrows. When a team member would like to import new software project information file (e.g., source code) into the version control repository, his/her user agent sends a request message to the versioncontrol agent. The versioncontrol agent then imports the requested file into the version control repository. Once the annotation agent perceives that the file is imported, it applies the ontological concept from the Software Engineering Ontology domain knowledge to semantically annotate the project information to identify new instances. The ontology agent then inserts these new instances into the ontology knowledge base. Finally, the Software Engineering Ontology is populated with new instances.

The query process is indicated by the green arrows. It starts with the user sending a query to his/her user agent. The user agent generates a message and sends it to the ontology agent which is responsible for querying the knowledge in the ontology. The ontology agent then retrieves the requested information and sends it back to the user agent.

The Software Engineering Ontology instantiations manipulation process is indicated by the brown arrows. Once the user agent receives a request to manipulate the instance knowledge, it generates a request message to the ontology agent. The ontology agent requests the recommender agent to send a recommendation regarding the impact of a change made to the instantiation to the user agent. If the user confirms the manipulation request, the ontology agent manipulates the instance knowledge and the recommender agent sends the notifications to relevant user agents about the change made and how it affects the users' workspaces.

The monitoring of particular instance knowledge is indicated by the purple arrows. There are three main events that trigger the recommender agent to send a notification to the corresponding user which are: i) a threshold is reached, ii) a specific condition occurs, and iii) every appointed period of time is met (day, week, month, etc.). More details in regard to proactively monitoring of software project information are given in Chapter 7, section 7.5.2.2.3.

#### 5.4.4 Structural design of the agent society

In the design phase of the macro-perspective, the AUML class diagram at the conceptual level is used to provide the structure of an agent society in order to describe the structural interdependencies between the agents (Huguet 2003). In addition, it provides an overview of the main types of knowledge assets or knowledge resources that an agent requires in order to fulfil its task (Figure 5-3).

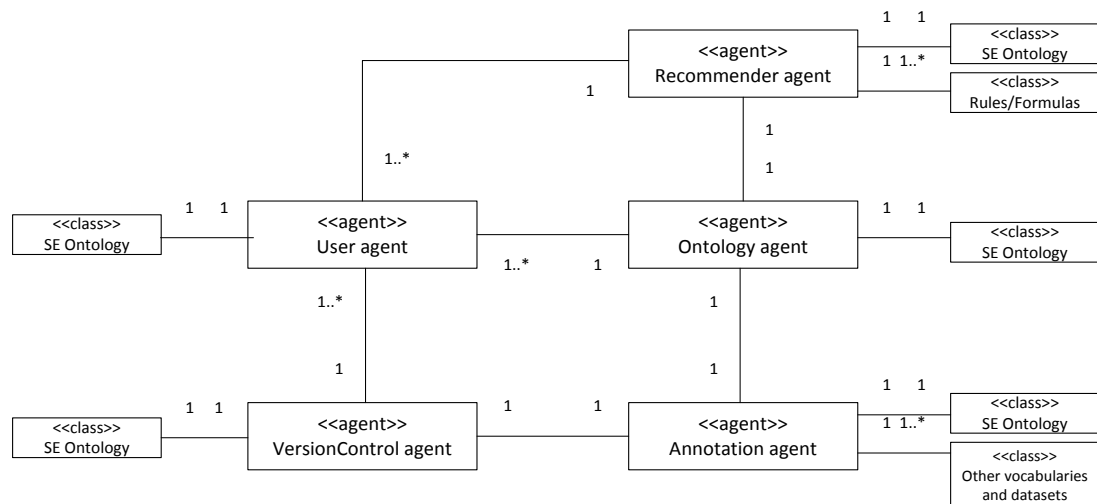


Figure 5-3: AUML class diagram of the macro-perspective of an agent society

In Figure 5-3, user agents have a many-to-one relationship with the versioncontrol agent. Each user agent sends a request to the versioncontrol agent to import new project information file into the version control repository. The versioncontrol agent has a direct link to the annotation agent with a one-to-one relationship to initiate the semantic annotation service. The annotation agent also has a one-to-one relationship with the ontology agent for the ontology population of new instances identified from the annotation process. The ontology agent has a one-to-one relationship with the recommender agent to request for generating recommendations and notifications. Lastly, the ontology agent and the recommender agent have a one-to-many relationship with user agents in order to deliver retrieved result, recommendations, and notifications to the corresponding user agents.

Furthermore, in this diagram, the knowledge assets that the agents require in order to fulfil their tasks are represented as classes. Every agent needs the Software



Engineering Ontology to facilitate consistent communication. The annotation agent requires knowledge from the Software Engineering Ontology to semantically annotate software project information. It also requires knowledge from additional classes such as other vocabularies and datasets in order to enrich semantic descriptions of the annotated software project information. Moreover, depending on the particular task, the recommender agent needs specific rules or formulas to apply to the retrieved instance knowledge in order to process the recommendations.

#### **5.4.5 Agent Interoperations**

In order to achieve the SEOMAS goals, multiple agents need to interact with one another according to their roles. The AUML sequence diagram is used to model the dynamic interactions within the agent society as presented in Figure 5-4. The standardised FIPA Request interaction protocol (FIPA 2002) is used as a main protocol in this research. The content of the messages is defined based on the Software Engineering Ontology in order to facilitate consistent communication among different agents.

According to semantic annotation of software project information and ontology population, once a developer finalises the source code, he/she makes a request to import it into the version control repository through his/her user agent. The user agent then creates an ACL request message and sends it to the versioncontrol agent. After the file is imported, the annotation agent is initiated to semantically annotate the source code with the Software Engineering Ontology domain concepts. It then sends a request to the ontology agent to populate the new instances identified from the annotation process into the Software Engineering Ontology repository.

When querying instance knowledge, a user agent sends a message to the ontology agent. The ontology agent retrieves the instance knowledge as requested and then sends the result back to the user agent. If not result is retrieved, a failure message is sent to the user agent.

In order to manipulate the Software Engineering Ontology instance knowledge, a user agent sends a request to the ontology agent. The ontology agent sends a request to the recommender agent to notify the user about the change impact

that can be occurred from the manipulation to make him/her aware of any unintended side effects. If the user confirms the request, the ontology agent manipulates the instance knowledge as requested. The recommender agent then sends the notifications about the instance knowledge manipulation and the potentially affected artefacts to the relevant user agents.

Regarding the monitoring of software project information, when the recommender agent monitors particular instance knowledge and can identify any deviation, the notification is generated and sent to the relevant user agents.

At the conclusion of the design phase of the macro-perspective, the outputs which are regarded as agent specifications for each agent type will be refined in the micro-perspective. These specifications are refined into resources, behaviours, and interactions and presented according to the proposed solutions of the framework of active Software Engineering Ontology in Chapters 6 and 7. In these two chapters, the prototype system is implemented and evaluated in terms of their ability to provide active support to remote team members working collaboratively in a multi-site software development setting.

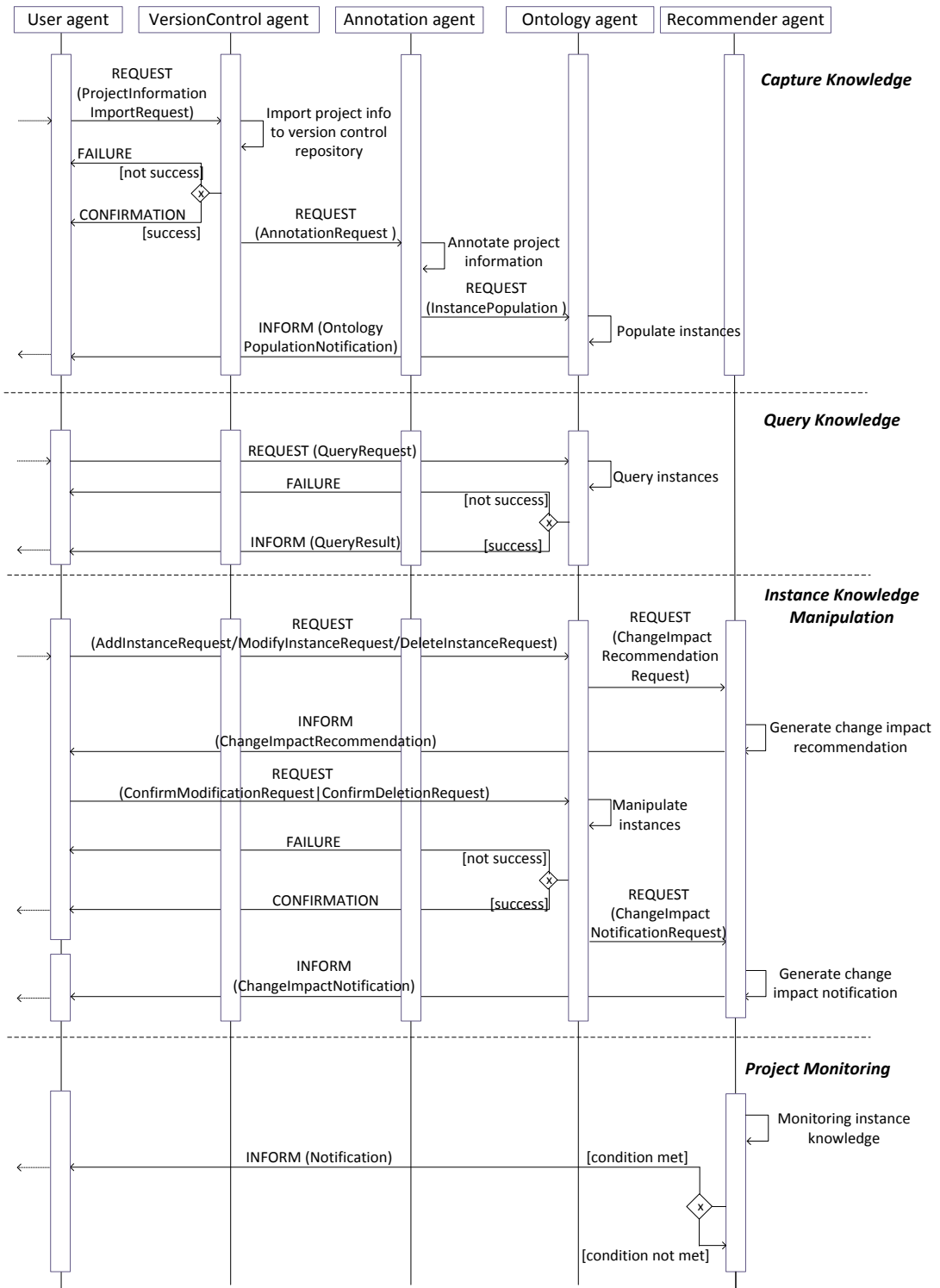


Figure 5-4: Overall interactions among the agents

## **5.5 Conclusion**

This chapter begins with a brief overview of several well-known existing agent-oriented software engineering methodologies. The integration of the macro-perspective and micro-perspective of agent-oriented software engineering as well as the AUML methodology are used to implement the active Software Engineering Ontology through multi-agent system framework. Then the analysis and design phases of the macro-perspective are discussed in detail to derive the overall solution including the structure of agent society and its dynamic interactions that results in the agent specifications of each agent type. These specifications will be used for the micro-perspective of the agent-oriented software engineering.

The next chapters describe the micro-perspective of the SEOMAS agents and the SEOMAS platforms in accordance with the proposed solutions of the framework for active Software Engineering Ontology in the order summarised. They are then followed by the evaluation chapter which details the evaluation of the complete framework.

Chapter 6 - Solution 1: Automated Knowledge Capture from Software Project Information

Chapter 7 – Solution 2: Software Engineering Ontology Instantiations Management

Chapter 8 – Solution 3: Active Platforms for Multi-site Software Development Environments

Chapter 9 – Active Software Engineering Ontology Framework Evaluation

## **5.6 References**

Aseeri, Ahmed Abdulridha. 2011. "Lightweight Community-Driven Approach to Support Ontology Evolution." Master of Philosophy (Information Systems), Curtin University.

- Bauer, Bernhard, and Jörg Müller. 2004. "Methodologies and modeling languages." *Luck M., Ashri R. D'Inverno M.(eds.): Agent-Based Software Development. Artech House Publishers, Boston, London.*
- Carlos, A. Iglesias, and Garijo Mercedes. 2005. "The agent-oriented methodology MAS-CommonKADS." In *Agent-Oriented Methodologies*, 46-78. Hershey, PA, USA: IGI Global.
- De Leenheer, Pieter. 2009. "On Community-based Ontology Evolution." Dissertation, Vrije Universiteit Brussel, Brussels, Belgium.
- DeLoach, Scott A. 2004. "The MaSE methodology." In *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, eds Federico Bergenti, Marie-Pierre Gleizes and Franco Zambonelli, 107-125. Boston, MA: Springer US.
- . 2014. "O-MaSE: An extensible methodology for multi-agent systems." In *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, eds Onn Shehory and Arnon Sturm, 173-191. Berlin, Heidelberg: Springer Berlin Heidelberg.
- FIPA. 2002. Foundation for intelligent physical agents: FIPA request interaction protocol specification. Accessed November 23, 2015, <http://www.fipa.org/specs/fipa00026/SC00026H.pdf>.
- Francisco, J. Garijo, J. Gomez-Sanz Jorge, and Massonet Philippe. 2005. "The MESSAGE methodology for agent-oriented analysis and design." In *Agent-Oriented Methodologies*, 203-235. Hershey, PA, USA: IGI Global.
- Gomez, C, D Isern, and A Moreno. 2007. *Software engineering methodologies to develop multi-agent systems: state-of-the-art*.
- Huget, Marc-Philippe. 2003. "Agent UML class diagrams revisited." In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, 49-60. Springer.

- Huget, Marc-Philippe, and James Odell. 2005. "Representing agent interaction protocols with agent UML." In *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004. Revised Selected Papers*, eds James Odell, Paolo Giorgini and Jörg P. Müller, 16-30. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Huget, Marc-Philippe, James Odell, and Bernhard Bauer. 2004. "The AUML approach." In *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, eds Federico Bergenti, Marie-Pierre Gleizes and Franco Zambonelli, 237-257. Boston, MA: Springer US.
- Kasisopha, Natsuda. 2013. "Development of Semantic Wiki as a Basis for Software Engineering Ontology Evolution." Master of Philosophy (Information Systems), Curtin University.
- Massimo, Cossentino. 2005. "From requirements to code with PASSI methodology." In *Agent-Oriented Methodologies*, eds Henderson-Sellers Brian and Giorgini Paolo, 79-106. Hershey, PA, USA: IGI Global.
- Nienaber, Rita C. 2008. "A Model for Enhancing Software Project Management Using Software Agent Technology." University of South Africa.
- Padgham, Lin, and Michael Winikoff. 2003. "Prometheus: A methodology for developing intelligent agents." In *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002 Bologna, Italy, July 15, 2002 Revised Papers and Invited Contributions*, eds Fausto Giunchiglia, James Odell and Gerhard Weiß, 174-185. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Palma, Raúl, Fouad Zablith, Peter Haase, and Oscar Corcho. 2012. "Ontology evolution." In *Ontology Engineering in a Networked World*, eds Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta and Aldo Gangemi, 235-255. Springer Berlin Heidelberg.
- Sturm, Arnon, and Onn Shehory. 2014. "The landscape of agent-oriented methodologies." In *Agent-Oriented Software Engineering: Reflections on*

*Architectures, Methodologies, Languages, and Frameworks*, eds Onn Shehory and Arnon Sturm, 137-154. Berlin, Heidelberg: Springer Berlin Heidelberg.

Wood, Mark F., and Scott A. DeLoach. 2001. "An overview of the multiagent systems engineering methodology." In *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000 Limerick, Ireland, June 10, 2000 Revised Papers*, eds Paolo Ciancarini and Michael J. Wooldridge, 207-221. Berlin, Heidelberg: Springer Berlin Heidelberg.

Wooldridge, Michael, Nicholas R. Jennings, and David Kinny. 2000. "The Gaia methodology for agent-oriented analysis and design." *Autonomous Agents and Multi-Agent Systems* 3 (3): 285-312. doi: 10.1023/a:1010071910869.

Wooldridge, Michael, and Paolo Ciancarini. 2001. "Agent-oriented software engineering: The state of the art." In *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000 Limerick, Ireland, June 10, 2000 Revised Papers*, eds Paolo Ciancarini and Michael J. Wooldridge, 1-28. Berlin, Heidelberg: Springer Berlin Heidelberg.

Zablith, Fouad, Grigoris Antoniou, Mathieu d'Aquin, Giorgos Flouris, Haridimos Kondylakis, Enrico Motta, Dimitris Plexousakis, and Marta Sabou. 2015. "Ontology evolution: a process-centric survey." *The Knowledge Engineering Review* 30 (01): 45-75. doi: 10.1017/S0269888913000349.

Zimmermann, Roland. 2006. *Agent-based Supply Network Event Management, Whitestein Series in Software Agent Technologies*. Switzerland: Birkhäuser Verlag.

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.

# **Chapter 6   Ontology-based Multi-agent Approach for Capturing Software Project Information**

## **6.1   Introduction**

The previous chapter focuses on the macro-perspective of agent-oriented software engineering. The result is a holistic model of an ontology-based multi-agent system which describes agent types, their roles, and basic interactions among agents. In this chapter, the micro-perspective design of each agent type involved in the semantic annotation of software project information and the ontology population is explored. The ultimate goal is to capture, with minimum human intervention, the knowledge pertaining to the software engineering domain contained in the project information. Once the knowledge is captured and populated in the Software Engineering Ontology, it can subsequently be used by project team members or software agents to clarify any ambiguity or to address major software development issues.

The chapter begins with the design of the internal models of the SEOMAS agents, namely, user agent, versioncontrol agent, annotation agent, and ontology agent who work collaboratively to capture knowledge of software project information. Then the proposed design is realised by the prototype system implementation for proof-of-concept experiments. Afterwards, the practical application based on a case study related to bug resolution activities in a multi-site software development project is demonstrated.



## 6.2 Ontology-based Multi-agent System to Capture Software Project Information

A large volume of software project information is produced in software projects. Manually transforming or mapping them into a semantically rich form for shared understanding is time-consuming, laborious, tedious and prone to error. Hence, it is important to use a systematic approach to automate the knowledge capture of software project information. Source code is considered as the main artefact centrally located and critical in software development; therefore, this chapter focuses on capturing software engineering knowledge from the source code. The Software Engineering Ontology is mainly used to provide software engineering domain knowledge by means of a semantic annotation process, and to link the instances generated. In addition, it is used to facilitate consistent communication between agents.

There are four main agent types involved in the knowledge capturing process, namely, user agent, version control agent, annotation agent, and ontology agent. They work collaboratively to assist project team members to capture knowledge from source code artefacts by being transparently integrated into daily software development tasks (i.e., version control). In the next section, details of each agent type are given within the micro-perspective of AOSE. These details include structures, agent behaviours and related agent interactions, and are described in AUML models as shown in Figure 6-1.

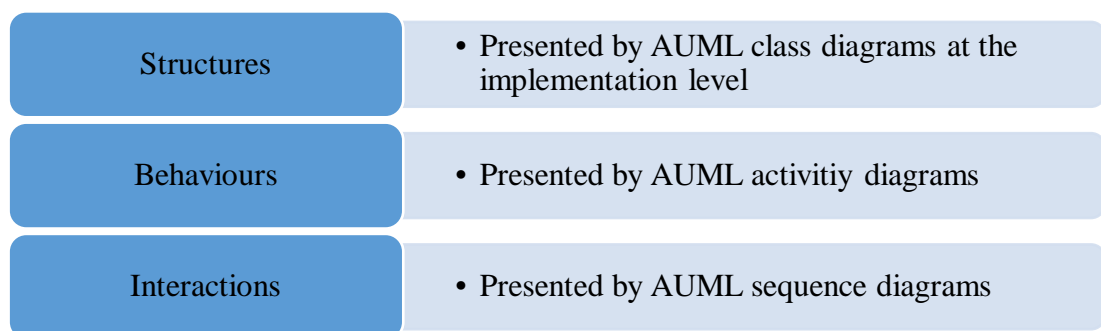


Figure 6-1: Micro-perspective of each agent type described in AUML models

The structure of an agent is presented by the AUML class diagram at the

implementation level presented in (Zimmermann 2006) which is adapted from (Huguet 2003; Bauer 2002). The AUML class diagram at the implementation level depicts several aspects of an agent as follows.

- Role - All roles given to an agent type from the micro-perspective are presented.
- Knowledge Asset – Main types of knowledge resources that an agent type requires in order to fulfil its task.
- Behaviour – Various behaviours that an agent requires to fulfil its objective are presented. They are similar to methods in object-oriented class diagrams. Each behaviour is demonstrated in the format of Behaviour-type [pre-condition] Behaviour name [post-condition]. If a pre-condition is achieved, the behaviour will be activated. A post-condition demonstrated the goal which is achieved. Three types of behaviours are distinguished according to (Zimmermann 2006; Huguet 2003) which are proactive, reactive, and internal. Proactive behaviours (Pro) are initiated by an agent based on their knowledge, goals, or some given conditions such as timer, exceptions, or conditions. Reactive behaviours (Reac) are triggered by a change in the environment received with the agent's sensor or in reaction to other agents' actions. Internal behaviours (Int) are not visible to other agents. They are initiated by a direct call from other behaviours or by a precondition which is the consequence of another agent's behaviour and are defined within the agent.
- Perception: The various types of inputs that an agent perceives through its sensors are presented.
- Protocol: The interaction protocols that an agent uses to construct a conversation message to communicate with other agents. The standardised FIPA Request interaction protocol (FIPA 2002) is used as a main protocol in this research.
- Collaborator: The other agent types that an agent collaborates with.

There are two steps to model behaviours of an agent. The first step is to structure the overview of all behaviours of an agent and the second step is to model details of a behaviour with its various activities, potential alternatives and required inputs and outputs. They are modelled based on UML activity diagrams proposed in (Zimmermann 2006) adapted from a proposal of (Huget 2003). The interactions among each agent type are modelled based on AUML sequence diagrams according to its behaviours that act upon incoming messages or sending messages to other agents.

## 6.3 User Agent

A user agent is mainly responsible for mediating between a user and the system. It manages user interface presentation and interactions. A user employs his/her user agent to perform tasks on his/her behalf. In this chapter, a user makes a request to import a software project information file to the version control repository through his/her user agent. It then sends a request to the versioncontrol agent to import the file. Once the file has been imported and has automatically captured and populated in the Software Engineering Ontology, a user agent also receives a messages to notify the user to validate and verify the logical consistency of the ontology instances.

It is to be noted that in this chapter, the focus is only on the design and development of the SEOMAS approach for the automated knowledge capture of software project information. Therefore, the user agent's features and behaviours associated with the management of Software Engineering Ontology instantiation are not mentioned here but will be described in Chapter 7.

### 6.3.1 Structure

An overview of the structural features of a user agent is illustrated with the AUML class diagram at implementation level in Figure 6-2. Two main roles are assigned to a user agent, namely, *ACLMessageGenerator* and *OutputGenerator*. The

knowledge asset or resource that any user agent requires to conduct its activities is the Software Engineering Ontology. A user agent uses it as a shared ontology that enables knowledge-level communication in the Software Engineering domain and facilitates consistent communication among agents. The concepts defined in the ontology are derived from the concepts structured in the Software Engineering to facilitate agent interoperability. Every software agent type uses the Software Engineering Ontology to assist in creating its ACL messages and to translate the content received from other agents. The benefit of using the Software Engineering Ontology for the agents' communication is that it enables semantic interoperability so that the messages will be understood by all agents. A user agent collaborates with the version control agent by asking it to import software project information file (i.e., source code file) into a version control repository. It also collaborates with the ontology agent by receiving a message to notify a team member about the ontology population.

The *ACLMessageGenerator* role derives the reactive behaviour *GenerateACLMessage*, while the *OutputGenerator* role derives the reactive behaviour *GenerateOutput*. These two behaviours are analysed in the subsequent section. Perceptions of a user agent are based on a request from its user (userRequest) and from the ACL messages received from the versioncontrol agent (confirmation or failure) and the ontology agent (ontologyPopulationNotification). Regarding the protocol being used, a user agent employs an ACL message with a FIPA-request protocol to request the versioncontrol agent to import a source code file to a version control repository.

<b>&lt;&lt;Agent&gt;&gt;</b> User Agent
<b>Role</b> - ACLMessageGenerator - OutputGenerator
<b>Knowledge Asset</b> Software Engineering Ontology
<b>Collaborator</b> - VersionControl agent - Ontology agent
<b>Behaviour</b> <<Reactive>> - Reac [userRequest] GenerateACLMessage [messageGenerated]

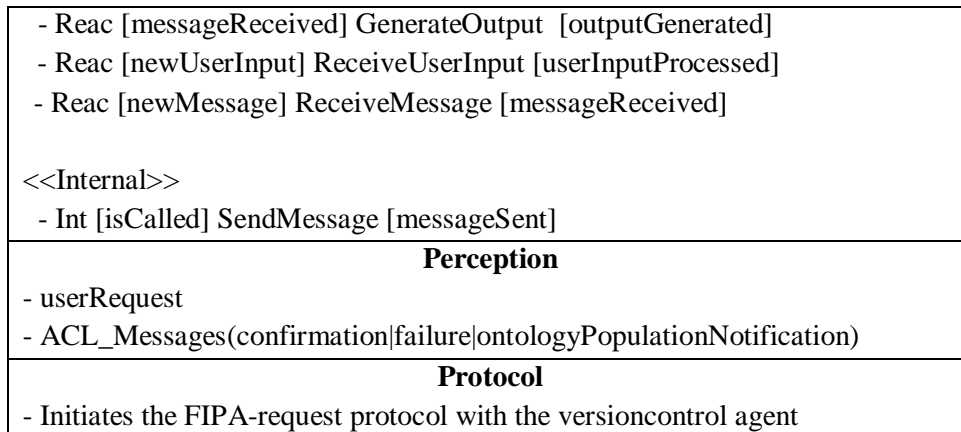


Figure 6-2: AUML class diagram at implementation level of a user agent

## 6.3.2 Behaviours

### 6.3.2.1 Overview

An overview of the behaviours associated with the roles of a user agent and the interdependencies between these behaviours is depicted in Figure 6-3. The behaviour diagrams based on UML activity diagrams adopted from (Zimmermann 2006) are used to describe these behaviours. There are three behaviours that realise the functions of sensors and effectors of a user agent:

1. The reactive behaviour *ReceiveUserInput* is triggered when a user makes a new request through his/her user agent.
2. The reactive behaviour *ReceiveMessage* is activated when a new agent message is received. It extracts the message content and makes it available to other behaviours of the agent.
3. The internal behaviour *SendMessage* is initiated by other behaviours of the agent. It works by sending out ACL messages to other agents.

It is to be noted that the behaviours *ReceiveMessage* and *SendMessage* are two basic behaviours used for realising functions of sensors and effectors of all agent types in the SEOMAS framework.

As an *ACLMessageGenerator* role, the reactive behaviour *GenerateACL-Message* acts upon user requests that are received by the *ReceiveUserInput* behaviour. It translates user requests into ACL messages and sends them to the

corresponding agent by the *SendMessage* behaviour. Within an *OutputGenerator* role, when a user agent receives a message by the *ReceiveMessage* behaviour, the reactive behaviour *GenerateOutput* is triggered. It transforms the ACL message into a meaningful output and displays it to the user.

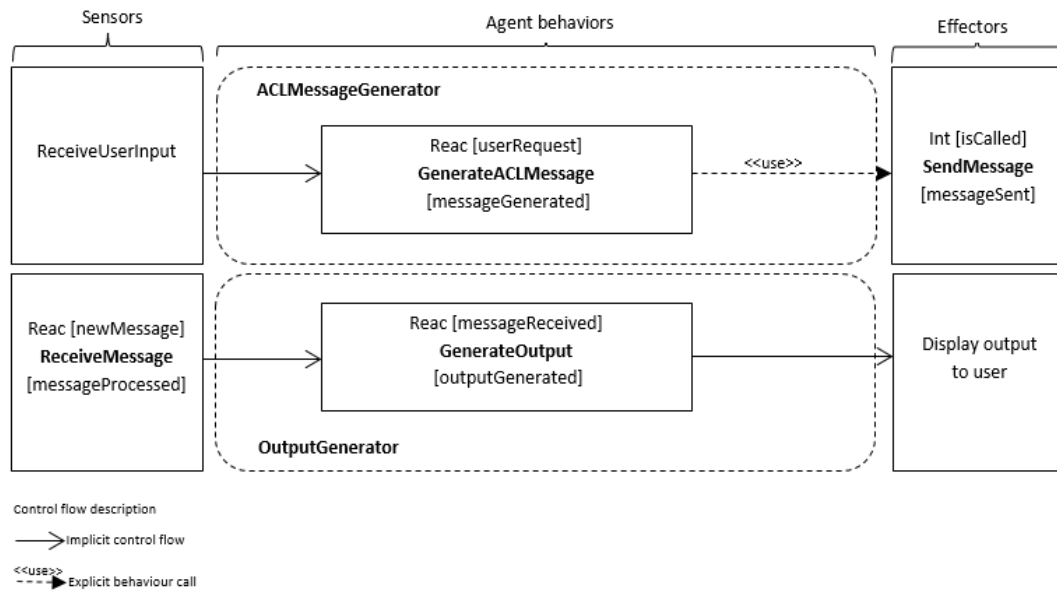


Figure 6-3: Overview of user agent behaviours

### 6.3.2.2 ACL message generation behaviour

The *ACLMessageGenerator* role is realised by the behaviour *GenerateACLMessage*. It constructs the ACL message based on a user request and user input parameter. Detailed AUML activity diagrams for this behaviour are shown in Figure 6-4. When a new request from a user is received, the *GenerateACLMessage* behaviour becomes active. First, it identifies the type of requested service (e.g. import software project information). Second, it defines the communicative act of a newly constructed ACL message (e.g. Request) to indicate the action that the message is meant to perform. Third, it specifies who is sending this message and to whom this message is being sent. In this chapter, a user agent sends a request for importing software project information to the versioncontrol agent. Then it sets the interaction protocols to construct an agent conversation message (e.g., FIPA-Request). The Software Engineering Ontology is registered to provide the context of the system and to support the interpretation of the content by the receiving agent.

Then it defines the message content and the content language to indicate the representation of the message content. Finally, a new ACL message is sent to other agent types through the *SendMessage* behaviour.

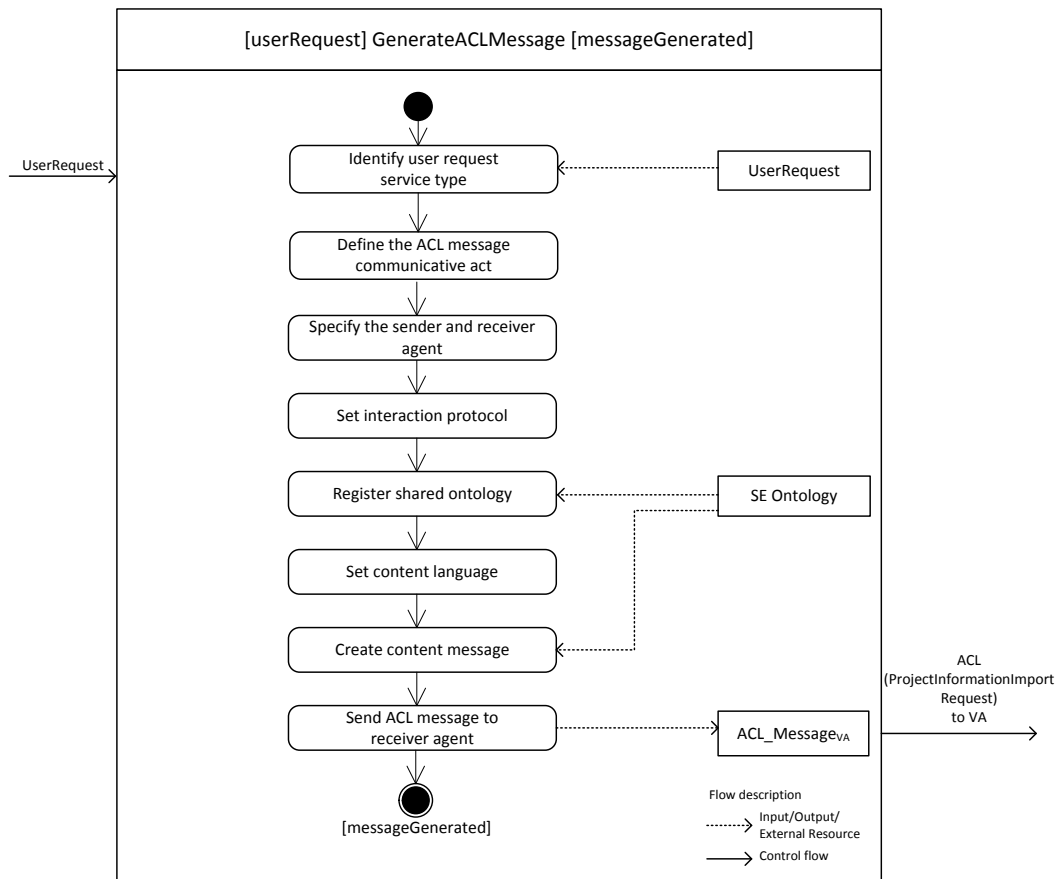


Figure 6-4: The GenerateACLMessage behaviour details

### 6.3.2.3 Output generation behaviour

When a user agent receives a message from other agents, the reactive behaviour *GenerateOutput* is activated (Figure 6-5). It first identifies a received ACL message content and the communicative act to analyse the type of message (e.g. inform, confirm, failure). The output is generated based on the communicative act type and the content of the message. Finally, the output is sent to be displayed to the user. In this chapter, a user agent perceives a message from the versioncontrol agent to notify the result of importing software project information file into the version control repository or a message from the ontology agent to notify about the ontology population.

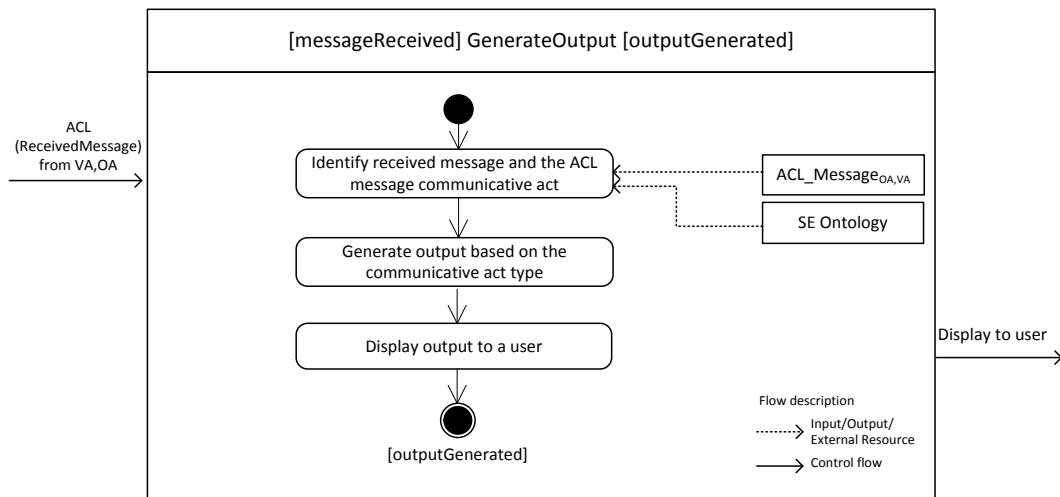


Figure 6-5: Details of GenerateOutput behaviour

### 6.3.3 Interactions

The AUML sequence diagram is used to demonstrate the interactions among different agent types. Figure 6-6 illustrates the main interactions among a user agent, the versioncontrol agent, and the ontology agent resulting from the behaviours of a user agent, namely, *GenerateACLMessage* and *GenerateOutput*. The *GenerateACLMessage* behaviour of a user agent generates an ACL message to request the versioncontrol agent to import software project information file into a version control repository. The *GenerateOutput* behaviour manages the ACL messages sent from the version control agent and the ontology agent by transforming them into meaningful outputs which are then delivered to a user.



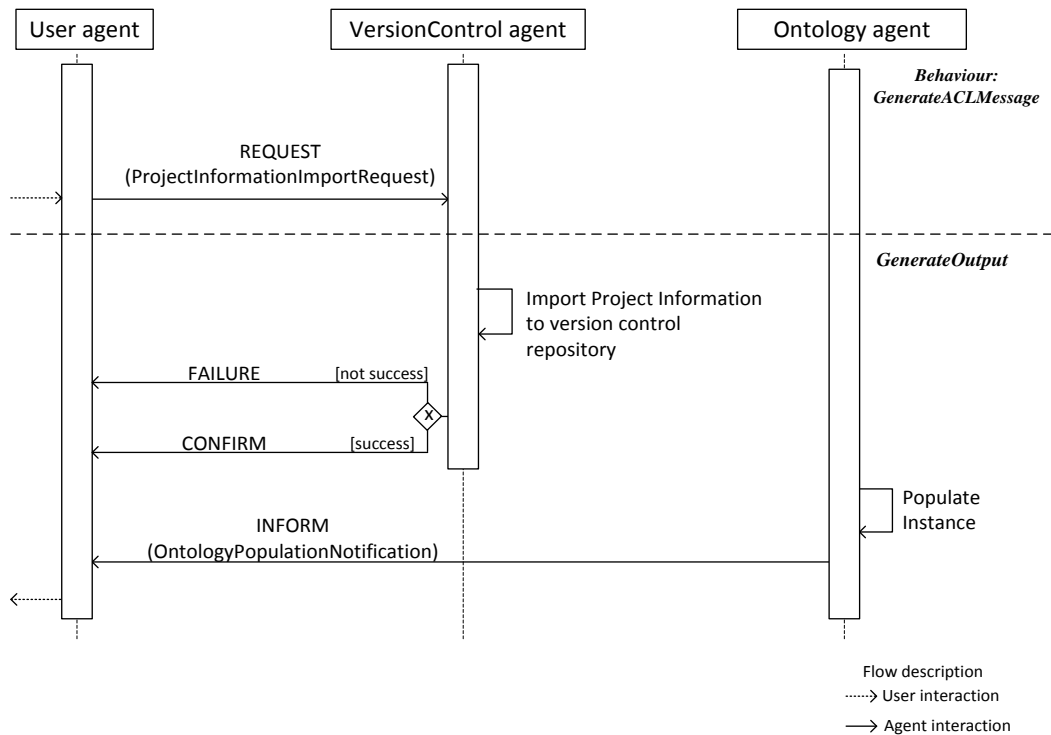


Figure 6-6: Agent interaction of a user agent

## 6.4 VersionControl Agent

The versioncontrol agent is mainly responsible for managing the version control repository. In this thesis, once it receives a request from a user agent, it imports the requested software project information file into the version control repository.

### 6.4.1 Structure

An overview of the structural features of the versioncontrol agent is provided in the AUML class diagram at the implementation level presented in Figure 6-7. There is only one role associated with this agent which is the *VersionControlManager* role. This role involves managing software project information in the version control repository. The main knowledge assets of the versioncontrol agent is the Software Engineering Ontology used as a shared ontology that enables knowledge-level communication in the Software Engineering domain in

order to facilitate consistent communication among agents.

The perception of the versioncontrol agent is an ACL request message to import software project information file issued by a user agent (*projectInformationImportRequest*). Its interactions are performed by employing ACL messages with the FIPA-request protocol to collaborate with a user agent and the annotation agent. It responds to the FIPA-request protocol with a user agent to import project information file into the version control repository while it initiates the FIPA-request protocol with the annotation agent to annotate software project information.

<b>&lt;&lt;Agent&gt;&gt;</b> VersionControl Agent
<b>Role</b> - VersionControlManager
<b>Knowledge Asset</b> Software Engineering Ontology
<b>Behaviour</b> <<Reactive>> - Reac [newMessage] ReceiveMessage [messageReceived] <<Internal>> - Int [projectInformationImportRequest] ImportProjectInformation [requestProcessed] - Int [isCalled] SendMessage [messageSent]
<b>Perception</b> ACL_Messages(ProjectInformationImportRequest)
<b>Protocol</b> - Responds to the FIPA-request protocol with a user agent to import project information file into the version control repository - Initiate the FIPA-request protocol with the annotation agent to annotate software project information
<b>Collaborator</b> - User agent - Annotation agent

Figure 6-7: AUML class diagram at implementation level of the versioncontrol agent

## 6.4.2 Behaviours

### 6.4.2.1 Overview

An overview of behaviours associated with the roles of the versioncontrol agent and the interdependencies between these behaviours is depicted in Figure 6-8. Two basic behaviours, *ReceiveMessage* and *SendMessage*, are defined to realise the

functions of sensors and effectors of the agents. *ReceiveMessage* is a reactive behaviour that responds to a new message received from other agents. It extracts the message content and makes it available to other behaviours of the agent. *SendMessage* is an internal behaviour used to generate and send out the ACL message to other agents.

A reactive behaviour *ImportProjectInformation*, acts upon a user agent's request perceived by the *ReceiveMessage* behaviour. It is responsible for retrieving the requested project information file from the software project information repository and importing it into the version control repository. Once the file is imported, it initiates the *SendMessage* behaviour to request the annotation agent to initiate the semantic annotation process.

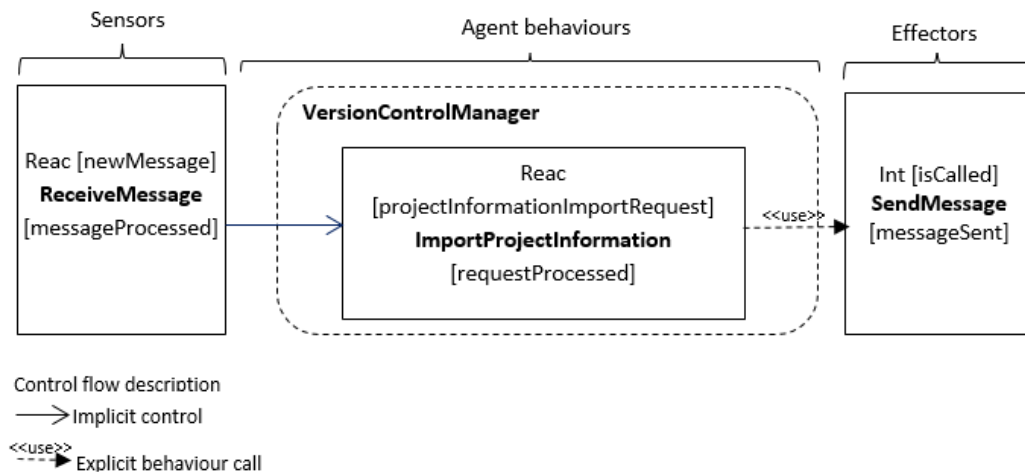


Figure 6-8: Overview of the versioncontrol agent behaviours

#### 6.4.2.2 VersionControlManager Behaviour

The VersionControlManager role is realised by a reactive behaviour, *ImportProjectInformation*. It becomes active when a request to import software project information is perceived by the behaviour *ReceiveMessage*. The behaviour *ImportProjectInformation* retrieves a requested project information file from the project information repository. If the project information file is retrieved successfully, it is imported into the version control repository. Then, a message to confirm the success of the file import is sent to a user agent and a request for a

semantic annotation process is sent to the annotation agent. However, if the file retrieval from the project information repository fails, a message to notify the failure is sent to a user agent (Figure 6-9).

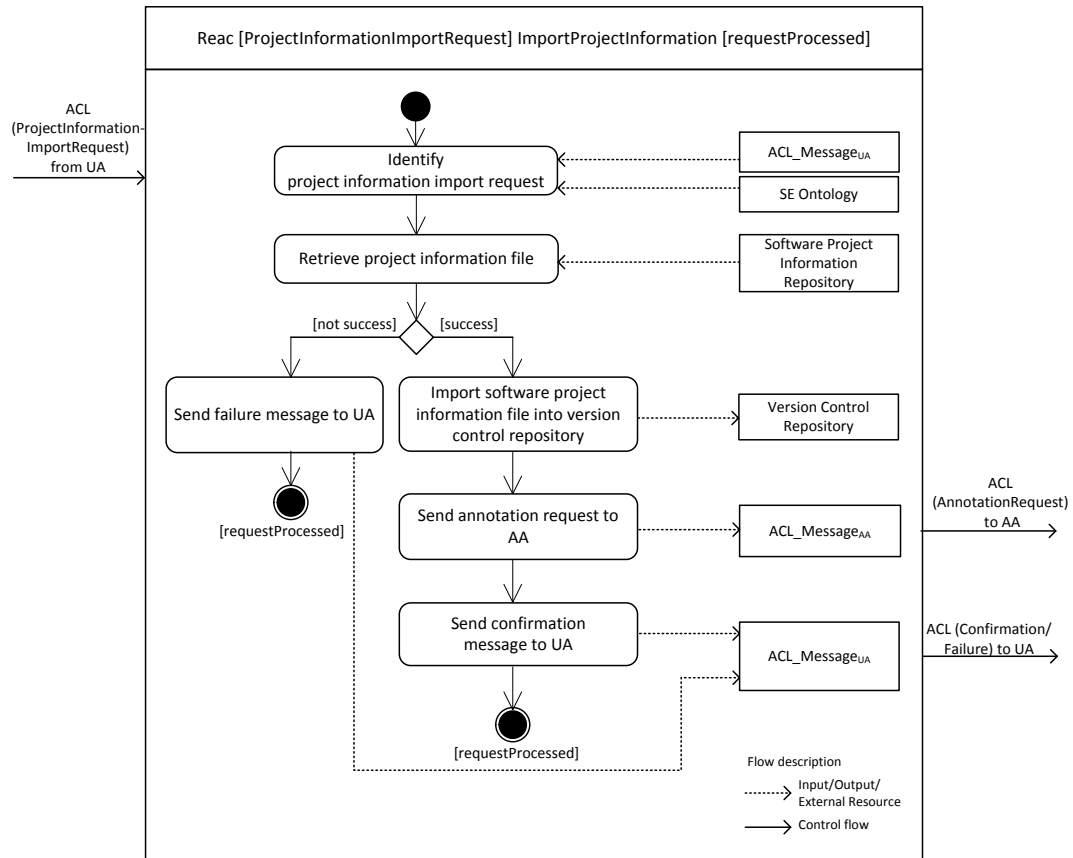


Figure 6-9: Activities of ImportProjectInformation behaviour

### 6.4.3 Interactions

Figure 6-10 illustrates the interactions among the versioncontrol agent, a user agent, and the annotation agent within the behaviour *ImportProjectInformation*. Once the versioncontrol agent receives a request from a user agent, it imports the software project information file into the version control repository and sends the results back to a user agent. It also interacts with the annotation agent by sending a request to annotate a software project information file.

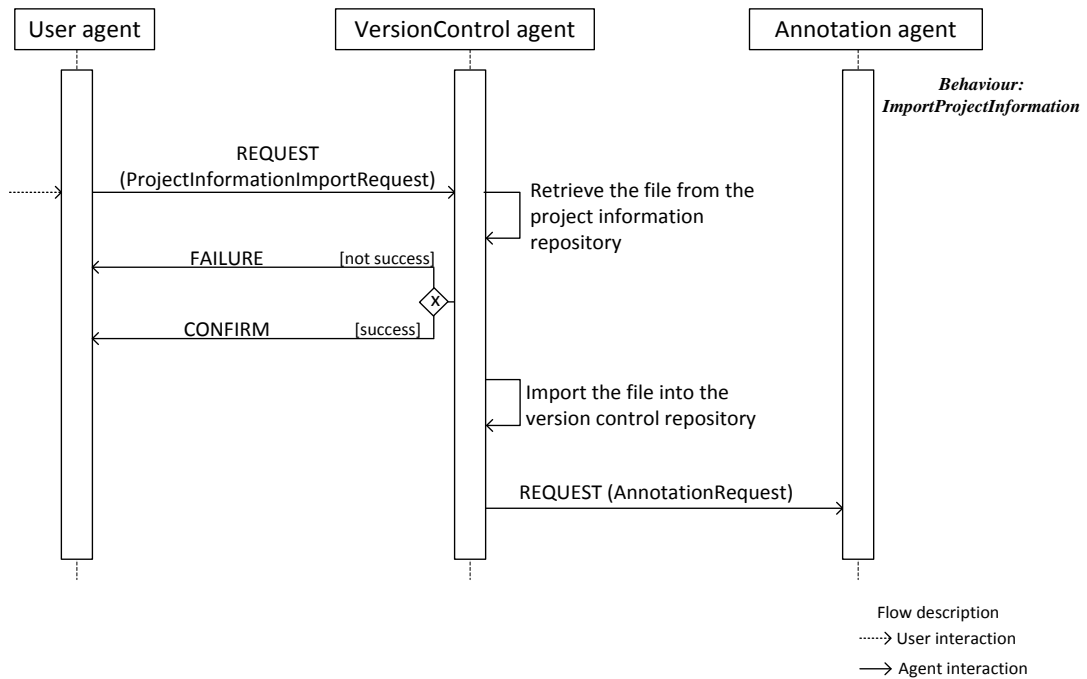


Figure 6-10: Agent interaction of the versioncontrol agent

## 6.5 Annotation Agent

The annotation agent is mainly responsible for semantically annotating software project information (i.e., source code artefact). It responds to an annotation request from the versioncontrol agent by carrying out a semantic annotation process in order to identify new instances of the Software Engineering Ontological concepts from the source code artefacts.

### 6.5.1 Structure

An overview of the structural features of the annotation agent is provided in the AUML class diagram at the implementation level presented in Figure 6-11. The role associated with this agent is *SemanticAnnotator*. The *SemanticAnnotator* role is to semantically annotate source code artefacts with the appropriate concepts defined in the Software Engineering Ontology. Its main resources comprise various knowledge assets, namely, Software Engineering Ontology, and other ontologies and

controlled vocabularies.

- The Software Engineering Ontology is used to provide domain knowledge to source code artefacts during the annotation process, and to link the instances generated from the annotated source code elements. Furthermore, it is used as a shared ontology that enables knowledge-level communication in the Software Engineering domain in order to facilitate consistent communication among agents.
- Other ontologies and controlled vocabularies are used to enrich the semantic description of the annotated source code. For example, Friend-of-a-Friend (FOAF) vocabulary (Brickley and Miller 2014) is used for information describing people and representing relationships. Simple Knowledge Organisation System (SKOS) (Miles et al. 2005) is used to aggregate concepts/terminologies into a single concept scheme. The Semantically-Interlinked Online Communities (SIOC) (Breslin et al. 2006) is used to describe information from online communities. Dublin Core (DC) (Weibel et al. IETF RFC 2413, 1998) is used for resource description. In addition, the annotated source code elements are also interlinked with the DBpedia dataset in order to provide an extended view of them. DBpedia is chosen here because it is a community effort to extract structured information from Wikipedia<sup>1</sup> and to make this information available on the web. It contributes to the Linked Data idea by interlinking with several data sources on the Web via RDF links (Auer et al. 2007).

The annotation agent fulfils its role with two main behaviours: *IdentifySourceCodeKeyConcepts* and *AnnotateSourceCode*. Details of these roles are analysed and discussed in the next section. The main perception of the annotation agent is the annotation request (`annotationRequest`) from the versioncontrol agent. Its interactions are performed by employing ACL messages with the FIPA-request protocol to collaborate with the versioncontrol agent and the ontology agent. It responds to the FIPA-request protocol with the versioncontrol agent for the source code annotation and it initiates the FIPA-request protocol with the ontology agent for

---

<sup>1</sup> <https://www.wikipedia.org/>

the ontology population.

<b>&lt;&lt;Agent&gt;&gt;</b> Annotation Agent
<b>Role</b>
- SemanticAnnotator
<b>Knowledge Asset</b>
Software Engineering Ontology, Other ontologies and controlled vocabularies
<b>Behaviour</b>
<<Reactive>> - Reac [annotationRequest] IdentifySourceCodeKeyConcepts [sourceCodeIdentified] - Reac [newMessage] ReceiveMessage [messageReceived]
<<Internal>> - Int [sourceCodeIdentified] AnnotateSourceCode [sourceCodeAnnotated] - Int [isCalled] SendMessage [messageSent]
<b>Perception</b>
ACL_Messages(annotationRequest)
<b>Protocol</b>
- Responds to the FIPA-request protocol with the versioncontrol agent for source code annotation - Initiates the FIPA-request protocol with the ontology agent for ontology population
<b>Collaborator</b>
- VersionControl agent - Ontology agent

Figure 6-11 AUML class diagram at implementation level of the annotation agent

## 6.5.2 Behaviours

### 6.5.2.1 Overview

An overview of behaviours associated with the roles of the annotation agent and the interdependencies between these behaviours is depicted in Figure 6-12. Two basic behaviours, *ReceiveMessage* and *SendMessage*, are defined to realise the functions of sensors and effectors of the agents. *ReceiveMessage* is a reactive behaviour that responds to a new message received from other agents. It extracts the message content and makes it available to other behaviours of the agent. *SendMessage* is an internal behaviour used to generate and send out the ACL message to other agents.

A reactive behaviour *IdentifySourceCodeKeyConcepts*, acts upon the

versioncontrol agent's request perceived by the *ReceiveMessage* behaviour. It is responsible for retrieving the requested source code file from the version control repository. It then identifies the key elements that are being used in the source code (e.g., class, field, method, and interface). *SourceCodeIdentified*, as the post-condition, is applicable as the pre-condition to an internal behaviour *AnnotateSourceCode*. This behaviour semantically annotates the source code key concepts using the Software Engineering Ontology in order to identify new instances of the ontological concepts. In addition, it also applies other relevant domain ontologies and controlled vocabularies to enrich and interlink their semantic descriptions. Once these processes are done, the *SendMessage* behaviour is activated to send a request to the ontology agent to populate these annotated source code elements into the ontology repository as new instantiations.

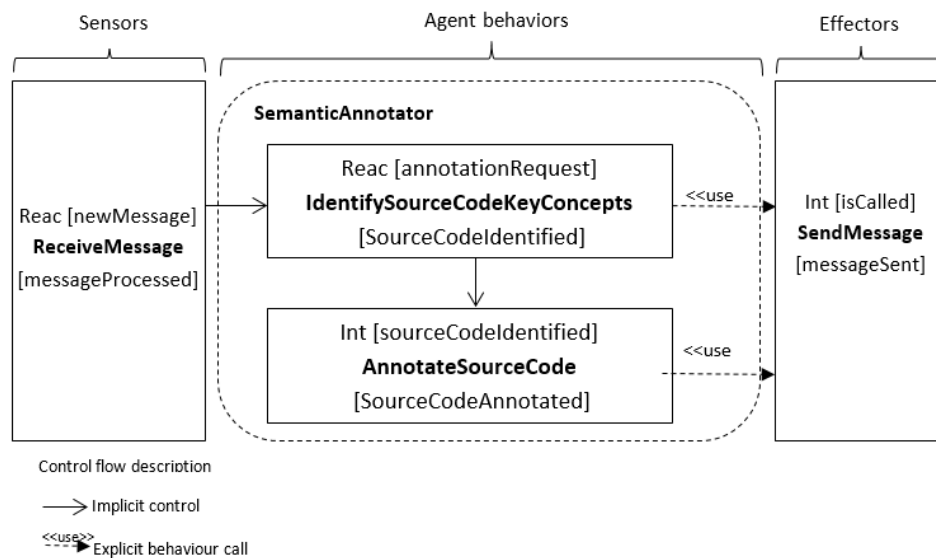


Figure 6-12: Behaviour overview diagram of the annotation agent

### 6.5.2.2 Semantic Annotation behaviours

The *SemanticAnnotator* role involves two behaviours: a reactive behaviour and an internal behaviour, which are *IdentifySourceCodeKeyConcepts* and *AnnotateSourceCode*, respectively.



### 6.5.2.2.1 IdentifySourceCodeKeyConcepts behaviours

An incoming request for source code annotation from a user agent is managed by the *IdentifySourceCodeKeyConcepts* behaviour. Two main steps are performed after a request has been identified:

#### 1. Source code retrieval

This step is to retrieve the requested source code file from the version control repository.

#### 2. Key concept identification

This step is to identify the key concepts that are being used in the source code. The source code is analysed and parsed to produce an abstract syntax tree (AST) which is a representation of the abstract syntactic structure of the source code written in a programming language, for example, classes, fields, methods, constructors, parameters as well as in-line comments (e.g., JavaDoc). For source code comments such as author, versions are also identified and parsed in order to obtain a meaningful term-based description of the source code (Figure 6-13).

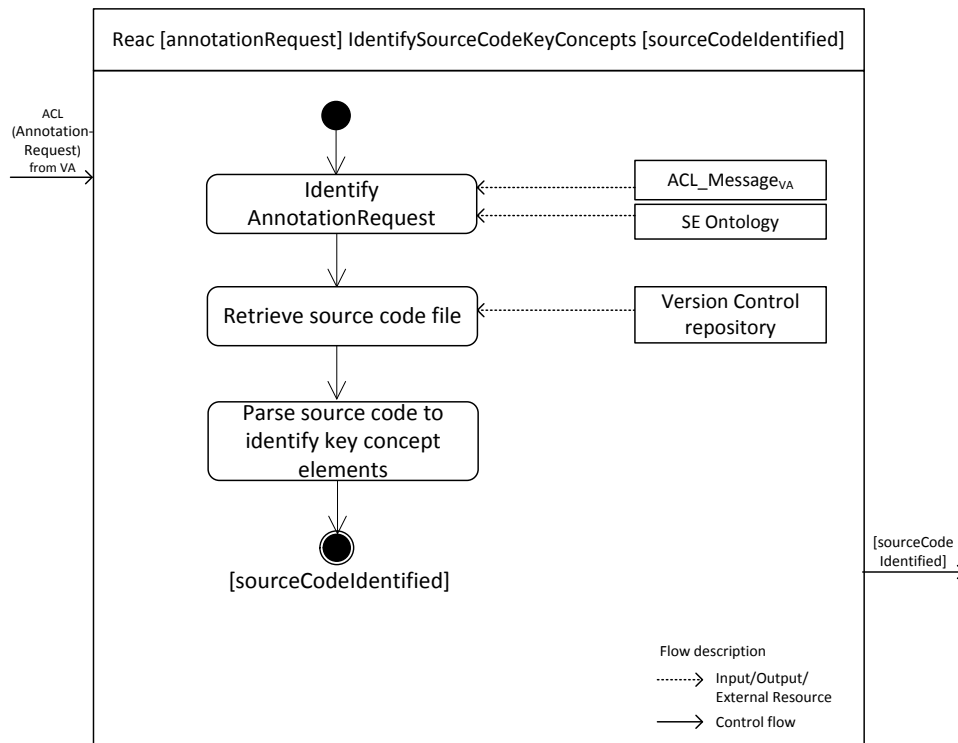


Figure 6-13: Details of IdentifySourceCodeKeyConcepts behaviour

### 6.5.2.2.2 AnnotateSourceCode behaviours

After the *IdentifySourceCodeKeyConcepts* behaviour accomplishes its task of key concept identification, the *AnnotateSourceCode* behaviour is initialised as indicated in the pre-condition [*sourcecodeIdentified*]. It annotates the source code elements with the appropriate concepts defined in the Software Engineering Ontology and other well-known ontologies and vocabularies, as well as to enrich and to interlink the annotated source code with similar concepts in other datasets (Figure 6-14). This behaviour comprises two main tasks:

#### 1) Source code annotation

The identified source code elements and other software artefacts are assigned software engineering domain concepts that correspond to their semantic description specified in the Software Engineering Ontology. Examples of these concepts are Class, Field, Method, Parameter, Modifier, etc. The source code elements that are assigned to those concepts are used to construct statements in the format of RDF/OWL triples which comprise three elements, namely, subject, predicate, and object (subject, predicate, object). The subject part identifies the thing that the statement is about. The predicate part identifies the property or characteristic of the subject that the statement specifies. The object part identifies the value of the property or characteristic (Beckett and McBride 2004). The RDF/OWL statement can be used to semantically describe:

- resource type of the source code elements such as (HelloWorld, type, Class),
- attribute of the source code elements such as (HelloWorld, isMainClass, “True”), or
- to define the relationship between source code elements such as (HelloWorld, hasMethod, main).

#### 2) Enrichment and Interlinking

In this step, other relevant domain ontologies and controlled vocabularies, namely, FOAF, DC, SKOS, SIOC are reused to enrich and interlink the semantic

description of the annotated source code. For example, all the source code elements (e.g., class, package, interface, etc.) are annotated with the relationship *rdf:type* as Dublin Core Metadata Initiative (DCMI) Type '*Software*'<sup>2</sup>. If the name of an author is available in the source code, then this relationship is defined in the resulting RDF/OWL triple by using *foaf:name*. The use of existing domain ontologies can enhance the re-useability factor and promote data interoperability (Ashraf, Hussain and Hussain 2012) as well as help to find semantic similarities with other similar entities described in different semantic repositories. Interlinking also includes the construction of semantic relationships between the annotated source code elements and other entities defined in other dataset on the Web, namely, Wikipedia. In other words, interlinking can enable extensive textual information related to the annotated source code elements or other project-related resources to be retrieved from the Wikipedia website. To extract structured information from Wikipedia and then transform it into RDF, DBpedia has been developed by the research community. The URI according to the format <http://dbpedia.org/resource/Name> corresponds with the URL of the source Wikipedia article, which has the pattern <http://en.wikipedia.org/wiki/Name> (Bizer et al. 2009). The annotation agent interlinks the annotated source code elements with the corresponding DBpedia entity by using the *owl:sameAs* property. This property is used to specify that the URIs of the annotated elements and those of DBpedia actually refer to the same entities.

After the source code has been annotated with the Software Engineering Ontology domain concepts as well as enriched and interlinked with other ontologies and controlled vocabularies, a message is sent to request the ontology agent to insert the annotated source code into the ontology as new instances.

---

<sup>2</sup> <http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=dcmitype#Software>

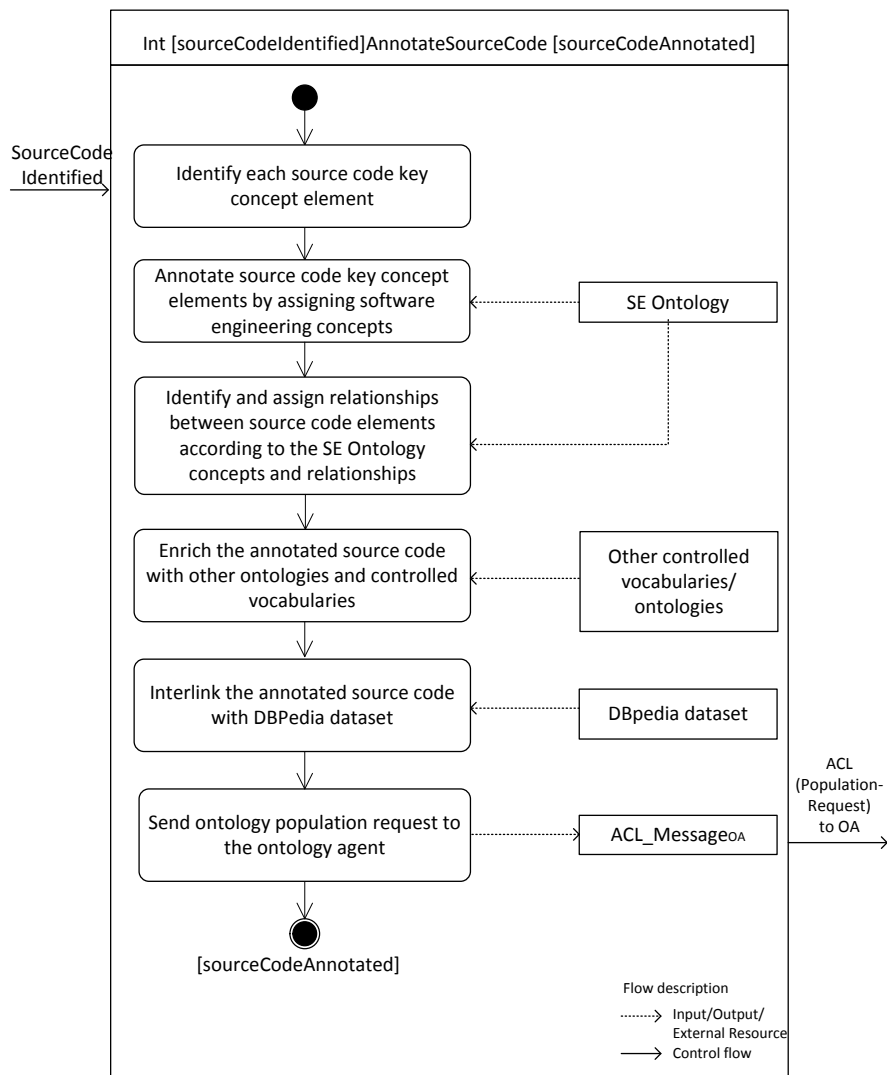


Figure 6-14: Details of AnnotateSourceCode behaviour

### 6.5.3 Interactions

Figure 6-15 illustrates the interactions among the annotation agent, the versioncontrol agent, and the ontology agent resulting from the behaviours of the annotation agent. Within the behaviour *IdentifySourceCodeKeyConcepts*, the annotation agent receives a request to annotate source code from the version control agent. It then retrieves a source code file from the version control repository and identifies key concepts from the source code elements. Once the source key concepts have been identified, the behaviour *AnnotateSourceCode* is triggered to semantically annotate the source code key concepts to identify new instances of the Software Engineering Ontology. Then it sends a request to populate these new instances to the ontology agent.

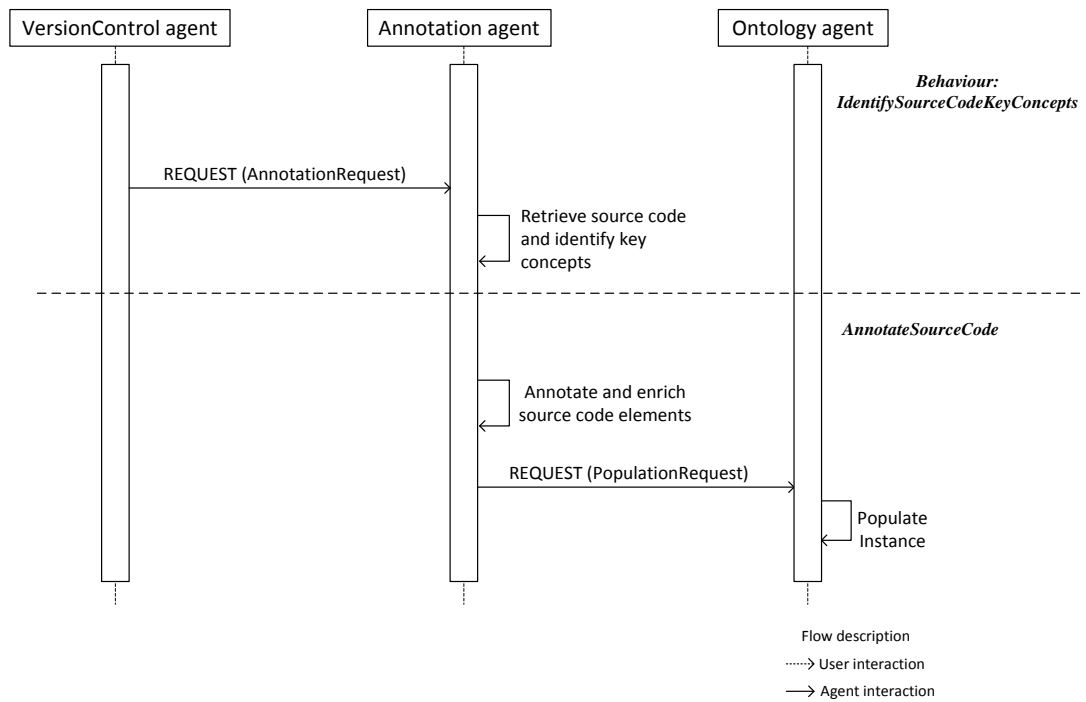


Figure 6-15: Agent interactions of the annotation agent

## 6.6 Ontology Agent

The ontology agent is mainly responsible for managing instance knowledge captured in the Software Engineering Ontology such as query, addition, modification, deletion. However, in this chapter, the details of the design and development of the ontology agent pertain only to the addition of new instance knowledge identified from the semantic annotation process. The *InstanceKnowledgeManager* role of the ontology agent is realised by a reactive behaviour *PopulateInstance* in this chapter given in the following section. More details of the ontology agent associated with this role used to manage Software Ontology Instantiations will be given in Chapter 7.

### 6.6.1 Structure

An overview of the structural features of the ontology agent is provided in Figure 6-16. In order to manage the instance knowledge, there is only one role associated with the ontology agent which is the *InstanceKnowledgeManager*. In this chapter, this role is to populate the Software Engineering Ontology with new instances identified by the annotation agent. The ontology agent's only knowledge asset is the Software Engineering Ontology. The Software Engineering Ontology domain knowledge is used to enable consistent communication among agents, while the Software Engineering Ontology repository is used to store new instances populated.

The ontology agent fulfils its role with the main behaviour *PopulateInstance*. The main perception of the ontology agent is the population request (populationRequest) from the annotation agent. Its interactions are performed by responding to the ACL message with the FIPA-request protocol to the annotation agent.

<b>&lt;&lt;Agent&gt;&gt;</b> Ontology Agent
<b>Role</b> - InstanceKnowledgeManager
<b>Knowledge Asset</b> Software Engineering Ontology
<b>Behaviour</b> <<Reactive>> - Reac [populationRequest] PopulateInstance [instancePopulated] - Reac [newMessage] ReceiveMessage [messageReceived] <<Internal>> - Int [isCalled] SendMessage [messageSent]
<b>Perception</b> ACL_Messages (populationRequest)
<b>Protocol</b> - Responds to the FIPA-request protocol with the annotation agent for the ontology population
<b>Collaborator</b> - Annotation agent - User agent

Figure 6-16 AUML class diagram at implementation level of the ontology agent

## 6.6.2 Behaviours

### 6.6.2.1 Overview

A behaviour overview diagram of the ontology agent only for the ontology population process is presented in Figure 6-17. The *InstanceKnowledgeManager* role is realised by the reactive behaviour *PopulateInstance*. It is activated when it perceives a request (populationRequest) as specified in the pre-condition. It extends the Software Engineering Ontology repository with new instances identified by the annotation agent.

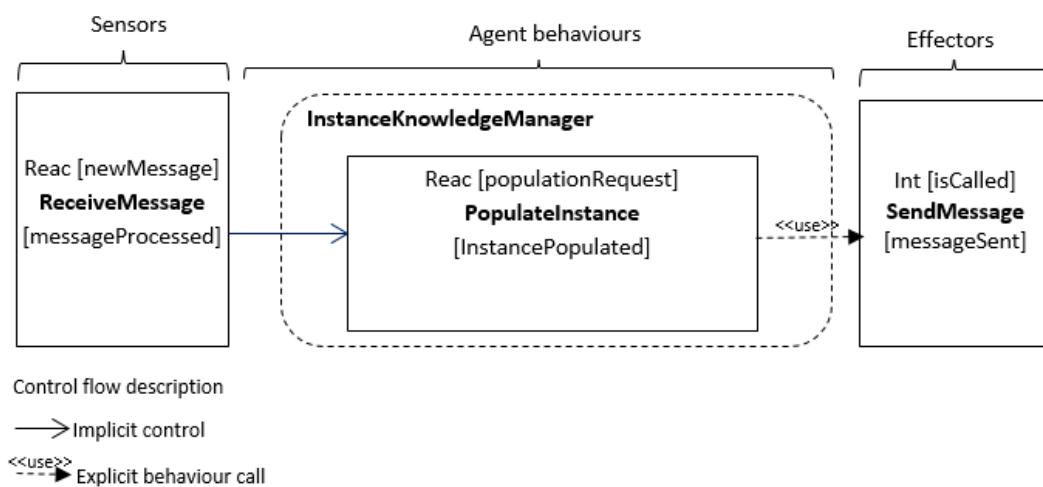


Figure 6-17: Behaviour overview diagram of the ontology agent for the ontology population

### 6.6.2.2 Ontology Population behaviour

Once the new instances are identified by the annotation agent by means of a semantic annotation process, they are ready to be populated into the Software Engineering Ontology repository as new ontological instances. In other words, the ontology population process is equivalent to the instance generation by means of inserting new instances of concepts, properties and relations into the Software Engineering Ontology instance knowledge base. According to description logics (DL) (Nardi and Brachman 2003), an ontology as a knowledge base comprises two knowledge components, namely, TBox and ABox. The TBox contains domain definitions that describe concepts and properties. The Abox, also called assertional

knowledge, specifies the individuals of the domain concepts derived from the TBox. To put it differently, the TBox comprises concepts and their relations while the ABox consists of instances of concepts or individuals. The Software Engineering Ontology and other ontologies are considered as the TBox while all annotated data with these ontologies are considered as the ABox. Therefore, the ontology population process performed by the *PopulateInstance* behaviour manages the insertion of annotated data as instances in the ABox.

The *PopulateInstance* behaviour is activated when a request from the annotation agent is perceived. It inserts the annotated and enriched source code elements into the Software Engineering Ontology instance knowledge base as new ontology instantiations. Once the population process is completed, it sends a message to notify the user agent who requests to import a source code file into the version control repository in order to validate and verify the logical consistency of the new instantiations (Figure 6-18).

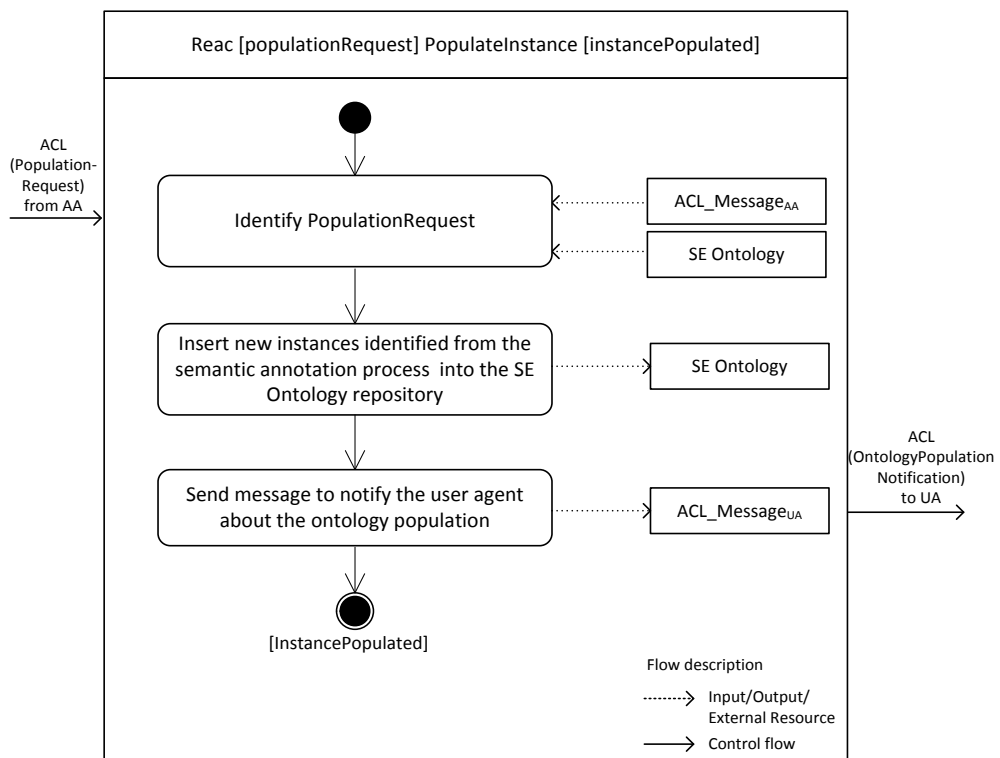


Figure 6-18: Details of *PopulateInstance* behaviour



### 6.6.3 Interactions

All interactions that result from the behaviours of the ontology agent are shown in Figure 6-19. The ontology agent receives an ontology population request from the annotation agent. Then it populates the Software Engineering Ontology with the new instances identified from the annotation process. Lastly, it sends the notification of the ontology population back to the user agent.

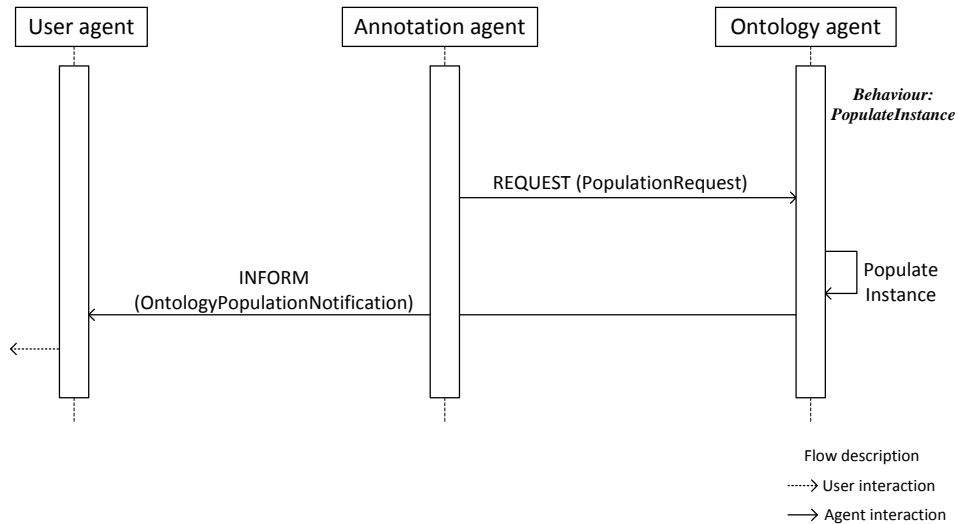


Figure 6-19: Agent interactions of the ontology agent

After the team member receives the notification about the ontology population of new instances, as a domain expert, he/she will validate and verify the instances to check their logical consistency. The new additional instances populated in the Software Engineering Ontology repository must be checked to determine whether they conform to the Software Engineering Ontology; this is done by checking the consistency of instances with reference to the ontology. This quality assurance process is done manually to ensure the production of quality information. This process can be done through ontology reasoners such as FaCT++, Pellet, RacerPro, HermiT, etc.

In order to summarise the semantic annotation and the ontology population process performed by the SEOMAS agents as described above, the whole process is shown graphically in Figure 6-20.

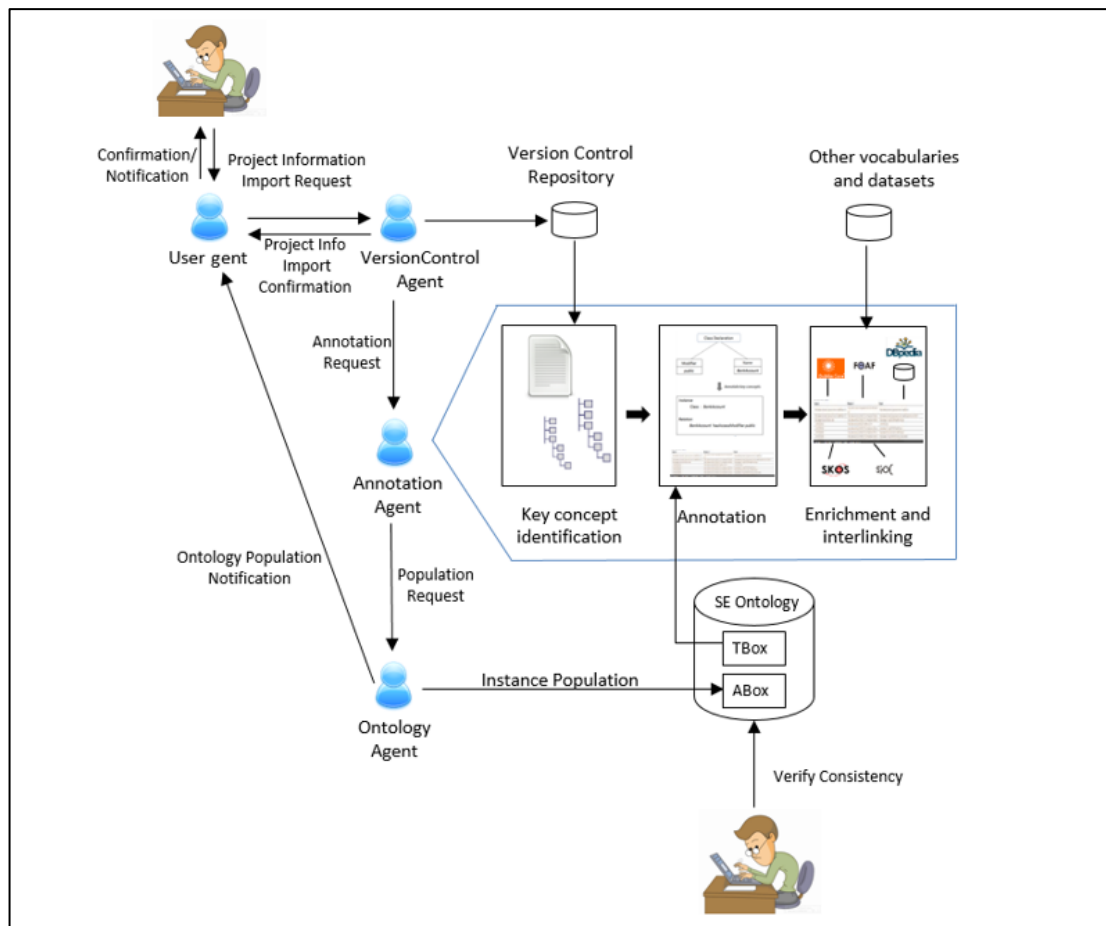


Figure 6-20: The automated knowledge capture by the SEOMAS approach

## 6.7 Implementation

The prototypes are used as proof-of-concept experiments of the proposed framework. Java source code is selected for a proof-of-concept implementation. Jena, a Java framework for building Semantic Web applications, is used to make a connection between agents and the Software Engineering Ontology and to provide several functionalities such as create, read, modify triples in RDF/OWL. Qdox is used as a parser for the extraction of source code elements. JADE, Java Agent Development Framework (Liao et al. 2011), which is an agent middleware, is chosen to implement the agent platform and to provide a development framework. JADE is developed from Java and is completely based on the Foundation for Intelligent Physical Agents (FIPA) specifications (Bellifemine, Caire and Greenwood 2007). Agent Communication Language (ACL) defined by FIPA is chosen as the language

of communication between agents. JADE provides various implemented FIPA-specified interaction protocols such as FIPA-Query, FIPA-Request and so on to construct agent conversation messages. As JADE complies with FIPA specifications, it offers several components necessary for agent management. These components are automatically activated at the agent platform start-up. For example, Directory Facilitator (DF) agents provide a naming service and yellow pages service. The Agent Management System (AMS) supervises access and usage of the agent platform. The Remote Monitoring Agent (RMA) keeps track of all registered agents. The Sniffer Agent (SA) monitors all message communications between agents. Both FIPA and JADE support the use of ontologies in their agent systems.

JADE helps to integrate ontologies to represent the application domain through its content reference model (Caire and Cabanillas 2010) as shown in Figure 6-21. The Software Engineering Ontology is registered to this model through the ontological elements, namely, predicates, concepts, and agent actions so that it can be accessed by JADE agents and used as the content of an ACL message.

- *Concept* is an expression that indicates an entity with a complex structure which can be defined in terms of slots, for example, (Class :name “BankAccount”) stating that the class name is BankAccount.
- *Predicate* is an expression that refers to something about the status of the world such as (hasMethod (Class :name “BankAccount”) (Method: name deposit)) stating that the class BankAccount has method deposit.
- *Agent action* is a special concept that indicates an action which can be performed by an agent, for instance, (Query (Class :name “BankAccount”)) stating that the agent can query the class BankAccount. In JADE, the agent actions are implemented as the objects of the class `jade.core.behaviours.Behaviour`.

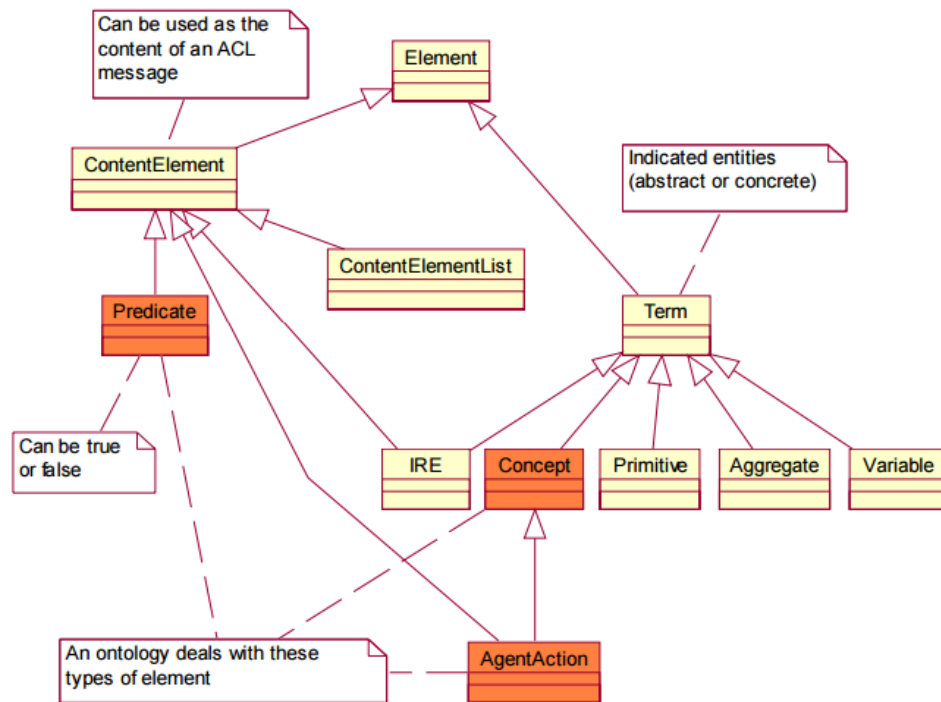


Figure 6-21: The content reference model in JADE (Caire and Cabanillas 2010)

The following five steps are used to integrate the Software Engineering Ontology into the JADE agent platform (Bellifemine, Caire and Greenwood 2007).

1. Define ontology in JADE including the schemas that define the types of predicates, agent actions and concepts relevant to the Software Engineering Ontology.

2. Develop ontological Java classes according to all types of predicates, agent actions and concepts in the ontology. This step can be done by using the Ontology Bean Generator Protégé plug-in (Aart 2007). It helps to generate JADE-compliant Java-classes from RDF(S), XML and Protégé projects.

3. Choose a suitable content language that is used by all agents to maintain the correct semantics and expression of terms for successful communication. In this work, the Semantic Language (SL) proposed by FIPA is chosen.

4. Register the selected content language (SL) and the defined ontology (SE Ontology) with the agent.

5. Create and handle content expression as Java objects that are instances of the classes in step 2. JADE will translate these Java objects to/from strings or sequences of bytes that fit the content slot of an ACL message. Table 6-1 shows the content slots for a request to semantically annotate a Java file sent from a user agent

to the recommender agent. The ACL message for this request is shown in Figure 6-22.

Table 6-1: Content slots for semantic annotation request

Sender	Alex user agent
Receiver	VersionControl agent
Communicative act	REQUEST
Language	fipa-sl
Ontology	SEOntology
Interaction protocol	FIPA-Request
Content	C:\Users\15643403\Documents\CodeTestAnno\BankAccount.java

```

Message: (REQUEST
:sender ( agent-identifier :name "Alex Agent@MASPlatform"
:addresses (sequence http://C-D-0004872.staff.ad.curtin.edu.au:7778/acc ))
:receiver (set ( agent-identifier :name VersionControlAgent@MASPlatform ) )
:content "C:\Users\15643403\Documents\CodeTestAnno\BankAccount.java"
:language fipa-sl
:ontology SEOntology
:protocol FIPA-Request
)

```

Figure 6-22: An ACL message requesting to import a Java source code file into the version control repository

The communicative act (or performative type) is a required parameter of all ACL messages to indicate the action that the message conveys. In this case, a REQUEST communicative act is used. The Software Engineering Ontology, is used as the ontology context for this content. It is a shared ontology based on the Software Engineering Ontology for facilitating agent consistent communication. The agents can use it to represent the knowledge and to send messages containing this knowledge. The interaction protocol specifies predefined sequences of messages that can be used to design agents' interactions. In this example, the FIPA-Request protocol is used to specify that a user agent wants to request the versioncontrol agent to import a source code file into the version control repository. The versioncontrol

agent extracts the content and obtains the name of the Java file (i.e. BankAccount.java). Then it imports the requested file into the version control repository as shown in Figure 6-23.

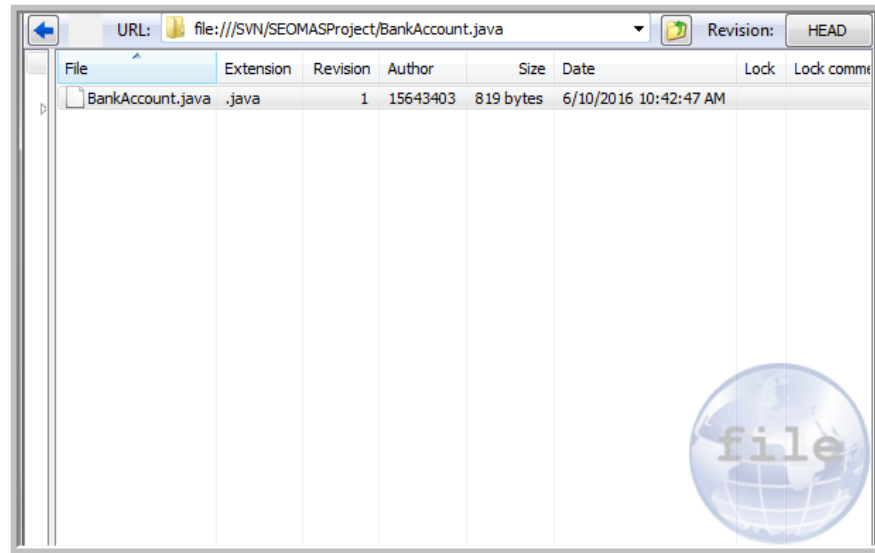


Figure 6-23: A Java source code file imported into the version control repository

Figure 6-24 presents the interactions among collaborative agents which are captured by a sniffer agent.

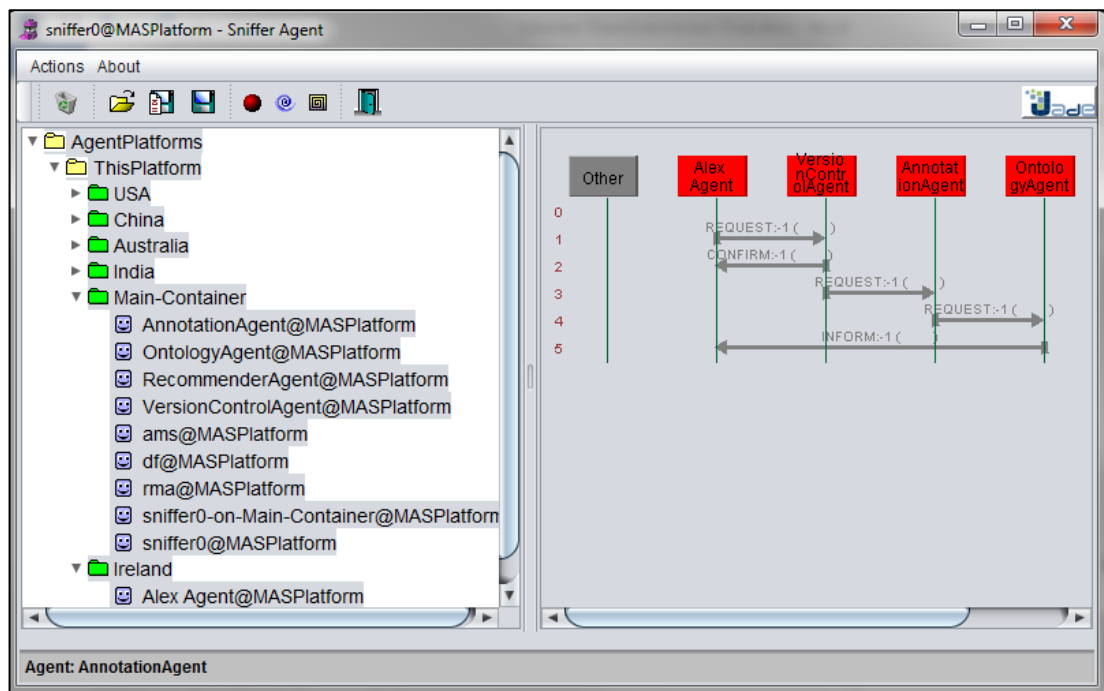


Figure 6-24: The interactions among agents captured by a sniffer agent

## 6.8 Results

In this chapter, the SEOMAS approach is capable of automatically capturing knowledge from the software project information (i.e., source code). After the version control agent imports a source code into the version control repository, the annotation agent starts the semantic annotation process. An example of how a single line of Java source code is parsed and annotated with the concepts and relationships defined in the Software Engineering Ontology by the annotation agent is shown in Figure 6-25. The code “public Class BankAccount” declares a class called BankAccount. The annotation agent parses this code to turn it into an AST as presented in Figure 6-25. It traverses this tree by processing the root node Class Declaration and realises that a Class instance will be generated. Then it processes the Name node to annotate and generate the instance of the Class class which is *BankAccount*. After that, it visits the Modifier node to annotate and generate the instance of the Modifier class which is *public*. In the ontology, class Class has a relationship *hasAccessModifier* with class Modifier; therefore, the annotation agent creates a relationship between the instances of these two classes (i.e., *BankAccount* and *public*) with the relationship *hasAccessModifier*.

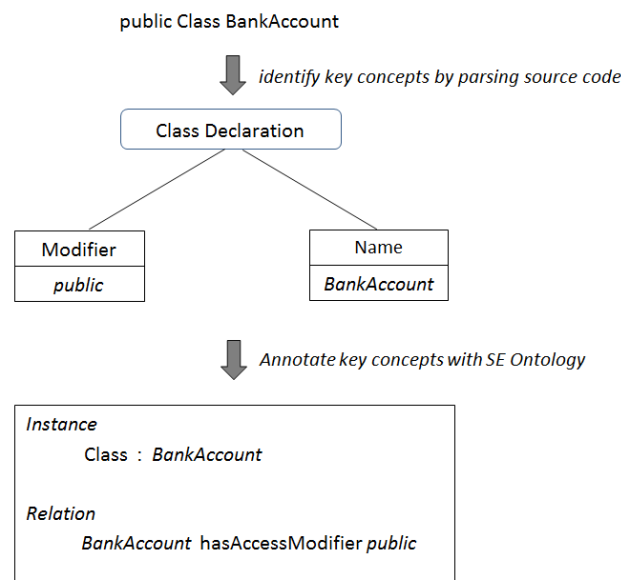


Figure 6-25: Parsed Source code to be annotated with the Software Engineering Ontology concepts and relations.

The annotation result of the Java source code, `BankAccount.java`<sup>3</sup> is shown in Figure 6-26. Each source code element is annotated with concepts and relations defined in the Software Engineering Ontology. For example, *BankAccount* is annotated as a Class and a Constructor, *public* is annotated as a Modifier, *balance* is annotated as a Field, *deposit* is annotated as a Method, etc. These elements are also identified as instances of their corresponding concepts. For instance, *BankAccount* becomes an instance of a class Class, *balance* is an instance of a class Field, *deposit* is an instance of a class Method, etc.

Each concept in the ontology has relationships to associate it with other concepts. For instance, a class Class has a relationship *hasAccessModifier* to relate it with a class Modifier and a relationship *hasMethod* to relate it with a class Method. The Class relationship also inherits its relationship to the instance level. Thus, *BankAccount* as an instance of class Class has a relationship *hasAccessModifier* to relate it with *public* as an instance of class Modifier. It also has a relationship *hasMethod* to relate it with *getBalance* which is an instance of class Method.

---

<sup>3</sup> <http://homepages.uel.ac.uk/A.Kans/pm1/week3.pdf>



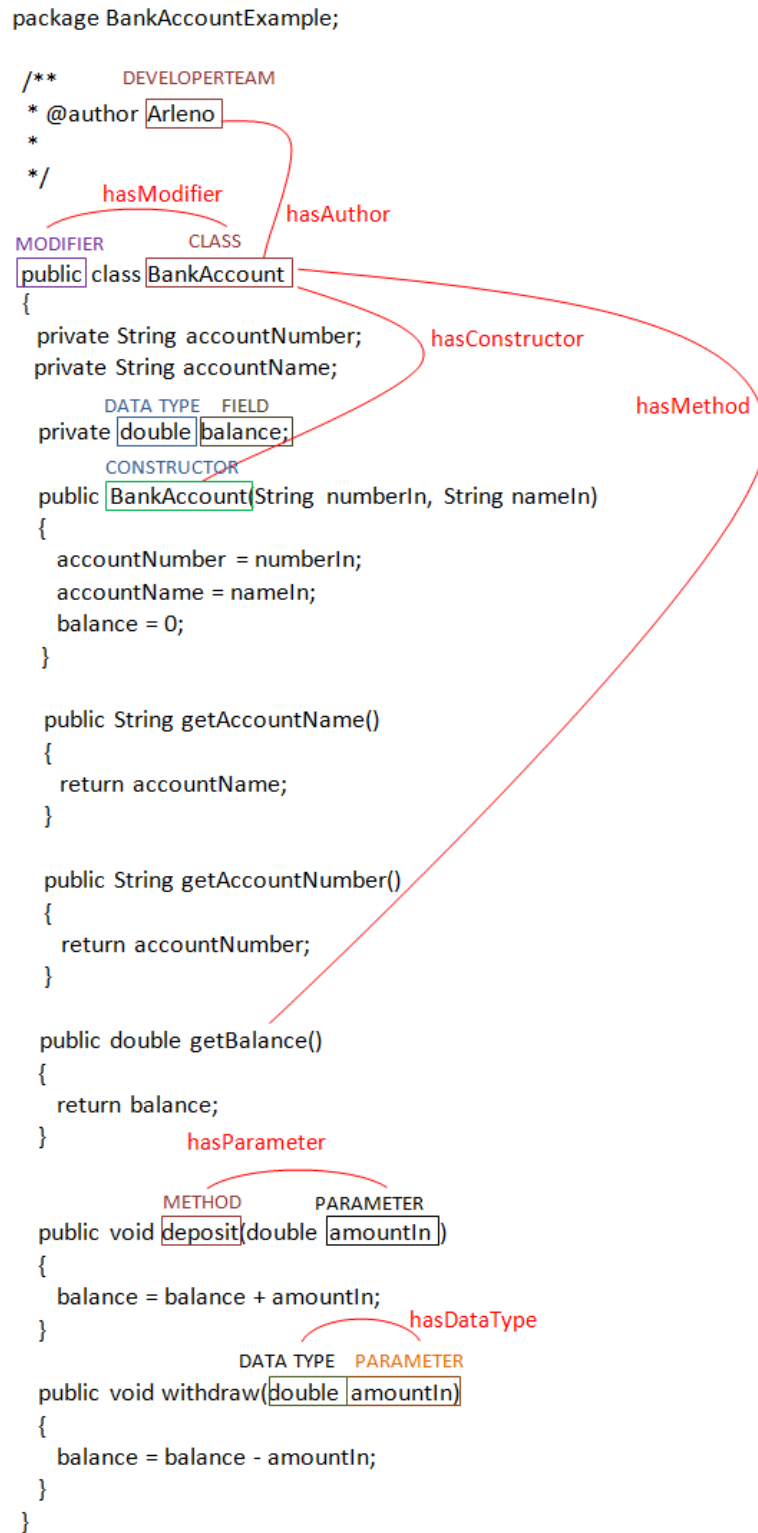


Figure 6-26: A Java source code being annotated with concepts and their relationships defined in the Software Engineering Ontology

Figure 6-27 shows the output from the semantic annotation of the BankAccount.java source code.

```
===== Semantic Annotation START =====  
File Name >> BankAccount.Java  
File Creation data >> 1462943261803  
Java Classes --> BankAccount  
Class belongs to package: BankAccountExample  
Class has following field(s):  
Field Name: accountNumber  AccessModifier: private  DataType: Java.lang.String  
Field Name: accountName  AccessModifier: private  DataType: Java.lang.String  
Field Name: balance  AccessModifier: private  DataType: double  
  
Class has AccessModifier: public  
Constructor: BankAccount  
  
Class has following method(s):  
BankAccount  
getAccountName  
getAccountNumber  
getBalance  
deposit  
withdraw  
  
Method name: BankAccount  
Modifier: public  
Parameters:  
Name: numberIn  DataType: Java.lang.String  
Name: nameIn  DataType: Java.lang.String  
  
Method name: getAccountName  
Method return type: Java.lang.String  
Modifier: public  
  
Method name: getAccountNumber  
Method return type: Java.lang.String  
Modifier: public
```

```

Method name: getBalance
Method return type: double
Modifier: public

Method name: deposit
Method return type: void
Modifier: public
Parameters:
Name: amountIn   DataType: double

Method name: withdraw
Method return type: void
Modifier: public
Parameters:
Name: amountIn   DataType: double
===== Semantic Annotation END =====

```

Figure 6-27: Output of annotation from the `BankAccount.java` source code

The ontology agent populates the Software Engineering Ontology by inserting new instances derived from the semantic annotation process into the ontology repository. For example, in Figure 6-28, *BankAccount* is semantically annotated and identified as instances of `ClassType (Class)`, `Constructor`, and `Method`. In addition, because the *BankAccount* instance is enriched with the Software concept of Dublin Core Metadata Initiative (DCMI), so it is an instance of a Software class as well. The annotated source code elements are also enriched by interlinking them with other relevant data source in order to provide an extended view of them. Figure 6-29 demonstrates that the annotated Java class `BankAccount` is interlinked with the DBpedia dataset named `http://dbpedia.org/page/Java_class_file`. The link is created by using an `owl:sameAs` property to specify that the URI of the annotated element and that of the DBpedia dataset refer to the same resource (Figure 6-29). As a consequence, additional information about the Java class file can be obtained or queried from DBpedia website (`http://dbpedia.org`) (Figure 6-30).

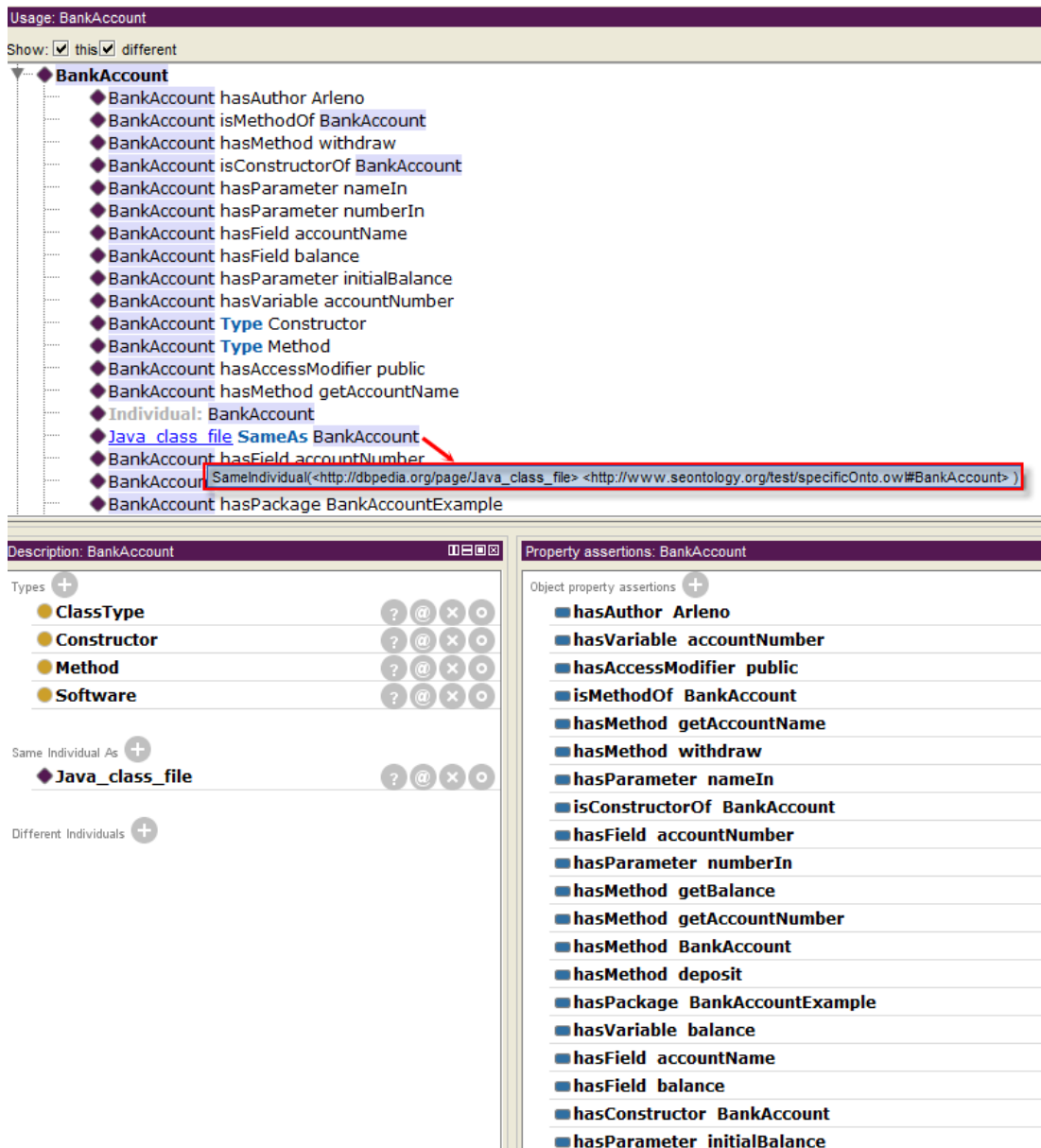


Figure 6-28: Populated instances and their relations presented in Protégé tool

```

<rdf:Description
rdf:about="http://www.seontology.org/test/specificOnto.owl#BankAccount">
.....
  <owl:sameAs rdf:resource="http://dbpedia.org/page/Java_class_file"/>
</rdf:Description>

```

Figure 6-29: BankAccount Java class being interlinked with a metadata term to its relevant entity in DBpedia dataset

DBpedia Browse using Formats Faceted Browser Sparql Endpoint

## About: Java class file

An Entity of Type : WikicatComputerFileFormats, from Named Graph : <http://dbpedia.org>, within Data Space : [dbpedia.org](http://dbpedia.org)

A Java class file is a file (with the .class filename extension) containing Java bytecode that can be executed on the Java Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (.java files) containing Java classes. If a source file has more than one class, each class is compiled into a separate class file. JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes Java platform-independent.

Property	Value
dbo:abstract	<ul style="list-style-type: none"> <li>A Java class file is a file (with the .class filename extension) containing Java bytecode that can be executed on the Java Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (.java files) containing Java classes. If a source file has more than one class, each class is compiled into a separate class file. JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes Java platform-independent. <sup>(en)</sup></li> </ul>
dbo:wikiPageExternalLink	<ul style="list-style-type: none"> <li><a href="http://java.sun.com/docs/books/vmspec/">http://java.sun.com/docs/books/vmspec/</a></li> <li><a href="https://docs.oracle.com/javase/specs/jvms/se6/html/VMSpecTOC.doc.html">https://docs.oracle.com/javase/specs/jvms/se6/html/VMSpecTOC.doc.html</a></li> <li><a href="http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html">http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html</a></li> </ul>
dbo:wikiPageID	<ul style="list-style-type: none"> <li>731735 (xsd:integer)</li> </ul>
dbo:wikiPageRevisionID	<ul style="list-style-type: none"> <li>697897741 (xsd:integer)</li> </ul>
det:subject	<ul style="list-style-type: none"> <li>dbc:Computer_file_formats</li> <li>dbc:Java_platform</li> </ul>
rdfs:type	<ul style="list-style-type: none"> <li>yago:Abstraction100002137</li> </ul>

Figure 6-30: Information about Java class file from DBpedia<sup>4</sup>

Figure 6-31 depicts the instances and relationships of class BankAccount. The graph is generated by the OntoGraf plug-in.

<sup>4</sup> [http://dbpedia.org/page/Java\\_class\\_file](http://dbpedia.org/page/Java_class_file)

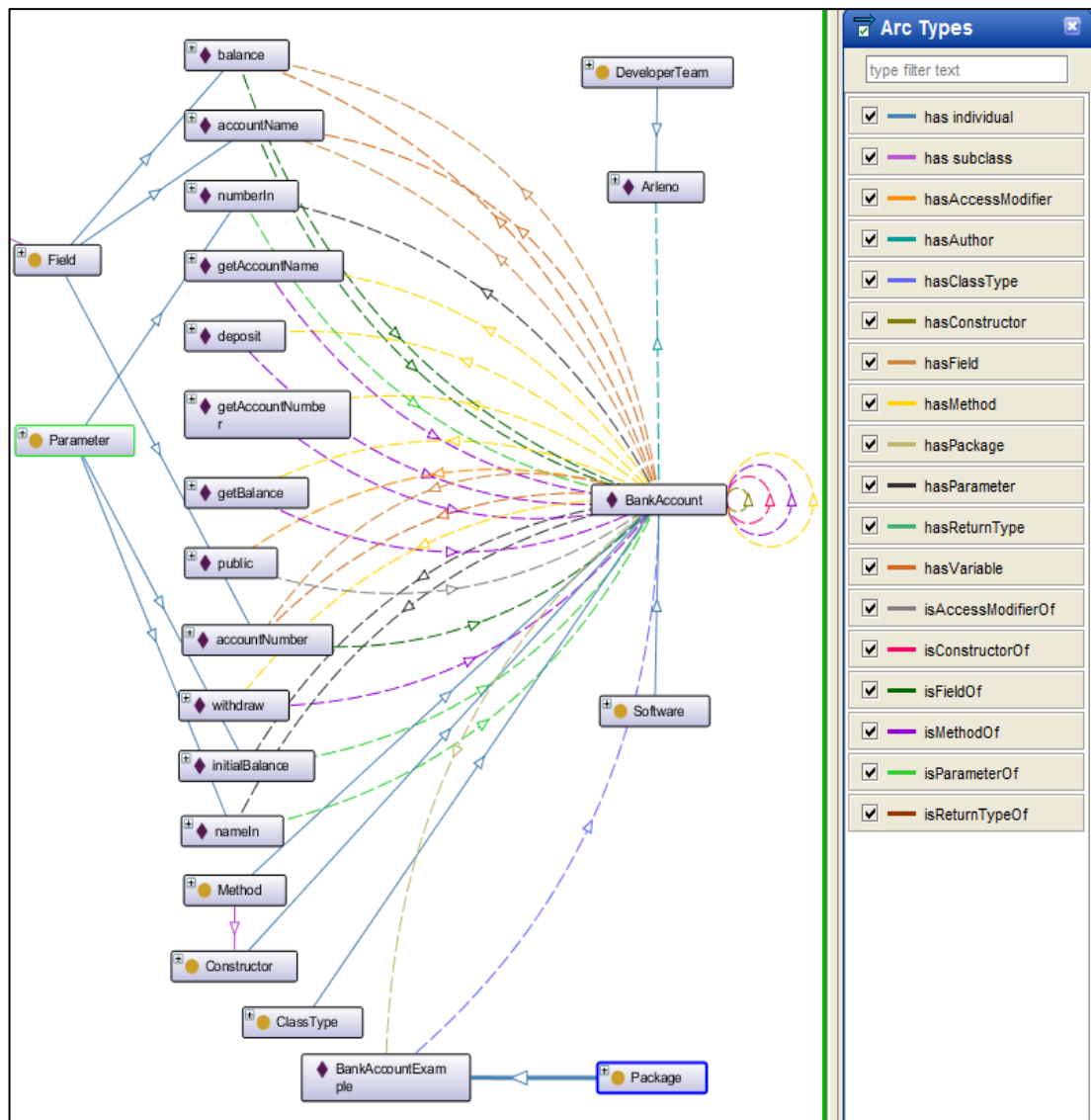


Figure 6-31: OntoGraf presentation of the BankAccount class instance.

In order to ensure that the new populated instances conform to the Software Engineering Ontology, they need to be verified by means of logical consistency checking. In this case, HerMiT 1.3.8, which is the built-in reasoner of Protégé 5.0.0, is chosen for this task. The result is shown in Figure 6-32. The system does not return any inconsistency warning, indicating that the new instances conform to the ontology. The time taken to reason the ontology is 2435ms.

```

C:\ Protege
Setting active ontology to OntologyID<OntologyIRI<<http://www.seontology.org/tes
t/specificOnto.owl>>>
Rebuilding entity indices...
... rebuilt in 1 ms
... active ontology changed
loading OntologyID<Anonymous-5> from file:/C:/NEE/PhDresearch/Jamshaid/NAVEED/OL
D-MASSsourceCode/MultiAgentSystem%2016-11-2015/memory_init/foafse.owl
Jul 15, 2016 1:10:12 PM org.semanticweb.owlapi.rdf.syntax.RDFParser$StartRDF sta
rtElement
INFO: Notice: root element does not have an xml:base. Relative IRIs will be reso
lved against file:/C:/NEE/PhDresearch/Jamshaid/NAVEED/OLD-MASSsourceCode/MultiAge
ntSystem%2016-11-2015/memory_init/foafse.owl
Rebuilding entity indices...
... rebuilt in 1 ms
Rebuilding entity indices...
... rebuilt in 1 ms
Initializing the reasoner by performing the following steps:
  class hierarchy
  object property hierarchy
  data property hierarchy
  class assertions
  object property assertions
  same individuals
Hermit 1.3.8 classified in 2435ms

```

Figure 6-32: Protégé reasoner’s logs for consistency checking of new populated instances

## 6.9 Practical uses

In this section, the practical uses of the SEOMAS approach to automate the semantic annotation of source code are demonstrated through a case study in a multi-site software development environment. The case study is derived from (Wongthongtham, Dillon and Chang 2011).

Suppose that a multi-national software company were running a multi-site software development project at four sites: Perth, Shanghai, Dublin, and Bangalore. After releasing V1.1 Build 20140205, a bug is found by Richard@Perth. Richard immediately files the bug in the project’s issue-tracking system. Given its great severity, the following day Richard files another urgent request in the issue-tracking system, hoping to increase its priority ranking so that it can draw greater attention from developers. The bug report is soon opened on the same day by Vishay@Bangalore. He comes up with a quick fix and adds a comment at the end of the report, giving the report the status of "re-evaluation pending". One week later, Arleno@Shanghai files a duplicate bug which is soon recognised as a repeated report

two days later. Arleno then realises that the solution (being a temporary measure) is not good enough. He discusses the issue with his team members and supervisor who add comments to the report and direct their concerns back to the Bangalore Lab. Based on Arleno's detailed information, Larry@Bangalore soon provides another bug fix solution. This fix is then picked up by Michael@Dublin, a technical lead who used to work with the component in the software package. Michael points out that Larry's fix might produce deadlocks in another related component and suggests reverting to the first fix. The next day, Larry soon fixes the bug based on Michael's instruction. Michael checks the fix, and marks the bug report status as "resolved" and closes the bug. The same day, Lisa@Shanghai states that the latest fix results in a connection timeout. Later on, Larry asks her to explain the affected component. At this point, Michael steps in, fixes the bug, and explains his fix. A few days later, Richard finally determines that the bug issue has been "resolved". The bug has led to a discussion in all three sites regarding the architecture of the component library.

From the scenario above, it is evident that even though the bug was not too complex and required a simple modification to fix the problem, it took a long time to resolve and close it. Difficulties arose because the software development-related information relevant to the bug was distributed across multiple software repositories with no links among them. This could result in duplicate bug reports. Moreover, because of the lack of integration of software artefacts that were similar or related, the information required to resolve the bug issue was not readily accessible. Therefore, even though the bug was simple to fix, developers had to spend more time on resolving the problem. Another challenge stemmed from there being no knowledge support enabling the identification of team members who were more likely to be able to resolve the bug issue. In this scenario, several attempts were made to fix the bugs by people with no expertise in dealing with this particular problem. Lastly, the lack of knowledge-sharing was one of the main issues that caused the delay. Larry did not know the impact of making a change to a certain component and there was no available document or information to which he could refer.

This scenario is used as a case study to demonstrate the application of the proposed approach. The case study is based on a vehicle registration system being developed by a multi-site team located across various sites. Software developers



communicate, coordinate, manage and share software development project information captured in the Software Engineering Ontology through the collaborative agents during the bug resolution process. A class diagram of a vehicle registration application in Figure 6-33 covers the main components of any normal Java-based project.

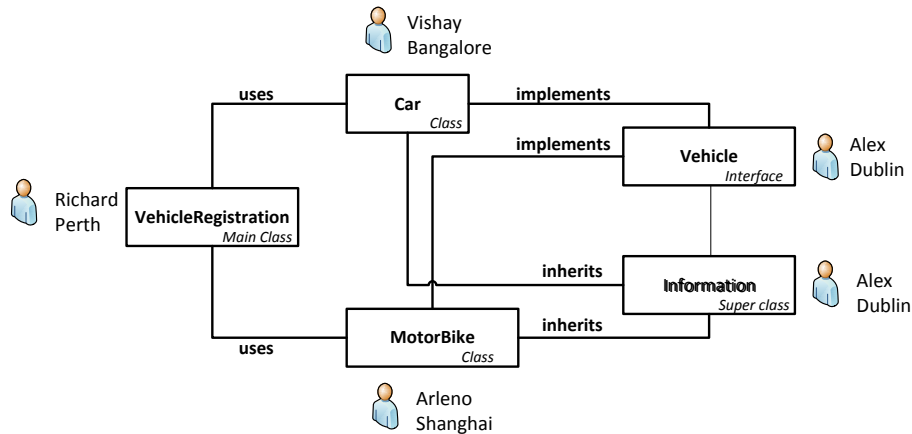


Figure 6-33: Vehicle registration class diagram in a multi-site development environment

The vehicle registration system is being developed in a multi-site development environment. Software developers are dispersed across four sites, namely, Perth, Bangalore, Dublin, and Shanghai. The Vehicle and Information classes provide generalised features of any type of vehicle. They are implemented and inherited by the Car and the MotorBike class. The VehicleRegistration is a main class that invokes either the Car or the MotorBike class based on a user selection.

All the classes above are annotated and populated in the Software Engineering Ontology repository by means of the SEOMAS approach. They are also semantically interlinked with other relevant software project information captured in the ontology. In other words, software project-related software information will not appear in isolation, but will be part of a large group of related information. Therefore, it is easily and readily accessible to software development teams. This information includes:

- Project description such as project name, description, list of

developers.

- Project development team information such as developer name, location, time zone, role, list of source code artefacts they are working on.
- Source code artefacts such as package, class, super class, interface, constructor, field, method, and developer.
- Source control commit log such as developers, source code, summary of a commit log (e.g., fix bug ID).
- Bug reports information such as bug ID, bug description, bug type, severity, reporter, and developer/fixer.
- Archived communication related to bug and issues such as discussion on the mailing list, forum, and developer.

The real power of the automated knowledge capture is realised when the knowledge is used to facilitate collaborative team members to communicate and coordinate effectively and efficiently as well as to enable knowledge sharing among them. Project teams can use this knowledge to assist their work or to address software development issues while working on software development projects. In order to demonstrate the practical use of the knowledge captured, the SEOMAS approach for the Software Engineering Ontology instantiations management which will be presented in the next chapter are utilised. In the following scenarios, the ontology agent and the recommender agent access the captured knowledge with the guidance of the Software Engineering Ontology to offer insight-providing information that can help to resolve the bug issues.

Figure 6-34 shows all agents in the SEOMAS platform. Each agent has a unique name and ID. Agents execute tasks and interact with each other by exchanging ACL messages. The SEOMAS platform consists of multiple containers which are implemented among distributed hosts. Each container represents each development site and it can host multiple agents. The Main-Container is where the JADE system agents, e.g., AMS, DF, RMA and the main SEOMAS agents, i.e., the

annotation agent, the versioncontrol agent, the ontology agent, and the recommender agent are situated. Each user agent lives in the container according to its software development site.

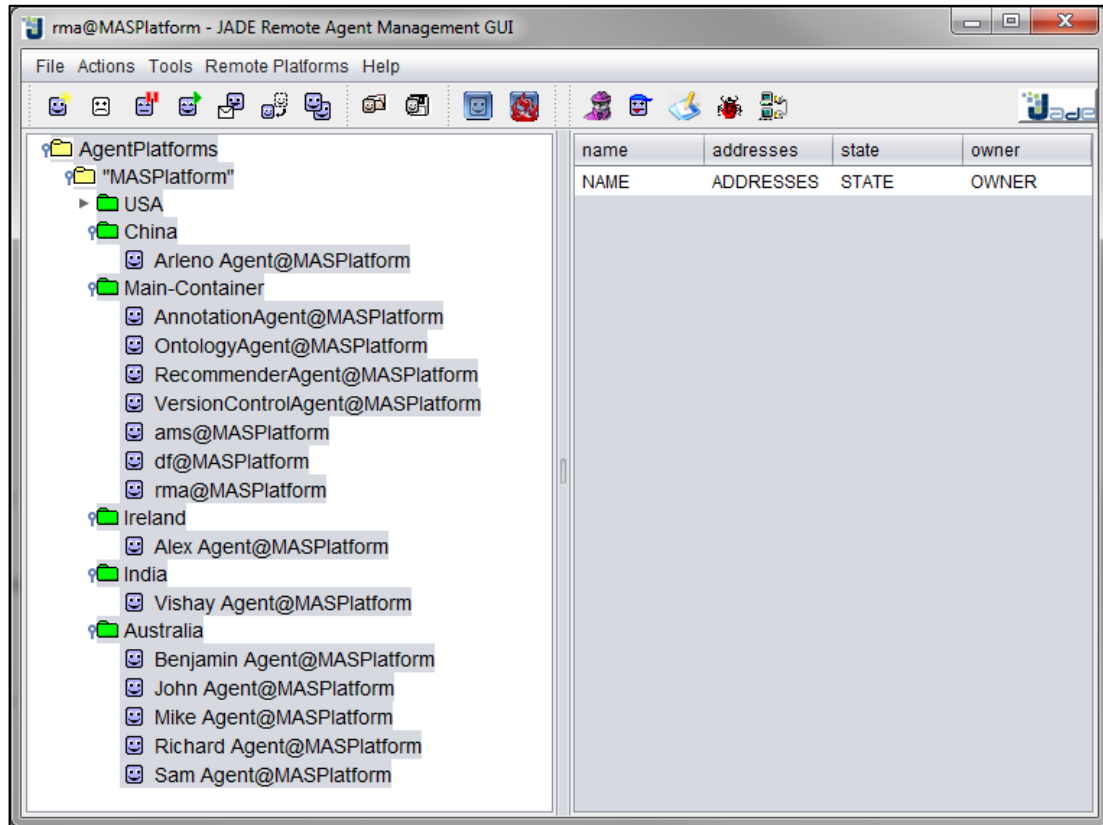


Figure 6-34: Agents in the SEOMAS platform

### Scenario 1- Provide information in regard to existing bug reports

During the system development process, a bug is found in a Car class by Alex. Before he enters a report into the issue-tracking system, he sends a query through his user agent to request information about any existing related problems. The ontology agent provides him with a list of previously reported bugs to a Car class as presented in Figure 6-35.

```
Output - MultiAgentSystem (run)
--- Message from OntologyAgent@MASPlatform ---
--- Message sent to Alex Agent@MASPlatform ---
Bug information that is related to the class Car

Bug:BugX010
BugID:BugX010
Bug Description:This bug is related to the wrong requirements of getMakeYear Method.
Bug Severity:Critical
Bug Status:Confirmed
Bug Fixing:BugFixerResourceForBugX010
Bug Type:IncorrectRequirements
Bug Fixed By:Arleno

Bug:BugX001
BugID:BugX001
Bug Description:This bug is related to the calculation of interest for car insurance
Bug Severity:Major
Bug Status:Open
Bug Fixing:BugFixerResourceForBugX001
Bug Type:LogicError
Bug Fixed By:Vishay
```

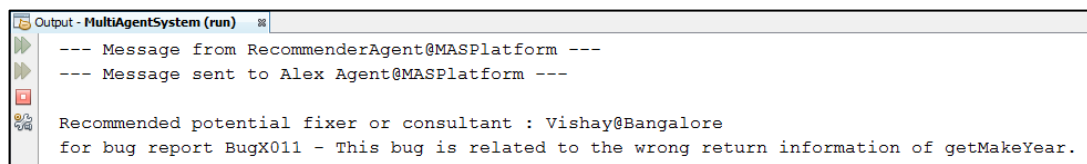
Figure 6-35: Existing bug reported to the Car class

Bug reports and other software artefacts such as source code or components are annotated and linked explicitly based on the Software Engineering Ontology concepts and its relation. Thus, individual issues can be raised along with other related issues. In this case, before filing a bug report, the ontology agent can help Alex to locate related problems in the issue-tracking system according to their associated class defined in the ontology and its instances. Alex can look at a list of associated problem reports and determine whether or not his issue is a duplicate one. In this way, duplicated bug reports could be avoided. This helps to dramatically reduce confusion and unnecessary information overload in the software project.

### Scenario 2 – Provide information with respect to potential fixer or expert

Once the bug is filed, the recommender agent identifies the person who is more likely to be able to resolve the bug problem and recommends this person to Alex. This is done by retrieving a full record of mappings of previously reported bugs to the 'Car' class and the names of developers who fixed those bugs. Then they are processed to find the person who most frequently resolved bugs reported to the Car class and who is therefore more likely to be an expert in this area. In Figure 6-36,

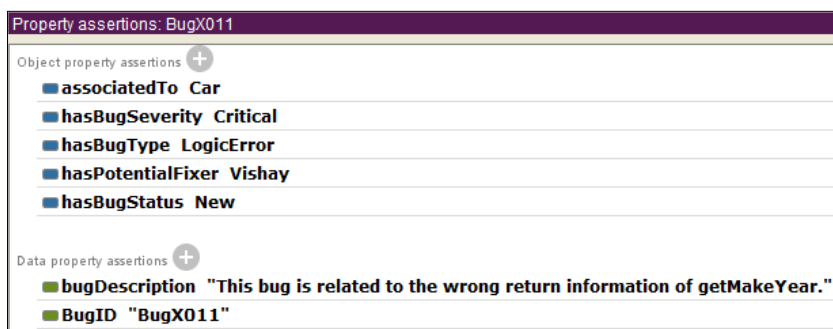
the recommender agent sends a message to Alex to recommend Vishay at the Bangalore site as an expert or a potential bug fixer for his report on the Car class.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Alex Agent@MASPlatform ---
Recommended potential fixer or consultant : Vishay@Bangalore
for bug report BugX011 - This bug is related to the wrong return information of getMakeYear.
```

Figure 6-36: Recommend an expert or a potential fixer for a new bug report

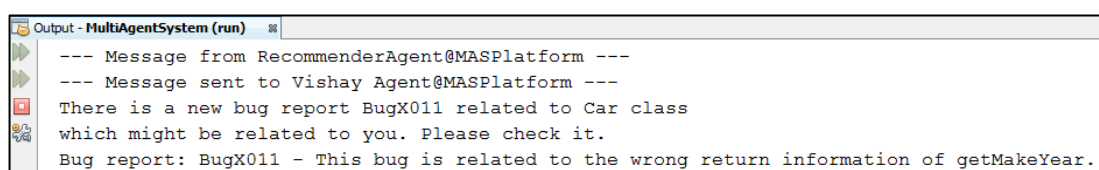
Additionally, the recommender agent attaches Vishay at the Bangalore site who is suggested as a potential fixer or consultant for the new bug report. Therefore, when other developers try to fix the bug (in case of a company's policy that allows only authorised people to change the code), they can directly ask Vishay for advice and help.



```
Property assertions: BugX011
Object property assertions +
  associatedTo Car
  hasBugSeverity Critical
  hasBugType LogicError
  hasPotentialFixer Vishay
  hasBugStatus New
Data property assertions +
  bugDescription "This bug is related to the wrong return information of getMakeYear."
  BugID "BugX011"
```

Figure 6-37: The potential fixer's name attached in the bug report

In order to attract the attention of the person who is more likely to be a potential fixer or an expert in this matter, the recommender agent also sends a message to notify Vishay at Bangalore site that there is a new bug report that might be related to his expertise, as presented in Figure 6-38. This makes him aware of the new report and he can take appropriate action such as inspecting the bug or giving suggestions to other developers.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Vishay Agent@MASPlatform ---
There is a new bug report BugX011 related to Car class
which might be related to you. Please check it.
Bug report: BugX011 - This bug is related to the wrong return information of getMakeYear.
```

Figure 6-38: A message to notify a potential bug fixer or an expert about a new bug

In the above scenarios, the agents can utilise the captured knowledge in the Software Engineering Ontology to identify an expert or a potential bug fixer for a new issue based on the previous bug report history. This information is also attached in the bug report for sharing among development team members. Thus, when the bug is picked up by another developer, he can ask the expert for advice. In this case, the responsibility for resolving the reported bug issue is shared between the bug fixer and a domain expert, which can help to increase the accuracy of a bug resolution and decrease the fixing time. In addition, the SEOMAS approach can improve the awareness of remote teams by providing a timely message sent to relevant project members. For example, a message is directly sent to notify the expert about a new bug report that might correspond with his expertise.

### **Scenario 3 – Provide information for bug diagnosing**

While a developer is diagnosing and fixing the bug, the ontology agent can provide information about the ‘Car’ class which includes author name and his information (e.g., site, time zone), version, class access modifier, fields, constructors, methods, and other properties such as its related software components (interfaces and/or super classes). Furthermore, the ontology agent can provide a history of all previous bugs reported to the ‘Car’ class and how they were fixed.

```
Output - MultiAgentSystem (run)
--- Message from OntologyAgent@MASPlatform ---
--- Message sent to Vishay Agent@MASPlatform ---
-- Class Information--
Class: Car
Inline Comment: This class is the class representing the registration process for Cars.
Author: Vishay
Version: 1.2
location: Bangalore, India
timezone: GMT+5.30
hasAccessModifier: public
SuperClass: Information
Developer: Vishay

Has Methods:
getInformation
getMakeYear

Implements:
Vehicle

--Bug Information--:
Bug: BugX001
BugTypeName: LogicError
BugTypeDesc: This bug is related to the calculation of interest for car insurance
FixedDate: 10Jan2014
FixedBy: Vishay
FixDescription: These are the comments of resource who fixed it.

Bug: BugX010
BugTypeName: IncorrectRequirements
BugTypeDesc: This bug is related to the wrong requirements of getMakeYear Method.
FixedDate: 20Jan2014
FixedBy: Arleno
FixDescription:

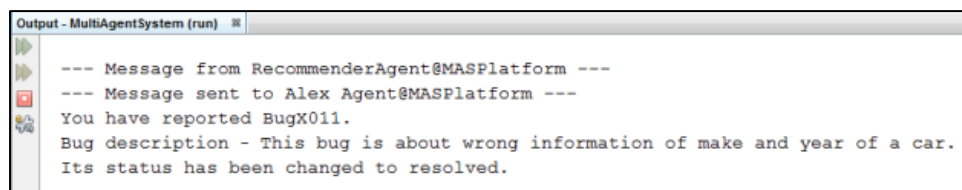
Bug: BugX011
BugTypeName: LogicError
BugTypeDesc: This bug is related to the wrong return information of getMakeYear.
FixedDate:
FixedBy:
FixDescription:
```

Figure 6-39: Related information about the Car class

Figure 6-39 shows the information about the Car class and its related bugs. This information is useful for diagnosing and fixing the bug. The information about dependencies among the class and other components is important for realising the impact of a change. It helps to prevent the unintended side effects which could lead to faults or future bugs. Moreover, the information about existing related bugs may assist a developer to identify the possible cause of the current bug as it may be a result of the fixes from previous bugs and/or the consequences of those fixes.

#### Scenario 4 – Monitor bug report status and notify the bug reporter when the bug issue is resolved

When the bug has been fixed and its status has been changed to “resolved”, the recommender agent sends a timely message to notify Alex, the bug reporter (Figure 6-40). Alex then can be aware of his report and verify the bug resolution. If he is satisfied with it, he can close the bug issue. In this case, the SEOMAS agents can help the developer become aware of the status of his report and alert him to investigate the resolution when it has been fixed. He does not need to keep track of the report by himself.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Alex Agent@MASPlatform ---
You have reported BugX011.
Bug description - This bug is about wrong information of make and year of a car.
Its status has been changed to resolved.
```

Figure 6-40: A message to the bug reporter giving a notice of the bug status

## 6.10 Discussion

Based on the use case scenarios given in the previous section, the benefit of the SEOMAS approach for automated knowledge capture of software project information is discussed as follows.

First, the SEOMAS approach requires very minimum effort from team members to transform software project information into a conceptually organised structure for common understanding. It is also transparently integrated into daily software development activities (e.g., integrated with the process for importing software artefacts into version control repositories). Hence, time-consuming, tedious, laborious, and prone-to-error tasks used to capture software development knowledge could be avoided. As a result, it could encourage project team members to share their knowledge, thereby improving the productivity of the software development process.

Second, once this software project information is captured and is well-organised, relevant concepts such as the semantic links between communication



threads (e.g., emails, mailing lists, discussion forums, documents, etc.), reports in the issue tracking system, and the software development artefacts, can be interlinked together. Therefore, relevant software project information will not appear isolated, but will be in a large group of related information for ready and easy access. The captured and interlinked knowledge can assist software teams to clarify any ambiguity during the communication and will facilitate team coordination. This knowledge is also in machine-readable and processable format. The software agents are able to understand and access this published knowledge with the guidance of the Software Engineering Ontology to provide useful and precise situational knowledge to project team members. For example, in the case study of a bug resolution issue, before filing a new bug report, the ontology agent can assist a developer to locate the related bugs in order to help to avoid duplicated report. It also can provide relevant information about the problem class and its previous issues to a bug fixer. The recommender agent can retrieve a full record of mappings of the previously reported bugs to the problem class and the information about the developer who fixed those bugs so that it can identify a potential fixer or a consultant and send a message to notify him/her to be aware of a new bug that might need his/her expertise to resolve. In this case, the coordination among team members can be improved because the agents can access and process the knowledge captured to proactively inform them of what is going on and when the coordination is needed.

Last, the SEOMAS approach uses the Software Engineering Ontology which is a comprehensive ontology covering all the aspect of software engineering to provide domain concepts during the annotation process. It also utilises other well-known ontologies and controlled vocabularies to enrich the semantic description of the annotated software project information wherever possible to enable reusability and to facilitate interoperability between different applications. Additionally, during the annotation process, the annotated source code elements are interlinked with the corresponding DBpedia concepts. The benefit of these links is that additional information about the annotated project information can be obtained from DBpedia which is a crowd-sourced community effort to extract structured information from Wikipedia website. Thus, the team members can extend their views of the captured knowledge by looking for additional information from DBpedia and its provided links.

## 6.11 Conclusion

In this chapter, the SEOMAS approach for automated knowledge capture of software project information is proposed. The agents utilise the Software Engineering Ontology to capture knowledge from software development artefacts during the daily software development activity. The captured knowledge is populated as new instances in the Software Engineering Ontology repository to allow project team members and software agents to access it. It has been demonstrated that the captured knowledge can be put to practical use to clarify any ambiguity in remote communication and to facilitate effective and efficient coordination and knowledge sharing within a software development project. In the next chapter, the design and development of the SEOMAS framework to manage the Software Engineering Ontology instantiations will be discussed.

## 6.12 References

- Aart, Chris van. 2007. Ontology Bean Generator. Accessed December 5, 2014, <http://protegewiki.stanford.edu/wiki/OntologyBeanGenerator>.
- Ashraf, Jamshaid, Omar Khadeer Hussain, and Farookh Khadeer Hussain. 2012. "A framework for measuring ontology usage on the Web." *The Computer Journal*. doi: 10.1093/comjnl/bxs134.
- Auer, Sören, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. "DBpedia: A nucleus for a Web of open data." In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 and ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, 722-735. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Bauer, Bernhard. 2002. "UML Class Diagrams Revisited in the Context of Agent-Based Systems." In *Agent-Oriented Software Engineering II: Second International Workshop, AOSE 2001 Montreal, Canada, May 29, 2001 Revised Papers and Invited Contributions*, eds Michael J. Wooldridge,

- Gerhard Weiß and Paolo Ciancarini, 101-118. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Beckett, Dave, and Brian McBride. 2004. "RDF/XML syntax specification (revised)." *W3C recommendation* 10.
- Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood. 2007. *Developing multi-agent systems with JADE*: John Wiley & Sons.
- Bizer, Christian, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. "DBpedia - A crystallization point for the Web of data." *Web Semantics: Science, Services and Agents on the World Wide Web* 7 (3): 154-165. doi: <http://dx.doi.org/10.1016/j.websem.2009.07.002>.
- Breslin, John G., Stefan Decker, Andreas Harth, and Uldis Bojars. 2006. "SIOC: an approach to connect web-based communities." *International Journal of Web Based Communities* 2 (2): 133-142. doi: 10.1504/ijwbc.2006.010305.
- Brickley, Dan, and Libby Miller. 2014. FOAF vocabulary specification 0.99. Accessed November 20, 2015, <http://xmlns.com/foaf/spec/>.
- Caire, Giovanni, and David Cabanillas. 2010. Jade tutorial application – Defined content languages and ontologies. Accessed 12 July 2016, <http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf>.
- FIPA. 2002. Foundation for intelligent physical agents: FIPA request interaction protocol specification. Accessed November 23, 2015, <http://www.fipa.org/specs/fipa00026/SC00026H.pdf>.
- Huget, Marc-Philippe. 2003. "Agent UML class diagrams revisited." In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, 49-60. Springer.
- Liao, Y., M. Lezoche, H. Panetto, and N. Boudjlida. 2011. "Semantic annotation model definition for systems interoperability." *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*: 61-70.

- Miles, Alistair, Brian Matthews, Michael Wilson, and Dan Brickley. 2005. "SKOS Core: Simple knowledge organisation for the Web." In *Proceedings of the International Conference on Dublin Core and Metadata Applications*, 3-10. <http://dcpapers.dublincore.org/pubs/article/view/798>.
- Nardi, Daniele, and Ronald J. Brachman. 2003. "An introduction to description logics." In *The description logic handbook*, eds Baader Franz, Calvanese Diego, L. McGuinness Deborah, Nardi Daniele and F. Patel-Schneider Peter, 1-40. Cambridge University Press.
- QDox. 2015. Accessed October 11, 2014, <http://qdox.codehaus.org>.
- Robillard, M., R. Walker, and T. Zimmermann. 2010. "Recommendation systems for software engineering." *Software, IEEE* 27 (4): 80-86. doi: 10.1109/ms.2009.161.
- Weibel, Stuart, John Kunze, Carl Lagoze, and Misha Wolf. IETF RFC 2413, 1998. *Dublin core metadata for resource discovery*. IETF RFC 2413.
- Wongthongtham, P., T. Dillon, and E. Chang. 2011. "State of the art of community-driven software engineering ontology evolution" *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, doi: 10.1109/DASC.2011.170.
- Zimmermann, Roland. 2006. *Agent-based Supply Network Event Management, Whitestein Series in Software Agent Technologies*. Switzerland: Birkhäuser Verlag.

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.

# **Chapter 7   Ontology-based Multi-agent Approach for Software Engineering Ontology Instantiations Management**

## **7.1   Introduction**

The previous chapter explored the design and development of the SEOMAS approach to automatically capture knowledge of software project information. The Software Engineering Ontology is used to provide domain knowledge to software development artefacts during the annotation process. Once the knowledge has been captured and populated in the ontology repository, it can be subsequently used by project team members or software agents to clarify any ambiguity or to address major software development issues.

This chapter focuses on the design and development of the SEOMAS approach to manage the instance knowledge after it has been captured into the ontology repository. It starts with the design of internal models of collaborative agents, namely, a user agent, the ontology agent, and the recommender agent. Then the prototype is used as a proof-of-concept experiment to demonstrate the practical uses of the framework which focuses on supporting requirement changes and software traceability tasks. Finally, the results from the prototype demonstration are discussed and compared with other related work.

## **7.2 Ontology-based Multi-agent Approach for Software Engineering Ontology Instantiations Management**

Once software engineering knowledge has been captured and populated in the Software Engineering Ontology, it is in a machine-accessible format that allows software agents to read and process it with the guidance of the ontology. In order to manage Software Engineering Ontology instantiations, three agent types are involved, namely, a user agent, the ontology agent, and the recommender agent. This approach enables the functionalities to support software development teams through multi-agent collaboration as follows.

- Knowledge query

The SEOMAS agents assist project teams to access software engineering knowledge shared in the Software Engineering Ontology and to query the semantic linked software project information.

- Instance knowledge manipulation

The manipulation of Software Engineering Ontology instance knowledge has three basic operations, namely, add, modify, and delete. For the add operation, the agent inserts new instantiations into the ontology knowledge base. It should be noted that this operation is similar to the ontology population process; however, it is more suitable for the insertion of a small number of instantiations. For the modify operation, the agent changes the value of a particular property of some instantiations in the ontology. For the delete operation, the agent removes some instantiations from the ontology knowledge base. It is to be noted that the agent does not remove only an instantiation explicitly requested by a user. It also removes its associated properties referred to in other instantiations in order to maintain the consistency of instantiations in the ontology repository. Furthermore, the agents provide useful recommendation regarding the impact of a change made to the instantiation and notify relevant team members to be aware of the change and its impact in a timely manner. These features are intended to improve the team's awareness with respect to software evolution to maintain consistency among software development artefacts.

- Instance knowledge monitoring

The monitoring of Software Engineering Ontology instance knowledge is intended to provide proactive monitoring of particular software project information and notification service in order to actively inform team members about relevant information or alert them to any event that is more likely to disrupt the project's pre-planned schedule or to affect the project's performance. The appropriate team member will be notified to be aware of such event before an actual issue arises.

In the next section, details of each agent type (i.e., a user agent, the ontology agent, and the recommender agent) involved in the management of Software Engineering Ontology instantiations are given within the micro-perspective of AOSE. These details describe resources, agent behaviours and related agent interactions with AUML models as proposed in Figure 6-1 of Chapter 6.

## 7.3 User Agent

As mentioned in Chapter 6, a user agent acts as a representative of each user and a mediator between a user and the system. Some of its design details have already been mentioned in Chapter 6 in section 6.3, but it is mainly focused on the knowledge capturing process. In this chapter, the focus will be on the management of Software Engineering Ontology instantiations; therefore, only those design details that are different from those given in Chapter 6 will be described.

### 7.3.1 Structure

An overview of the structural features of a user agent is illustrated with the AUML class diagram at implementation level in Figure 7-1. The details of the structural features that are different from those given in Chapter 6 are in bold italic. In order to assist its user to manage the Software Engineering Ontology instance knowledge, a user agent needs to collaborate with the ontology agent and the recommender agent. Perceptions of a user agent are based on a request from its user (i.e., *userRequest*) and from the ACL messages received from the ontology agent

(i.e., `retrievedResult`, `confirmation`, `failure`) and the recommender agent (i.e., `changeImpactRecommendation`, `changeImpactNotification`, `Notification`). A user agent employs an ACL message with a FIPA-request protocol to request the ontology agent to query knowledge or manipulate the ontology instantiations.

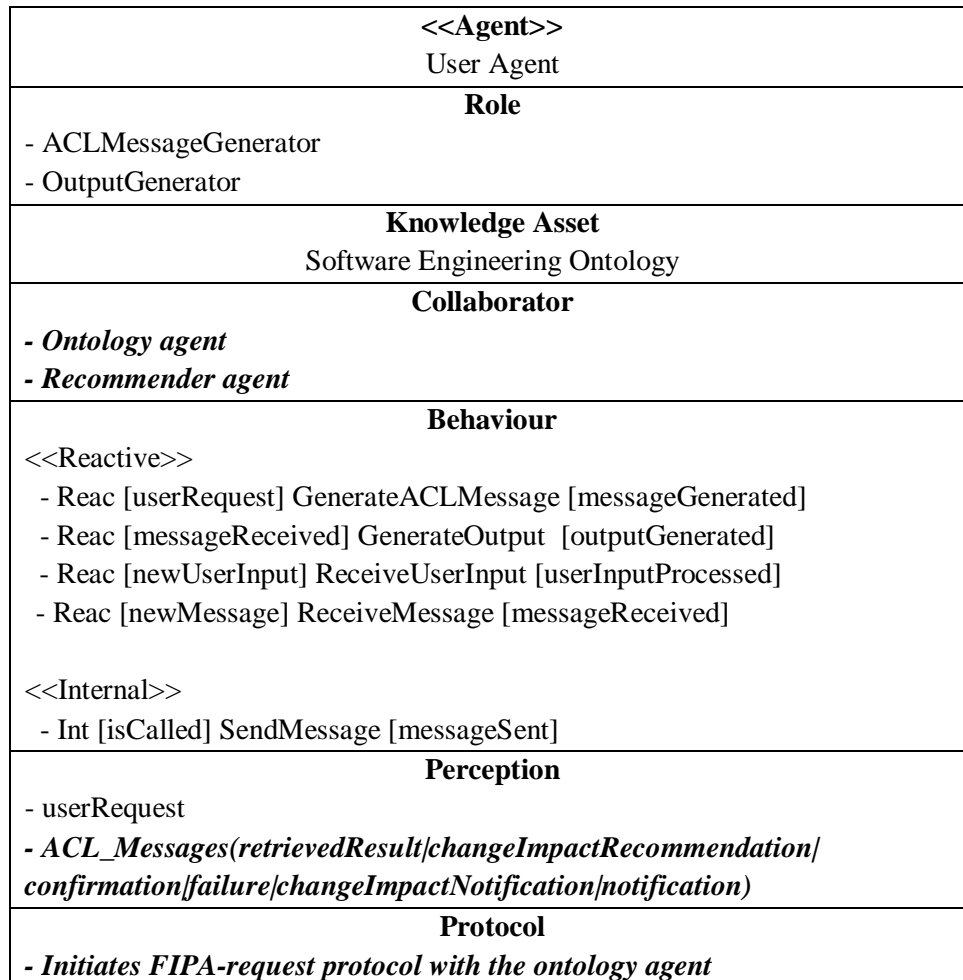


Figure 7-1: AUML class diagram at implementation level of a user agent

### 7.3.2 Behaviours

In this section, an overview of behaviours associated with the roles of a user agent and interdependencies between these behaviours is similar to the one presented in section 6.3.2. Therefore, only the details of a user agent's behaviours which are *GenerateACLMessage* and *GenerateOutput* are discussed.



### 7.3.2.1 ACL message generation behaviour

When a new request from a user is received, the *GenerateACLMessage* behaviour becomes active. It generates an ACL request message based on the request type (i.e., *queryRequest* or *manipulation* request). Then this message is sent to the ontology agent through the *SendMessage* behaviour (Figure 7-2).

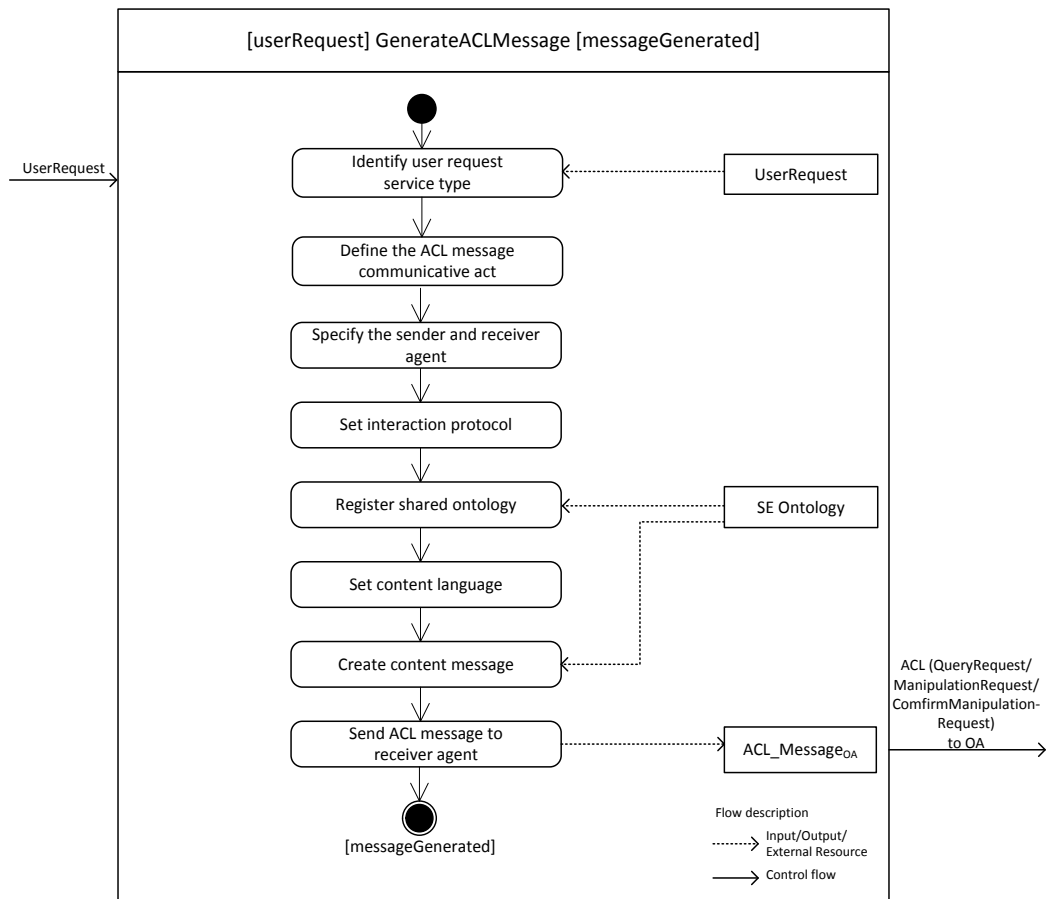


Figure 7-2: The GenerateACLMessage behaviour detail

### 7.3.2.2 Output Generation Behaviour

When a user agent detects a message received from other agent types that are involved in the Software Engineering Ontology instantiations management (i.e., the ontology agent or the recommender agent), the reactive behaviour *GenerateOutput* is activated (Figure 6-5). It first identifies the received ACL message content and the communicative act to analyse the type of the message (e.g. inform, confirm, failure). The message content is extracted and displayed to the user.

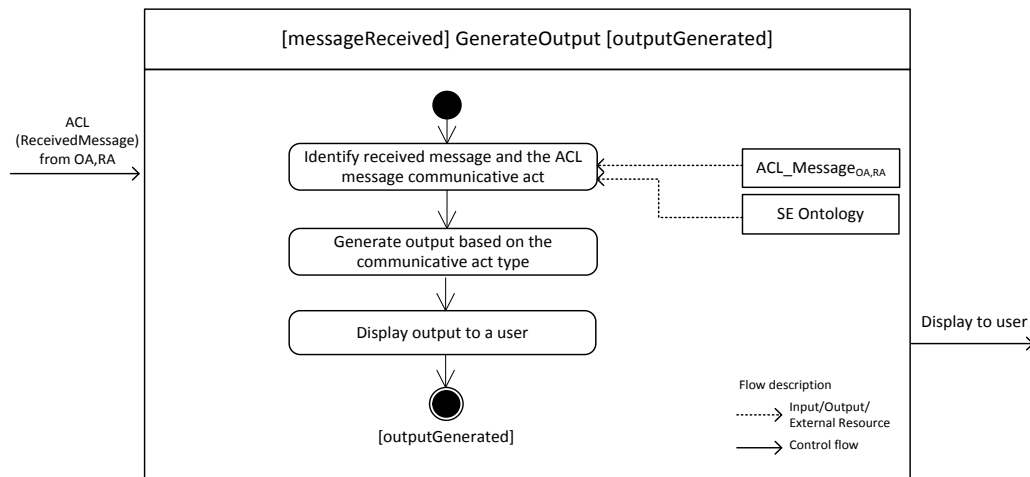


Figure 7-3: The GenerateACLMessage behaviour detail

### 7.3.3 Interactions

The AUML sequence diagram is used to describe the interactions between a user agent, the ontology agent, and the recommender agent according to its behaviours. As soon as a user agent receives a request from its user to query or manipulate the instance knowledge, the behaviour *GenerateACLMessage* is initiated. It generates the ACL message based on the request type. The request message is then sent to the ontology agent to process. The behaviour *GenerateOutput* receives and generates the output received from the ontology agent or the recommender agent as shown in Figure 7-4.

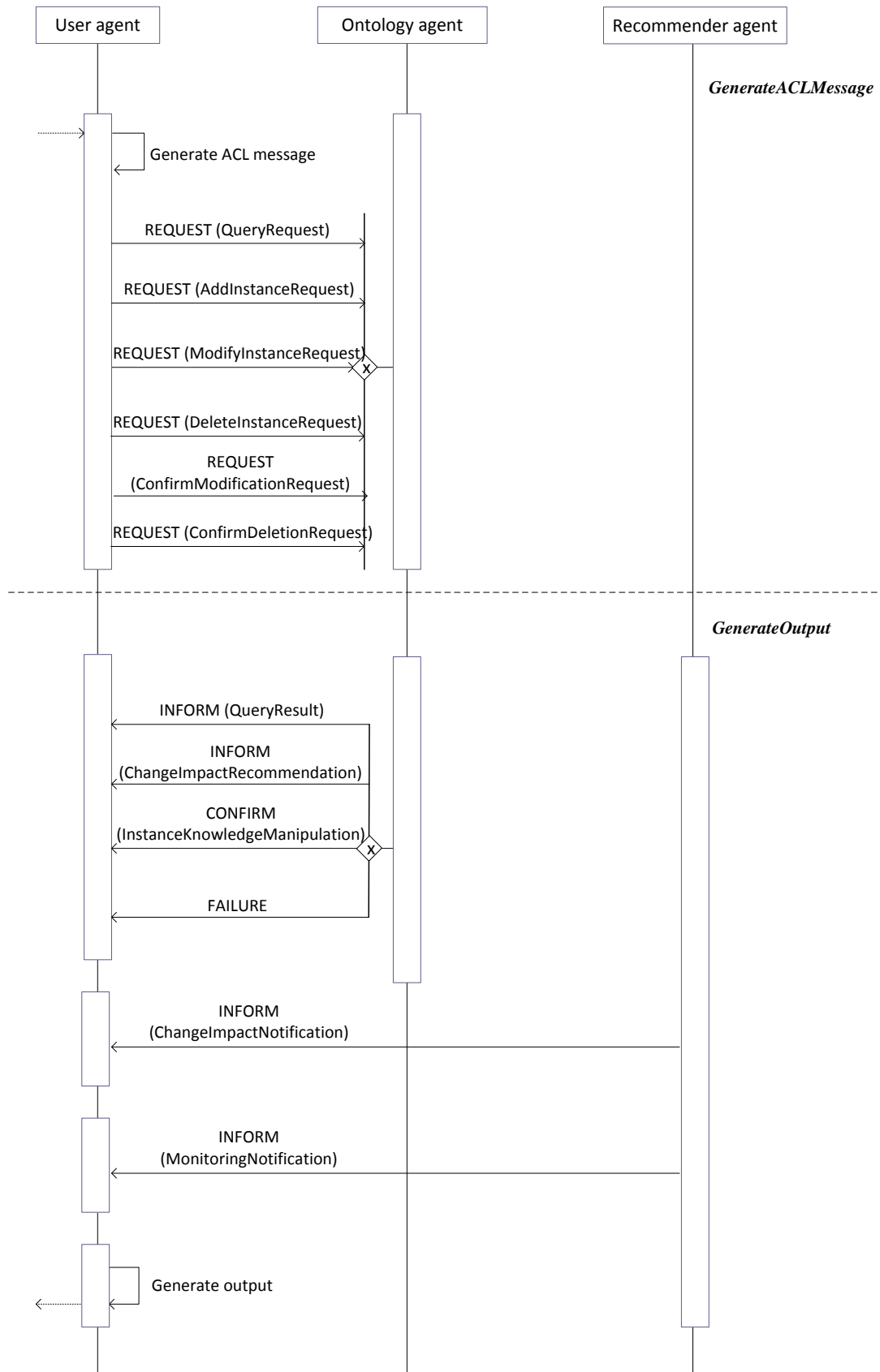


Figure 7-4: Agent interactions of a user agent

## 7.4 Ontology Agent

The ontology agent serves as an interface to manage the connection to the Software Engineering Ontology. It performs a crucial role in accessing and manipulating the Software Engineering Ontology instantiations. It also provides a proactive feature to monitor software project information captured as instance knowledge. These responsibilities are specifically reflected in the structure and behaviours of the ontology agent as follows.

### 7.4.1 Structure

The above responsibilities of the ontology agent are inherited by the role *InstanceKnowledgeManager*. An overview of the structural features of the ontology agent, including its resources, is illustrated with the AUML class diagram at implementation level in Figure 7-5. Its main resource is the Software Engineering Ontology. The Software Engineering Ontology domain knowledge is used to create agent messages and to translate the content for consistent communication, while its instance knowledge is used to retrieve semantically-linked software project information.

The ontology agents' roles include several main behaviours, namely, *QueryKnowledge*, *AddInstanceKnowledge*, *ModifyInstanceKnowledge*, and *DeleteInstanceKnowledge*. These behaviours are discussed in detail in the next section. The ontology agent depends on the perceptions of its sensors that provide information about the environment. Two agent message types from user agents are received by the agent, namely, query request (*queryRequest*), add instance knowledge request (*addInstanceRequest*), modify instance knowledge request (*modifyInstanceRequest*), delete instance knowledge request (*deleteInstanceRequest*), confirm modification request (*confirmModificationRequest*), and confirm deletion request (*confirmDeletionRequest*). The ontology agent's interaction that responds to a user agent and the recommender agent are performed by means of the agent communication language message with the FIPA-request protocol.

<b>&lt;&lt;Agent&gt;&gt;</b> Ontology Agent
<b>Role</b> - InstanceKnowledgeManager
<b>Knowledge Asset</b> Software Engineering Ontology
<b>Behaviour</b> <<Reactive>> - Reac [queryRequest] QueryKnowledge [requestProcessed] - Reac [addInstanceRequest] AddInstanceKnowledge [instanceKnowledgeManipulated] - Reac [modifyInstanceRequest] ModifyInstanceKnowledge [instanceKnowledgeNotManipulated instanceKnowledgeManipulated] - Reac [deleteInstanceRequest] DeleteInstanceKnowledge [instanceKnowledgeNotManipulated instanceKnowledgeManipulated] - Reac [newMessage] ReceiveMessage [messageReceived] <<Internal>> - Int [isCalled] SendMessage [messageSent]
<b>Perception</b> ACL_Messages(queryRequest addInstanceRequest modifyInstanceRequest deleteInstanceRequest confirmModificationRequest confirmDeletionRequest)
<b>Protocol</b> - Responds to the FIPA-request protocol with user agents for knowledge query and instance knowledge manipulation - Initiates the FIPA-request protocol with the recommender agent to request recommendation and notification
<b>Collaborator</b> - User agent - Recommender agent

Figure 7-5: AUML class diagram at implementation level of the ontology agent

## 7.4.2 Behaviours

### 7.4.2.1 Overview

A behaviour overview diagram of the ontology agent is presented in Figure 7-6. The ontology agent has a role, *InstanceKnowledgeManager* that has four reactive behaviours activated in response to user agents' requests, namely, *QueryKnowledge*, *AddInstanceKnowledge*, *ModifyInstanceKnowledge*, and *DeleteInstanceKnowledge*. The behaviour *QueryKnowledge* reacts to a perceived agent message with the perception type *queryRequest* as specified in the precondition of this behaviour. It queries the instance knowledge of the Software Engineering Ontology and provides the retrieved results to a user agent. The

behaviour *AddInstanceKnowledge* is responsible for inserting new instance knowledge into the Software Engineering Ontology according to a user request. The behaviour *ModifyInstanceKnowledge* and *DeleteInstanceKnowledge* modifies and deletes the requested instance knowledge respectively.

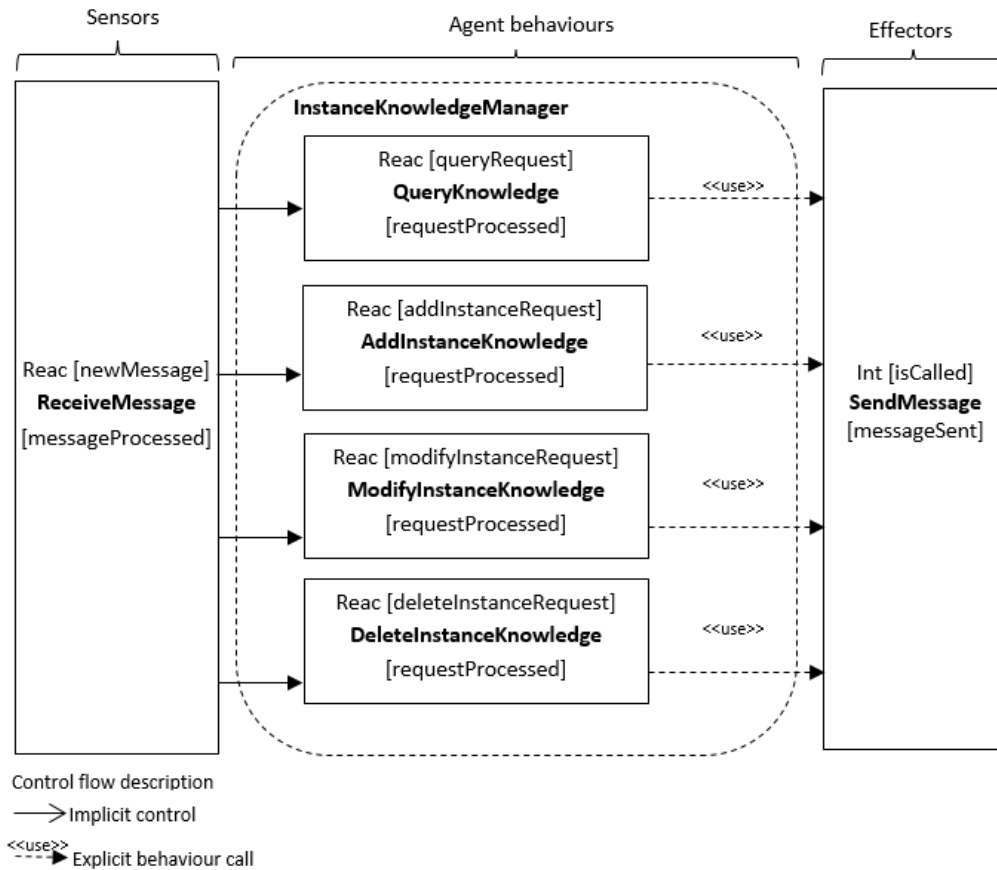


Figure 7-6: Behaviour overview diagram of the ontology agent

#### 7.4.2.2 Instance Knowledge Management Behaviours

The *InstanceKnowledgeManager* role deals with the management of the Software Engineering Ontology instantiations. Four main behaviours are associated with this role, namely, *QueryKnowledge*, *AddInstanceKnowledge*, *ModifyInstanceKnowledge*, and *DeleteInstanceKnowledge*.

### 7.4.2.2.1 QueryKnowledge behaviour

In the case where a new query request from a user agent is perceived by the behaviour *ReceiveMessage*, the behaviour *QueryKnowledge* becomes active (Figure 7-7). It identifies the query request by utilising the Software Engineering Ontology to define the context of the system and to facilitate the agent's communication. It retrieves the instance knowledge in the Software Engineering Ontology according to the request. The retrieved result is sent directly back to a user agent. If no result is retrieved, a failure message is sent to the user agent. It should be noted that the ontology agent exploits existing reasoning mechanisms of the OWL ontology to derive the knowledge that is most relevant to the query. These reasoning mechanisms include *TransitiveProperty*, *subClassOf*, *subPropertyOf*, *equivalentClass*, and *equivalentProperty*.

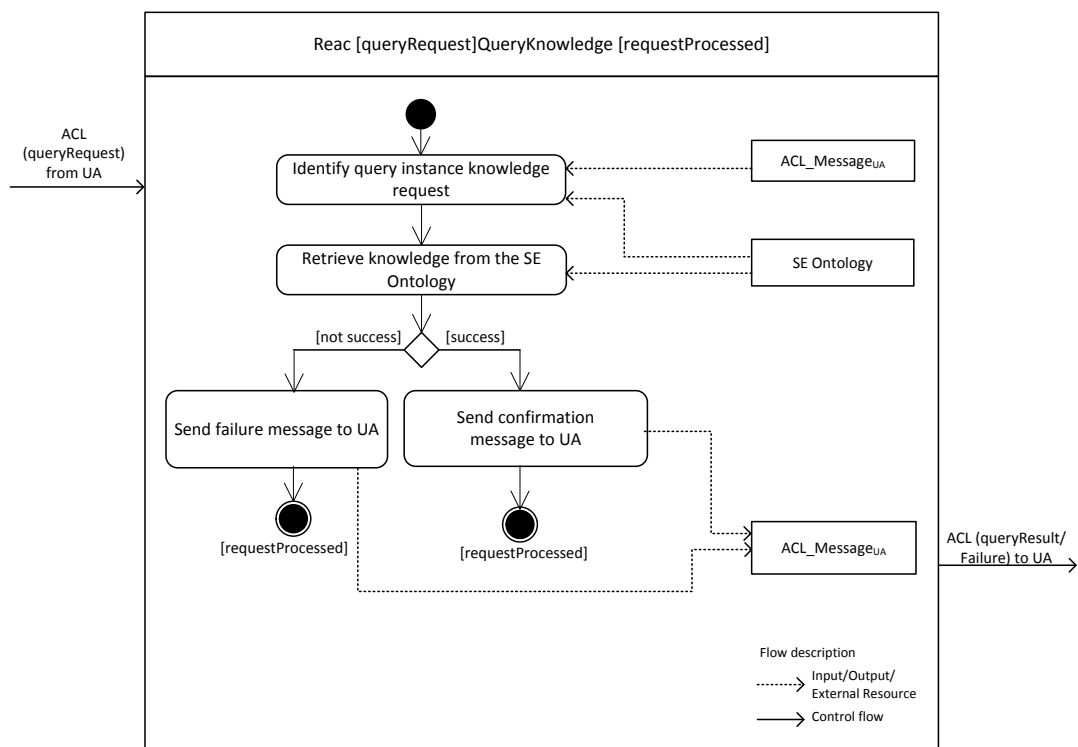


Figure 7-7: Activities of QueryKnowledge behaviour

### 7.4.2.2.2 AddInstanceKnowledge behaviour

In order to modify the instance knowledge in the Software Engineering Ontology knowledge base, the reactive behaviour *AddInstanceKnowledge* is activated after the agent perceives a request from the behaviour *ReceiveMessage*. It adds a new instantiation according to the request. If it is successful, a message to confirm the addition is generated. However, if it is not successful (e.g., not accepted by software engineering domain knowledge asserted in the ontology), a failure message is created. The message is then sent out by the behaviour *SendMessage* to a user agent who requests the addition of a new instantiation (Figure 7-8).

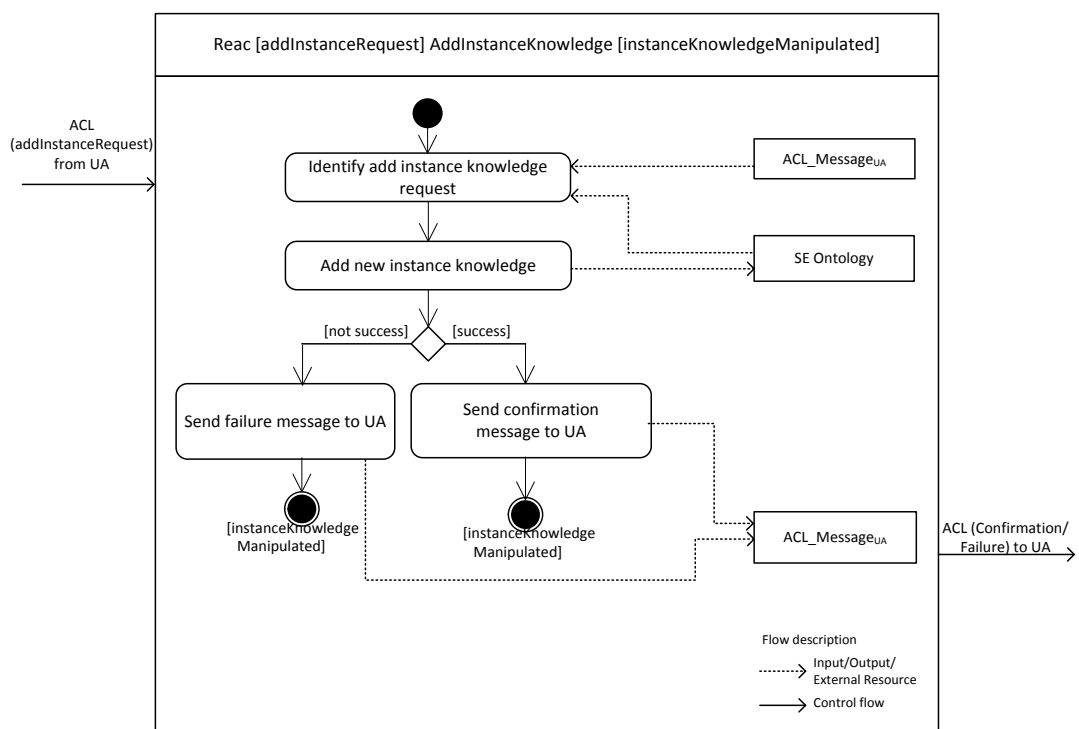


Figure 7-8: Activities of AddInstanceKnowledge behaviour

### 7.4.2.2.3 ModifyInstanceKnowledge behaviour

In order to modify the instance knowledge in the Software Engineering Ontology, the reactive behaviour *ModifyInstanceKnowledge* is activated after the ontology agent receives a manipulation request from the behaviour *ReceiveMessage* (Figure 7-9). The Software Engineering Ontology is also used to define the context of the system and to facilitate agents' communication. The behaviour



*ModifyInstanceKnowledge* identifies a request and then sends a message to the recommender agent to request the recommendation about the change impact analysis in order to make a user aware of any unintended side effect. Then it will wait for the confirmation of the modification from a user. If a user confirms the modification, it modifies the instantiation according to the request. It then sends a message back to a user agent to confirm the modification, and requests that the recommender agent notify relevant user agents who potentially may be affected by the modification made.

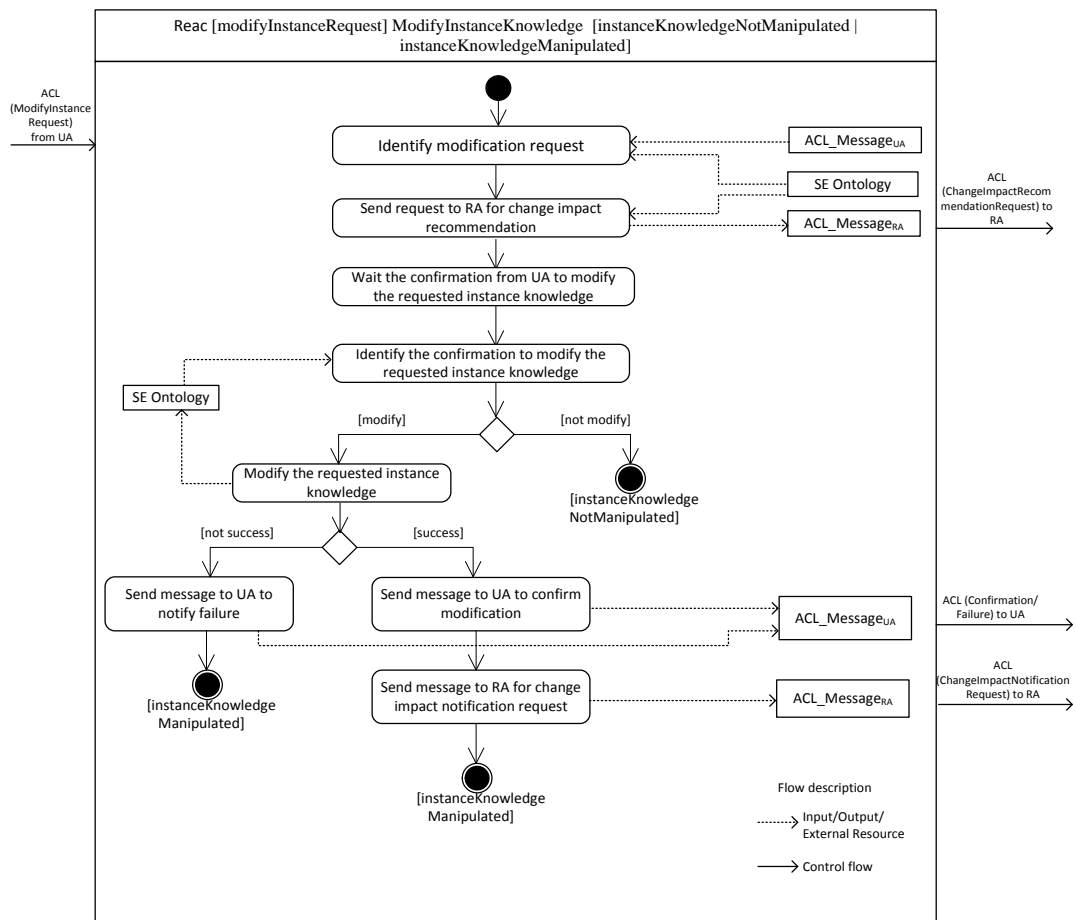


Figure 7-9: Activities of ModifyInstanceKnowledge behaviour

#### 7.4.2.2.4 DeleteInstanceKnowledge behaviour

The reactive behaviour *DeleteInstanceKnowledge* is triggered when the ontology agent receives a request to delete an instantiation from the behaviour *ReceiveMessage* (Figure 7-10). It identifies a request and then sends a message to the recommender agent to request the recommendation about the change impact analysis

from the deletion and wait for the confirmation from a user. If a user confirms that the instantiation can be deleted, the behaviour *DeleteInstanceKnowledge* removes it from the ontology knowledge base. Furthermore, this behaviour does not remove only the instantiation explicitly requested by a user. It also removes the properties associated with other instantiations in order to maintain the consistency of instantiations in the ontology repository. It then sends a message back to a user agent to confirm the deletion and asks the recommender agent to notify relevant users who own the artefacts that potentially may be affected by the deletion of the instantiation.

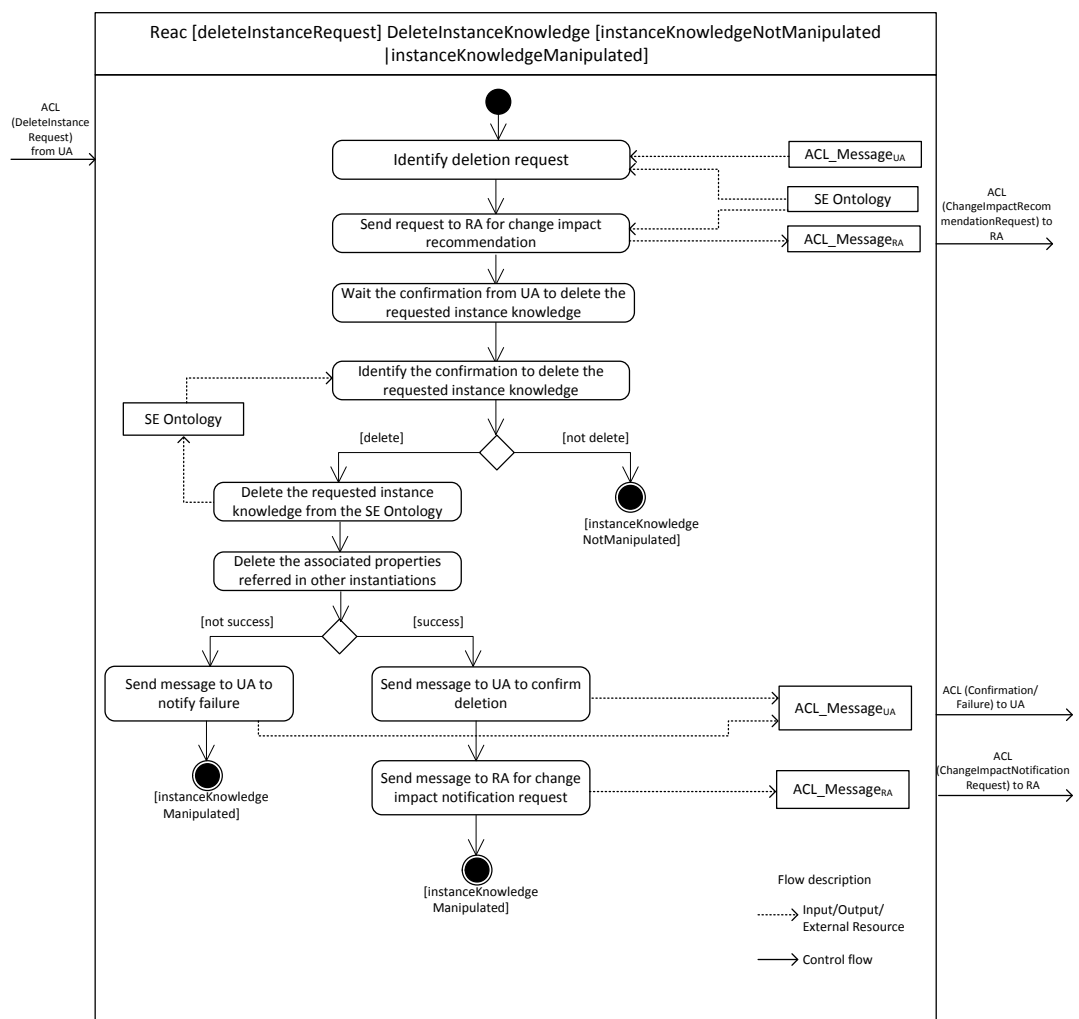


Figure 7-10: Activities of DeleteInstanceKnowledge behaviour

### 7.4.3 Interactions

The interactions between agent types that result from the behaviours of the ontology agent *QueryKnowledge*, *AddInstanceKnowledge*, *ModifyInstanceKnowledge*, and *DeleteInstanceKnowledge* are depicted in Figure 7-11.

In the case of the *QueryKnowledge* behaviour, a user agent sends a query request to the ontology agent. The ontology agent retrieves the results from the Software Engineering Ontology. If the retrieved results are not available, it responds to a user agent with a message to notify the failure. If the results are available, they are sent back to the user agent.

If the ontology agent perceives a request to add new instantiation, it responds to the request by adding a new requested instantiation to the Software Engineering Ontology repository. If the addition is success, a message is sent to confirm a user agent. If it is not successful, a message is sent to notify the user agent of a failure. In order to modify or delete instance knowledge, the behaviours *ModifyInstanceKnowledge* or *DeleteInstanceKnowledge* are activated. When, the ontology agent perceives a request from a user agent, it sends a request to the recommender agent for a recommendation regarding the impact of the change. If a user makes a decision to modify or delete such instance knowledge, a user agent sends a confirmation message to the ontology agent. When the instance knowledge is modified or deleted successfully, the ontology agent sends a message to the user agent to confirm the change made so that the user can validate and verify the logical consistency of the ontology instances. The ontology agent also sends a request to the recommender agent to send notifications to inform other user agents that potentially may be affected by the proposed change.

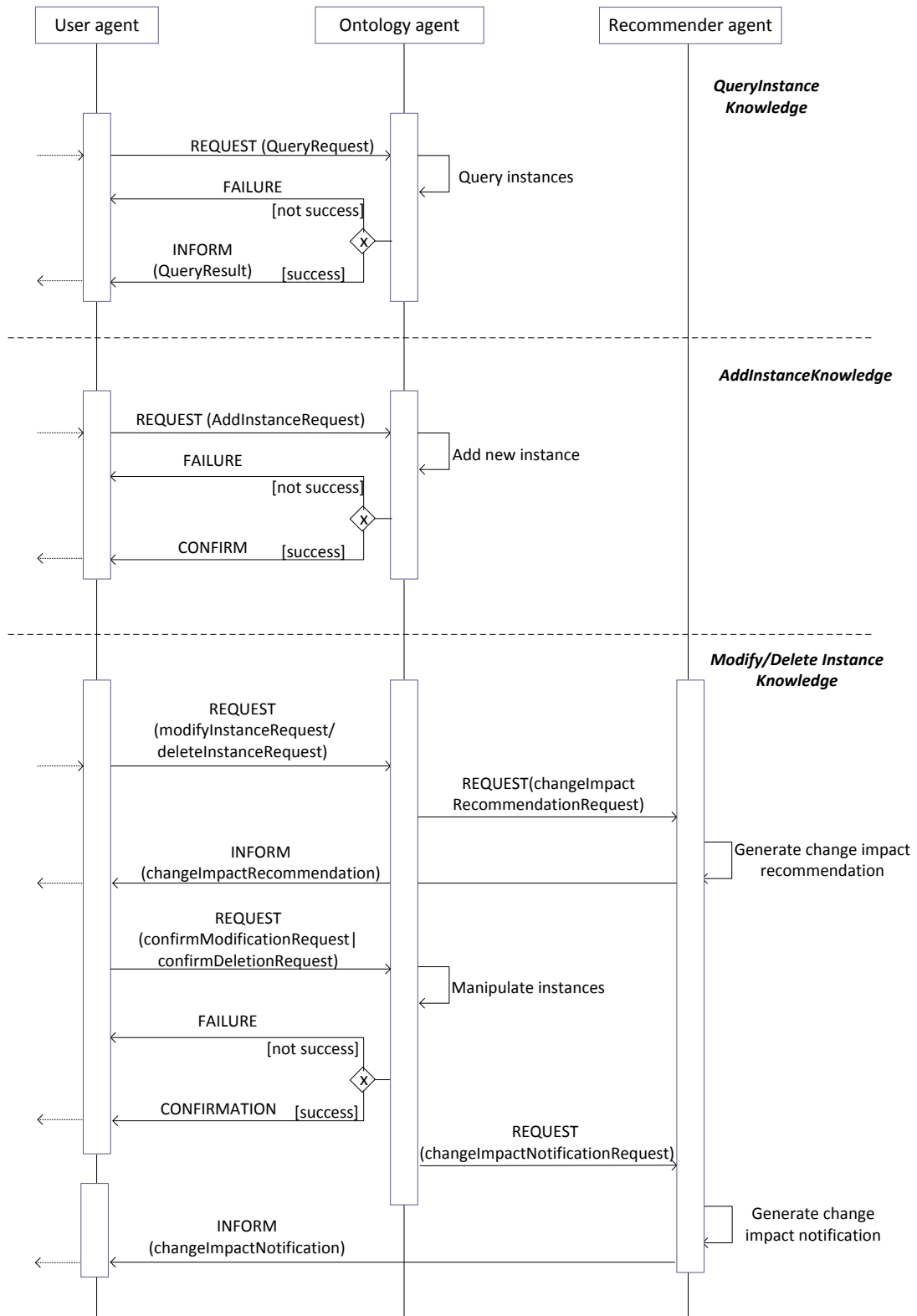


Figure 7-11: Interactions among agent types of the ontology agent

## 7.5 Recommender Agent

The recommender agent is responsible for accessing Software Engineering Ontology instance knowledge and processing it to generate useful recommendations and notifications (e.g, change impact analysis, potential bug fixer, potential deviation or disruptive event). These responsibilities are reflected in the structure and behaviours of the recommender agent as follows.

### 7.5.1 Structure

The responsibilities of the recommender agent are inherited by the role *Recommender*. An overview of the structural features of the recommender agent, including its resources, is illustrated with the AUML class diagram at implementation level in Figure 7-5. Its main resources comprise two main knowledge assets, namely, Software Engineering Ontology repository, and rules or formulas. The Software Engineering Ontology domain knowledge is used to create agent messages and to translate the content for consistent communication while its instance knowledge is used to retrieve semantically-linked software project information. Rules or formulas are applied to the results retrieved from the Software Engineering Ontology repository to generate a recommendation. It should be noted that the rules or formulas are pre-defined according to specific tasks. For example, when processing recommendations about the change impact analysis, the rules as described in section 7.6.1 in Table 7-2 are applied to the retrieved result.

The recommender agent collaborates with the ontology agent and user agents. It has three main behaviours, namely, *ManageMonitoring*, *GenerateChangeImpact-Recommendation*, and *GenerateChangeImpactNotification*. Details of these behaviours are explained in the next section. The recommender agent depends on the perceptions of its sensors that provide information about the environment which are the change impact analysis request (*changeImpactRecommendationRequest*) and the change impact notification request (*changeImpactNotificationRequest*) from the ontology agent. The recommender agent's interactions that respond to the ontology agent are performed by means of the agent communication language message with the FIPA-request protocol.

<b>&lt;&lt;Agent&gt;&gt;</b> Recommender Agent
<b>Role</b> - Recommender
<b>Knowledge Asset</b> Software Engineering Ontology, Rules or Formulas
<p style="text-align: center;"><b>Behaviour</b></p> <p>&lt;&lt;Proactive&gt;&gt; - Pro [cyclic instanceKnowledgeManipulated] ManageMonitoring [monitoringStatusInitiated]</p> <p>&lt;&lt;Reactive&gt;&gt; - Reac [newMessage] ReceiveMessage [messageReceived] - Reac [changeImpactRecommendationRequest] GenerateChangeImpactRecommendation [changeImpactRecommendationGenerated] - Reac [changeImpactNotificationRequest] GenerateChangeImpactNotification [changeImpactNotificationGenerated]</p> <p>&lt;&lt;Internal&gt;&gt; - Int [isCalled] SendMessage [messageSent]</p>
<p style="text-align: center;"><b>Perception</b></p> <p>ACL_Messages (changeImpactRecommendationRequest  changeImpactNotificationRequest)</p>
<p style="text-align: center;"><b>Protocol</b></p> <p>- Responds to the FIPA-request protocol with the ontology agent for generating recommendations</p>
<p style="text-align: center;"><b>Collaborator</b></p> <p>- Ontology agent - User agent</p>

Figure 7-12: AUML class diagram at implementation level of the recommender agent

## 7.5.2 Behaviours

### 7.5.2.1 Overview

A behaviour overview diagram of the recommender agent is presented in Figure 7-13. The recommender agent has only the *Recommender* role. As a *Recommender* role, a reactive behaviour *GenerateChangeImpactRecommendation* is activated in response to the ontology agent's request received by the *ReceiveMessage* behaviour. It reacts to the perceived message with the perception type *changeImpactRecommendationRequest* as specified in the pre-condition. It generates a recommendation regarding the impact of the requested change and then sends it to the user agent who requested the change to make him/her aware of any unintended

side effects. If the user agent confirms that a change can be made and the ontology agent has manipulated the instance knowledge, a reactive behaviour *GenerateChangeImpactNotification* is initiated to generate messages to propagate a change to relevant user agents who are potentially affected by the change made. Furthermore, the recommender agent provides a proactive behaviour *ManageMonitoring* that controls its decisions regarding particular software project information that needs to be monitored proactively. This behaviour is triggered cyclically or when the instance knowledge is manipulated. If the pre-defined condition is met, a message is sent to notify the relevant user agent.

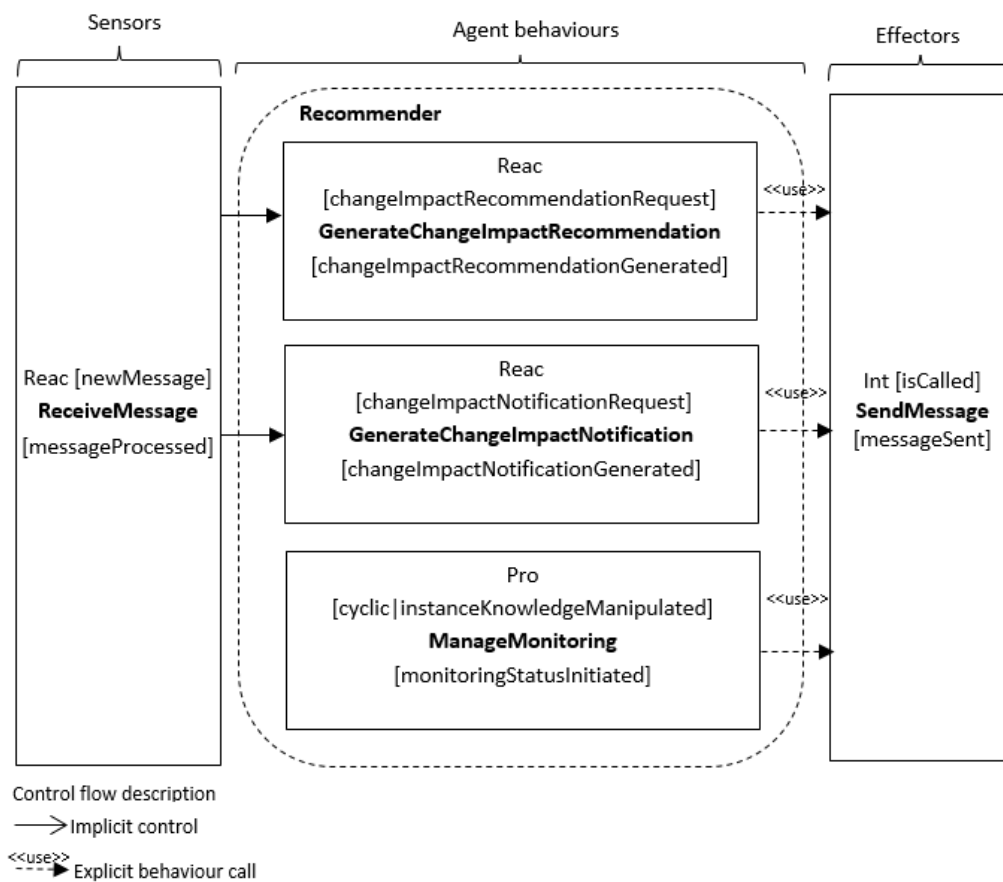


Figure 7-13: Behaviour overview diagram of the recommender agent

### 7.5.2.2 Recommendation Management Behaviours

The *Recommender* role is responsible for generating recommendations and notifications. Three main behaviours are associated with this role of the recommender agent, namely, *GenerateChangeImpactRecommendation*,

*GenerateChangeImpact-Notification, and ManageMonitoring.*

### 7.5.2.2.1 GenerateChangeImpactRecommendation behaviour

The *GenerateChangeImpactRecommendation* behaviour acts upon a request from the ontology agent. It identifies the software artefacts that may potentially be affected by the change request based on associated relations defined in the Software Engineering Ontology. Then, pre-defined rules/formulas are applied to them according to the type of those artefacts (e.g., requirement change impact rules in section 7.6.1 in Table 7-2) to generate a recommendation about the potential impact of a change. The change impact recommendation is sent to the user who requested a change in order to make him/her aware of the change impact so that s/he can decide whether or not to make the change (Figure 7-14).

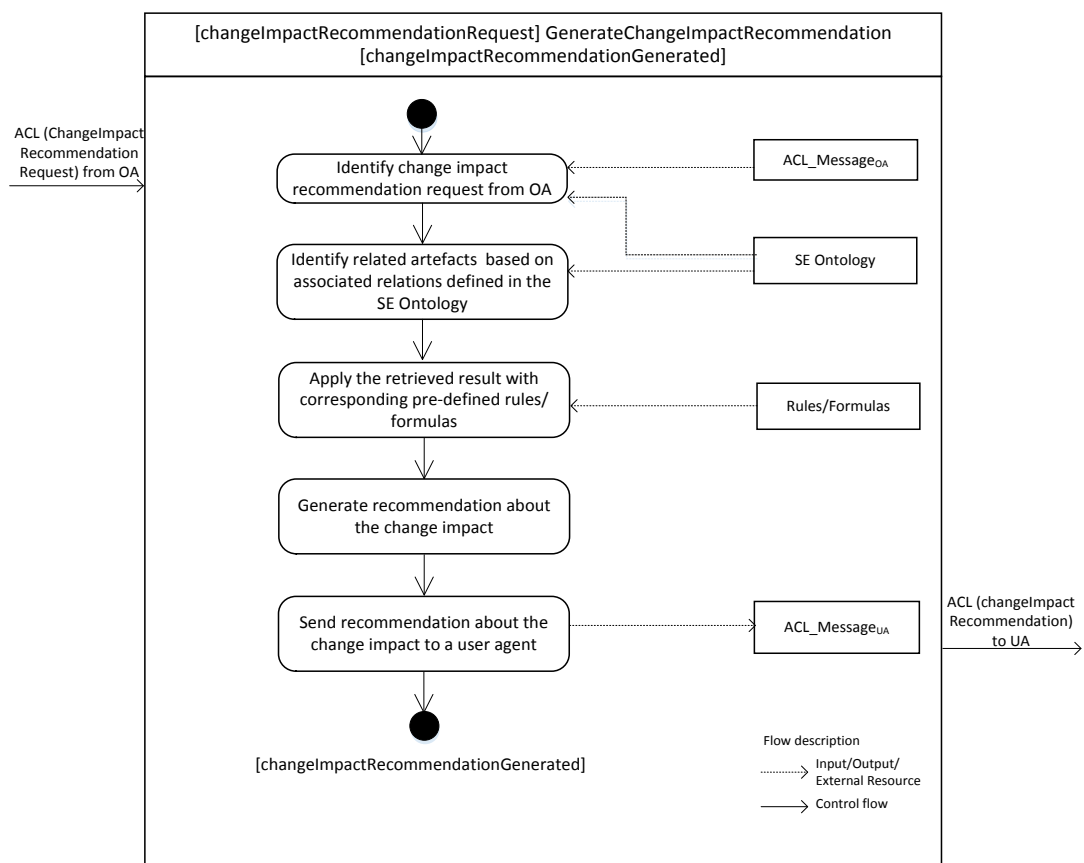


Figure 7-14: Activities of GenerateChangeImpactRecommendation behaviour



### 7.5.2.2.2 GenerateChangeImpactNotification behaviour

Once the instance knowledge has been manipulated according to a change request, the *GenerateChangeImpactNotification* is initiated. It identifies the owners of those artefacts that may be potentially affected based on associated relations defined in the Software Engineering Ontology. Then the notifications are forwarded to them so that they can be aware of the change and its impact (Figure 7-15).

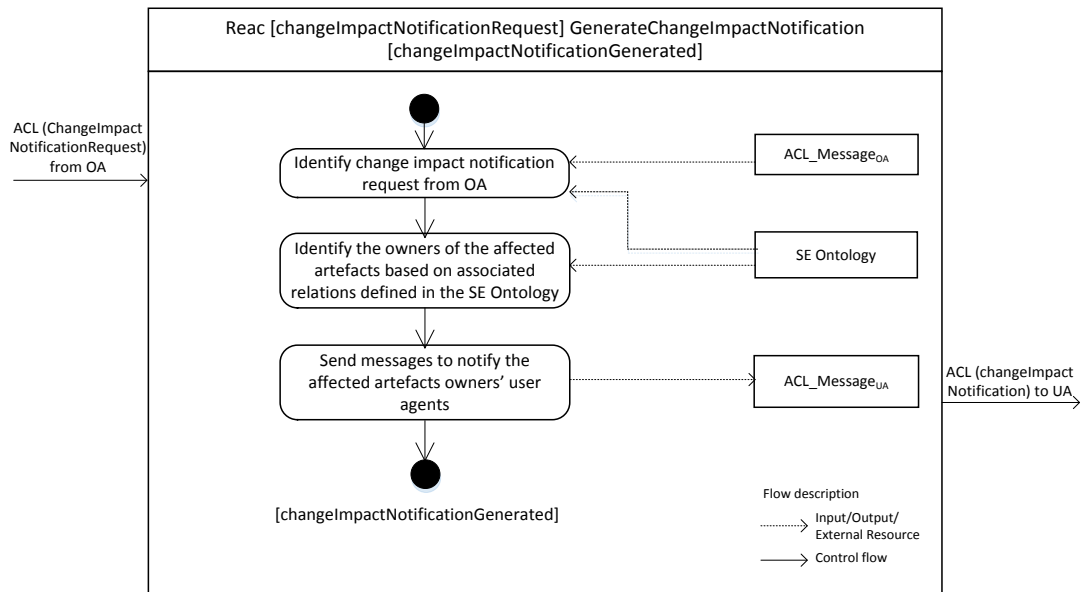


Figure 7-15: Activities of GenerateChangeImpactNotification behaviour

### 7.5.2.2.3 ManageMonitoring behaviour

In addition to the aforementioned reactive behaviours, the recommender agent offers a proactive behaviour, *ManageMonitoring*, which controls its decisions regarding the identification of events that are more likely to deviate or disrupt the project from the pre-planned schedule or to affect the project performance. It is initiated cyclically or when the recommender agent perceives that the instance knowledge has been manipulated (Figure 7-16). It monitors the specific instance knowledge and notifies the person in charge about the event or the deviation when one of the following conditions is met: a threshold is reached, a specific condition occurs, and every appointed period of time is met (e.g., every day, every week, and every month).

In case that a threshold is reached, particular instance knowledge is retrieved and counted for the summation. If the summation reaches a pre-defined threshold, a notification is generated and sent to the corresponding team member. For example, if the behaviour *ManageMonitoring* can identify that the number of bugs reported to a particular class is above a threshold (e.g. greater than 20 reports), a class author is asked to investigate the issue. In case that a specific condition is met, once the instance knowledge is manipulated and then its property matches the pre-defined condition, a notification is generated and sent to the corresponding team member. For instance, once the bug report status has been changed to ‘resolved’, a bug reporter is notified to validate the solution. Finally, at every appointed period of time (e.g. every day, every week, the end of the month, etc.), particular instance knowledge is checked or processed and if the pre-defined condition is met, a notification is generated and sent to the corresponding team member. For example, a message is sent to notify the project manager about the issues which have remained unassigned to the fixers for a week.

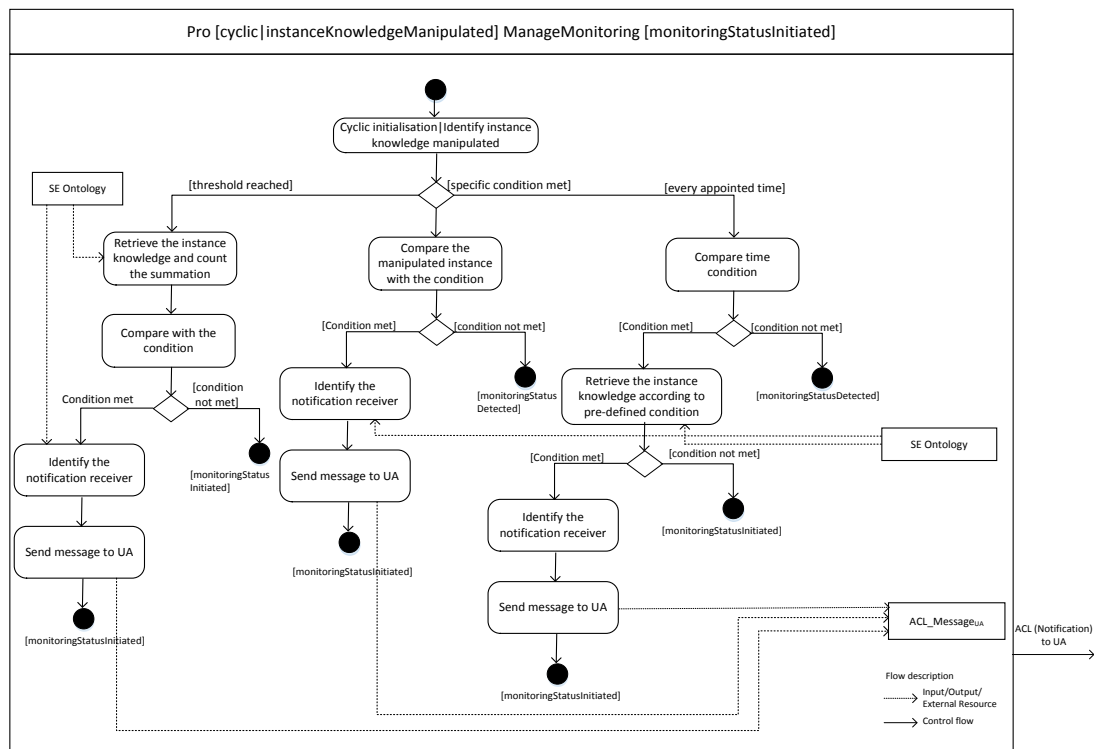


Figure 7-16: Activities of ManageMonitoring behaviour

### 7.5.3 Interactions

The AUML sequence diagram is used to demonstrate the interactions between agent types for the recommender agent. Figure 7-17 illustrates the main interactions between the recommender agent, the ontology agent and user agents which result from the behaviours *GenerateChangeImpactRecommendation*, *GenerateChangeImpact-Notification*, and *ManageMonitoring*.

In the case of the *GenerateChangeImpactRecommendation* behaviour, the recommender agent receives a message from the ontology agent to request recommendations about the impact of a change (e.g., modify or delete) made to particular instance knowledge. The recommendations are sent back to the user agent who requested the change. When the instance knowledge is modified or deleted, the recommender agent sends notifications about the change and its impacts to relevant user agents. This is done by the behaviour *GenerateChangeImpactNotification*.

Concerning the behaviour *ManageMonitoring*, when the recommender agent monitors some particular instance knowledge and identifies any potential deviation or any event that meets the condition set, the notification is generated and sent to the relevant user agents.

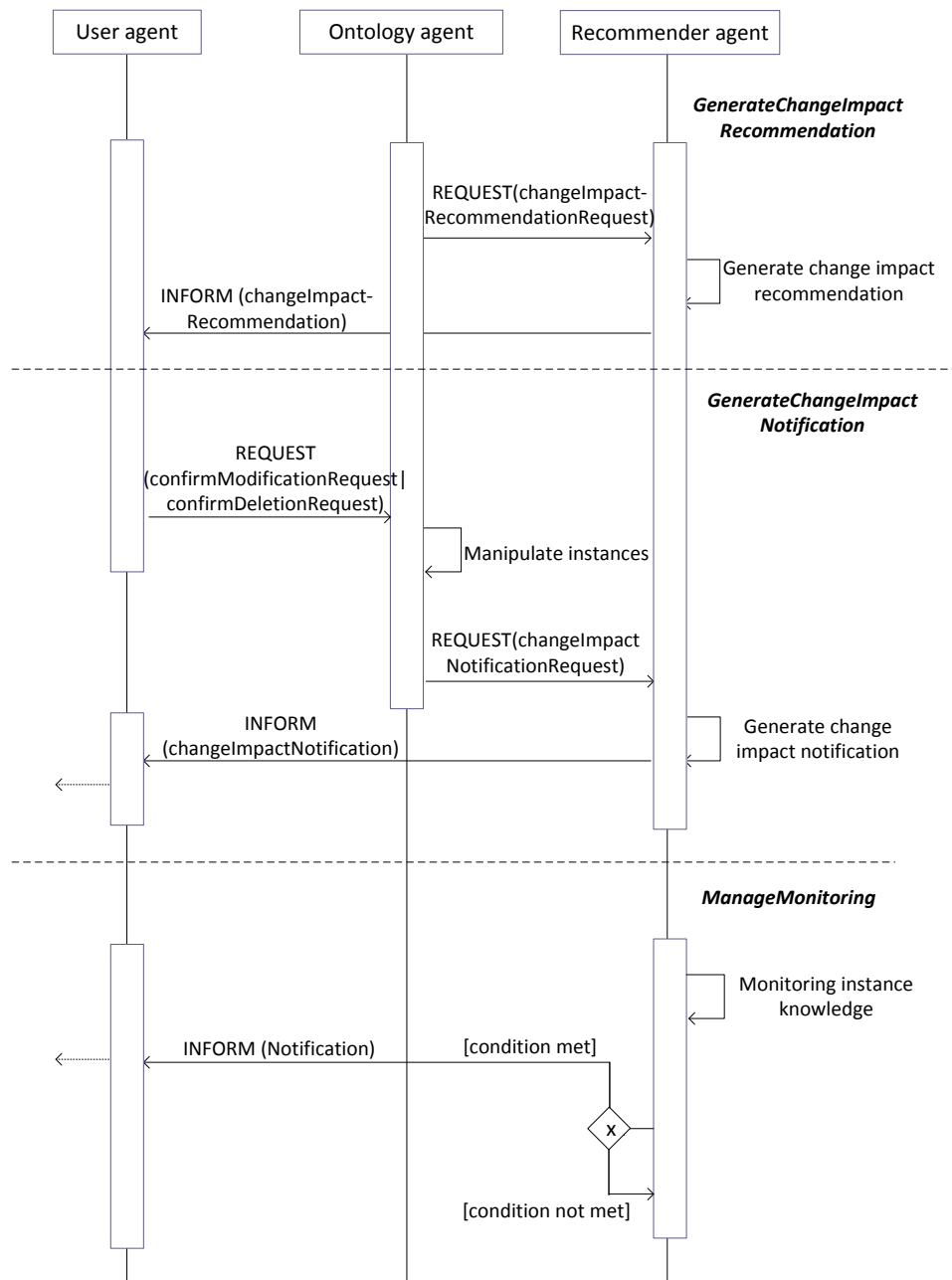


Figure 7-17: Interactions among agent types of the recommender agent

## **7.6 Practical Uses of the SEOMAS approach to Support Requirement Traceability**

This section discusses how to apply the SEOMAS approach for the management of Software Engineering Ontology instantiations to support automated requirements traceability tasks. Requirements traceability is one of the essential activities of requirement management. It refers to “the ability to describe and follow the life of a requirement in both forward and backward direction” (Gotel and Finkelstein 1994). The manual performance of this task takes a great deal of effort and is time-consuming, laborious, and prone to error. Requirements traceability is challenging in centralised software development and even more so in a multi-site environment where software teams are located across several sites. An effective and proactive approach is needed to enable software teams to manage and be aware of changes in requirements. The SEOMAS agents work cooperatively to trace and identify potentially affected software artefacts and notify the relevant team members about any change to requirements in order to provide them with timely awareness. In the following sub-sections, details of the interdependency of requirements and the rules regarding the impact of changes to requirements are presented to illustrate how requirements are related to and affect each other. A case study of online shopping system development is used to demonstrate how the SEOMAS approach can assist software development teams to manage requirement change, particularly in terms of requirement traceability.

### **7.6.1 Requirements Interdependencies Modelling**

Requirements interdependencies refers to the way that requirements relate to and affect each other. A number of researches such as (Pohl 1996; Dahlstedt and Persson 2005) have proposed various classifications of interdependency. In (Dahlstedt and Persson 2003), the authors have compiled different views of requirements’ interdependencies and have developed a neutral classification of fundamental interdependencies grouped into two main categories which are structural and cost/value interdependencies as shown in Table 7-1.

Table 7-1: Dahlstedt's Interdependency model

Categories	Description	Type
<b>Structural</b>	Concerns the structures of requirements	Requires, Explains Similar_to, Influences Conflicts_with
<b>Cost/Value</b>	Concerns the cost and value involved in implementing requirements	Increase/Decrease_cost_of Increase/Decrease_value_of

The above requirement interdependencies are used to define the semantic relations among requirements captured in the Software Engineering Ontology. This research focuses mainly on identifying the impact of changes made to requirements by recommending direct and indirect software artefacts which include other affected requirements, use cases, classes, and test cases. Changes to requirements may include additions, deletions and modifications. It is noted that the requirement update in this research mainly concerns any change to the description of a requirement. The focus here is particularly on the structural interdependency types which are *Requires*, *Explains*, and *Conflicts\_with*. In this work, they are used to represent the relationships of requirements. However, to facilitate better understanding in our context, the word *Refines* is used instead of *Explains*.

A *Requires* relation means that the fulfilment of one requirement depends on the fulfilment of another requirement. For example, for Requirement R<sub>1</sub>- the system allows the user to place an order and for Requirement R<sub>2</sub>- the user has to log in successfully before placing an order. This demonstrates that R<sub>1</sub> requires R<sub>2</sub>.

*Refines* means that a requirement is derived from another requirement and adds more specific details. For instance, for Requirement R<sub>1</sub>- the system can manage the payment via credit card and PayPal and for Requirement R<sub>2</sub> – the system allows the user to pay online. The interdependency is that R<sub>1</sub> refines R<sub>2</sub>.

*Conflicts\_with* describes the contradictory relationship among requirements. A requirement conflicts with another requirement if they cannot exist simultaneously. For instance, for Requirement R<sub>1</sub>- only the system administrator can manage user passwords and for Requirement R<sub>2</sub> – the users can change their passwords online. This demonstrates that R<sub>1</sub> conflicts with R<sub>2</sub>.

In (Göknil, Kurtev and van den Berg 2008), the authors define the change impact rules for requirement changes. These rules are adopted and modified for this work as presented in Table 7-2. For example, if requirement  $R_1$  has interdependency ‘requires’ with requirement  $R_2$ , if  $R_2$  is deleted then  $R_1$  is considered as an actual affected requirement.

Table 7-2: Change impact rules adapted from(Göknil, Kurtev and van den Berg 2008)

<b>Interdependency Type</b>	<b>R1 Requires R2</b>	<b>R1 Refines R2</b>	<b>R1 Conflicts with R2</b>
R <sub>1</sub> is modified	R <sub>2</sub> is not affected	R <sub>2</sub> is not affected	R <sub>2</sub> is not affected
R <sub>2</sub> is modified	R <sub>1</sub> is candidate affected	R <sub>1</sub> is candidate affected	R <sub>1</sub> is not affected
R <sub>1</sub> is deleted	R <sub>2</sub> is not affected	R <sub>2</sub> is not affected	R <sub>2</sub> is not affected
R <sub>2</sub> is deleted	R <sub>1</sub> is affected	R <sub>1</sub> is candidate affected	R <sub>1</sub> is not affected
New R is added to R <sub>1</sub>	R <sub>1</sub> is affected R <sub>2</sub> is not affected	R <sub>1</sub> is affected R <sub>2</sub> is not affected	R <sub>1</sub> is affected R <sub>2</sub> is not affected
New R is added to R <sub>2</sub>	R <sub>1</sub> is candidate affected R <sub>2</sub> is affected	R <sub>1</sub> is candidate affected R <sub>2</sub> is affected	R <sub>1</sub> is not affected R <sub>2</sub> is affected

### 7.6.2 Agent Capabilities

Software systems continue to develop over time. Changes occur frequently in software development. They may result from modifications to users’ original requirements, modifications to the environment in which the software operates, and bug fixing (Naslavsky et al. 2005). The SEOMAS approach can assist software teams to manage requirements information captured in the Software Engineering Ontology as instantiations and to facilitate automated requirements traceability tasks as follows.

### 7.6.2.1 Manipulating a requirement captured in the Software Engineering Ontology

When a team member requests a change to a requirement (add/modify/delete) through his user agent, a user agent translates the request into an ACL message and sends it to the ontology agent. The ontology agent requests the recommender agent sends a recommendation regarding the impact of a change to other relevant software artefacts (e.g., other requirements, use cases, source codes, test cases). If a team member confirms that a change can be made, the ontology agent manipulates the requirement captured as instance knowledge in the Software Engineering Ontology as requested.

### 7.6.2.2 Recommend change impact on related requirements

When the recommender agent receives a request to manipulate a requirement, it retrieves the requirements that are related to the proposed change and identifies them as the affected requirements or the candidate-affected requirements. In Figure 7-18, if the retrieved requirement has an interdependency type with the changed requirement, the recommender agent needs to process it against a change impact rule to identify the type of impact. However, if the retrieved requirement has no interdependency type but shares the same use case with the changed requirement, the recommender agent considers it as a candidate-affected requirement.

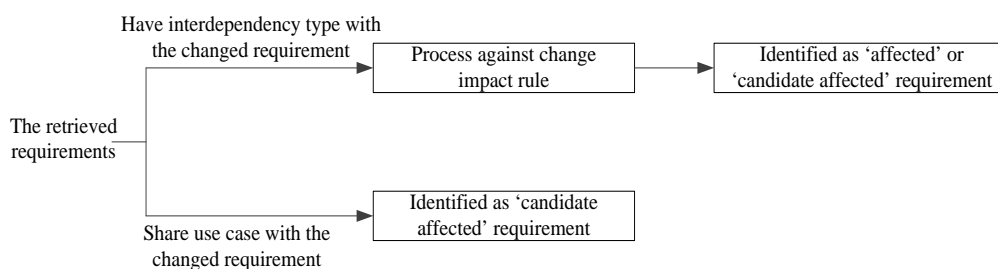


Figure 7-18: Types of potentially affected requirements

In the case where a new requirement is added to the existing requirement, for instance, the system currently accepts payment by credit card and a new requirement is added to allow the user to pay by PayPal. The recommender agent will consider this case similar to modify the existing requirement. It determines the type of



interdependency that exists between the existing requirements and then it applies the change impact rules shown in Table 7-2 to identify the impact of adding a new requirement. For example, in Figure 7-19, a new requirement R is added to R<sub>2</sub>; therefore, R<sub>2</sub> is changed and regarded as an affected requirement. Because R<sub>1</sub> requires R<sub>2</sub>, so R<sub>1</sub> is considered as a candidate-affected requirement.

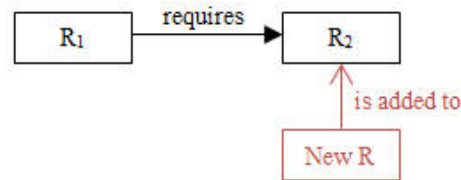


Figure 7-19: New R is added to existing requirement R<sub>2</sub>

### 7.6.2.3 Recommend change impact on other related software artefacts

The Software Engineering Ontology stores software project development information as instance knowledge. Information about software artefacts such as requirements, use cases, classes, and test cases is semantically linked and can be used for traceability recovery. For example, Requirement A requires Requirement B, Requirement B implements use case C relating to class D and deriving test case E. The recommender agent retrieves traceability information to identify the directly-, indirectly-, or candidate-affected software artefacts, namely, use cases, classes and test cases as well as the authors of those artefacts (Figure 7-20).

- If they are related to the changed requirement, the recommender agent identifies them as directly-affected artefacts.
- If they are related to the requirement affected by the proposed change, the recommender agent identifies them as indirectly-affected artefacts.
- If they are related to the candidate-affected requirement of the changed requirement, the recommender agent identifies them as candidate-affected artefacts.

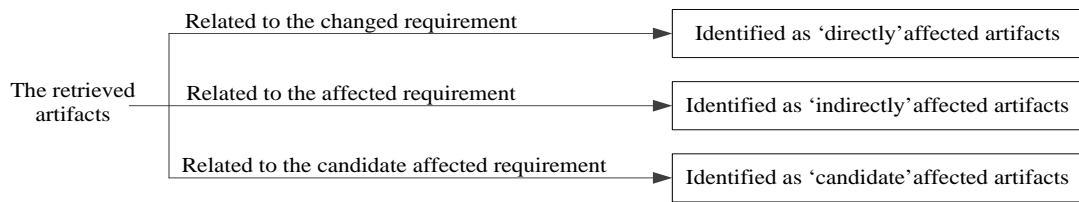


Figure 7-20: Types of potential affected artefacts

#### 7.6.2.4 Notify relevant team members about the impact of the requirement change.

To enable other software teams to become aware of the change made to a requirement, the recommender agent sends messages to notify those artefacts' owners about a change in requirements that may affect their own artefacts as soon as the Software Engineering Ontology instance knowledge is manipulated. They can become aware of the change to requirements in a timely manner and then undertake further investigation regarding the effect on their artefacts if it is needed.

#### 7.6.2.5 Generate traceability matrix

The ontology agent is capable of generating a dynamic traceability matrix to represent the links between the requirement and the other software artefacts by retrieving the related instance knowledge based on its associated concepts defined in the Software Engineering Ontology. These artefacts include use cases, classes and test cases. The ontology agent also allows its user to specify the requirement ID to filter the output of the traceability matrix or leave it blank to present all information.

### 7.6.3 Case Study

The implemented prototype has been demonstrated through the case study of an online shopping software development (Gupta 2013). Additional information (e.g., requirement dependencies, test cases, new requirements) is added for the purpose of evaluating the SEOMAS approach as presented in Appendix A. The SEOMAS approach for the Software Engineering Ontology instantiations management is utilised in this project to assist software development teams to

manage changes in requirements. Figure 7-21 shows all agents in the SEOMAS platform. Each user agent lives in the container according to its software development site.

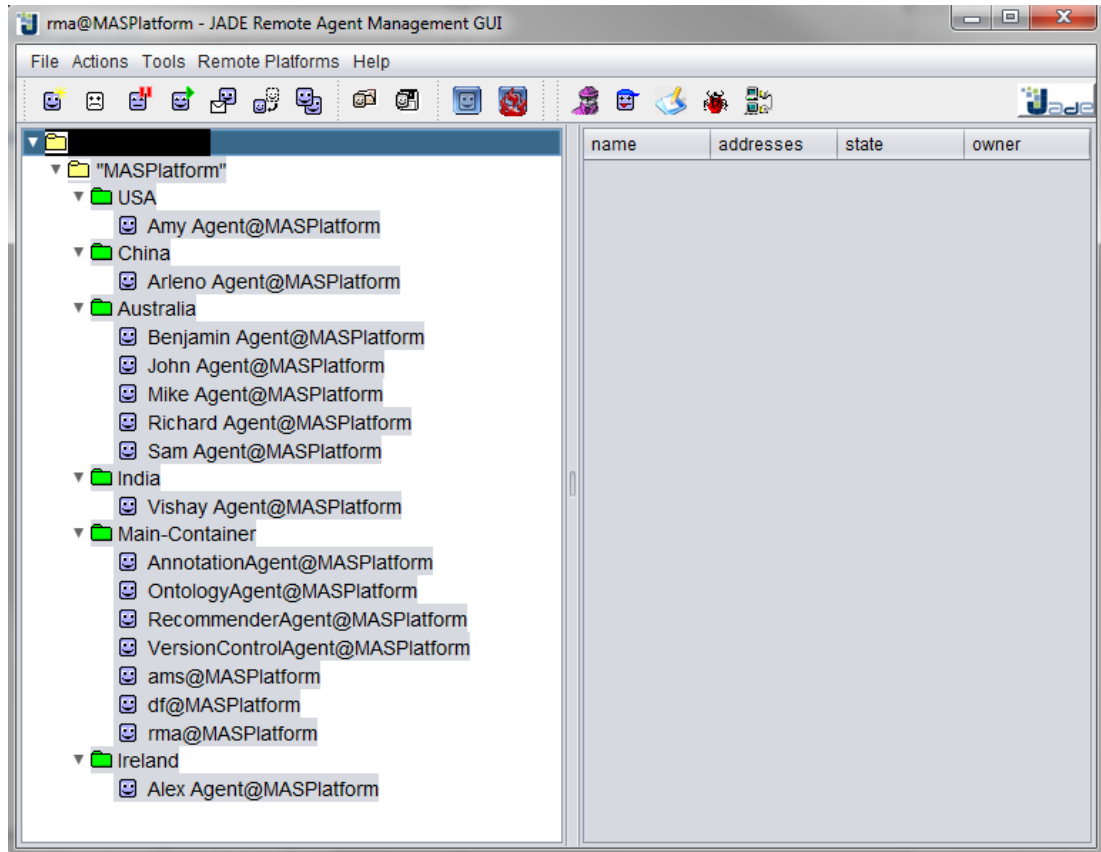


Figure 7-21: Agents in the SEOMAS platform

### 7.6.3.1 Scenario 1 - Querying instance knowledge

This scenario demonstrates the ability of the SEOMAS approach to query the instance knowledge in the Software Engineering Ontology. The analyst, Mike in Australia, wants to establish a trace from requirements to test cases in order to verify whether all the requirements have been taken into account in the developed system. Thus, he asks his user agent for requirement traceability information through the query platform. His user agent then sends a query to the ontology agent to retrieve all related traceability information from the Software Engineering Ontology knowledge base.

The ontology agent sends all information back to Mike's user agent to

generate the output in the form of a requirement traceability matrix as presented in Figure 7-22

```

-- Requirement Traceability Matrix
-----
| Requirement | UseCase | Class | TestCase |
=====
| seo:FR01 | seo:UC1 | seo:DBController | seo:TC1.1 |
| seo:FR02 | seo:UC2 | seo:Items | seo:TC2.1 |
| seo:FR02 | seo:UC2 | seo:Items | seo:TC4.1 |
| seo:FR03 | seo:UC3 | seo:Cart | seo:TC3.1 |
| seo:FR03 | seo:UC3 | seo:Cart | seo:TC3.2 |
| seo:FR03 | seo:UC3 | seo:Cart | seo:TC3.3 |
| seo:FR03 | seo:UC3 | seo:Cart | seo:TC3.4 |
| seo:FR04 | seo:UC2 | seo:Items | seo:TC2.1 |
| seo:FR04 | seo:UC2 | seo:Items | seo:TC4.1 |
| seo:FR05 | seo:UC4 | seo:Cart | seo:TC5.1 |
| seo:FR05 | seo:UC4 | seo:Cart | seo:TC5.2 |
| seo:FR05 | seo:UC4 | seo:Cart | seo:TC7.1 |
| seo:FR06 | | | |
| seo:FR07 | seo:UC4 | seo:Cart | seo:TC5.1 |
| seo:FR07 | seo:UC4 | seo:Cart | seo:TC5.2 |
| seo:FR07 | seo:UC4 | seo:Cart | seo:TC7.1 |
| seo:FR08 | seo:UC6 | seo:Cart | seo:TC11.1 |
| seo:FR08 | seo:UC6 | seo:Cart | seo:TC11.2 |
| seo:FR08 | seo:UC6 | seo:Cart | seo:TC8.1 |
| seo:FR08 | seo:UC6 | seo:Checkout | seo:TC11.1 |
| seo:FR08 | seo:UC6 | seo:Checkout | seo:TC11.2 |
| seo:FR08 | seo:UC6 | seo:Checkout | seo:TC8.1 |
| seo:FR09 | seo:UC7 | seo:Items | seo:TC9.1 |
| seo:FR10 | seo:UC8 | seo:Items | |
| seo:FR11 | seo:UC6 | seo:Cart | seo:TC11.1 |
| seo:FR11 | seo:UC6 | seo:Cart | seo:TC11.2 |
| seo:FR11 | seo:UC6 | seo:Cart | seo:TC8.1 |
| seo:FR11 | seo:UC6 | seo:Checkout | seo:TC11.1 |
| seo:FR11 | seo:UC6 | seo:Checkout | seo:TC11.2 |
| seo:FR11 | seo:UC6 | seo:Checkout | seo:TC8.1 |
| seo:FR12 | seo:UC9 | seo:DBController | seo:TC12.1 |

```

Figure 7-22: Excerpt of a traceability matrix

### 7.6.3.2 Scenario 2 - Modifying instance knowledge

This scenario demonstrates the ability of the SEOMAS approach to modify the requirements information captured in the Software Engineering Ontology as instance knowledge. The analyst, Mike at the Australia site, requests a modification to the requirement FR01 by changing its description to “*The users shall not be able to view the categories on the applications home page*” through his user agent. The agent constructs and sends an ACL message to the ontology agent to modify the FR01’s requirement description as requested (Figure 7-23).

```

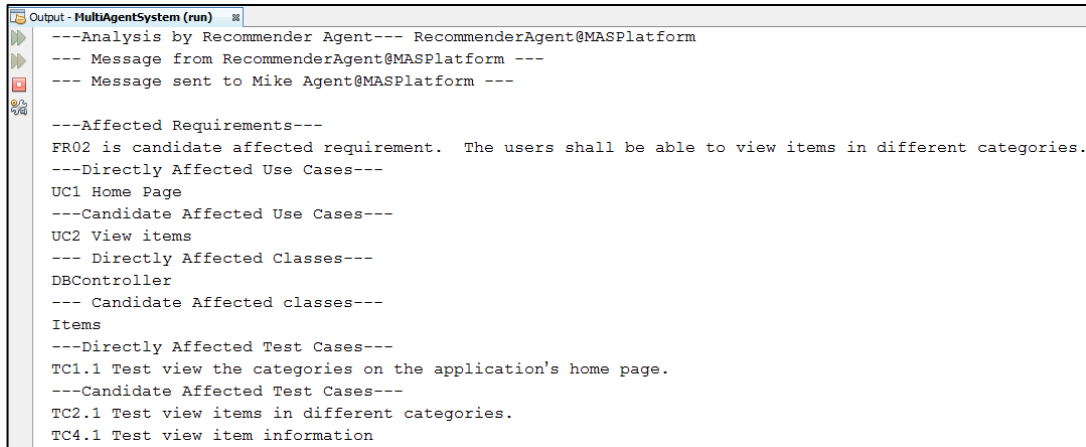
Message: (REQUEST
:sender (agent-identifier :name "Mike Agent@MASPlatform" :addresses
(sequence http://C-A0012783.staff.ad.curtin.edu.au:7778/acc))
:receiver (set (agent-identifier :name OntologyAgent@MASPlatform ))
:content "((action (agent-identifier :name \"Mike Agent@MASPlatform\"
:addresses (sequence http://C-A0012783.staff.ad.curtin.edu.au:7778/acc))
(UpdateRequirement :reqContent \"The users shall not be able to view the
categories on the applications home page.\" :name FR01)))"
:language fipa-sl
:ontology SEOntology
:protocol FIPA-Request
)

```

Figure 7-23: An ACL message requesting to modify the Requirement FR01

The ontology agent extracts the content of the ACL message and evaluates the requested action. It sends a request to the recommender agent to recommend about the impact of the modification of the requirement FR01 to Mike as shown in Figure 7-24. It recommends requirement FR02 as a candidate-affected requirement because its interdependency with FR01 is '*FR02 Requires FR01*' and the request is the '*modify*' type. Use case UC1 is also recommended as a directly affected use case because it is realised from FR01. Use case UC2 is considered as a candidate-affected use case because it is realised from the candidate-affected requirement FR02. Class DBController is considered as a directly-affected class because it relates to use case UC1, a directly affected use case. Class Items is regarded as a candidate-affected class because it relates to use case UC2, a candidate-affected use case. Test case TC1.1 is a directly affected test case because it is derived from use case UC1, a directly affected use case. Test case TC2.1 and TC4.1 are candidate-affected test cases because they are derived from the candidate-affected use case UC2. If Mike confirms the modification, the ontology modifies the requirement FR01 as requested and sends a message to Mike's user agent to confirm the modification in the Software Engineering Ontology (Figure 7-25). The recommender agent then sends messages to notify the relevant user agents of the authors of those affected artefacts

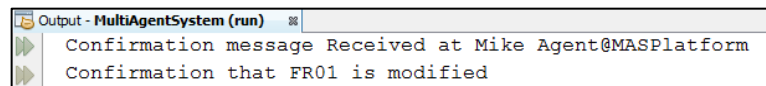
as shown in Figure 7-26. For instance, John’s user agent receives a message to notify that FR01 is modified and it is a directly affected UC1 (Home page) use case. John can then be aware of the requirement change and take appropriate action in response to the change.



```
Output - MultiAgentSystem (run)
---Analysis by Recommender Agent--- RecommenderAgent@MASPlatform
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Mike Agent@MASPlatform ---

---Affected Requirements---
FR02 is candidate affected requirement. The users shall be able to view items in different categories.
---Directly Affected Use Cases---
UC1 Home Page
---Candidate Affected Use Cases---
UC2 View items
--- Directly Affected Classes---
DBController
--- Candidate Affected classes---
Items
---Directly Affected Test Cases---
TC1.1 Test view the categories on the application's home page.
---Candidate Affected Test Cases---
TC2.1 Test view items in different categories.
TC4.1 Test view item information
```

Figure 7-24: Recommendation of potentially affected artefacts



```
Output - MultiAgentSystem (run)
Confirmation message Received at Mike Agent@MASPlatform
Confirmation that FR01 is modified
```

Figure 7-25: A message to confirm the modification of the requirement FR01

```

Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Amy Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate following test case(s) for the directly affected test cases.
  TC1.1 Test view the categories on the application's home page.

--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Sam Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate the following use case(s) for the candidate affected use cases.
  UC2 View items

--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to John Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate the following use case(s) for the directly affected use cases.
  UC1 Home Page

--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Amy Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate following test case(s) for the candidate affected test cases.
  TC2.1 Test view items in different categories.
  TC4.1 Test view item information

---Requested changes are saved to ontology by --- RecommenderAgent@MASPlatform
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Vishay Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate following class(es) for the directly affected classes.
  DBController

--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Richard Agent@MASPlatform ---
Requirement FR01 is modified.
Recommend to investigate following class(es) for the candidate affected classes.
  Items

```

Figure 7-26: Messages to notify the authors of potentially affected artefacts

From this scenario, it can be seen that the SEOMAS approach not only assists the distributed teams to manage a change in requirement captured in the Software Engineering Ontology, but also has the ability to recommend the impact of a change and to notify relevant team members to be aware of a change made from the remote sites in a timely manner. The updated requirement information is also instantly available for sharing among other team members across multiple sites.

### 7.6.3.3 Scenario 3 - Adding new instance knowledge

This scenario demonstrates the ability of the SEOMAS approach to extend the existing instantiations of the Software Engineering Ontology with new

instantiations. The analyst, Sam at the Australia site, requests the addition of a new requirement FR21 “*The user shall complete the captcha when logging in for the purpose of differentiating a human being from the computer program*”. Sam’s user agent creates and sends an ACL request message to add a new requirement FR21 together with its information to the ontology agent.

```
Message: (REQUEST
:sender ( agent-identifier :name "Sam Agent@MASPlatform" :addresses (sequence
http://C-A0012783.staff.ad.curtin.edu.au:7778/acc ))
:receiver (set ( agent-identifier :name RecommenderAgent@MASPlatform ) )
:content "((action (agent-identifier :name \"Sam Agent@MASPlatform\" :addresses
(sequence http://C-A0012783.staff.ad.curtin.edu.au:7778/acc)) (AddRequirement
:reqContent \"The user shall complete the captcha when log in for the purpose of
differentiating a human being from the computer program.\" :priority Desirable
:status Stability :useCase UC14 :newfrname FR21)))"
:language fipa-sl
:ontology SEOntology
:protocol FIPA-Request
)
```

The ontology agent extracts the content of the ACL message and evaluates the requested action. In this case, the action is the addition of a requirement. A message content slot consists of information about a new requirement, namely the requirement’s content, requirement’s changed status, requirement’s priority, and its realised use case. The ontology agent creates a new instance of a Requirement class and inserts the information from a message content slot into its data properties and object properties. A new requirement instance FR21 instantiation is inserted into the ontology repository as shown in Figure 7-27.



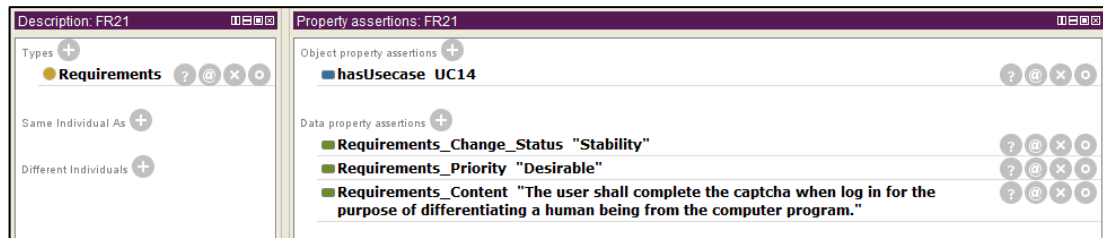


Figure 7-27: A new added instantiation of a Requirements class

## 7.7 Discussion

In this section, results from the case study using the SEOMAS approach to support the requirement change management focusing on requirements traceability tasks are discussed and compared with other related work.

Requirements traceability, considered as a sub-part of requirements management, is challenging in centralised software development and even more so in a multi-site environment where software teams are located across several sites. Software development projects need to deal with a variety of changes including changes made to requirements during the whole software development life cycle. A single change can impact on other artefacts because of their interdependencies. An effective and proactive approach is needed to establish and maintain consistency between these software artefacts and alert remote team members to any changes in requirements. In the case study, it has been proven that the SEOMAS approach enables effective and efficient coordination of software teams by improving real-time awareness of the teams when a requirement is changed. Additionally, it provides efficient and effective communication through timely notification directed to the relevant team members.

There are extensive research works on topics related to traceability when there is a change in requirements. In (Assawamekin, Sunetnanta and Pluempitiwiriyaewej 2009; Hayes, Dekhtyar and Sundaram 2006), the authors focus only on vertical traceability of requirement artefacts. This may result in a limited view of the artefacts potentially affected by a change request. The development of large and complex software systems generates various artefacts with different levels

of abstraction throughout the life cycle. Therefore, both vertical and horizontal traceability approaches are required to manage the links between related artefacts at these different levels of abstraction. In (Goknil et al. 2011), the authors proposed a requirement meta-model with formal relation types together with a tool named Tool for Requirements Inferencing and Consistency checking (TRIC). The purpose is to manage requirements and to support automatic inferencing and consistency change management. Their approach focuses only on tracing between requirements and requirements, as well as between requirements and architectural components, but not on other software artefacts. The authors of (de Almeida Falbo, Braga and Machado 2014) propose IMSD-Req, an extension of the work in (de Oliveira Arantes and de Almeida Falbo 2010) by introducing new requirements-specific features including analysing the impact of changes to requirements, evaluating consistency of requirement prioritisation, generating a requirement traceability matrix, and verifying requirements using checklists. This approach is similar in concept to our own, but it mainly provides traceability support to co-located teams. It does not have enough capability to ensure team awareness of changes to requirements in a multi-site software development project. Team members at different sites may be unaware of the effect of the change. This can create inconsistencies between development artefacts.

Of the commercial tools available for the management of requirements, IBM Rational RequisitePro (RequisitePro® 2015) is one of the well-known products that helps project teams to manage and maintain their requirements. It allows software developers to manage traceability among requirements and other related artefacts such as classes, and test cases. However, it offers only two general relation types expressing the direction of dependency between requirements: traceFrom and traceTo. Therefore, if one requirement is changed, then all requirements traced from the changed requirement are considered to be affected and need to be investigated for the change. This can produce many false positives.

However, the SEOMAS approach can address the shortcomings of these works in the following ways.

- 1) To enable software artefacts to be traced by means of both vertical and horizontal traceability, the Software Engineering Ontology, which is an ontology that

defines common shareable software engineering knowledge and represents the concepts in the software engineering domain, is used to establish and maintain the traceability information. It can provide the support needed to unify and relate these artefacts to trace between requirements and other requirements, and to trace across related artefacts produced throughout the phases of the project's software life cycle.

2) The semantics of requirements relations are defined and they are integrated with the proposed change impact rules to determine which related software artefacts are required to change. This approach can help to create more valid impacts and reduce the number of false positives. Moreover, the SEOMAS approach differentiates the type of impact on the artefacts whether it is direct or indirect, or an actual or just candidate artefact.

3) In multi-site software development environments, software teams are dispersed across various sites. Team members at one site may be unaware of what is going on at other sites. The SEOMAS framework is based on agent-based technology so it has advantages over the traditional software systems for software traceability in terms of supporting autonomous, reactive and proactive features in a distributed working environment. For instance, it can provide automated support for the traceability recovery and change impact analysis on an autonomous basis. Notifications are sent to inform the relevant team members who are the owners of the affected artefacts of a change in real-time once software project information captured in the ontology is manipulated. These features can help to improve the software team's awareness of software evolution and to maintain consistency among project artefacts.

## **7.8 Conclusion**

This chapter has focused on the design and development of the ontology-based multi-agent system for Software Engineering Ontology instantiations management. The specifications of the involved agents (i.e. user agent, the ontology agent, and the recommender agent) are refined in regard to three aspects, namely, resources, behaviours, and interactions. The SEOMAS approach is evaluated through

a case study of an online shopping software development by focusing on supporting requirement traceability tasks.

The chapter concludes with a comparison of the results obtained from the SEOMAS approach with those of the other approaches for supporting requirement traceability tasks. It also presents a discussion regarding how the approach can improve team awareness of software evolution and can help to maintain consistency among project artefacts in a proactive manner. The next chapter provides an analysis and discussion of the SEOMAS framework for active platforms for multi-site software development environments.

## 7.9 References

- Assawamekin, N., T. Sunetnanta, and C. Pluempitiwiriyaewej. 2009. "MUPRET: An ontology-driven traceability tool for multiperspective requirements artifacts." In *Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on*, June 1-3, 2009. 943-948. doi: 10.1109/ICIS.2009.55.
- Dahlstedt, Asa G, and Anne Persson. 2003. "Requirements interdependencies-moulding the state of research into a research agenda." In *9th International Workshop on Requirements Engineering–Foundation for Software Quality (RefsQ'03), Klagenfurt/Velden, Austria*, June 16 -17, 2003. 55-64.
- Dahlstedt, ÅsaG, and Anne Persson. 2005. "Requirements interdependencies: State of the art and future challenges." In *Engineering and Managing Software Requirements*, eds Aybüke Aurum and Claes Wohlin, 95-116. Springer Berlin Heidelberg.
- de Almeida Falbo, Ricardo, Carlos Eduardo C Braga, and Bruno Nandolpho Machado. 2014. "Semantic documentation in requirements engineering." In *17th Workshop on Requirements Engineering (WER 2014), Pucón, Chile*, April 23-25, 2014.

- de Oliveira Arantes, Lucas, and Ricardo de Almeida Falbo. 2010. "An infrastructure for managing semantic documents." In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International*, October 25-29, 2010. 235-244. IEEE.
- Göknil, Arda, Ivan Kurtev, and KG van den Berg. 2008. "Change impact analysis based on formalization of trace relations for requirements." In *Traceability Workshop - European Conference on Model Driven Architecture Foundations and Applications (ECMDA-TW 2008), Berlin, Germany, June 9-12, 2008*
- Goknil, Arda, Ivan Kurtev, Klaas van den Berg, and Jan-Willem Veldhuis. 2011. "Semantics of trace relations in requirements models for consistency checking and inferencing." *Software & Systems Modeling* 10 (1): 31-54.
- Gotel, Orlena CZ, and Anthony CW Finkelstein. 1994. "An analysis of the requirements traceability problem." In *1st International Conference on Requirements Engineering., Colorado Springs, CO, April 18-22., 94-101*. IEEE.
- Gupta, Swati. 2013. "Online Shopping Cart Application." Dissertation, Agriculture and Applied Science, North Dakota State University, USA. [http://library.ndsu.edu/tools/dspace/load/?file=/repository/bitstream/handle/10365/23054/Swati\\_Online%20Shopping%20Cart%20Application.pdf?sequence=1](http://library.ndsu.edu/tools/dspace/load/?file=/repository/bitstream/handle/10365/23054/Swati_Online%20Shopping%20Cart%20Application.pdf?sequence=1).
- Hayes, J. H., A. Dekhtyar, and S. K. Sundaram. 2006. "Advancing candidate link generation for requirements tracing: the study of methods." *IEEE Transactions on Software Engineering* 32 (1): 4-19. doi: 10.1109/TSE.2006.3.
- Naslavsky, Leila, Thomas A Alspaugh, Debra J Richardson, and Hadar Ziv. 2005. "Using scenarios to support traceability" *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*: ACM.
- Pohl, Klaus. 1996. *Process-Centered Requirements Engineering*. New York, USA: John Wiley & Sons.

RequisitePro®, IBM Rational. 2015. <http://www-01.ibm.com/software/in/awdtools/-reqpro/>.

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.

# **Chapter 8 Active Platforms for Multi-site Software Development Environments**

## **8.1 Introduction**

This chapter discusses the four active platforms for multi-site software development environments. In the framework for active Software Engineering Ontology through a multi-agent system (SEOMAS), the Software Engineering Ontology is a main component capturing software engineering domain knowledge together with software development project information. The platforms are equipped with active support to facilitate the management of software development project information and enable knowledge sharing through collaborative software agents situated in the foreground of the ontology. They interact and mediate between the ontology and software project team members. The platforms define standards pertaining to the framework to support collaborative software development with software engineering knowledge throughout various activities of the software life cycle. They provide project team members with relevant and useful information that can assist them to carry out software development activities. Furthermore, the platforms can enable team members to be more productive by automating certain time-consuming and tedious tasks such as knowledge capturing or searching.

The chapter begins by presenting the platforms framework, followed by details of each platform. Then the practical uses of the platforms are discussed to demonstrate their capabilities to actively assist software development teams to manage and share knowledge. The chapter concludes with an enumeration of the benefits of using the platforms.

## 8.2 Platforms Framework

In this section, the SEOMAS platforms are discussed in terms of how they can assist software project team members to access and manage software engineering knowledge captured in the Software Engineering Ontology. The SEOMAS framework consists of four platforms: knowledge capture platform, knowledge query platform, knowledge monitoring platform, and knowledge manipulation platform. These platforms support knowledge process activities including knowledge capture, knowledge search, knowledge dissemination, and knowledge maintenance (Natali and Falbo 2002). The Software Engineering Ontology is located at the core of the knowledge infrastructure to support knowledge management, knowledge sharing and reuse (Figure 8-1).

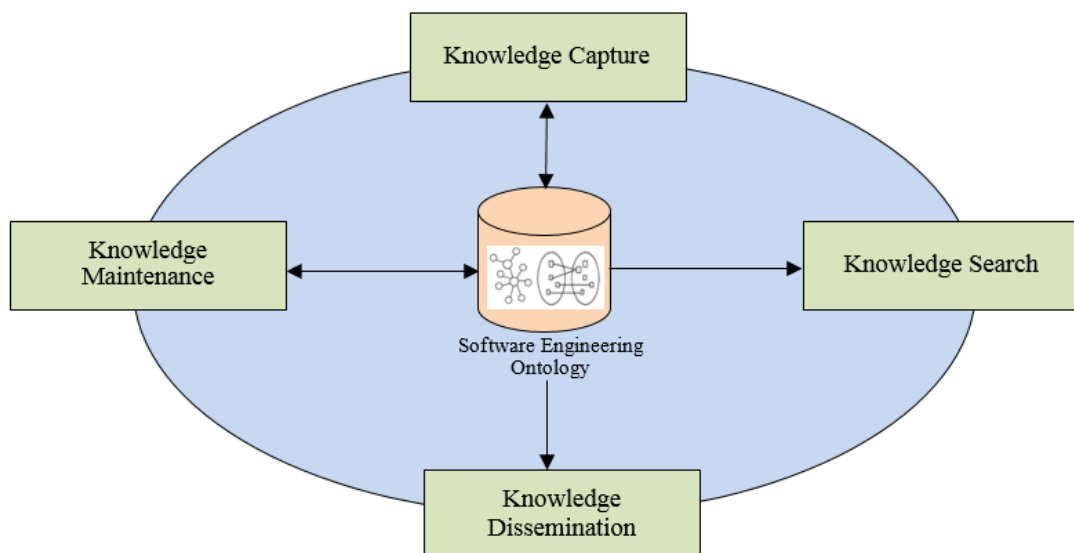


Figure 8-1: Software Engineering knowledge management and sharing infrastructure

### Knowledge Capture

Large amounts of software project information are produced during a software project. The task of manually capturing of domain specific knowledge in a formal conceptual model of this project information is laborious, costly, and error-prone. The automated capturing of software project information and populating it in the Software Engineering Ontology repository are done through the knowledge capture platform.



### **Knowledge Search**

Team members query or search for software engineering domain knowledge or software project information captured in the Software Engineering Ontology to satisfy their particular information need. This search is considered as a user-initiated search which means that a user has to specify the information that s/he requires in order to formulate the query. Team members can access required knowledge by querying or searching through the knowledge query platform.

### **Knowledge Dissemination**

Knowledge dissemination is different from knowledge search in the sense that it is initiated by the system and does not require a user to explicitly request it. In other words, the knowledge that is potentially useful to the users is transferred to them in a proactive manner without them having to explicitly request it. The platforms can recommend relevant knowledge for particular situational tasks (e.g. a match between bug and expert) or proactively monitor software project information and notify appropriate team members when there is any potential deviation. This proactive assistance for knowledge dissemination is provided through the knowledge manipulation platform and the knowledge monitoring platform.

### **Knowledge Maintenance**

Software project information often evolves as a result of changes made to requirements or to design processes. One of the main challenges related to software evolution is to maintain the consistency among related software development artefacts. In this thesis, the maintenance of software project information captured in the Software Engineering Ontology is done through the knowledge manipulation platform. The platform also provides active support for automated traceability recovery to maintain consistency among software artefacts and to facilitate team's awareness in real time during software evolution.

These activities are conducted through the Software Engineering Ontology-based multi-agent system for knowledge management and knowledge sharing platforms, or the SEOMAS platforms for short. These platform levels are depicted in Figure 8-2.

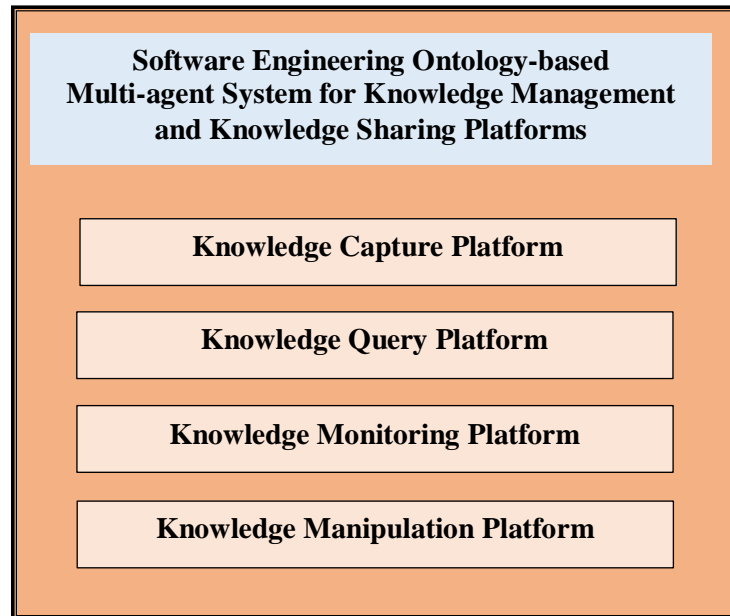


Figure 8-2: Four levels of Software Engineering Ontology-based multi-agent system knowledge management and knowledge sharing platforms

The **knowledge capture platform** assists team members to automatically capture software engineering knowledge from software project information by means of the semantic annotation and the ontology population process. The Software Engineering Ontology is a main component that provides domain knowledge throughout the semantic annotation process. Software project information is captured and transformed into a conceptually organised form and can be semantically interlinked with other relevant information sources. In this thesis, the knowledge capture platform focuses on capturing software engineering knowledge from source code artefacts. This is because they are centrally located and critical in software development.

The **knowledge query platform**, or the query platform for short, assists project team members to query or search for software engineering domain knowledge and semantically-linked software project information captured in the Software Engineering Ontology. The platform has the ability to exploit existing reasoning mechanisms of the ontology (e.g., class subsumption, instance checking) to derive knowledge that is more relevant to the search.

The **knowledge monitoring platform**, or monitoring platform for short, is

proactively monitors the software project information, i.e., instance knowledge that is more likely to encounter deviations or disruptive events which may lead to poor quality of the final software product, project delay, and budget overrun. This information is used to support effective decision making by the software teams in response to these deviations or disruptive events.

The **knowledge manipulation platform**, or manipulation platform for short, enables all team members to make any change (i.e., add, modify, and delete) to the software project information captured in the ontology. In order to do so, a team member interacts with the platform to make a request to manipulate the instance knowledge. The platform also provides a recommendation regarding the impact that the change may have on related software artefacts, and notifies relevant team members of any change made by others. The recommendation and notification are offered to project members on an information push-based basis.

Figure 8-3 presents a flow chart of the processes when utilising the platforms to access, manage, and share software project information captured in the ontology. Basically, the platforms offer many usage possibilities depending on a user's particular requirement. For example, software project information is semantically annotated and populated in the Software Engineering Ontology knowledge base. Then team members can query for particular information and/or manipulate the knowledge according to their particular tasks. Another example is that once the knowledge has been captured, the platforms proactively monitor it and notify relevant members when a deviation is identified. The team leader can query to inspect the situation and/or manipulate the knowledge if it is needed.

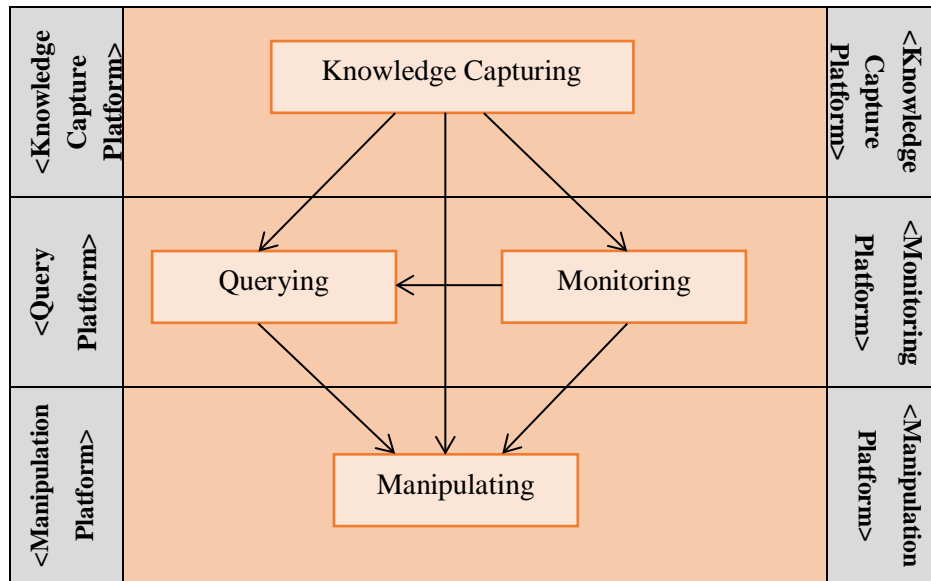


Figure 8-3: A flow of the processes when utilising the platforms

The SEOMAS platforms are intended to actively assist software development teams to access, manage, and share software project information throughout the various phases of the software life cycle. They are also intended to facilitate effective and efficient communication and coordination among team members. In the next sections, details of each platform are discussed.

### 8.3 Knowledge Capture Platform

The knowledge capture platform involves the process of capturing software engineering knowledge from the software project information during a daily software development activity. A team member interacts with this platform by making a request to import a source code file into the version control repository. When the file is imported, it is also semantically annotated with the appropriate concepts and relations defined in the Software Engineering Ontology in order to identify new instances of the ontological concepts. They are also enriched with other controlled vocabularies (e.g. FOAF, DC, DBpedia) to encourage ontology reuse and information interoperability. Additionally, they are interlinked with similar entities in the DBpedia dataset in order to provide additional information in regard to the concepts or entities represented by the concept. The annotated and enriched source

code elements are populated into the Software Engineering Ontology as new instantiations. Once populated, they are available for use to facilitate remote communication and coordination among project team members, among the SEOMAS agents, and between members and the agents. Figure 8-4 demonstrates a request to import source code through the platform. A developer requests to import the source code “Employee.java” to the project version control repository. The source code used as an example here is derived from the book “Java™ for Programmers” (Deitel and Deitel 2011, 268-269).

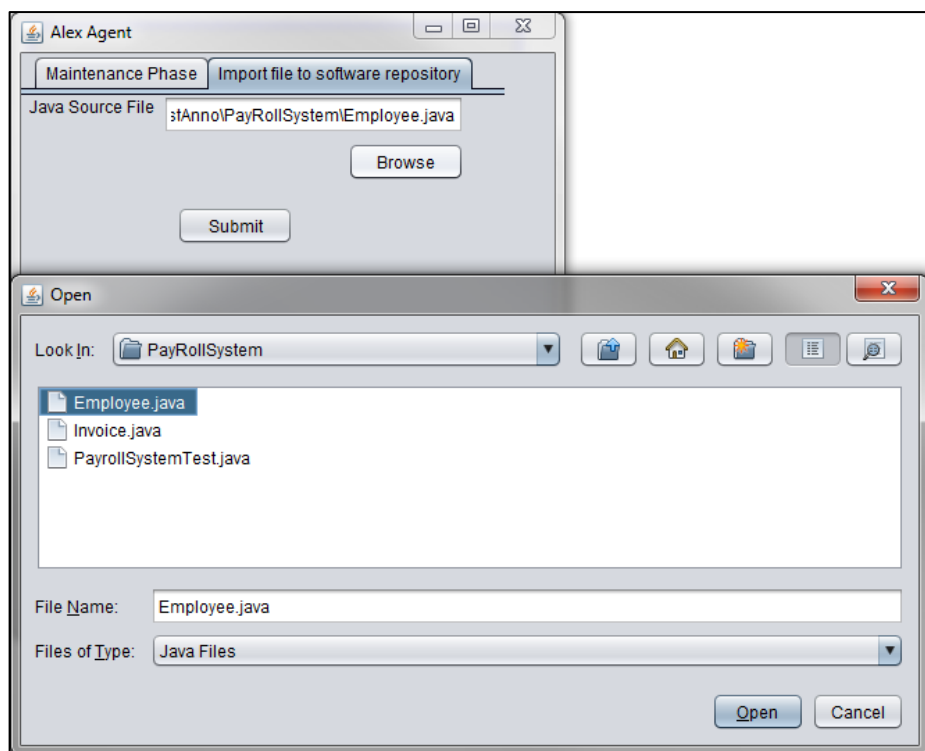


Figure 8-4: Alex user agent requests to import a Java source code file into the version control repository

Once the source code has been imported, it is semantically annotated with the Software Engineering domain concepts. The result of the annotation from the Employee Java source code is shown in Figure 8-5.

=====Semantic Annotation START =====

File Name >> Employee.java

File Creation data >> 1463544962841

INFO [OntologyAgent] (Config.java:27) - Trying to initiate config

Java Classes --> Employee

Class has following field(s):

Field Name: firstName AccessModifier: private DataType: java.lang.String

Field Name: lastName AccessModifier: private DataType: java.lang.String

Field Name: socialSecurityNumber AccessModifier: private DataType: java.lang.String

Class has AccessModifier: public

Implemented Interfaces: Payable

Constructor: Employee

Class has following method(s):

Employee

setFirstName

getFirstName

setLastName

getLastName

setSocialSecurityNumber

getSocialSecurityNumber

toString

Method name: Employee

Modifier: public

Parameters:

Name: first DataType: java.lang.String

Name: last DataType: java.lang.String

Name: ssn DataType: java.lang.String

Method name: setFirstName

Method return type: void

Modifier: public

Parameters:

```
Name: first   DataType: java.lang.String

Method name: getFirstName
Method return type: java.lang.String
Modifier: public

Method name: setLastName
Method return type: void
Modifier: public
Parameters:
Name: last   DataType: java.lang.String

Method name: getLastName
Method return type: java.lang.String
Modifier: public

Method name: setSocialSecurityNumber
Method return type: void
Modifier: public
Parameters:
Name: ssn   DataType: java.lang.String

Method name: getSocialSecurityNumber
Method return type: java.lang.String
Modifier: public

Method name: toString
Method return type: java.lang.String
Modifier: public

===== Semantic Annotation END =====
```

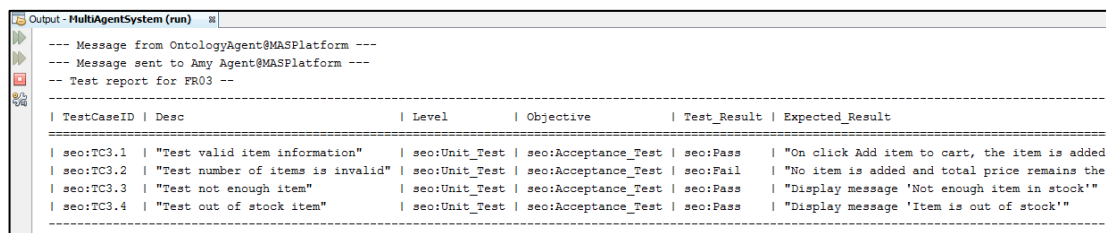
Figure 8-5: Result of annotation from the Employee Java source code

The annotated source codes are then populated in the Software Engineering Ontology as instance knowledge. They are subsequently used in other platforms to clarify any ambiguity in remote communication and facilitate effective and efficient

coordination among project development teams. With assistance from the knowledge capture platform, the tedious and time-consuming process of manually capturing the software project information becomes automated and requires only minimum effort from the team members.

## 8.4 Query Platform

The query platform is where team members can query the semantic linked software project information to facilitate their work. A team member sends a request to query software project information through the query platform. The request is processed to retrieve the query result from the Software Engineering Ontology and it is sent back to a team member. Figure 8-6 shows an example of querying test results of a particular requirement (i.e., FR03) from the query platform.



```

--- Message from OntologyAgent@MASPlatform ---
--- Message sent to Amy Agent@MASPlatform ---
--- Test report for FR03 ---
-----
| TestCaseID | Desc | Level | Objective | Test_Result | Expected_Result |
-----
| seo:TC3.1 | "Test valid item information" | seo:Unit_Test | seo:Acceptance_Test | seo:Pass | "On click Add item to cart, the item is added" |
| seo:TC3.2 | "Test number of items is invalid" | seo:Unit_Test | seo:Acceptance_Test | seo:Fail | "No item is added and total price remains the" |
| seo:TC3.3 | "Test not enough item" | seo:Unit_Test | seo:Acceptance_Test | seo:Pass | "Display message 'Not enough item in stock'" |
| seo:TC3.4 | "Test out of stock item" | seo:Unit_Test | seo:Acceptance_Test | seo:Pass | "Display message 'Item is out of stock'" |
-----

```

Figure 8-6: Excerpt of querying test report of the requirement FR03

It is to be noted that in this platform, the existing reasoning mechanisms of the ontology are exploited to derive the knowledge that is more relevant to the search. The next example demonstrates the platform's ability to utilise the concept of reasoning over the Software Engineering Ontology. According to a case study of the online shopping system development in Chapter 6, a Tester, Amy at the USA site, is implementing a test plan. She consults her user agent to provide information about all use cases and test cases associated with the use case UC9 (Place order). The relationships between use case UC9 and other use cases are shown in Figure 8-7. Her user agent sends a query to the ontology agent and receives a result as depicted in Figure 8-8.

When a use case UC9 is tested, UC10 (Login), UC11(Register), and UC14 (Verify Captcha) also need to be tested. The reason is that a use case UC9 includes



use case UC10. Use case UC10 also includes use case UC14. Therefore, from the reasoning capability over the Software Engineering Ontology (transitive closure), if Amy wants to test a use case UC9, the ontology agent also includes use case UC10 and UC14 in the test plan. Use case UC11 extends use case UC10; thus, it is suggested that UC11 should be tested as well. Consequently, the platform suggests that all the test cases of a use case UC9 (i.e., test cases TC12.1, TC12.2, TC12.3), UC10 (i.e., test cases TC13.1, TC13.2, TC13.3), UC11 (i.e., test cases TC13.4 and TC13.5) and UC14 (i.e., TC14.1 and TC14.2) should be included in the test plan when user UC 9 is tested.

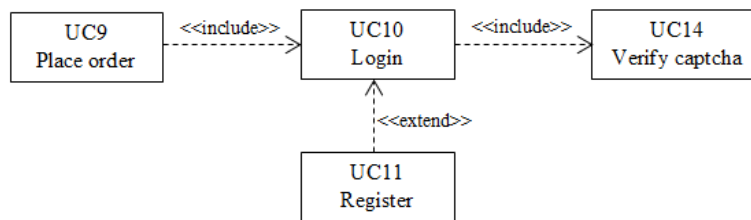


Figure 8-7: Use cases related to a use case UC9

```

Output - MultiAgentSystem (run)
--- Message from OntologyAgent@MASPlatform ---
--- Message sent to Amy Agent@MASPlatform ---

---Related Use Cases---
UC10
UC11
UC14

---All test cases that have to be tested ---
TC12.1
TC12.2
TC12.3
TC13.1
TC13.2
TC13.3
TC13.4
TC13.5
TC14.1
TC14.2
TC25.1
TC25.2
  
```

Figure 8-8: Result of use cases and test cases required to be tested with a use case UC9

From this scenario, it is evident that the platform can utilise the Software Engineering Ontology for reasoning purposes. This is the capability that extends the

traditional keyword-based search in order to increase the relevancy of the search and query retrieval result.

## 8.5 Monitoring Platform

The monitoring platform is concerned with proactively monitoring the particular software project information that might potentially encounter deviations or disruptive events. This information could be useful to avoid problems before they actually occur. The appropriate team member is notified in a timely manner so that s/he can be aware of the situation and respond with appropriate actions.

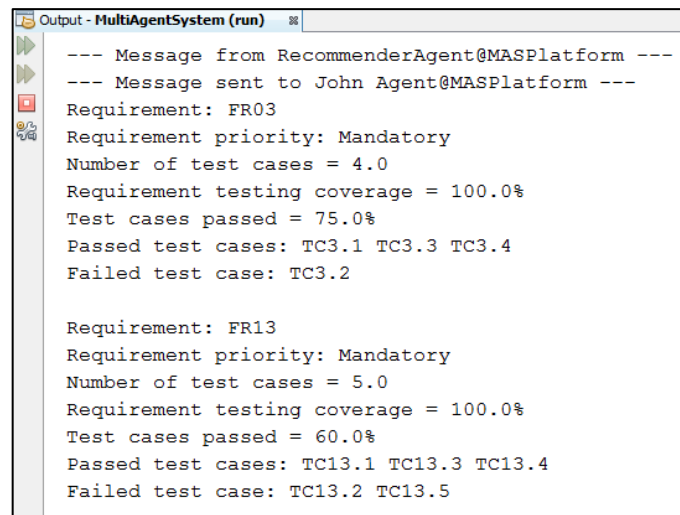
Mainly, the monitoring platform involves proactively tracking particular instance knowledge and responding to the event that may occur. Proactivity means that the platform can assist team members to identify a potential problem before it actually arises. The recommender agent is able to read and process the instance knowledge captured in the Software Engineering Ontology according to pre-defined conditions and engages in dialogue with user agents. A message to notify the person in charge about the deviations or disruptive events is sent in real time when one of the following conditions exists.

1. **A threshold is reached.** For instance, the platform sends a message to notify the class owner that the number of bugs reported to the class is above a certain threshold. The purpose is to alert him/her to the need to investigate the cause(s) of the problem so that s/he can allocate resources to solving and testing efficiently.
2. **A specific condition occurs.** For example, the platform tracks the reported bug and notifies the reporter when the bug issue has been resolved and is closed.
3. **Every appointed period of time** (day, week, month, etc.). For instance, a message is sent to notify the project manager about the issues which have remained unassigned to the fixers for a certain period. Another

example is when a requirement has no linked use case or does not have a test case established for a certain period. This may indicate that it might not have been properly implemented or has gone untested.

In the following example, the platforms demonstrate their ability to proactively monitor the requirement test coverage. Basically, each requirement is mapped to one or more test cases that are used to validate whether the functionality works as expected. Since the test cases cover a requirement, if the test cases associated with a particular requirement cannot be executed successfully, then the requirement is not completely validated. Typically, if a requirement has low priority (e.g., optional), even though the requirement is not completely validated, this could be ignored or postponed for a later fix (Srinivasan and Gopaldaswamy 2006). However, if a requirement has high priority (e.g., highly desirable, desirable, mandatory) and is not successfully validated, it should be inspected and the defects should be fixed; otherwise, it may prevent a product release.

In Figure 8-9, the monitoring platform identifies two requirements (i.e., FR03 and FR13) with the 'Mandatory' requirement priority. Once all test cases associated with these two requirements have been executed, the percentage of their test cases passed is not 100%. Therefore, the platform sends a message to notify the appropriate project member analyst, John, regarding the requirement test coverage report, the mapping between requirements and test cases, and the information about the test cases that have passed and those that have failed. Then he can look at the full test report or consult with the tester to check whether the defects corresponding to these requirements need to be fixed. Subsequently, the failed test cases are re-executed.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to John Agent@MASPlatform ---
Requirement: FR03
Requirement priority: Mandatory
Number of test cases = 4.0
Requirement testing coverage = 100.0%
Test cases passed = 75.0%
Passed test cases: TC3.1 TC3.3 TC3.4
Failed test case: TC3.2

Requirement: FR13
Requirement priority: Mandatory
Number of test cases = 5.0
Requirement testing coverage = 100.0%
Test cases passed = 60.0%
Passed test cases: TC13.1 TC13.3 TC13.4
Failed test case: TC13.2 TC13.5
```

Figure 8-9: Message to give notice of requirement testing coverage

## 8.6 Manipulation Platform

The manipulation platform is where the instance knowledge is manipulated and includes modification, addition, and deletion. A team member makes a request to manipulate the instance knowledge through the platform. His/her user agent translates the request and passes it to the ontology agent to process the manipulation request. The platform can not only manipulate the instance knowledge, but it is also able to recommend the impact of a change to a team member to make him/her be aware of any unintended side effects from the change requested. Furthermore, once the manipulation is committed, timely notifications are sent to inform relevant team members to be aware of the proposed change. They can then investigate and perform certain actions to respond to the change. In this platform, the Software Engineering Ontology is used as a means of semantic tracing to derive dependencies among software development artefacts. The recommender agent consults the Software Engineering Ontology and applies a defined change impact rule to determine potentially affected software artefacts in order to generate recommendations and notifications.

For example, according to the case study presented in section 6.9, when Alex requests a change to the method `getMakeYear` of the `Vehicle` interface by modifying

a method return type through the manipulation platform, the recommender agent can make him aware of the potential impact to other software components. In object-oriented system development, a subclass is dependent on the super class that it inherits or the interface that it implements; therefore, a change in the super class or the interface will impact on its subclass (Khatri and Chillar 2011). Figure 8-10 presents the recommendation of potentially affected artefacts sent to Alex. MotorBike and Car class are suggested as affected classes when the getMakeYear method is modified because they implement the Vehicle interface. VehicleRegistration is also suggested as the affected class because it is the main call which invokes either Car or MotorBike class. Figure 8-11 illustrates messages sent to notify the authors of those potentially affected artefacts to be aware of the change in the Vehicle interface. In this example, the manipulation platform does not only assist team members to manage the software project information captured in the Software Engineering Ontology, but it also provides useful and precise situational knowledge regarding the change impact analysis to improve team members' awareness and alert them to the need for coordination.

```

Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Alex Agent@MASPlatform ---
--Affected Subclasses--
MotorBike implements Vehicle
Car implements Vehicle
--Affected main classes--
VehicleRegistration calls getMakeYear

```

Figure 8-10: Recommendation of potentially affected artefacts

```

Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Arleno Agent@MASPlatform ---
There is a change in getMakeYear of Vehicle class that might affect MotorBike class.
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Vishay Agent@MASPlatform ---
There is a change in getMakeYear of Vehicle class that might affect Car class.
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Richard Agent@MASPlatform ---
There is a change in getMakeYear of Vehicle class that might affect VehicleRegistration class.

```

Figure 8-11: Messages to notify the authors of potentially affected artefacts

## **8.7 Practical Uses**

In this section, the practical uses of the SEOMAS platforms are demonstrated through examples of a case study. It begins with the analysis of problems encountered in the software development project. Then it demonstrates how the SEOMAS platforms are able to assist software teams to proactively identify potential problems or deviations before an actual issue arises, and to manage and share software project information captured in the Software Engineering Ontology.

### **8.7.1 Problem Analysis**

During a software development project, use cases are important for eliciting and documenting functional requirements. They contain useful information that is appropriate for corresponding processes and requirements. Because they are input to various activities in the project, their quality reflects the quality of the whole development project. If a use case is missing, it may result in some necessary functionalities not being implemented (Anda, Hansen and Sand 2009). During the validation of the user requirements, the identification of missing use cases may be necessary (Mefteh, Bouassida and Ben-Abdallah 2014).

In some projects, missing use cases might not be identified until the system is validated at the testing phase. An appropriate example is the case study of the Home Lighting Automation System (HOLIS) project as described in the book “Managing Software Requirements: A Unified Approach” (Leffingwell and Widrig 2000, 354-356). The testing team was validating the system in order to confirm that the implemented system conformed to the requirements established for it. However, the team found that some requirements that had no associated use case and some requirements were not linked to any test case. If this missing information is not discovered or it is identified too late, it could result in a final software product that does not meet customer needs. The use case and test case fragment of traceability in the HOLIS project are depicted in Figure 8-12.

Relationships - Direct Only	TC1:On/Off Dim Test TC2:Round-trip Message TC3:Superfluous Test	Relationships - Direct Only	TC1:On/Off Dim Test TC2:Round-trip Message TC3:Superfluous Test
UC1: On/Off Dim	<	SR1: System Clock The system shall	
UC2: Initiate Emergency		SR2: OnLevel Illumination parameter	
UC3: Do Useless Thing		SR3: HOLIS shall support up to 255	<
		SR4: Message protocol from Control	<
		SR5: The CCU must have no known	

Figure 8-12: Use case and test case fragment of traceability (Leffingwell and Widrig 2000, 354)

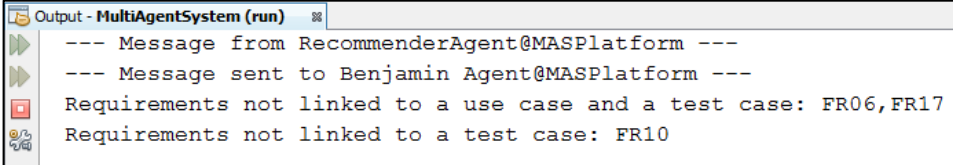
The situation described above indicates that only a manual project review would cause the unintentional error. In the next section, the SEOMAS platforms are demonstrated to provide active support software teams to proactively monitor software project information. The purpose is to identify a deviation or disruptive event that could result in an issue during the software project. In addition, when such an event is identified, the change made to related software project information can be done through the SEOMAS platforms to enable real-time knowledge sharing among project team members.

### 8.7.2 Platform Uses

Examples of the practical uses of the SEOMAS platforms are given throughout this section. A case study of an online shopping system development used in section 7.6.3 is also used here to demonstrate the SEOMAS platforms' capabilities. After software project information has been transformed and populated into the Software Engineering Ontology, the monitoring platform can monitor it in order to verify the status of the software project. In this example, it proactively monitors and identifies incomplete requirement information such as a requirement that is not linked to any use case or test case. When this missing information is identified, a notification is sent to notify the appropriate team member to investigate

this issue instead of going around to irrelevant people.

Figure 8-13 presents a notification sent to Benjamin, a requirement engineer at a site in Australia to notify him about requirements FR06 and FR17 that have no linked use cases. This can indicate that they might not have been properly considered or implemented. FR10 does not have an established test case so this implies that it might have missed out on being tested. The absence of this important information might cause software functionalities to not work as expected.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Benjamin Agent@MASPlatform ---
Requirements not linked to a use case and a test case: FR06,FR17
Requirements not linked to a test case: FR10
```

Figure 8-13: Notification about missing requirement information

After receiving the notification, Benjamin investigates the reported missing use cases and test cases for those requirements through the traceability matrix by utilising the query platform as shown in Figure 8-14. He then uses the manipulation platform to modify requirements FR06, FR17, and FR10 by adding the associated use cases and test cases. In order to resolve missing use case information of requirement FR06 and FR17, he connects FR06 to use case UC2 and connects FR17 to use case UC13. For the missing test case that is related to requirement FR10, he links test case TC8.1 to use case UC8. Figure 8-15 presents the traceability matrix after the missing information has been included in the ontology repository.



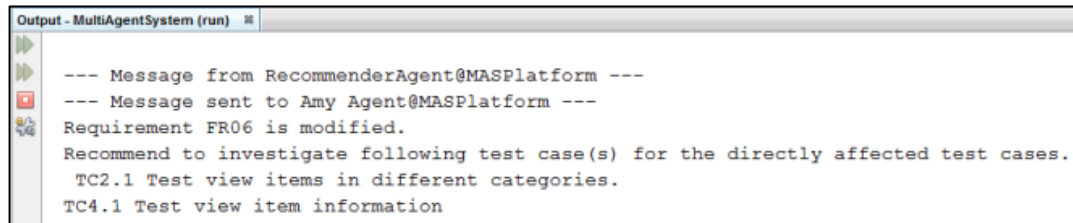
Requirement	UseCase	Class	TestCase
seo:FR01	seo:UC1	seo:DBController	seo:TC1.1
seo:FR02	seo:UC2	seo:Items	seo:TC2.1
seo:FR02	seo:UC2	seo:Items	seo:TC4.1
seo:FR03	seo:UC3	seo:Cart	seo:TC3.1
seo:FR03	seo:UC3	seo:Cart	seo:TC3.2
seo:FR03	seo:UC3	seo:Cart	seo:TC3.3
seo:FR03	seo:UC3	seo:Cart	seo:TC3.4
seo:FR04	seo:UC2	seo:Items	seo:TC2.1
seo:FR04	seo:UC2	seo:Items	seo:TC4.1
seo:FR05	seo:UC4	seo:Cart	seo:TC5.1
seo:FR05	seo:UC4	seo:Cart	seo:TC5.2
seo:FR05	seo:UC4	seo:Cart	seo:TC7.1
seo:FR06			
seo:FR07	seo:UC4	seo:Cart	seo:TC5.1
seo:FR07	seo:UC4	seo:Cart	seo:TC5.2
seo:FR07	seo:UC4	seo:Cart	seo:TC7.1
seo:FR08	seo:UC6	seo:Cart	seo:TC11.1
seo:FR08	seo:UC6	seo:Cart	seo:TC11.2
seo:FR08	seo:UC6	seo:Cart	seo:TC8.1
seo:FR08	seo:UC6	seo:Checkout	seo:TC11.1
seo:FR08	seo:UC6	seo:Checkout	seo:TC11.2
seo:FR08	seo:UC6	seo:Checkout	seo:TC8.1
seo:FR09	seo:UC7	seo:Items	seo:TC9.1
seo:FR10	seo:UC8	seo:Items	seo:TC8.1
seo:FR11	seo:UC6	seo:Cart	seo:TC11.1
seo:FR11	seo:UC6	seo:Cart	seo:TC11.2
seo:FR11	seo:UC6	seo:Cart	seo:TC8.1

Figure 8-14: Excerpt of querying traceability matrix with missing use cases and test cases

Requirement	UseCase	Class	TestCase
seo:FR01	seo:UC1	seo:DBController	seo:TC1.1
seo:FR02	seo:UC2	seo:Items	seo:TC2.1
seo:FR02	seo:UC2	seo:Items	seo:TC4.1
seo:FR03	seo:UC3	seo:Cart	seo:TC3.1
seo:FR03	seo:UC3	seo:Cart	seo:TC3.2
seo:FR03	seo:UC3	seo:Cart	seo:TC3.3
seo:FR03	seo:UC3	seo:Cart	seo:TC3.4
seo:FR04	seo:UC2	seo:Items	seo:TC2.1
seo:FR04	seo:UC2	seo:Items	seo:TC4.1
seo:FR05	seo:UC4	seo:Cart	seo:TC5.1
seo:FR05	seo:UC4	seo:Cart	seo:TC5.2
seo:FR05	seo:UC4	seo:Cart	seo:TC7.1
seo:FR06	seo:UC2	seo:Items	seo:TC2.1
seo:FR06	seo:UC2	seo:Items	seo:TC4.1
seo:FR07	seo:UC4	seo:Cart	seo:TC5.1
seo:FR07	seo:UC4	seo:Cart	seo:TC5.2
seo:FR07	seo:UC4	seo:Cart	seo:TC7.1
seo:FR08	seo:UC6	seo:Cart	seo:TC11.1
seo:FR08	seo:UC6	seo:Cart	seo:TC11.2
seo:FR08	seo:UC6	seo:Cart	seo:TC8.1
seo:FR08	seo:UC6	seo:Checkout	seo:TC11.1
seo:FR08	seo:UC6	seo:Checkout	seo:TC11.2
seo:FR08	seo:UC6	seo:Checkout	seo:TC8.1
seo:FR09	seo:UC7	seo:Items	seo:TC9.1
seo:FR10	seo:UC8	seo:Items	seo:TC8.1
seo:FR11	seo:UC6	seo:Cart	seo:TC11.1
seo:FR11	seo:UC6	seo:Cart	seo:TC11.2
seo:FR11	seo:UC6	seo:Cart	seo:TC8.1

Figure 8-15: Excerpt of querying traceability matrix after adding missing use cases and test cases

Because FR06 is modified by adding associated use case UC2, which links to test cases TC2.1 and TC4.1, the platform sends a message to notify the tester team at the USA site to be aware of this change (Figure 8-16). Amy, a tester who is responsible for these test cases is notified so that she can take appropriate actions such as updating the test cases' details to align them with requirement FR06.



```
Output - MultiAgentSystem (run)
--- Message from RecommenderAgent@MASPlatform ---
--- Message sent to Amy Agent@MASPlatform ---
Requirement FR06 is modified.
Recommend to investigate following test case(s) for the directly affected test cases.
  TC2.1 Test view items in different categories.
  TC4.1 Test view item information
```

Figure 8-16: Message to notify a tester regarding FR06 is updated

## 8.8 Discussion

In the aforementioned section, the functionality of the SEOMAS platforms is demonstrated, specifically in assisting distributed project teams to effectively manage and share software engineering knowledge when they are working on various software development activities throughout the software life cycle. In this section, the active platforms are discussed in relation to the issues defined in Chapter 3 which are:

- Software engineering knowledge management
- Knowledge sharing and reuse
- Communication
- Coordination
- Timely awareness

## **8.8.1 Software Engineering Knowledge Management**

The SEOMAS platforms actively assist project development teams to access and manage software engineering knowledge captured in the Software Engineering Ontology ranging through a series of stages from its creation to its use. In other words, they can provide active support for various knowledge management activities, namely, knowledge capture, knowledge search, knowledge dissemination, and knowledge maintenance.

### **8.8.1.1 Knowledge Capture**

Conventional knowledge capturing approaches require their users to manually extract knowledge from software artefacts, and then formalise the knowledge at the conceptual level. However, a large amount of information is produced during a software development project. Therefore, the manual transformation or mapping of this information into semantically rich form is time-consuming, laborious, tedious, and prone to error. The knowledge capture platform assists team members by providing an automated knowledge capturing approach which is seamlessly integrated into daily software development activities. In other words, with active support from the knowledge capture platform, pertinent software engineering knowledge contained within the software project information is automatically captured in the Software Engineering Ontology knowledge base with minimum need for human intervention. Thus, it could encourage project team members to share their knowledge, thereby improving the software productivity.

Once software project information has been captured and populated in the Software Engineering Ontology, it is available to be used for communicating and sharing among team members, among software agents, and between team members and software agents. This information can be linked to other related information to create the dependencies among them in order to support semantic query or semantic search facilities.

### **8.8.1.2 Knowledge Search**

The query platform is mainly responsible for assisting with the retrieval of software project information captured in the ontology. Project team members can

query or search the semantically-linked software project information to facilitate their work. They can have some knowledge of an issue, rather than precise knowledge about the concepts and relationships defined in the ontology. Put differently, a user specifies the information that he/she needs and then the query platform will formulate the query, retrieve knowledge from the ontology and deliver the result to the user. For instance, when a developer wants to search for the bugs reported to a particular class, he then specifies the class name. The query platform can actively assist him to formulate the query and retrieve information regarding the problem class and its related bugs as well as deliver the result to a developer. Additionally, the query platform utilises the reasoning capability of the ontology to increase the relevance of the search and query results.

#### **8.8.1.3 Knowledge Dissemination**

During software a development project, particularly in a multi-site distributed setting, software team members might not be aware of the existence of certain knowledge or might not be able to find it effectively (e.g., a new member who has just joined a project). The SEOMAS platforms, i.e., the monitoring platform and the manipulation platform, support knowledge dissemination in a proactive manner. The platforms can provide project members with useful information without requiring them to explicitly express their needs. Relevant and timely information, associated with their working context, is delivered to software teams. For instance, they can alert a requirement engineer in regard to a requirement that has missed some information for a certain period or when a bug report is filed, the platforms can suggest a match between the bug and expert. Accordingly, even though team members may not be aware of the existence of certain knowledge or may not be able to find it effectively, the useful and relevant knowledge is still provided to them on a push-based delivery basis.

#### **8.8.1.4 Knowledge Maintenance**

Software development is a knowledge-intensive activity. Once the software systems have been developed and deployed, they are subject to ongoing maintenance to correct failures, adapt to changes in the system's environment, or adapt to changes in users' requirements. The manipulation platform can assist team members to

manage the evolution of software project information captured in the Software Engineering Ontology to reflect the project's development. It offers proactive features to propagate changes of project information to relevant team members including recommendations about the impact of the change, and timely notifications to inform relevant development team members about the change and its impact. The benefits of these features are to help to avoid unintended side effects arising from the change made, and they improve team members' awareness of software evolution in real time; moreover, they can help to maintain consistency among software development artefacts.

### **8.8.2 Knowledge Sharing and Reuse**

As mentioned earlier, software development is knowledge-intensive where knowledge sharing plays an important role in team collaboration. However, in the real working environment, project team members may not have the time or incentive to share their knowledge with others. Accordingly, the active support provided by the SEOMAS platforms can become a key enabler to encourage team members to share their knowledge. The knowledge capture platform provides the automated support to capture knowledge of software project information into semantically rich form that can be subsequently shared and reused by software teams. The query platform can enable project teams to reuse existing knowledge and past experience to resolve software development issues (e.g. knowledge about particular bugs and how they are fixed). This information can help them save time spent on resolving the issue and can bring about software productivity benefits. The manipulation platform assists with the updating of software project information captured in the Software Engineering Ontology, and also provides real-time knowledge sharing regarding the update to relevant team members.

### **8.8.3 Communication**

The SEOMAS platforms are intended to enable effective and efficient communication which is critical in a collaborative software development environment. Software project information is captured according to the software

engineering domain knowledge through the knowledge capture platform. Thus, the semantics of the project information are made to be more explicit so that it enables a meaningful communication which eliminates misinterpretations, misunderstandings, and miscommunications among team members. Furthermore, the platforms facilitate efficient communication that is targeted and timely to keep team members well-informed of the project progress. In other words, relevant and useful information is directed to the team members who need to know about it early enough for them to drive their decision making or to perform appropriate actions within the time constraint. For example, software change propagation and its impact that are sent to relevant team members to be aware of a change made by others, or the notification sent to corresponding members when a deviation or a disruptive event is identified. The effective and efficient communication provided by the SEOMAS platforms can assist team members to reduce traditional communication efforts (e.g., phone calls, emails, online chats, etc.) which are not very conducive to semantic understanding.

#### **8.8.4 Coordination**

Software development projects involve various work dependencies and linkages which need information about others' activities and their coordination. Coordination become more complex as the degree of distribution of the team increases, and the lack of team awareness is a critical factor. The SEOMAS platforms can provide active assistance to improve effective and efficient coordination among project team members. The platforms proactively inform them of other team members' actions in order to maintain real-time team awareness and make them aware of coordination needs to manage work dependencies. For example, when a bug is filed, the platforms can match a bug and a potential expert and then notify him/her about a bug that needs his/her expertise to fix. If several bug reports referring to the same class are filed until their number is above a certain threshold, the platforms proactively inform the class author in order to bring his attention to the need to diagnose the issue. Another example is that the platform notifies the project manager about the issues which have remained unassigned to the fixers for a certain period. In the above examples, it can be seen that the SEOMAS platforms help to

promote effective and efficient coordination within project teams which can bring about a decrease in task resolution time, eliminate redundant tasks, and prevent software defects that compromise the quality of a software system.

## 8.9 Conclusion

This chapter has focused on the active platforms for multi-site software development environments, namely, knowledge capture platform, query platform, monitoring platform, and manipulation platform. These platforms are intended to provide active support to remote project team members to manage and share knowledge effectively throughout the various software development activities in the software life cycle. The practical uses of the platforms are demonstrated through the case study of the development of an online shopping system. The chapter concluded with a discussion about the benefits of using the platforms in accordance with the key issues identified in Chapter 3. In the next chapter, the framework for active Software Engineering Ontology will be evaluated.

## 8.10 References

- Anda, Bente, Kai Hansen, and Gunhild Sand. 2009. "An investigation of use case quality in a large safety-critical software development project." *Information and Software Technology* 51 (12): 1699-1711. doi: <http://dx.doi.org/10.1016/j.infsof.2009.04.005>.
- Deitel, Paul, and Harvey M Deitel. 2011. *Java™ for Programmers*. second ed: Prentice Hall Professional.
- Khatri, Sujata, and RS Chillar. 2011. "Analysis of features affecting testing in object oriented systems." *Analysis* 3 (2): 17-21.
- Leffingwell, Dean, and Don Widrig. 2000. *Managing Software Requirements: A Unified Approach*. Addison-Wesley Professional.

Mefteh, Mariem, Nadia Bouassida, and Hanène Ben-Abdallah. 2014. "Feature model extraction from documented UML use case diagrams." *Ada User* 35 (2): 107.

Natali, Ana Candida Cruz, and RA Falbo. 2002. "Knowledge management in software engineering environments" *Proceedings of the XVI Brazilian Symposium on Software Engineering (SBES'2002)*,

Srinivasan, D, and R Gopaldaswamy. 2006. "Software testing: Principles and practices." *Pearson Education, New Delhi India*.

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.



# **Chapter 9 Evaluation of the Framework for Active Software Engineering Ontology**

## **9.1 Introduction**

In Chapter 5, a conceptual framework for the active Software Engineering Ontology through a Multi-Agent System (SEOMAS) was discussed. In Chapter 6 and Chapter 7, the development of the ontology-based multi-agent approach for capturing knowledge from software project information and the development of the ontology-based multi-agent approach for the Software Engineering Ontology instantiations management were discussed respectively. In Chapter 8, the active platforms for multi-site software development environments were developed as a working prototype to demonstrate the feasibility of using the SEOMAS platforms to assist collaborative team members to manage and share software engineering knowledge effectively throughout various activities in a software development life cycle.

In this chapter, the prototype system is used as proof-of-concept experiments to evaluate the framework for active Software Engineering Ontology. The evaluation is carried out in accordance with a framework for evaluation in design science research addressed by Venable, Pries-Heje, and Baskerville (2012). The chapter starts with the framework solution requirements for active Software Engineering Ontology. The proposed framework will be observed and measured to determine how well it can provide solutions for the research issues. This activity involves comparing the framework requirements and observing results from the use of the prototypes in the demonstration. The quantitative parameters including the time to complete the task, the number of team members involving, and the required number of team members' actions, are used to measure the efficiency of deploying the SEOMAS framework and platforms to assist software development activities. The chapter concludes with a discussion of results, taking an integrated view that is

appropriate for the framework solution requirements.

## **9.2 Framework Requirements**

In Chapter 3, three main research issues were identified and led to the active Software Engineering Ontology framework requirements. The prototype system was developed according to these requirements and these are evaluated in the next section. The framework requirements are:

- Automated Knowledge Capture of Software Project Information
- Software Engineering Ontology Instantiations Management
- Active Platforms for Multi-site Software Development Environments

### **9.2.1 Automated Knowledge Capture of Software Project Information**

The first requirement of the framework for active Software Engineering Ontology is automated knowledge capture of software project information. A software development project produces a large volume of software artefacts. However, these are in syntactic form so their structures are not conducive to an understanding of the semantics, and therefore may create ambiguities. The Software Engineering Ontology was developed to define common sharable software engineering knowledge and to enable knowledge integration in a multi-site software development environment. Software project information can be captured according to the concepts defined in the ontology and subsequently used to clarify any ambiguity in communication and to enable knowledge sharing among software project development teams. Nevertheless, manually capturing knowledge of software project information into conceptualised form according to the concepts defined in the Software Engineering Ontology is a time-consuming, labour-intensive, tedious and prone-to-error task. Therefore, the framework requires automated knowledge capture of software project information that is seamlessly integrated in a software development process in order to help project team members capture software

engineering knowledge with very minimum effort. It is found that the ontology-based multi-agent approach is an appropriate means of achieving the aforementioned requirement. Nonetheless, it is important to prove through the prototypes that the approach conforms to its claims.

### **9.2.2 Software Engineering Ontology Instantiations Management**

Once software development knowledge has been captured in the Software Engineering Ontology, the next requirement is to manage the knowledge which includes accessing and manipulating the information (i.e., add, modify, delete) to reflect software evolution. In order to obtain or manipulate the knowledge captured in the Software Engineering Ontology, project team members need to know exactly the concepts and relationships to which they are referring. However, quite often a person who utilise the ontology may try to resolve an issue but s/he cannot translate it into the exact concepts and relations formed in the ontology. In addition, due to the considerable amount of knowledge captured, it could be possible that software teams are not aware of the existence of certain knowledge in the ontology, so this potentially useful knowledge may be overlooked. Therefore, the framework needs to assist software development teams to obtain the most relevant and precise situational knowledge and project information.

Moreover, software project information is always evolving as a result of modifications to changes in users' requirements, adaptation to changes in the system's environment, and ongoing maintenance to correct failures. Making a single change may affect other software artefacts. Thus, the framework requires effective management of software project information captured in the Software Engineering Ontology by enhancing a software team's real-time awareness regarding software evolution and maintaining consistency among software development artefacts. It is found that the ontology-based multi-agent system is a solution for effective instantiations management. The prototypes demonstrate Software Engineering Ontology instantiations management capacities.

### **9.2.3 Active Platforms for Multi-site Software Development Environments**

The last requirement of the framework for active Software Engineering Ontology is to put the framework into practice to illustrate the benefits of the platforms. The platforms are intended to actively assist multi-site software development teams to effectively manage and share software engineering knowledge throughout the software development life cycle. Project team members connect through the platforms via their user agents. The collaborative agents interact and mediate between the Software Engineering Ontology and team members to enable effective and efficient communication and coordination. Because software development activities are interconnected, the platforms play significant roles in supporting software development activities throughout the software development life cycle.

## **9.3 Prototype Systems Evaluation**

Hevner et al. (2004) consider the evaluation of the designed artefact as an important component of a design science research process. According to a design science research framework process model proposed by Peffers et al. (2007), evaluation is a crucial activity that indicates how effectively and efficiently the artefact provides a solution to the problem. The process of evaluation involves comparing the objectives of a solution with the results obtained from using the designed artefact in the demonstration. In this research, the conceptual framework and the research framework are evaluated in accordance with a framework for evaluation in design science research addressed by Venable, Pries-Heje, and Baskerville (2012) through the prototype system as proof-of-concept experiments. Several scenario experiments based on real case studies in the literature are conducted to evaluate the effectiveness and efficiency of the proposed framework through the implemented prototypes.

It is to be noted that because of the time constraint, the artificial evaluation technique (Venable, Pries-Heje and Baskerville 2012) is chosen to evaluate the proposed framework. The artificial data based on the case studies in the literature is

executed on the real system (prototype system) but the users are not real. Case study is mentioned in (Peffer et al. 2007) as one of the research evaluation methods that can be used to demonstrate the use of the design artefact to solve one or more instances of the problems. The use of the case study is common for evaluating the multi-agent based systems in several researches such as those of (Ossowski et al. 2004; Pomar, López and Pomar 2011; Calyam et al. 2014; Mahesh, Ong and Nee 2007). Therefore, it is selected as the best case to address the four problems articulated in this thesis.

The evaluation engages various use case scenarios to assess the effectiveness and efficiency of the framework through the prototype system. It focuses on evaluating the prototypes according to the following three aspects of the framework requirements:

- Evaluation of Automated Knowledge Capture of Software Project Information
- Evaluation of Software Engineering Ontology Instantiations Management
- Evaluation of Active Platforms for Multi-site Software Development Environments

### **9.3.1 Evaluation of Automated Knowledge Capture of Software Project Information**

In this section, the evaluation of automated knowledge capture of software project information is demonstrated through the case study derived from (Wongthongtham, Dillon and Chang 2011). It is also used to demonstrate the practical use of the SEOMAS framework and the output screenshots are presented in Chapter 6, section 6.9. Table 9-1 describes the bug resolution process mentioned in the case study when the SEOMAS framework is not utilised.

Table 9-1: Bug resolution process described in (Wongthongtham, Dillon and Chang 2011)

No.	Date	Actor	Actions
1.	3 Aug 2009	Richard@Perth	Richard filed a bug report in the project issue tracking system with high priority.
2.	4 Aug 2009	Richard@Perth	Richard filed another bug report with an urgent request hoping to increase its priority and draw greater attention from developers.
3.	4 Aug 2009	Vishay@Bangalore	Vishay came up with a quick fix and added a comment at the end of the report, putting the report into the status of "re-evaluation pending".
4.	11 Aug 2009	Arleno@Shanghai	Arleno filed a duplicate bug which was soon recognized as a repeated report two days later.
5.	15 Aug 2009	Arleno@Shanghai	Arleno discussed with his team members and supervisor, who added comments to the report and directed their concerns back to the Bangalore Lab
6.	17 Aug 2009	Larry@Bangalore	Larry provided another bug fix solution
7.	17 Aug 2009	Michael@Dublin	Michael picked up the fix and pointed out that Larry's fix might produce deadlocks in another related component and suggested reverting back to the first fix.
8.	18 Aug 2009	Larry@Bangalore	Larry fixed the bug based on Michael's instruction
9.	24 Aug 2009	Michael@Dublin	Michael checked the fix and marked the bug report status as "resolved" and closed the bug.
10.	24 Aug 2009	Lisa@Shanghai	Lisa suggested that the latest fix resulted in a connection timeout.
11.	25 Aug 2009	Larry@Bangalore	Larry asked Lisa to explain the affected component
12.	25 Aug 2009	Michael@Dublin	Michael fixed the bug, and explained his fix.
13.	29 Aug 2009	Richard@Perth	Richard closed the bug as "resolved".
<b>Total</b>	<b>27 days</b>	<b>6 actors</b>	<b>13 actions</b>

From Table 9-1, it can be seen that even though the bug was not too complicated and needed only a simple modification to fix the problem, it took 27 days to finalise the resolution which might cause a project delay. Difficulties arose from the lack of common semantics. First, the information related to the bug was dispersed among several software repositories with no links to indicate that they were related to each other. Therefore, the same bug report was filed repeatedly. Second, the bug was initially fixed by developers who had no expertise in this area, resulting in several iterations of invalid fixes. Without the knowledge support to match the bug with the expert, the bug-fixing time could be prolonged. Finally, the inadequate sharing of project information and knowledge, such as the dependencies among software components, can delay the bug fixing. As discussed above, Larry did not know what the affected component was, so he needed someone to clarify this information because there was no available and explicit reference that he could

access.

In order to address the abovementioned issues, software project information (e.g., source code, bug reports, communication threads) should be captured so that software development knowledge becomes conceptualised, organised, and can be semantically linked among related knowledge. The SEOMAS framework can help to automate knowledge capture process by means of the semantic annotation and the ontology population tasks which are seamlessly integrated into the software development process (e.g., version control). Once this software project information has been captured and integrated, it is available for sharing among software project teams to facilitate software development activities or to address project issues by, for example, assisting with a bug resolution process as described in Table 9-2.

Table 9-2: Bug resolution process with supporting from the SEOMAS approach

No.	Date	Actor	Actor Actions	Agent	Agent Actions
1.				VersionControl agent Annotation agent Ontology agent	1. The versioncontrol agent imported a new software project information file into the version control repository. 2. The annotation agent annotated software development artefacts to identify new instances. 3. The ontology agent populated the Software Engineering Ontology with new instances.
2.	3 Aug 2009	Richard@Perth	Before filing a bug report, Richard checked whether the bug had been reported through the query platform	Richard's user agent	Richard's user agent sent a query request to the ontology agent
3.	3 Aug 2009			Ontology agent	The ontology agent retrieved existing bug reports related to the problem class and sent them back to the user agent.
4.	3 Aug 2009	Richard@Perth	Richard filed a new bug report with high priority.		
5.	3 Aug 2009			Recommender agent	The recommender agent 1. identified Michael@

No.	Date	Actor	Actor Actions	Agent	Agent Actions
					Dublin as the most likely person to be able to solve the new filed bug report;  2. attached Michael@Dublin as the potential fixer into the bug report;  3. sent a message to notify Michael@Dublin to draw his attention to the new bug report that may need his expertise to resolve.
6.	3 Aug 2009	Michael@Dublin	Michael received a message to notify him of a new bug report.	Michael's user agent	Michael's user agent translates a message from the recommender agent and display to Michael
7.	3 Aug 2009			Ontology agent	The ontology agent provided Michael with: 1. information about the problem class and its related software components; and  2. history of all previous bugs reported to the problem class and how they were fixed.
8.	4 Aug 2009	Michael@Dublin	1. Michael fixed the bug based on information provided by the ontology agent. 2. Michael marked the bug report status as "resolved".		
9.	4 Aug 2009			Recommender agent	The recommender agent sent a message to notify Richard that the status of the bug had been changed to "resolved".
10.	5 Aug 2009	Richard@Perth	Richard read the message, verified the resolution, and then closed the bug.		
<b>Total 3 days</b>		<b>2 actors</b>	<b>6 actions by real user</b>	<b>6 agents</b>	<b>12 actions by agents</b>
<b>Total number of actions</b>			<b>18 actions</b>		

In Chapter 6, it can be seen that the SEOMAS framework can automatically and transparently assist software teams with the knowledge capture process because it is integrated into the software development activities. Therefore, team members do not have to expend any time or effort on this task. Once the software project



information has been captured, it is accessible and processable by the software agents. They can use this knowledge to facilitate effective and efficient communication and coordination, and to enable knowledge sharing among project team members.

As demonstrated in Table 9-2, the bug resolution process involves bug understanding, bug triage, and bug fixing as well as additional steps to avoid the recurrence of similar bugs in the future. It is considered as one of the most complex activities particularly in a multi-site distributed software development project because it requires significant collaboration of information from various sources (e.g. bug reports, software components, forum discussions) and various stakeholders. From the comparison provided in Table 9-1 and Table 9-2, it is evident that the SEOMAS framework can help multi-site distributed software development teams to resolve the bug issues by improving the effectiveness and the efficiency of communication and coordination as well as enabling knowledge sharing as follows.

1. Before filing a bug report, the ontology agent can help a software developer to locate related bug reports based on their associated concepts defined in the Software Engineering Ontology and its instances. Then s/he can view a list of existing bugs reported to a particular class and determine whether the current bug is a duplicate. In this case, duplicated bug reports could be identified early and avoided. This can reduce the unnecessary information overload and considerably reduce confusion as well as help to prevent tedious conflict.

2. After a bug has been filed, the recommender agent can recommend a person who is most likely able to resolve the bug issue, and sends a message to alert him about the new bug report that potentially needs his expertise to resolve. This can help to match a bug to a potential fixer or consultant in order to avoid the inadequate fixes from someone without expertise with this particular bug. In addition, the recommender agent attaches the potential bug fixer's name to the bug report so that when other developers try to fix the bug (in case of a company's policy that allows only authorised people to change the code), they can directly ask the expert for advice and help.

3. When the bug is being fixed, the ontology agent can provide relevant

information that is necessary for fixing the bug such as the history of bugs reported to the problem class and their resolution, or related software components and their owners. Then the developer can know what dependencies exist and check with relevant people before making a change in order to prevent unintended side effects from a change made.

4. When a developer makes a change to the source code, he is also proactively informed about the components that potentially may be affected by a change. This can reduce unintended side effects from the impact of the bug fixing, and avoid future problems.

5. The recommender agent sends a message to notify the bug reporter as soon as the bug status is changed to “resolved”. The reporter then knows that the issue that he reported has been resolved, so he can verify the solution. Once he is satisfied with the solution, the bug report can be closed. The SEOMAS agents can improve real-time awareness of team members and enable efficient coordination without overloading them.

### **Parameters for Efficiency Measurement**

In the above scenarios, the efficiency of bug resolution by utilising the SEOMAS framework is measured by three parameters, namely, time to complete the task, the number of team members involving in the bug resolution, and the number of team members’ actions.

#### **1. Time to complete the task**

Without the support of SEOMAS, the estimated time that would be taken to resolve a single bug issue is 27 days. However, when SEOMAS is utilised, it takes only three days to fix the same bug. This significant reduction in time is due to the fact that source code artefacts and other software-related project information (e.g., bug reports, archived communications, project documents, etc.) are all captured and can be integrated to generate interconnections among them. The ontology agent and the recommender agent can utilise this interlinked knowledge space to deliver useful and timely information to development teams. The delivered information is also

based on previous historical data in the software project. Information such as a match between a bug and expert, related software components and related bugs, can assist developers to diagnose and fix the bug more effectively and efficiently. Therefore, the response time required to correct failures in order to complete the bug resolution task is reduced.

## **2. The number of team members involving in the bug resolution**

As seen in Table 9-1, six team members are involved in the bug resolution process. Even though the bug is not a complicated one and may require only a simple modification by an expert, without utilising the SEOMAS platform, it goes around across multiple sites which leads to several iterations of inappropriate fixes from someone without expertise in fixing this kind of bug; moreover it unnecessarily prolongs the bug resolution process. With the support from the SEOMAS framework, fewer team members are involved in the bug resolution process because the number of people reporting duplicate bugs can be reduced and the bug can be directly assigned to the appropriate team member who has the expertise required to resolve the issue instead of going around to several people.

## **3. The number of team members' actions**

In the bug resolution scenario without support from the SEOMAS framework, it can be seen that there are a number of unnecessary actions from the team members. For example, personnel are filing duplicated bugs or iteratively fixing the same bug. This is because the information and interactions which relate to the bug are stored in various software artefacts without links between them. When the SEOMAS platform is utilised, the source code is annotated using meta-data that is semantically rich to enable it to be interlinked with other relevant information. Hence, the development artefacts are all related, not independent. Therefore, the ontology agent can help to locate related problems and deliver them to the team members to prevent the same bugs from being reported multiple times. Additionally, it can provide necessary context-relevant information of a problem class to assist the developer to fix the bug. Therefore, the number of team members' actions is decreased from thirteen actions to six actions.

From Table 9-2, it can be seen that the total number of actions with SEOMAS support is higher than without the SEOMAS support in Table 9-1. This is because several additional actions are performed by the SEOMAS agents in order to achieve their goal and to enable team members to perform their tasks more efficiently. These actions include, for instance, the translation between team members and their user agents, identifying expert and recommending useful information about related software components, sending messages sent to relevant team members. However, these actions are autonomously performed by the agents and do not impact on team members' performance.

### **9.3.2 Evaluation of Software Engineering Ontology Instantiations Management**

In this section, the evaluation of Software Engineering Ontology Instantiations Management is demonstrated by using the following scenario derived from a case study in (Lai and Ali 2013, 46-51).

The software organization ALPHA designs, defines and delivers a broad range of IT solutions. The main site (headquarters) of ALPHA is located in Australia, and the offshore sites are located in India and China. Client XYZ contacted ALPHA for the development of an online shopping system. Based on conversations with the client, the analysts and requirements engineer of ALPHA extracted and analysed details about prospective system requirements. As a result, they identified that there would be two modules in the project: one related to services (that is, purchases, order tracking, seller information) and one related to payment (that is, basic payment and authentication mechanisms). Thereafter, members from these teams established a repository to record details of the online shopping system, generated a requirements traceability matrix, sent project requirements to two offshore locations for global software development, and communicated and discussed changes in requirements with other development teams. The client, requirements engineer and project analysts of ALPHA are located in Australia; however, the offshore teams are located in China and India. The Chinese and Indian teams are responsible for the development of the payment and service modules, respectively.

As things become clearer to the client with the passage of time, the client

wants to make a few changes to the payment module of the project. In the initial set of requirements, end users can only use their credit cards for payment purposes in the shopping system. To provide flexibility to end users in making online payments, the client wants to make changes to the payment module by adding the “PayPal” facility so that the end users can have different options available for online payments. For this purpose, the managers of the client’s organization contacted members of the analyst team and discussed the required changes. Mike, the requirements engineer working at the Australian site is responsible for this requirement change. Table 9-3 describes the process of a requirement change.

Table 9-3: Requirement change process described in (Lai and Ali 2013, 46-51)

No.	Date	Actor	Actions
1.	1 February 2013	Mike@Australia	<p>1. Mike extracted details from the project repository about the requirements relating to the payment module and the team responsible for its development.</p> <p>2. He examined the change impact from the requirement traceability matrix and he found that to make change to the Requirement ID 15 by adding payment with PayPal would directly affect Use case 15 and related codes responsible by China team. In addition, other requirements i.e., Requirement ID 3, Requirement ID 6, Use Case 3, Use Case 6, and related codes responsible by India team would be affected too.</p>
2.	2 February 2013	Mike@Australia	Mike made the real change to the Requirement ID 15 by adding additional payment method ‘PayPal’.
3.	2 February 2013	Mike@Australia	Mike discussed with the development team managers Mr. JKL@China site and Mr. ABC@India site about potential impact of the changes via Skype.
4.	3 February 2013	Mr. JKL Developer@China	<p>Mr. JKL looked at the project repository and inspected the change in Requirement ID 15 whether it was actually affected Use case 15 and which source code module might be affected.</p> <p>If so, he discussed the need to update those artefacts with his development teams in order to reflect the change.</p>
5.	3 February 2013	Mr. ABC Developer@India	<p>Mr. ABC looked at the project repository and inspected the change in Requirement ID 15 whether it was actually affected Use case 3, Use case 6 and which source code module might be affected.</p> <p>If so, he discussed the need to update those artefacts with his development teams in order to reflect the change.</p>
6.	5 February 2013	Mike@Australia Developer@China Developer @India	Mike, Chinese and Indian developers manually updated the requirement traceability matrix according to the project repository to make the matrix up-to-date.
<b>Total</b>	<b>5 days</b>	<b>5 actors</b>	<b>9 actions</b>

From Table 9-3, it can be seen that when a change needed to be made to a requirement, the requirements engineer had to inspect the potential impacts of the change and discuss these himself with the relevant team members. He began by manually extracting details from the project repository about the requirements relating to the payment module and the team responsible for its development. He discovered that the development teams at the Chinese and Indian sites would potentially be affected by this change. Unfortunately, this method is prone to error because he might unintentionally overlook some portion of the traceability information. Therefore, some related artefacts might not be traced and adjusted to conform to the change. Furthermore, in the above example, Mike himself had to inform other team members at different sites to make them aware of the change. Lastly, when such changes occurred, team members had to manually update the traceability matrix to reflect these changes which could be costly in terms of time and effort if the changes were enormous. Even though there are some commercial tools that aim at supporting the identification of affected artefacts when there are changes in requirements, they usually do not provide support to ensure that the links and impacted artefacts are properly maintained in a timely fashion (Kannenbergh and Saiedian 2009). This is particularly challenging in distributed settings. Changes can be expected throughout the life cycle of software projects, so maintaining traceability throughout changes to the system is significant important. Table 9-4 presents a possible alternative way to manage the change in a requirement through the SEOMAS framework by utilising its capability to manage software project information captured in the Software Engineering Ontology.

Table 9-4: Requirement change process with support from the SEOMAS approach

No.	Date	Actor	Actor Actions	Agent	Agent Actions
1.	1 February 2013	Mike@Australia	Mike requests to modify Requirement ID 15 by adding additional requirement about PayPal payment through his user agent.		
2.	1 February 2013			Mike's user agent	Mike's user agent translated his request and sent it to the ontology agent.

No.	Date	Actor	Actor Actions	Agent	Agent Actions
3.	1 February 2013			Ontology agent	The ontology agent sent a request for recommendation about the change impact to the recommender agent.
4.	1 February 2013			Recommender agent	The recommender agent: 1. identified potentially impacted requirements by applying the defined change impact rule;  2. identified other potentially impacted software artefacts;  3. sent the recommendations about the change impact to Mike's user agent
5.	1 February 2013	Mike@Australia	Mike examined the change impact recommendation and decided to commit the change.		
6.	1 February 2013			Ontology agent	The ontology agent modified the Requirement ID 15 in the ontology repository.
				Recommender agent	The recommender agent sent messages to notify the developer at China and the developer at India sites to be aware about the change in Requirement ID 15 and its impact to their artefacts.
4.	2 February 2013	-Developer@China  -Developer@India	The China and Indian developers examined the change and its impact to their artefacts.  In case that the change really affected their artefacts, they request to update their artefacts	User agents of : - developer@China - developer@India	1. The user agents of developers at China and India sites translated the messages received from the recommender agent and displayed to their users.  2. They sent requests to the ontology agent to update their artefacts.

No.	Date	Actor	Actor Actions	Agent	Agent Actions
5.	2 February 2013			Ontology agent	The ontology agent updated the instance knowledge according to the change made.
<b>Total</b>	<b>2 days</b>	<b>3 actors</b>	<b>4 actions by real users</b>	<b>5 agents</b>	<b>10 actions by agents</b>
<b>Total number of actions</b>			<b>15 actions</b>		

The above scenarios evaluate the effectiveness of using the SEOMAS framework to manage software project information captured in the Software Engineering Ontology. As shown by comparing Table 9-3 and Table 9-4, the SEOMAS framework can be utilised to improve the effectiveness and efficiency of communicating and coordination as well as enable knowledge sharing in a multi-site software development setting regarding a change in requirements as follows.

1. When a team member requests a change in requirements, the SEOMAS agents collaboratively work to respond to the request. A user agent translates a requirement change request from concepts and relationships defined in the Software Engineering Ontology, and sends it to the ontology agent. The ontology agent then collaborates with the recommender agent to identify the change impact recommendation on other requirements and other artefact types. The recommendation is sent back to the team member for consideration. This can help to prevent unintended side effects from any change made to the instantiations.

2. If a team member decides to make the change, the ontology agent modifies the software project information captured in the Software Engineering Ontology according to the request. Once the change is has been made, the software project information is updated and instantaneously available for sharing among the remote members.

3. The recommender agent proactively sends messages to notify other relevant team members who are potentially impacted, about the change. Then they can be aware of the change made by others at different remote sites and respond in order to coordinate appropriately. In this case, the SEOMAS framework can maintain timely awareness of the geographically dispersed teams effectively and efficiently through proactive and real-time messages.



## **Parameters for Efficiency Measurement**

From the abovementioned efficiency measurement of using the SEOMAS framework to manage a requirement change process, it can be seen that the framework can actively assist team members to complete their software development activities more efficiently by shortening task completion time, reducing the number of team members involved in the tasks and reducing the number of team members' actions that need to be performed in order to complete the tasks.

### **1. Time to complete the task**

Without the SEOMAS framework, the estimated time required to complete the task, including the management of a change in requirements takes five days until all information is updated in the project repository. However, with the support from the SEOMAS framework, it takes only two days which indicates that the SEOMAS framework can reduce the completion time. The main reason is that the recommender agent can proactively assist team members to identify the impact of a requirement change based on the semantic relationship captured in the Software Engineering Ontology. Therefore, they do not have to spend time trying to discover this information manually. Additionally, once the change has been made, the recommender agent proactively sends messages about the change to relevant team members at different remote sites in real-time. Dispersed project members are therefore made aware of the change in a timely manner and can take further action in response to the change within the appropriate time constraint.

### **2. The number of team members involving**

As can be seen in Table 9-3, without utilising the SEOMAS framework, there are five team members involved in the requirement change process. When a requirement ID 15 is changed, Mike, a requirement engineers, must manually inspect the potential impact of the change and spend time to discuss the potential affects with the development team managers at the remote sites (i.e., Mr.JKL at China and Mr.ABC at India sites). The development team managers also need to discuss the need to update those artefacts with their development teams in response to the change. Therefore, even for a single change, several project team members need to be involved, particularly on the change propagation and change impact identification

tasks.

However, with the support from the SEOMAS framework, the number of team members involved in the process can be reduced from five actors to three actors. The main reason is that the software agents can autonomously manage the change propagation and change impact identification task instead of this being done manually by project team members. The recommender agent consults the Software Engineering Ontology to identify which artefacts or modules potentially will be affected and who should be notified about this issue. It can assist Mike to be aware of the impact of the change in order to prevent any unintended side effects. In addition, once the requirement update has been done, the recommender agent proactively propagates the change and its impact to other relevant team members in a timely manner so that they can be aware of the change and coordinate with appropriate actions.

### **3. The number of team members' actions**

In Table 9-3, there are several actions that team members must perform manually in order to manage a change in requirements appropriately. For example, Mike must examine the change impact of the requested requirement change from the requirement traceability matrix. Manual traceability is prone to error and some artefacts might not be identified as potentially being affected by a change. In addition, the change has to be manually propagated to relevant team members. Furthermore, when a team member makes the change, the traceability matrix needs to be updated and this is manually performed by project team members.

However, with support from the SEOMAS framework, the number of team members' actions can be reduced from nine actions to only four actions. Several tedious, laborious and error-prone tasks such as tracing the changed requirements to other artefacts are carried out by the software agents utilising semantic representation and semantic relationship defined in the ontology. In Table 9-4, it is to be noted that the total number of actions is more than those without the SEOMAS framework in Table 9-3. The reason is that there are some additional actions that need to be taken by the SEOMAS agents in order to achieve the goals. For example, the translation between team members and their user agents, messages

sent to relevant team members, etc. However, these actions are autonomously performed by the agents and do not affect team member performance.

### 9.3.3 Evaluation of Active Platforms for Multi-site Software Development Environments

In this section, the evaluation of active platforms for multi-site software development environments through a case study from (Leffingwell and Widrig 2000) is discussed. The use case and test case fragment of traceability are depicted in Figure 9-1.

Relationships - Direct Only	TC1:On/Off Dim Test TC2:Round-trip Message TC3:Superfluous Test	Relationships - Direct Only	TC1:On/Off Dim Test TC2:Round-trip Message TC3:Superfluous Test
UC1: On/Off Dim	▼	☒ SR1: System Clock The system shall	
UC2: Initiate Emergency		SR2: OnLevel Illumination parameter	
UC3: Do Useless Thing		☒ SR3: HOLIS shall support up to 255	▼
		☒ SR4: Message protocol from Control	▼
		SR5: The CCU must have no known	

Figure 9-1: Use case and test case fragment of traceability

A testing team was validating the Home Lighting Automation System (HOLIS) project in order to confirm that the implemented system conformed to the requirements established for it. All actions are described in Table 9-5.

Table 9-5: Scenarios for investigating missing project information (adapted from Leffingwell and Widrig 2000, 354-356)

No.	Date	Actor	Actions
1.	1 November 2013	QA leader@USA	The QA leader found that there were some test cases (i.e., TC1, TC3) that remained unexecuted for a certain period.
2.	1 November 2013	QA leader@USA	The QA leader asked the tester team to check this issue.
3.	2 November 2013	Tester@USA	The tester inspected the issue from the requirement traceability matrix and he also found that there were some requirements that were not linked to any test case (i.e.SR1, SR2, SR5). So he reported this issue to the team leader.

No.	Date	Actor	Actions
4.	3 November 2013	Team leader@ Australia	The team leader checked the traceability matrix and found that these requirements had no associated use case.
5.	3 November 2013	Team leader@ Australia	The team leader contacted the requirement engineer to inspect this issue.
6.	4 November 2013	Requirement engineer@ Australia	The requirement engineer checked the problem and added the missing use cases and test cases for those requirements.
7.	4 November 2013	Requirement engineer@ Australia	The requirement engineer informed the tester about the update of associated use cases and test cases to those requirements.
8.	5 November 2013	Tester@USA	The tester checked the update from requirement traceability matrix and prepared for the further test plans.
<b>Total</b>	<b>5 days</b>	<b>4 actors</b>	<b>8 actions</b>

From Table 9-5, it can be seen that during the testing phase, the testing team at USA site coincidentally discovered missing project information, namely, several requirements that were not linked to any test case. It also led to the discovery that some requirements also did not have an associated use case. If this missing information is not discovered early, it might result in an unsuccessful final software product which does not meet customer needs. This type of situation indicates that only a manual project review might have caused the unintentional error. Furthermore, when the tester found the requirements without test case established, he was not sure about whom to contact regarding this issue. Thus, he directed his issue to the team leader at Australia site. The team leader then directed this issue to the requirement engineering who was responsible for those requirements. Finally, when missing use cases and test cases were added, the requirement engineer himself had to manually inform relevant team members of this change.

The same situation occurs when the SEOMAS platform is utilised to proactively monitor software project information in Table 9-6.

Table 9-6: Proactively monitoring of project information by the SEOMAS platforms

No.	Date	Actor	Actor Actions	Agent	Agent Actions
1.	1 November 2013			Recommender agent	The recommender agent proactively monitored software project information and found that the requirements SR1, SR2, and SR5 had no use case and test case established.

No.	Date	Actor	Actor Actions	Agent	Agent Actions
2.	1 November 2013			Recommender agent	The recommender agent sent a message to notify the requirement engineer regarding this issue
3.	1 November 2013			Requirement engineer's user agent	The requirement engineer's user agent delivered the message to the requirement engineer.
4.	2 November 2013	Requirement engineer@Australia	The requirement engineer checked the problem and sent a request through his user agent to update associated use cases and test cases to those requirements.		
5.	2 November 2013			Requirement engineer's user agent	The requirement engineer's user agent sent a requirement update request to the ontology agent to add associated test cases to those requirements.
6.	2 November 2013			Ontology agent	1) The ontology agent modified the requirement as requested.  2) The recommender agent sent a message to notify the tester about the update.
7.	2 November 2013			Tester's user agent	The tester's user agent delivered the message to the tester.
8.	2 November 2013	Tester@USA	The tester could realise about a set of new test cases that were just linked to those requirements and prepared for the further test plans.		
<b>Total</b>	<b>2 days</b>	<b>2 actors</b>	<b>2 actions by real users</b>	<b>4 agents</b>	<b>7 actions by agents</b>
<b>Total number of actions</b>			<b>9 actions</b>		

In this section, the SEOMAS platforms are evaluated in terms of their effectiveness in providing active support to access and manage software project information and to enable knowledge sharing among multi-site distributed software development teams. As seen in Table 9-6, the monitoring platform can proactively monitor software development project information captured in the Software

Engineering Ontology. When it identifies a deviation or disruptive event that is likely to affect the desired project outcome, it sends a message to notify the person in charge that s/he should investigate and take appropriate action before the actual problem arises. In this example, the requirement engineer is proactively informed about the requirements that are not associated with a use case and test case. He investigates the issue and decides to update the requirements with associated use cases and test cases through the manipulation platform. The platform modifies the instantiations of requirements SR1, SR2, and SR5 as requested and sends messages to notify relevant team members (e.g., tester) to be aware of the changes. Finally, all updated knowledge is promptly made available for sharing among project team members.

### **Parameters for Efficiency Measurement**

In the above scenarios, the efficiency of platforms in assisting software team members to manage and share software engineering knowledge is measured by three parameters, namely, time to complete the task, the number of team members involved in the process, and the number of actions that team members need to perform.

#### **1. Time to complete the task**

Without the SEOMAS support, the estimated time required to complete the task which involves investigating and finding missing project information, takes five days until the test cases added to validate those requirements are realised by the tester team. However, with the support from the SEOMAS, it takes only two days to complete the process which indicates that the SEOMAS platforms can reduce the lengthy process. The reason is that the monitoring platform can proactively monitor software project information and, when it identifies the requirements that have no associated use case and test case for a certain period, it sends a message to notify the requirement engineer about this missing information. Furthermore, when use cases and test cases are updated to the associated requirements, relevant team members at different development sites (e.g., tester) are notified in real time so they can be aware of the update and use this information to continue their work as soon as the information is updated.

## **2. The number of team members involving**

As seen in Table 9-5, there are four team members involving to the process of investigating missing project information which are a QA leader who found remained unexecuted test cases, a tester who found some requirements with no links to test cases, a team leader who discovered that these requirements had no associated use case, and a requirement engineer who updated these missing information. However, with support from the SEOMAS framework, the number of team members involved in this whole process is less because the collaborative agents can perform some actions on behalf of the real users. For instance, the recommender agent can proactively monitor project information by itself; therefore, this action does not need a manual effort from any team member. Furthermore, with the ability of the agents to access the semantic knowledge captured in the Software Engineering Ontology, it is possible to infer the knowledge that is suitable for a given team member or to whom that specific information should be disseminated. When the recommender agent identifies the requirements without associated use cases and test cases, it directly sends a message to notify the requirement engineer who is responsible for those requirements to investigate and update the missing information. In this case, there is no need for the team leader to be involved in this issue. Thus, the number of team members involved can be reduced from four actors to only two actors.

## **3. The number of team members' actions**

In the scenario of investigating missing project information without support from the SEOMAS platform, it can be seen that a number of actions are performed manually by the team members. For example, a tester, a team leader, and a requirement engineer have to trace the missing information using a requirement traceability matrix. If this issue is dismissed or not identified early, more actions may need to be performed in order to fix the defects in the final software product. When missing use cases and test cases are added to the requirements, the requirement engineer himself also has to notify relevant team member. However, with the active support from the SEOMAS platforms, most of the abovementioned actions are performed autonomously by the agents. For instance, software project information is proactively monitored and missing information is automatically identified by the

recommender agent. It also manages the tasks of notifying the update of the requirements to other relevant team members. As a result, the actions that need to be performed by the real users can be reduced from eight actions to only two actions.

## **9.4 Discussion of Results**

In this section, results from the previous section are discussed in an integrated view. The discussion is categorised into three sections corresponding to the framework requirements as follows:

- Automated Knowledge Capture of Software Project Information
- Effective Software Engineering Ontology Instantiations Management
- Active Platforms for Multi-site Software Development Environments

### **9.4.1 Automated Knowledge Capture of Software Project Information**

Software project information is produced throughout the software development life cycle. In particular, source code is considered critical and is central to software development. The software maintenance is usually made at the source code level and it can be prone to ambiguity in communication. The SEOMAS framework has the capacity to annotate semantic source code for automatic knowledge acquisition and management. With the support of the SEOMAS framework, the manual process of capturing knowledge from source code which is tedious, laborious, and prone to error can be replaced and performed automatically by the software agents. The time spent on this process is also considerably decreased. In addition, the capturing process is transparent to project teams as it is integrated into daily software development activities. Once the software project information has been captured according to the software engineering domain knowledge defined in the ontology, it eventually becomes meaningful so that project team members can use it to facilitate shared understanding and to clarify any ambiguity in communication. Moreover, software development knowledge captured in the



ontology is well-organised and relevant concepts are interlinked. Related software project information will not appear isolated, but will be in a large group of related information that can be readily and easily accessed. This captured knowledge is also in machine-understandable form. Consequently, the software agents can read and process them with the guidance of the Software Engineering Ontology in order to assist team members to address software development issues. For example, when a new bug report is filed, the agent can autonomously retrieve a full record of mappings of the previously reported bugs to the problem class and the information about the developer who fixed those bugs. It then processes this information to identify the potential bug fixer and proactively informed him/her to have awareness of the new bug that needs his/her expertise to fix. In this case, the framework can enable relevant team members to be notified of others' actions and be aware of the coordination needed so that they can coordinate while the development is still underway.

In section 9.3.1, the prototypes are evaluated to observe and measure how effective and efficient the SEOMAS framework is in supporting project software teams to address software development issues, namely, bug resolution activity. It can be seen that the real power of the automated knowledge capture is realised when the knowledge is used to assist collaborative team members to communicate and coordinate effectively and efficiently. As a consequence, software development activities are performed more efficiently as bug issues can be resolved more quickly, fewer team members are needed for the tasks, and there is a reduction in the number of team members' actions that need to be performed in order to complete the tasks.

#### **9.4.2 Effective Management of Software Engineering Ontology Instantiations**

Once the software project information has been captured in the ontology repository, it is important that this knowledge be managed effectively so that it can be easily accessed and manipulated. As mentioned earlier, in order to obtain the knowledge captured, software teams need to explicitly request and know exactly which concepts and relations that they are referring to; otherwise, they may not be able to obtain the required knowledge. Furthermore, software systems are subject to maintenance and evolution, so changes are inevitable in all stages of a software

development project. One of the main challenges of software evolution is the management of dependencies that exist between software development artefacts.

In section 9.3.2, the prototypes are evaluated to observe and measure how effective and efficient the SEOMAS framework for Software Engineering Ontology instantiations management is in assisting project team members to manage software evolution in regard to a change in requirements. It can be seen that the agents can help to manipulate instance knowledge in order to reflect the change and manage dependencies among related artefacts. The collaborative agents can make use of the software engineering knowledge captured in the ontology to provide recommendations about the impacts of the change and deliver them to relevant team members in a proactive manner without an explicit request from them. In this case, group awareness and timely awareness are well-maintained to avoid the conflicts or inconsistencies of software artefacts resulting from the change made. Thus, the results have shown that it is efficient to improve software team's productivity by shortening time spent to manage a change in requirements, reducing the number of team members involved in the task, and reducing the number of team members' actions that need to perform in order to complete the task.

### **9.4.3 Active Platforms for Multi-site Software Development Environments**

Software development comprises various knowledge-intensive tasks that require collaborative project teams to manage and share software development knowledge effectively. This is particularly so in multi-site software development environments where team members are geographically dispersed, and inadequate communication and coordination are the main factors that can hinder the success of a software project. The SEOMAS framework provides active support to software teams that collaborate and interact with each other to manage and share knowledge through the SEOMAS platforms, namely, knowledge capture platform, query platform, monitoring platform, and manipulation platform that cover various development activities in the software life cycle. They support the management and sharing of software engineering knowledge throughout a series of stages including knowledge capture, knowledge search, knowledge dissemination, and knowledge maintenance.

In section 9.3.3, the prototype results demonstrate the potential and the advantages of the platforms. Once the software project information has been captured in the ontology through the knowledge capture platform, it can be proactively monitored by the monitoring platform to identify any potential deviation or disruptive event which may lead to scenarios such as poor quality of the final software product, project delay and budget overrun. When the unusual event is identified, the notification is sent to the corresponding person to diagnose the issue before an unexpected event occurs. The query platform can be used to assist team members to investigate the issue. If the software project information needs to be modified, the platforms also facilitate the manipulation of instantiations to reflect the change. It can be seen that the platforms play an important role in enabling active knowledge sharing by providing relevant and situational knowledge without requiring team members to explicitly express their needs. In addition, because the Software Engineering Ontology is a comprehensive ontology covering all aspects of software engineering, the platforms can utilise this knowledge to facilitate remote team members throughout the various phases in the software life cycle. The results have shown that the SEOMAS platforms help to promote effective and efficient communication and coordination and to enable effective knowledge sharing within remote project teams. Accordingly, it results in a decrease in task resolution time, eliminate redundant tasks, and help to avoid software defects that compromise the quality of a software system.

## **9.5 Conclusion**

In this chapter, the active Software Engineering Ontology framework requirements are presented. They consist of: i) automated knowledge capture of software project information; ii) Software Engineering Ontology instantiations management; and iii) active platforms for multi-site software development environment. The prototypes are used as proof-of-concept experiments to evaluate the proposed framework. The evaluation is carried out in accordance with the evaluation framework for design science research. The proposed framework is examined and evaluated in terms of how well it can provide solutions for the

research issues. Three quantitative parameters - time to complete the task, number of team members involved, and number of team members' actions - are used to measure the efficiency of the framework in facilitating software development activities and to address software development issues. The chapter is then concluded with the discussion in an integrated perspective according to the framework requirements.

## 9.6 References

- Calyam, Prasad, Lakshmi Kumarasamy, Chang-Gun Lee, and Fusun Ozguner. 2014. "Ontology-based semantic priority scheduling for multi-domain active measurements." *Journal of Network and Systems Management* 22 (3): 331-365.
- Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. "Design science in Information Systems research." *MIS Quarterly* 28 (1): 75-105.
- Kannenbergh, Andrew, and Hossein Saiedian. 2009. "Why software requirements traceability remains a challenge." *CrossTalk The Journal of Defense Software Engineering* 22 (5): 14-19.
- Lai, Richard, and Naveed Ali. 2013. "A requirements management method for global software development." *Advances in Information Sciences (AIS)* 1 (1): 38-58.
- Leffingwell, Dean, and Don Widrig. 2000. *Managing Software Requirements: A Unified Approach*. Addison-Wesley Professional.
- Mahesh, M., S. K. Ong, and A. Y. C. Nee. 2007. "A web-based multi-agent system for distributed digital manufacturing." *International Journal of Computer Integrated Manufacturing* 20 (1): 11-27. doi: 10.1080/09511920600710927.
- Ossowski, Sascha, Josefa Z Hernandez, María Victoria Belmonte, Jose Maseda, Alberto Fernández, Ana García-Serrano, Francisco Triguero, Juan Manuel Serrano, and José Luis Pérez-De-La-Cruz. 2004. "Multi-agent systems for

decision support: A case study in the transportation management domain." *Applied Artificial Intelligence* 18 (9-10): 779-795.

Peppers, Ken, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. "A design science research methodology for Information systems research." *Journal of Management Information Systems* 24 (3): 45-77.

Pomar, J, V López, and C Pomar. 2011. "Agent-based simulation framework for virtual prototyping of advanced livestock precision feeding systems." *Computers and electronics in agriculture* 78 (1): 88-97.

Venable, John, Jan Pries-Heje, and Richard Baskerville. 2012. "A comprehensive framework for evaluation in design science research." In *Design Science Research in Information Systems. Advances in Theory and Practice*, 423-438. Springer.

Wongthongtham, P., T. Dillon, and E. Chang. 2011. "State of the art of community-driven software engineering ontology evolution" *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, doi: 10.1109/DASC.2011.170.

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.

# Chapter 10 Recapitulation and Future Work

## 10.1 Introduction

Multi-site distributed software development projects take place in an environment where development teams are dispersed throughout various remote sites. The drivers of this multi-site setting are the advantage of economic factors and the availability of global talent pools. The main purpose is to optimise human resources in order to develop higher quality products at a lower cost. Nonetheless, besides the benefits, this remote environment has created several additional challenges in terms of communication, coordination, group awareness, etc. These challenges can have great impact on the budget and project schedule as well as the quality of the final product. To maintain collaborative work through effective communication and coordination, the Software Engineering Ontology was developed to clarify the software engineering concepts and project information and to enable knowledge sharing among the dispersed teams. However, the nature of the Software Engineering Ontology is still passive being similar to that of existing ontologies in the Web, some challenges regarding knowledge assimilation and knowledge dissemination arise. As a consequence, it is important to have a mechanism to make the Software Engineering Ontology active so that it can provide effective support to facilitate and assist software development team members with software engineering knowledge when they are working on software development projects. In this thesis, SEOMAS, the comprehensive framework for active Software Engineering Ontology is presented. The framework consists of multiple components and approaches. Each of them were explained and developed in the previous chapters.

In the next section, the issues that have been addressed in this thesis are recapitulated. In section 10.3, the contributions made by this thesis are discussed. In section 10.4, areas for future work are identified, and section 10.5 concludes the chapter.

## 10.2 Recapitulation

The Software Engineering Ontology (Wongthongtham et al. 2009; Wongthongtham et al. 2005) was proposed to facilitate effective communication and coordination among project team members within a multi-site distributed software development environment. It provides shared understanding and consistent communication to remote team members by allowing them to navigate shared software engineering knowledge and to query the semantically-linked project information. Nevertheless, with its passive structure being similar to that of existing ontologies in the Web, some challenges regarding knowledge assimilation and knowledge dissemination arise.

The main objective of this thesis is to develop a comprehensive framework to make the Software Engineering Ontology active. This will enable it to effectively facilitate various tasks and assist its users to manage and share software engineering knowledge when they are working in software development projects particularly in multi-site distributed software development environments that require a high degree of collaboration and knowledge sharing. The issues addressed in this thesis are as follows.

### 1) Propose the framework of active Software Engineering Ontology

The passive structure of Software Engineering Ontology produces challenges associated with effectively obtaining or manipulating software engineering knowledge captured in the ontology. Therefore, the framework of an active Software Engineering Ontology is proposed in this thesis. Active Software Engineering Ontology refers to the Software Engineering Ontology which is equipped with the active support that can be used to proactively facilitate and assist its users with software engineering knowledge when they are working on software development projects. This active support is aimed at covering both phases/stages associated with the ontology deployment stage, namely, knowledge assimilation and knowledge dissemination.

- 2) Propose an approach to automate semantic annotation and ontology population in order to capture semantics of software project information into the Software Engineering Ontology.

Given the huge volume of software project information associated with a software project, the manual approach that relies primarily on software teams' processing to map software project information to the concepts defined in the Software Engineering Ontology, is not practical. An extensive manual capture of large software project information can be an extremely time-consuming, laborious, tedious, and error-prone task. Therefore, there is a need to have a specific area of research regarding knowledge assimilation to automate semantic capturing of software project information. The proposed approach automates the semantic annotation and ontology population process in order to identify software engineering knowledge from software project information and then populates the captured knowledge in the ontology knowledge base.

- 3) Propose an approach to effectively access and manage software engineering knowledge captured in the Software Engineering Ontology

Software engineering knowledge and software project information are captured and organised according to concepts and relations specified by the Software Engineering Ontology. However, due to the large amount of knowledge captured, it could be possible that software teams are not aware of the existence of certain knowledge in the ontology or even if they are, they might not be able to obtain or manipulate it effectively. Therefore, certain useful knowledge might be dismissed. The proposed approach can assist team members by making it quicker and easier for them to perform several time-consuming tasks such as searching for relevant information to address project development issues, locating an expert, analysing the impact of a change in software artefacts as well as propagating the change and its impact to relevant team members, and proactively monitoring particular software project information to identify potential deviation. Therefore, even though software teams may not be aware of the existence of the knowledge, the proposed approach can help them to locate and deliver the knowledge



required. The proactive knowledge delivery can reduce the laborious task of having to search for explicit, useful knowledge. Put succinctly, it can enable the right knowledge to be delivered to the right people who need to have it at the right time.

4) Propose the active platforms for multi-site software development environments

Software development is a knowledge and collaborative-intensive process where its success depends on the effectiveness and efficiency to manage and share software engineering knowledge among project team members. This is particularly critical in a multi-site distributed software development context where the collaboration and knowledge sharing are affected by physical and temporal distances. The passive structure of software Engineering Ontology results in some limitations to promote effective knowledge management for knowledge sharing such as manual knowledge capture and passive knowledge distribution. Therefore, there is a need for active platforms that can proactively support software team members to manage and share software engineering knowledge effectively in order to facilitate remote collaborative work. In addition, because software development is a knowledge-intensive and complex activity, the quality of a software product depends mainly on the quality of the software process which is the result of activities carried out during the software development process. Therefore, the proposed active platforms that can provide support to multi-site software project teams to work on various development activities throughout the software life cycle are valuable.

5) Evaluate the proposed framework

The aim of this thesis is to develop a comprehensive framework for active Software Engineering Ontology. There are currently no effective frameworks that can be successfully adopted to make the Software Engineering Ontology active so that software team members can gain useful support from deploying it particularly within a multi-site software development project. Therefore, in this thesis, the prototype system, a

realisation of the proposed framework, is used as a proof-of-concept and is evaluated based on several existing case studies found in the literature.

In the next section, the contributions made by this thesis are summarised.

## **10.3 Contribution of the Thesis**

The state-of-the-art survey on the existing literature presented in Chapter 2 is one of the contributions of this research. The comprehensive, extensive, and recent survey is conducted specifically to provide the necessary background and context, to discover the shortcomings of current approaches, and to determine the issues to be investigated in this research. The literature has been reviewed and classified according to the following groups:

The contributions of this thesis to the existing body of knowledge are as follows.

### **10.3.1 Contribution 1: Current State-of-the-art Research**

The state-of-the-art survey on the existing literature presented in Chapter 2 is one of the contributions of this research. The comprehensive, extensive, and recent survey is particularly conducted to provide necessary background and context and to discover the shortcomings of current approaches and to determine the issues to be investigated in this research. The literatures have been reviewed and classified into the following groups:

- Ontology-based semantic annotation
- Ontology-based multi-agent systems
- Assistive systems for software engineering

### **10.3.2 Contribution 2: Conceptual Framework**

In Chapter 5, the SEOMAS framework has been proposed to provide active support to assist software development team members with software engineering knowledge when they are working on software development projects. The main components of the proposed framework are the Software Engineering Ontology and the multi-agent system. The Software Engineering Ontology is an underlying knowledge representation of software engineering knowledge that enables all team members working in a multi-site environment to have a common understanding of the software development project. However, it does not possess a degree of autonomy or the capacity for dynamic adaptation to any changes to the situation such as the capturing of new software project information, proactively delivering useful information without the user's explicit request, or alerting team members to an unusual event that might change the project plan. In other words, it still heavily relies on the user's effort to manage such situations although the Software Engineering Ontology is in use. The agent-based technology can be integrated with the ontology to provide the autonomous and flexible features. The basic software engineering processes such as analysis, design, implementation, evaluation are used and integrated with the Agent Unified Modelling Language (AUML) in order to provide the complete development processes for the framework of active Software Engineering Ontology.

In recent literature, several ontology-based multi-agent systems have been proposed as presented in Chapter 2. To the best of our knowledge, the existing literature does not propose any effective framework that incorporates a multi-agent approach with the ontology designed for multi-site software development to assist distributed software project teams to manage and share software engineering knowledge when they carry out software development activities throughout the software development life cycle.

### **10.3.3 Contribution 3: A Systematic Approach to Capture Semantics of Software Project Information**

The third contribution of this thesis is the development of a systematic approach to capture the semantics of software project information which is

seamlessly integrated into the daily software development process in order to avoid extra effort from project team members. The approach is presented in Chapter 6. It consists of two main processes, namely, semantic annotation and ontology population. The semantic annotation process focuses on extracting software engineering knowledge from software project information in order to identify new instances of the ontological concepts according to the Software Engineering Ontology. The ontology population process focuses on instantiating the ontology knowledge base with new instances resulting from the semantic annotation process.

Once the software project information has been captured, it is in machine-readable form so that software agents are able to autonomously understand this knowledge. Therefore, the agents can use this knowledge to provide useful information to distributed project teams in order to clarify any ambiguity resulting from remote communication, to address major software development issues, and to facilitate effective and efficient coordination.

Several research studies regarding ontology-based semantic annotation have been introduced in the literature presented in Chapter 2. To the best of our knowledge, the existing literature does not propose any effective approach to automate the semantic annotation of software project information and populate it into the ontology knowledge base during the software development process.

In brief, the salient features of this approach are highlighted as follows:

- It proposes the semantic annotation process utilising the Software Engineering Ontology to provide software engineering knowledge to software development artefacts. The Software Engineering Ontology has been developed to facilitate common understanding among project team members within a multi-site distributed software development environment. It is a comprehensive ontology covering all aspects of the software engineering domain.
- It proposes the ontology population process to extend the ontology knowledge base with new instances identified during the semantic annotation process. The advantage is that the ontology has been expressed in Web Ontology Language (OWL). OWL is very expressive and the relations

between classes can be formally defined based on description logics. Accordingly, knowledge can be captured by utilising logic reasoning. In other words, hidden knowledge can be inferred by formulating logic expressions even though it is not explicitly defined in the ontology.

- The proposed approaches to capture semantic of software project information including semantic annotation and ontology population are automated and seamlessly integrated into the software development process (i.e., version control) in order to avoid extra effort from project team members.

#### **10.3.4 Contribution 4: A Systematic Approach to Manage Software Engineering Ontology Instantiations.**

The fourth contribution of this thesis is the development of the approach to manage Software Engineering Ontology instantiations presented in Chapter 7. The set of Software Engineering Ontology management instantiations includes knowledge retrieval and instance knowledge manipulation. The knowledge retrieval operation includes query, search, and proactive monitoring of software project information in order to identify the possibility of a deviation before an issue actually occurs. A proactive notification is provided to corresponding team member on a push-based delivery mode. On the other hand, the instance knowledge manipulation operation includes adding, modifying, and deleting instance knowledge as well as identifying the potential impact of the change made to the instantiations based on the relationship defined in the Software Engineering Ontology. Notifications are propagated to relevant team members to make them aware of the change. In a multi-site software development environment, communication and coordination are critical challenges because of physical and temporal distances. Team awareness with respect to any change made by other members at different sites is important. Therefore, proactive and timely knowledge delivery to avoid any confusion and integration risks can help to reduce the challenge presented by distance.

To the best of our knowledge, the existing literature does not propose any effective approach for the management of Software Engineering Ontology instantiations.

In brief, the salient features of this approach are highlighted as follows.

- It helps software development teams to obtain useful and situational knowledge that they may not be aware of or cannot find effectively.
- It provides a proactive software project information monitoring service in order to identify any possibility of encountering deviation before an actual issue occurs.
- It proposes a proactive notification feature to alert the corresponding team member in a push-based delivery mode.
- It not only manipulates the ontology instantiations, but also identifies the potential impact of the change made to the instantiations based on the associated relationship defined in the Software Engineering Ontology. Notifications are propagated in a timely manner to relevant team members to make them aware of the change that has been made.

### **10.3.5 Contribution 5: Active Platforms for Multi-site Software Development Environments**

The fifth contribution of this thesis is that it proposes the development of active platforms for multi-site software development environments aimed at assisting multi-site distributed software development teams to manage and share software engineering knowledge throughout the software development life cycle. Details of these platforms are presented in Chapter 8. Software development activities and their artefacts are interconnected. The work or a change in one activity may have an effect on the work in other activities. Therefore, software development team members need support across various development activities or phases.

To the best of our knowledge, the existing literature does not propose any effective platforms that provide active assistance to project team members, who are geographically distributed, to manage and share software engineering knowledge required for various software development activities in a software development life cycle.

In brief, the salient features of the proposed active platforms for multi-site software development environments are highlighted as follows.

- They provide assistance to manage and share software project information captured in the Software Engineering Ontology throughout a series of stages in knowledge management including knowledge capture, knowledge search, knowledge dissemination, and knowledge maintenance.
- They provide an integrated collection of services that can proactively facilitate software engineering activities during the various phases of the software life development cycle by applying knowledge management practice.
- They provide a mechanism for maintaining timely group awareness in order to manage work dependencies in a multi-site distributed software development environment by means of instant message notification. This addresses the major challenges regarding remote communication and coordination imposed by physical and temporal distances.
- They play an important role in enabling effective knowledge sharing by providing relevant and situational knowledge without requiring team members to explicitly express their needs. Relevant and timely information, associated with their working context, is delivered to software teams.

### **10.3.6 Contribution 6: Prototype Implementation and Evaluation**

The prototype system that realises the SEOMAS conceptual framework is implemented and presented in Chapters 6-8. Software agents are the main components that provide active support by interacting with and mediating between the Software Engineering Ontology and its users. The Java Agent Development Framework (JADE) is employed as the main development tools/environments for the SEOMAS prototype.

According to the framework evaluation presented in Chapter 9, several experiments based on real case study scenarios in the literature are simulated to evaluate the effectiveness and efficiency of the proposed approach through the

implemented prototype system. Three aspects of the framework solution requirements are evaluated. These are:

- Automated Knowledge Capture of Software Project Information
- Software Engineering Ontology Instantiations Management
- Active Platforms for Multi-site Software Development Environments

The experiment findings show that the SEOMAS prototype system is a realised software platform that can assist software development teams to acquire useful software engineering knowledge in order to manage software development activities or issues throughout the software development life cycle. It can help to shorten task completion time, to reduce the number of team members involved in the tasks, and to reduce the number of team members' actions that need to be performed in order to complete the tasks. It contributes to the literature by providing a reference for the implementation of the SEOMAS conceptual framework.

## **10.4 Future work**

In this thesis, the SEOMAS framework is introduced as an important area of work for making the Software Engineering Ontology active so that it can be used to facilitate and assist software team members with relevant software engineering knowledge when they are working on software development projects. Nonetheless, due to resource restrictions on this research, there are some limitations and potential enhancements that can be addressed and marked as future work in order to improve and extend the functionality of the proposed framework. In the following, the challenges or areas of improvement of the current work are discussed as the future work.

### **10.4.1 Future Work Focusing on the Framework Enhancement**

The scalability of the SEOMAS framework has not been fully examined hence it can be enhanced particularly focusing on the number of team members and



system complexity. Other framework enhancement for future work are presented as follows.

- In this thesis, the SEOMAS framework focuses only on the utilisation of the Software Engineering Ontology as a domain ontology. However, in the future work, application ontologies, which are ontologies engineered for a specific use or a particular application (Malone and Parkinson 2010) can be integrated into the current framework. The benefit of using application ontologies is to model concepts that are required to support software applications being developed in order to facilitate domain crossing. For example, if the aim of a software project is to develop an accounting information system, an ontology pertaining to the accounting domain can be integrated into the SEOMAS framework for better quality of semantic annotation or knowledge retrieval.
- The SEOMAS framework can be extended to capture the semantics of other types of software artefacts. The extension can cover the semantic annotation of both structured information (e.g., UML diagrams, issue tracking, commit data) and unstructured information (e.g., requirement documents, bug reports, forum discussion).
- The Enrichment and interlinking tasks of semantic annotation process can be extended to include additional ontologies such as Meaning of a Tag (MOAT) (Passant and Laublet 2008), Bug and Enhancement Tracking Language (BAETLE)<sup>1</sup>, vCard (Iannella 2001), etc. The purpose is to enrich the semantic description of software project information elements by reusing standard and well-known shared vocabularies wherever possible. The adoption of these ontologies can enable reusability and facilitates interoperability between different applications (Ashraf, Hussain and Hussain 2012).
- In this thesis, during the semantic annotation process, the *owl:sameAs* relationships are created by manually linking the terms of the annotated project information resources with their corresponding entry in DBpedia. However, this interlinking task can be improved by utilising duplicate

---

<sup>1</sup> <https://code.google.com/archive/p/baetle>

detection algorithms and frameworks such as Silk (Volz et al. 2009). Silk allows a developer to specify the types of RDF links that should be identified between data sources and the conditions that have to be met in order to establish the interlinking. Besides Silk, Swoosh (Benjelloun et al. 2009), or Duke<sup>2</sup> can also be considered for facilitating the interlinking task with other relevant datasets.

#### **10.4.2 Future Work on the Intelligent System Enhancement**

The area of further study and development to extend the functionality of the SEOMAS framework towards an intelligent system are as follows.

- The recommender agent's capability can be enhanced by means of incorporating with various recommendation algorithms such as:
  - The collaborative filtering approaches which produce a recommendation based on the similarity between users (Deng et al. 2011).
  - The content-based filtering techniques which deliver a recommendation that resembles the ones that a specific user formerly preferred (Adomavicius and Tuzhilin 2005).
  - Hybrid recommender systems that combine the aforementioned approaches in order to improve performance and resolve the problems associated with certain approaches (Burke 2007).
  - Semantic-based recommendation systems that integrate the semantic knowledge in their processes and their performances are based on a knowledge base (Gao, Yan and Liu 2008).

The integration of the abovementioned recommendation techniques can help to enhance the quality of the recommendations generated by the recommender agent and will provide a more comprehensive personalised service.

---

<sup>2</sup> <https://github.com/larsga/Duke>

- Intelligence mechanisms can be applied to the SEOMAS agents in order to incorporate rational behaviour and enhance the performance of each individual agent. They can help to increase the ability to learn from and adapt to the new environment and other agents. Therefore, the integration of these mechanisms into SEOMAS will offer a truly autonomous and more adaptable framework. Examples of the intelligence mechanisms are:
  - Machine learning methods such as supervised learning, unsupervised learning, and reinforcement learning, can be embedded in the SEOMAS agents to increase the ability to discover a problem solution on their own. They can enable the agents to act more proactively in a dynamic environment.
  - Data mining techniques (e.g., case-based reasoning, decision trees, and Bayesian networks) can be applied in the SEOMAS agents to enhance the performance of the decision making.
- The reasoning mechanism of the SEOMAS framework proposed in this thesis is based on the existing reasoning mechanism via the ontology inference. However, the reasoning capability of the agents can be enhanced by integrating other types of reasoning techniques such as rule-based knowledge. For example, Semantic Web Rule Language (SWRL) can be applied to the framework as a rule-based inference engine to determine how to deduce knowledge based on the semantics defined in the ontology and the domain-heuristic rules.
- The SEOMAS framework can be enhanced by increasing the ability of the user agents to monitor users' activities in order to obtain working context information. This information can then be analysed and processed to identify context similarity. As a consequence, the recommendations can be generated based on reusing related experiences that other team members have had in the past in a similar context.

## 10.5 Conclusion

In this chapter, the work that has been carried out in this thesis to address the identified research issues is recapitulated. The contributions made to the literature through this thesis are outlined. This is followed by a brief description of several research directions for future work to extend the proposed framework developed in this thesis.

The work that has been undertaken in this thesis has been published extensively as a part of proceedings in peer-reviewed international journals and conferences. A complete list of the publications arising from this thesis is given at the beginning of the thesis and some selected publications are included in the Appendix B at the end of the thesis.

## 10.6 References

- Adomavicius, G., and A. Tuzhilin. 2005. "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions." *Knowledge and Data Engineering, IEEE Transactions on* 17 (6): 734-749. doi: 10.1109/tkde.2005.99.
- Ashraf, Jamshaid, Omar Khadeer Hussain, and Farookh Khadeer Hussain. 2012. "A framework for measuring ontology usage on the Web." *The Computer Journal*. doi: 10.1093/comjnl/bxs134.
- Benjelloun, Omar, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. 2009. "Swoosh: a generic approach to entity resolution." *The VLDB Journal* 18 (1): 255-276. doi: 10.1007/s00778-008-0098-x.
- Burke, Robin. 2007. "Hybrid Web Recommender Systems." eds Peter Brusilovsky, Alfred Kobsa and Wolfgang Nejdl, 377-408. Springer Berlin / Heidelberg.

- Deng, Yong, Zhonghai Wu, Huayou Si, Hu Xiong, and Zhong Chen. 2011. "A collaborative filtering approach to making recommendations based on ontology in the movie domain." *Energy Procedia* 13: 228-236. doi: 10.1016/j.egypro.2011.11.036.
- Gao, Q., J. Yan, and M. Liu. 2008. "A semantic approach to recommendation system based on user ontology and spreading activation model" *2008 IFIP International Conference on Network and Parallel Computing*, doi: 10.1109/NPC.2008.74.
- Iannella, R. 2001. Representing vCard objects in RDF/XML. W3C Note. Accessed October 11, 2016, <https://www.w3.org/TR/vcard-rdf>.
- Malone, James, and Helen Parkinson. 2010. Reference and application ontologies. Accessed November 3, 2016, <http://ontogenesis.knowledgeblog.org/295>.
- Passant, Alexandre, and Philippe Laublet. 2008. "Meaning Of A Tag: A collaborative approach to bridge the gap between tagging and Linked Data." In *The WWW 2008 Workshop Linked Data on the Web (LDOW 2008), Beijing, China*.
- Volz, Julius, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. 2009. "Silk-A link discovery framework for the Web of data." In *The 2nd Workshop about Linked Data on the Web (LDOW2009), Madrid, Spain*, 20 April 2009.
- Wongthongtham, P., E. Chang, T.S. Dillon, and I. Sommerville. 2009. "Development of a software engineering ontology for multi-site software development." *IEEE Transactions on Knowledge and Data Engineering* 21 (8): 1205-1217. doi: 10.1109/TKDE.2008.209.
- Wongthongtham, Pornpit, Elizabeth Chang, Chan Cheah, and Tharam S Dillon. 2005. "Software engineering sub-ontology for specific software development." In *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA, Maryland, USA, April 7, 2005*. 27-33. IEEE. doi: 10.1109/SEW.2005.4.

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.

# Appendix A Additional Information of the Case Study of an Online Shopping Software Development

The case study of an online shopping software development presented in Chapter 7 and Chapter 8 is taken from (Gupta 2013). However, some additional information (e.g., requirement dependencies, test cases, new requirements) is added for the purpose of evaluating the SEOMAS framework as presented in this appendix.

Requirement ID	Description	Dependencies	Use Cases	Classes	Test Cases
FR01	The users shall be able to view the categories on the application's home page.		UC1 Home Page	DBController	TC1.1 Test view the categories on the application's home page.
FR02	The users shall be able to view items in different categories.	Requires FR01	UC2 View items	Items	TC2.1 Test view items in different categories.
FR03	The users shall be able to add items to the cart.	Requires FR02	UC3 Add Items to Cart	Cart	TC3.1 Test valid item information TC3.2 Test number of items is invalid TC3.3 Test not enough item TC3.4 Test out of stock item
FR04	The users shall be able to view more information about an item before adding it to the cart.	Refines FR03	UC2 View items		TC4.1 Test view item information
FR05	The users shall be able to view the shopping cart.		UC4 View Cart		TC5.1 Test view shopping cart successfully TC5.2 Test no items in the cart
FR06	The users shall be able to browse through the available items.	Requires FR05	Leave blank for testing missing use case.		
FR07	The users shall be able to view the items added to the cart.	Requires FR03	UC4 View Cart		TC7.1 Test view items added to the cart successfully
FR08	The users shall be able to check out with the current items in the cart.		UC 6Check Out		TC8.1 Test check out with the current items in the cart

Requirement ID	Description	Dependencies	Use Cases	Classes	Test Cases
FR09	The users shall be able to continue shopping after adding items in the cart.	Requires FR03	UC7 Continue Shopping		TC9.1 Test continue shopping
FR10	The users shall be able to delete items from the cart.	Requires FR06	UC8 Delete items		Leave blank for testing missing test case.
FR11	The users shall be able to check out items only when there are items in the shopping cart.		UC6 Check Out	Check-Out Cart	TC11.1 Test check out successfully TC11.2 Cart is empty
FR12	The users shall login or register using the user authentication form before placing order.		UC9 Place Order (include UC10)	Place-Order DBController	TC12.1 Test valid user authentication TC12.2 Test user has not login TC12.3 Test user has not registered
FR13	The users shall not be able to login or register if the information is incomplete or invalid.	Refines FR12	UC10 Log in UC11 Registration	User Authentication	TC13.1 Test valid username and password TC13.2 Test valid user name but invalid password TC13.3 Test invalid username TC13.4 Test complete user registration TC13.5 Test incomplete user registration information
FR14	The users shall place an order by completing the information in the order form.	Requires FR12	UC9 Place Order	Place-Order DBController	TC14.1 Test complete order form TC14.2 Test incomplete order form
FR15	The administrator shall be able to view all the users' information that completes the order form and the checkout process.	Requires FR14	UC12 View Database		TC15.1 Test view users' information
FR16	The administrator shall be able to add new items to the list of shopping items.		UC13 Manage Items	Items	TC16.1 Test manage item successfully TC16.2 Test invalid item
FR17	The administrator shall be able to modify/update an item's price and description.		Leave blank for testing missing use case.		
FR18	The administrator shall be able to delete items from the main page of the shopping-cart		UC13 Manage Items	Items	TC16.1 Test manage item successfully TC16.2 Test invalid



Requirement ID	Description	Dependencies	Use Cases	Classes	Test Cases
	application.				item
FR19	The administrator shall be able to view the entire history of the checked-out items.	Requires FR14	UC12 View Database		TC19.1 Test view the entire history of checked-out items
FR20	The administrator shall be able to view the entire history for the users who successfully complete the checkout process.	Requires FR14	UC12 View Database		TC20.1 Test view the entire history of users who successfully complete the checkout process.
FR21	The user shall complete the captcha when log in for the purpose of differentiating a human being from the computer program.	Requires FR12	UC14 Verify captcha	UserAuthentication	TC25.1 Test verify captcha successfully TC25.2 Test invalid captcha

**Note** FR21 is added to verify the reasoning capability over the Software Engineering Ontology as described in Chapter 8.