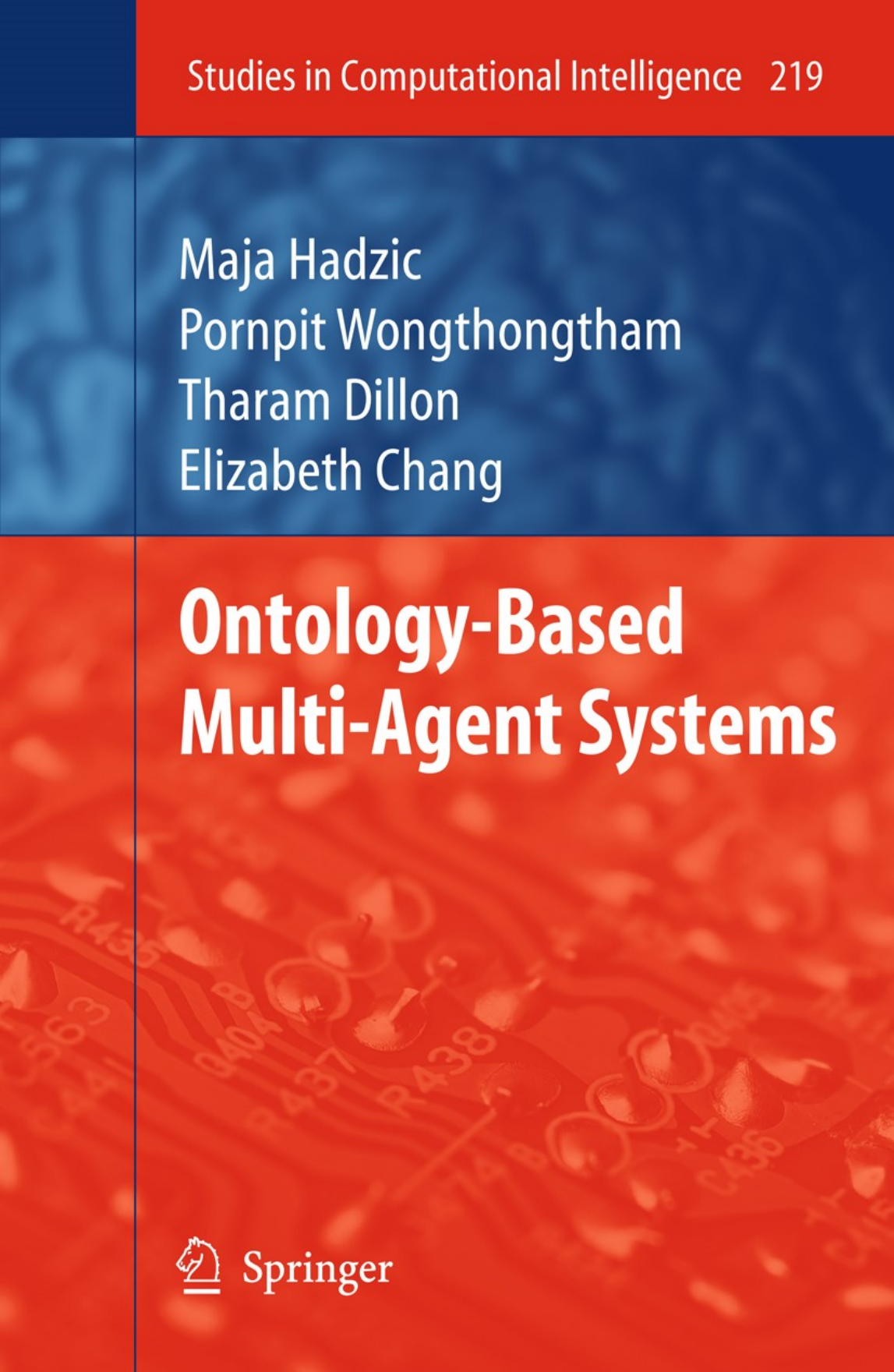
Studies in Computational Intelligence 219

Maja Hadzic
Pornpit Wongthongtham
Tharam Dillon
Elizabeth Chang

Ontology-Based Multi-Agent Systems



Springer

Studies in Computational Intelligence, Volume 219

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 197. Mauro Birattari
Tuning Metaheuristics, 2009
ISBN 978-3-642-00482-7

Vol. 198. Efrén Mezura-Montes (Ed.)
Constraint-Handling in Evolutionary Optimization, 2009
ISBN 978-3-642-00618-0

Vol. 199. Kazumi Nakamatsu, Gloria Phillips-Wren, Lakhmi C. Jain, and Robert J. Howlett (Eds.)
New Advances in Intelligent Decision Technologies, 2009
ISBN 978-3-642-00908-2

Vol. 200. Dimitri Plemenos and Georgios Miaoulis *Visual Complexity and Intelligent Computer Graphics Techniques Enhancements,* 2009
ISBN 978-3-642-01258-7

Vol. 201. Aboul-Ella Hassanien, Ajith Abraham, Athanasios V. Vasilakos, and Witold Pedrycz (Eds.)
Foundations of Computational Intelligence Volume 1, 2009
ISBN 978-3-642-01081-1

Vol. 202. Aboul-Ella Hassanien, Ajith Abraham, and Francisco Herrera (Eds.)
Foundations of Computational Intelligence Volume 2, 2009
ISBN 978-3-642-01532-8

Vol. 203. Ajith Abraham, Aboul-Ella Hassanien, Patrick Siarry, and Andries Engelbrecht (Eds.)
Foundations of Computational Intelligence Volume 3, 2009
ISBN 978-3-642-01084-2

Vol. 204. Ajith Abraham, Aboul-Ella Hassanien, and André Ponce de Leon F. de Carvalho (Eds.)
Foundations of Computational Intelligence Volume 4, 2009
ISBN 978-3-642-01087-3

Vol. 205. Ajith Abraham, Aboul-Ella Hassanien, and Václav Snášel (Eds.)
Foundations of Computational Intelligence Volume 5, 2009
ISBN 978-3-642-01535-9

Vol. 206. Ajith Abraham, Aboul-Ella Hassanien, André Ponce de Leon F. de Carvalho, and Václav Snášel (Eds.)
Foundations of Computational Intelligence Volume 6, 2009
ISBN 978-3-642-01090-3

Vol. 207. Santo Fortunato, Giuseppe Mangioni, Ronaldo Menezes, and Vincenzo Nicosia (Eds.)
Complex Networks, 2009
ISBN 978-3-642-01205-1

Vol. 208. Roger Lee, Gongzu Hu, and Huaikou Miao (Eds.)
Computer and Information Science 2009, 2009
ISBN 978-3-642-01208-2

Vol. 209. Roger Lee and Naohiro Ishii (Eds.)
Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2009
ISBN 978-3-642-01202-0

Vol. 210. Andrew Lewis, Sanaz Mostaghim, and Marcus Randall (Eds.)
Biologically-Inspired Optimisation Methods, 2009
ISBN 978-3-642-01261-7

Vol. 211. Godfrey C. Onwubolu (Ed.)
Hybrid Self-Organizing Modeling Systems, 2009
ISBN 978-3-642-01529-8

Vol. 212. Viktor M. Kureychik, Sergey P. Malyukov, Vladimir V. Kureychik, and Alexander S. Malyoukov
Genetic Algorithms for Applied CAD Problems, 2009
ISBN 978-3-540-85280-3

Vol. 213. Stefano Cagnoni (Ed.)
Evolutionary Image Analysis and Signal Processing, 2009
ISBN 978-3-642-01635-6

Vol. 214. Been-Chian Chien and Tzung-Pei Hong (Eds.)
Opportunities and Challenges for Next-Generation Applied Intelligence, 2009
ISBN 978-3-540-92813-3

Vol. 215. Habib M. Ammari
Opportunities and Challenges of Connected k-Covered Wireless Sensor Networks, 2009
ISBN 978-3-642-01876-3

Vol. 216. Matthew Taylor
Transfer in Reinforcement Learning Domains, 2009
ISBN 978-3-642-01881-7

Vol. 217. Horia-Nicolai Teodorescu, Junzo Watada, and Lakhmi C. Jain (Eds.)
Intelligent Systems and Technologies, 2009
ISBN 978-3-642-01884-8

Vol. 218. Maria do Carmo Nicoletti and Lakhmi C. Jain (Eds.)
Computational Intelligence Techniques for Bioprocess Modelling, Supervision and Control, 2009
ISBN 978-3-642-01887-9

Vol. 219. Maja Hadzic, Pornpit Wongthongtham, Tharam Dillon, and Elizabeth Chang
Ontology-Based Multi-Agent Systems, 2009
ISBN 978-3-642-01903-6

Maja Hadzic, Pornpit Wongthongtham, Tharam Dillon, and Elizabeth Chang

Ontology-Based Multi-Agent Systems

Springer

Dr. Maja Hadzic
Digital Ecosystems and
Business Intelligence Institute
GPO Box U1987
Perth, Western Australia 6845
Australia

Dr. Pornpit Wongthongtham
Digital Ecosystems and
Business Intelligence Institute
GPO Box U1987
Perth, Western Australia 6845
Australia

Prof. Tharam Dillon
Digital Ecosystems and
Business Intelligence Institute
GPO Box U1987
Perth, Western Australia 6845
Australia

Prof. Elizabeth Chang
Digital Ecosystems and
Business Intelligence Institute
GPO Box U1987
Perth, Western Australia 6845
Australia

ISBN 978-3-642-01903-6 e-ISBN 978-3-642-01904-3

DOI 10.1007/978-3-642-01904-3

Studies in Computational Intelligence ISSN 1860949X

Library of Congress Control Number: Applied for

Typeset & *Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

During the last two decades, the idea of Semantic Web has received a great deal of attention. An extensive body of knowledge has emerged to describe technologies that seek to help us create and use aspects of the Semantic Web. Ontology and agent-based technologies are understood to be the two important technologies here. A large number of articles and a number of books exist to describe the use individually of the two technologies and the design of systems that use each of these technologies individually, but little focus has been given on how one can design systems that carryout integrated use of the two different technologies.

In this book we describe ontology and agent-based systems individually, and highlight advantages of integration of the two different and complementary technologies. We also present a methodology that will guide us in the design of the integrated ontology-based multi-agent systems and illustrate this methodology on two use cases from the health and software engineering domain.

This book is organized as follows:

- **Chapter I**, Current issues and the need for ontologies and agents**,** describes existing problems associated with uncontrollable information overload and explains how ontologies and agent-based systems can help address these issues.
- **Chapter II**, Introduction to multi-agent systems**,** defines agents and their main characteristics and features including mobility, communications and collaboration between different agents. It also presents different types of agents on the basis of classifications done by different authors.
- **Chapter III,** Introduction to ontology**,** defines ontologies, their origins, properties, characteristics and representations of ontology models. It also explains difference between ontologies and knowledge bases, and between ontology design and data modeling. Different ways of classifying of ontologies is also discussed in this chapter.
- **Chapter IV,** Design approaches for multi-agent-based systems, introduces a number of different methodologies and approaches to the design of agent-based systems. Each methodology takes a unique perspective and focuses on a specific aspect of the system.
- **Chapter V,** Ontology design approaches, discussed a number of different methodologies used in the ontology design. We have chosen to discuss a set of the most frequently used methodologies. Interestingly, this set consists of complementary ontology-design approaches.

- **Chapter VI**, Significance of ontologies, agents and their integration, highlights the numerous advantages and applications of ontologies and agent-based systems. It also explains synergetic effects of integration of the two technologies.
- **Chapter VII**, Design methodology for integrated systems - Part I (ontology design)**,** unifies selected features of the different ontology design methodologies described in Chapter V into a single methodology. The Part I methodology consists of five steps, and each step and its sub-steps are explained in this chapter.
- **Chapter VIII**, Design methodology for integrated systems - Part II (multi-agent system design), unifies selected features of the different agent design methodologies described in Chapter IV into a single methodology. Additionally, it incorporates use of the ontologies into the multi-agent system. The Part II methodology consists of five steps, and each step and its sub-steps are explained in this chapter.
- **Chapter IX**, Notations for the integrated ontology and multi-agent system design**,** describes notations that can be used to model ontologies (class, generalization, property, restriction, association and instances) and agent-based systems (role, communication and agent).
- **Chapter X**, Architecture of the integrated ontology and multi-agent system, discusses the ontology server including ontology repositories, evolution and management. It also describes the agent platform used to build the intelligent agents that will operate on the basis of the developed ontology.
- **Chapter XI**, Case study I: Ontology-based Multi-agent System for Human Disease Studies, describes the use of the methodology described in Chapters VII and VIII to design a ontology-based multi-agent system for the retrieval of human disease information. It also gives a number of illustrative examples on the use and significance of the resulting system.
- **Chapter XII**, Case study II: Ontology-based Multi-agent System for Software Engineering Studies, describes the use of the methodology described in Chapters VII and VIII to design a ontology-based multi-agent system to carry out software engineering studies. Practical uses of this system are also discussed.
- **Chapter XIII**, Potential applications of integrated ontology-based multi-agent systems**,** we briefly mentioned potential application of the integrated systems in the collaborative environments, intelligent information retrieval and data mining studies.

Acknowledgements

We express our sincere thanks to Ms Bruna Pomella for proofreading the book, Mr Darshan Dillon for helping us in development of notation for modeling of agent-based systems and Mr Behrang Zadjabbari for formatting the book.

Contents

1 Current Issues and the Need for Ontologies and Agents……. 1
1.1 Introduction…… 1
1.2 Information Variety……... 1
1.3 Hindrances to Successful Information Retrieval……... 2
1.3.1 Distributed and Heterogeneous Information without Semantics…… 3
1.3.2 Underlying Knowledge Base Is Not Available…….. 4
1.3.3 Autonomous, Heterogeneous and Dynamic Information Resources…… 4
1.4 Information Retrieval in Science……... 4
1.5 Search Engines……..... 7
1.6 Web Semantics……..... 9
1.7 Agents for Dynamic Information Retrieval……... 11
1.8 Ontologies for Intelligent Information Retrieval……... 12
1.9 Conclusion……..... 12
References……..... 12

2 Introduction to Multi-Agent Systems…… 15
2.1 Introduction…… 15
2.2 Agent Definition…… 15
2.3 Agent's Environment……. 16
2.4 Agent's Characteristics…….. 17
2.5 Internal Data Structure of Agents…….. 18
2.6 Mobile Agents…… 19
2.7 Dependency Relationships between Agents…….. 22
2.8 Agent's Communication…… 24
2.8.1 Communication Parties……... 24
2.8.2 Communication Place……. 24
2.8.3 Communication Time……. 24
2.8.4 Communication Languages…… 25
2.8.4.1 Desired Features of Agent Communication Languages…… 25
2.8.4.2 Agent Communication Languages…….. 26
2.8.4.3 Ontologies for Agent Communication…… 28
2.9 Collaborative Problem-Solving Process in Multi-Agent Systems… 29
2.10 Different Types of Agents…… 30

2.10.1 Haag's Classification..... 30
2.10.2 Dillenbourg et al.'s Agents Classification..... 31
2.10.3 Maes's Classification..... 32
2.10.4 Classification According Agent's Functions..... 33
2.11 Conclusion..... 34
References..... 35

3 Introduction to Ontology..... 37
3.1 Introduction..... 37
3.2 Ontology Origins..... 37
3.3 Ontology Definition..... 38
3.4 Ontology Commitments..... 40
3.5 Ontology Community..... 41
3.6 Generalization/Specialization of Ontologies..... 41
3.7 Properties of Ontologies..... 43
3.8 Characteristics of Ontology Models..... 44
3.9 Representation of Ontology Domain..... 47
3.10 Ontology Design Versus Data Modelling..... 47
3.11 Ontology Versus Knowledge Base..... 49
3.12 Classification of Ontologies..... 51
3.12.1 Degree of Formality..... 52
3.12.2 Degree of Granularity..... 53
3.12.3 Level of Generality..... 53
3.12.4 Amount, Type and Subject of Conceptualization..... 54
3.12.5 Expressiveness of Ontologies..... 56
3.13 Conclusion..... 58
References..... 59

4 Design Approaches for Multi-Agent-Based Systems..... 61
4.1 Introduction..... 61
4.2 Agent Design Criteria..... 61
4.3 Agent Design Methodologies..... 62
4.3.1 Belief, Desire and Intention (BDI) Approach..... 62
4.3.2 Gaia..... 63
4.3.3 Agent UML..... 64
4.3.4 Agents in Z..... 65
4.3.5 DESIRE..... 66
4.3.6 Cassiopeia..... 67
4.3.7 Multi-Agent System Engineering..... 69
4.3.8 TROPOS..... 70
4.3.9 Prometheus..... 71
4.4 Conclusion..... 72
References..... 74

5 Ontology Design Approaches…….. 75
5.1 Introduction…… 75
5.2 Ontology Design Criteria……... 75
5.3 Ontology Design Methodologies……... 77
5.3.1 Knowledge Engineering Methodology……... 77
5.3.2 DOGMA…… 82
5.3.3 TOVE Methodology……..…… 83
5.3.4 METHONTOLOGY……... 86
5.3.5 SENSUS Methodology……... 87
5.3.6 DILIGENT Methodology……... 89
5.4 Conclusion……. 90
References……..... 90

6 Significance of Ontologies, Agents and Their Integration……... 93
6.1 Introduction…… 93
6.2 Advantages of Ontologies…….. 93
6.2.1 Ontologies for Data Semantics……... 93
6.2.2 Ontologies as Basis for Knowledge Sharing…….. 94
6.2.3 Ontologies as Basis for Knowledge Representation……... 98
6.2.4 Ontologies as Basis for Knowledge Management…….. 98
6.2.5 Ontologies for Intelligent Information Retrieval…… 99
6.2.6 Ontologies for Mediation…… 99
6.2.7 Ontologies for Natural Language Applications…….. 100
6.3 Advantages of Agent-Based Systems…… 101
6.3.1 Agents are Autonomous…….. 102
6.3.2 Agents Support Computational Intelligence……... 103
6.3.3 Distributed Mobile Agent-Based Computing……. 103
6.3.4 Agents Collaboration and Cooperation in Their Activities..... 105
6.3.5 Agents Support Automated Service Discovery…….. 105
6.4 Ontology and Agent-Based Systems Complementing Each Other... 106
6.4.1 Problem Decomposition……. 106
6.4.2 Locating and Retrieving of Information……. 107
6.4.3 Agent Communication…… 107
6.4.4 Information Analysis and Manipulation……. 108
6.5 Conclusion…… 109
References…….. 109

7 Design Methodology for Integrated Systems - Part I (Ontology Design)..…… 111
7.1 Introduction…… 111
7.2 Generalization and Conceptualization of the Domain……... 111
7.2.1 Ontology Communities……... 111
7.2.2 Purpose of the Ontology…… 112
7.2.3 Ontology Domain……... 113
7.2.4 Application of the Ontology……... 114
7.3 Aligning and Merging Ontologies……. 115

7.3.1 Identify Suitable Ontologies for Reuse…….. 115
7.3.2 Define Merging and Alignment Tool……. 116
7.3.3 Import the Source Ontologies……. 116
7.3.4 Identify Ontology Correspondences……... 117
7.3.5 Align and/or Merge Ontologies…….. 117
7.4 Formal Specification of Conceptualization……... 118
7.4.1 Ontology Concepts……. 119
7.4.2 Relationships between Concepts…… 119
7.4.3 Groups of Related Concepts……... 119
7.5 Formal Specification of Ontology Commitments…….. 120
7.5.1 Identify the Ontology Commitments…….. 120
7.5.2 Formalize the Commitments…….. 121
7.5.3 Identify Reusable Knowledge Components……... 121
7.6 Ontology Evaluation…….. 122
7.6.1 Ontology Design Quality Evaluation…….. 122
7.6.2 Ontology Usability Evaluation……... 123
7.7 Overview of the Ontology Development……... 123
7.8 Conclusion……. 125
References……. 126

8 Design Methodology for Integrated Systems - Part II (Multi-Agent System Design)…… 127
8.1 Introduction…… 127
8.2 Overview of the Agent Methodology……..…….. 127
8.3 Classification of Agents According to Their Responsibilities…….. 128
8.3.1 Establish Intuitive Flow of Problem Solving, Task and Result Sharing……. 129
8.3.2 Identify Corresponding Agent Functions……... 129
8.3.3 Identify Corresponding Agent Types……. 130
8.4 Identify the Need for an Ontology to Support Agent's Intelligence..... 131
8.4.1 Problem Decomposition……... 131
8.4.2 Information Retrieval…….. 132
8.4.3 Agent Communication…… 132
8.4.4 Information Analysis and Manipulation……. 132
8.4.5 Meaningful Information Presentation……. 133
8.5 Define Agent's Collaborations…….. 133
8.5.1 Establish Efficient Organization of Agents…… 134
8.5.2 Establish Correspondence between Agent Types and Their Organization…….. 134
8.6 Construction of Individual Agents……. 135
8.6.1 Identify Required Agent Components…… 136
8.6.2 Construct Various Agents……... 136
8.7 Protect the System by Implementing Security Requirements……... 137
8.7.1 Identify Security Requirements…….. 137
8.7.2 Implement the Requirements…….. 138

8.8 Conclusion…… 140
8.9 Advantages of the Onto-Agents Methodology…….. 141
References…….. 142

9 Notations for the Integrated Ontology and Multi-Agent System Design…… 143
9.1 Introduction…… 143
9.2 Modelling Notations…….. 144
9.2.1 Ontology Modelling Notations……... 144
9.2.1.1 Class Notation……... 148
9.2.1.2 Generalisation Notation…… 149
9.2.1.3 Property Notation…….. 150
9.2.1.4 Restriction Notation…….. 154
9.2.1.5 Associated Class Notation…… 155
9.2.1.6 Ontology Instances Notation…… 156
9.2.2 Agent Modelling Notations…… 157
9.2.2.1 Role Notation…… 158
9.2.2.2 Communication Notation……. 158
9.2.2.3 Agent Notation…….. 159
9.3 Diagrams…… 160
9.4 Conclusion……. 163
References…….. 163

10 Architecture of the Integrated Ontology and Multi-Agent System… 165
10.1 Introduction…….. 165
10.2 Ontology Server……... 166
10.2.1 Ontology Repositories…… 169
10.2.2 Ontology Evolution……. 170
10.2.3 Ontology Management……... 173
10.3 Agent Platform…… 173
10.4 Conclusion……... 177
References…….. 177

11 Case Study I: Ontology-Based Multi-Agent System for Human Disease Studies……. 179
11.1 Introduction……. 179
11.2 Generalization and Conceptualization of the Medical Domain…... 179
11.2.1 Community Associated with the Ontology.…… 180
11.2.2 Purpose of the Ontology..……... 180
11.2.3 Ontology Domain……..…… 181
11.2.4 Application Based on the Designed Ontology……... 182
11.2.5 Top-Level Hierarchy of GHDO……..... 182
11.3 Aligning and Merging Existing Medical Ontologies…….. 184
11.3.1 Identifying Ontologies Suitable to Be Reused…… 184
11.3.2 Define Alignment and Merge Tool…… 184
11.3.3 Import Source Ontologies……..... 185

11.3.4 Identify Correspondences between Different Ontologies... 185
11.3.5 Align and Merge Ontologies..... 185
11.4 Formal Specification of Human Disease Domain Conceptualization..... 185
11.4.1 Ontology Concepts..... 186
11.4.2 Relationships between Concepts..... 186
11.4.3 Lexons..... 186
11.4.4 Relationships between Lexons..... 187
11.4.5 Groups of Related Lexons..... 188
11.5 Formal Specification of Human Disease Ontology Commitments..... 189
11.5.1 Identify Intra- and Inter-Commitments..... 189
11.5.2 Formalize the Ontology Commitments..... 190
11.5.3 Identify Reusable Knowledge Components..... 190
11.6 Human Disease Ontology Evaluation..... 191
11.7 Classification of Agents According to their Responsibilities within the Human Disease Information Retrieval System..... 191
11.7.1 Establish Intuitive Flow of Problem Solving, Task and Result Sharing..... 191
11.7.2 Identify Corresponding Agent Functions..... 191
11.7.3 Identify Corresponding Agent Types..... 191
11.8 Identify the Need for Human Disease Ontology to Support Agents' Intelligence..... 193
11.8.1 Problem Decomposition and Task Assignments..... 193
11.8.2 Information Retrieval..... 195
11.8.3 Agent Communication..... 195
11.8.4 Information Analysis and Manipulation..... 197
11.8.5 Meaningful Information Presentation..... 198
11.9 Define Agent's Collaboration within the Intelligent Human Disease Information Retrieval System..... 198
11.9.1 Establish Efficient Organization of Agents..... 198
11.9.2 Establish Correspondence between Agent Types and Their Organization..... 199
11.9.3 Query Processing and Information Integration within GHMS..... 200
11.10 Construction of Individual Agents of the Human Disease Information Retrieval System..... 202
11.10.1 Identify Required Agents' Components..... 202
11.10.2 Construct Various Agents..... 205
11.11 Protect the Human Disease Information Retrieval System by Implementing Security Requirements..... 205
11.11.1 Identify Security Requirements..... 205
11.11.2 Implement the Security Requirements..... 205
11.12 Examples of Use of the Intelligent Human Disease Information Retrieval System..... 207
11.12.1 Example 1: Help Physician to Identify Disease..... 208

11.12.2 Example 2: Support Physician to Choose Disease Treatments…… 210
11.12.3 Example 3: Help Patients and General Public to Prevent a Disease…….. 210
11.12.4 Example 4: Help Medical Researchers to Identify Disease Causes…….. 211
11.12.5 Example 5: Help Medical Researcher Study Complex Diseases…… 213
11.13 Conclusion……. 215
References……. 215

12 CASE STUDY II: Ontology-Based Multi-Agent System for Software Engineering Studies……. 217
12.1 Introduction…….. 217
12.2 Generalization and Conceptualisation of the Software Engineering Domain…… 217
12.2.1 Purpose of the Software Engineering Ontology……. 217
12.2.2 The Context of Software Engineering Domain Knowledge……. 218
12.2.3 Community……. 221
12.2.4 Application Based on the Software Engineering Ontology……. 223
12.3 Formal Specifications of Software Engineering Domain Conceptualisation…… 234
12.3.1 SE Ontology Concepts…… 234
12.3.2 SE Ontology Relations and Constraints…….. 235
12.3.3 SE Ontology Instances…… 240
12.4 SE Ontology Validation……... 243
12.4.1 Validation of Communications……... 243
12.4.2 Validation of Knowledge Sharing…….. 246
12.4.3 Validation of Knowledge Management…….. 250
12.5 Classificaiton of Agents……... 256
12.6 Need of the Software Engineering Ontology to Support Agent's Intelligence…… 257
12.7 Agent's Collaborations…… 257
12.8 Construction of Individual Agents……... 261
12.8.1 User Agents…… 261
12.8.2 Safeguard Agents…… 262
12.8.3 Ontology Agents……. 262
12.8.4 Decision Making Agents.....…… 263
12.9 Practical Uses……... 264
12.10 Conclusion……. 269
References……. 269

13 Potential Applications of Ontology-Based Multi-Agent Systems……..... 271
13.1 Overview……..…… 271
13.2 Collaborative Environments……..…….. 271
13.3 Information Access and Retrieval……... 272
13.4 Data Mining……. 272
13.5 Open Issues……. 273

Chapter 1
Current Issues and the Need for Ontologies and Agents

1.1 Introduction

The internet had become a major source of information in many knowledge domains. The general public uses Google predominantly to obtain in-formation pertaining to a variety of knowledge domains. Generally, users will have different access to, and understanding of, the results they obtain from their 'Google' search. As Google is not built to separate authoritative from dubious in-formation sources, users may have to rely on specialized search engines.

The large volume of published information that is being accounted for is an additional problem that complicates the search. For example, biomedical re-searchers may use PubMed which is a service of the U.S. National Library of Medicine that includes over 16 million citations from life science journals for biomedical articles going back to the 1950s. Using the PubMed search engine, the user receives a list of journals related to the given keyword. It is then left to the user to read each journal individually and to try to establish links within this in-formation. This would be easy if the journal list consisted of a small number of journals. But the journal list usually consists of thousands of journals, and medical researchers usually do not have time to go through these results thoroughly. There is a high chance that some important information will be omitted.

There is a need to design an intelligent search engine that performs searches not only on keywords, but also on the meaning of the information. The search engine would go through the available information, understand this infor-mation, and select highly relevant information; moreover, it would link this infor-mation and present it in a meaningful format to the users.

In this chapter, we will briefly introduce the technologies underpinning such meaningful representation of information and the use of an intelligent and suppor-tive retrieval approach. We examine current issues related to information representation, information access and information retrieval on the web. We will introduce the meaning of web semantics and the role of ontologies and agent tech-nologies in the creation of semantically rich environments.

1.2 Information Variety

The number of information resources and of active users of this information is increasing each day. The severity of the problem that this poses is obvious when

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 1–13.
springerlink.com

one considers the total amount of stored information. In 2006, 161 exabytes (108 terabytes) of information were created or replicated worldwide. This exceeds the total amount of information created or replicated during the last 5,000 years (Brodie 2007). We are constantly drawing in information. Adding content to the web is quite an easy task and access to the web is uncomplicated and fast. The available information covers numerous knowledge domains and various disciplines such as astronomy, geology, biology, computer science, history, the arts, religion etc. (Figure 1.1).

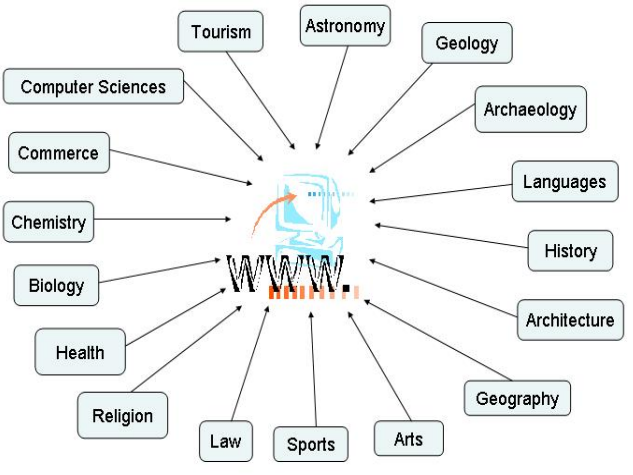


Fig. 1.1 Information on the web covers numerous and various knowledge domains

Finding relevant information from among the huge number of information resources is a difficult task. The sheer volume of currently available information limits access and retrieval of relevant information. Current use of web content is based on browsing and syntactic key-searches. Browsing makes use of hyperlinks in order to go from one web page to the next. In syntactic key-word searches, the key-words, which consist of patterns of characters, are matched to strings occurring in resources on the web. At present, these Google-like search engines are fundamental for retrieving information, but the search results are still often voluminous, frequently meaningless and often completely outside the domain in which the users are interested.

1.3 Hindrances to Successful Information Retrieval

The usefulness of available information needs to be maximized. This is becoming more important each day. The organization and management of the available information and existing information resources are an important focus.

We identified the following underlying issues in regard to difficulties in accessing, retrieving and sharing knowledge:

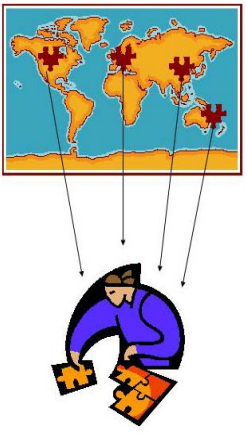
- There is an increasing body of distributed and heterogeneous information.
- There is no underlying knowledge base available to enable shared understanding of concepts, or to facilitate information retrieval and knowledge sharing.
- Databases and information resources are autonomous, heterogeneous and dynamic (content is continuously updated). They are developed by independent developers and there is no efficient tool to help coordinate them.

1.3.1 Distributed and Heterogeneous Information without Semantics

The rapid increase in the available information greatly increases the time required to locate the appropriate information by browsing and searching. Organizations are increasingly being forced to manage their knowledge more effectively and efficiently, and to share this information among all potential users.

The available information is distributed over various information resources. The internet contains a huge number of documents and information which cover different knowledge domains and different areas of the same knowledge domain. Much of the available information is 'repeated' within numerous databases. Usually, the knowledge required to solve a problem is spatially distributed over different information resources as shown in Figure 1.2. In this situation, sharing of information becomes crucial.

Fig. 1.2 Information distribution



The information is found in different formats relative to each other. Thus, the information could be unstructured information such as video, audio, image or text, semi-structured such XML documents, or structured such as relational or other data. Two different databases may contain information on the same topic but the way this information is structured and organized within those databases can be different. In these situations, it is quite difficult to automatically compare and analyze available information. Moreover, the available data is structured in such a way as to suit a particular purpose of the database and the associated community.

Important information in one database may be found to be insignificant for another database, and may be removed regardless of the importance of this information within a larger context. The heterogeneity of the information is becoming one of the main problems in efficient information use.

1.3.2 Underlying Knowledge Base Is Not Available

Knowledge integration methodologies are becoming more important as technology advances and interconnections between organizations grow. Efficient storage, acquisition and dissemination of knowledge require a structured and standardized organization of the data. The basic concepts, used to describe the information that needs to be reused and shared, must themselves also be shared. These basic concepts may be organized differently for different specialized systems, but the fundamental concepts must be common to all applications. A unifying framework that represents shared understanding of the domain knowledge would enable intelligent management of the available information across the domain.

1.3.3 Autonomous, Heterogeneous and Dynamic Information Resources

Currently, information resources exist autonomously and independently of each other (Goble 2003). In most cases, they are designed for one purpose, independently of the others. Each separate knowledge domain generates its own data and its own information sources. The integration of those autonomous information resources has been promoted over the last few years.

The information resources are quite heterogeneous relative to each other in their content, data structure, organization, information management etc. A range of analysis tools may be available, but their capabilities may be limited because each of those analysis tools is typically associated with a particular database format or information structure. Because each of the information sources usually have their own structures, content and query languages, those analysis tools often work only on a limited subset of the available information. Heterogeneity of the information resources makes their integration difficult.

One of the main issues is the dynamics of the internet environment. Content within the information resources on the web is changeable as it is continuously being updated and modified. The data and information can sometimes be added hourly or daily, and they may be removed in the next minute. A solution needs to be developed that can address the issues associated with the dynamic nature of these information resources.

1.4 Information Retrieval in Science

Information and computer-based support are becoming more and more important in a large number of disciplines. Computers are used to store and manage the

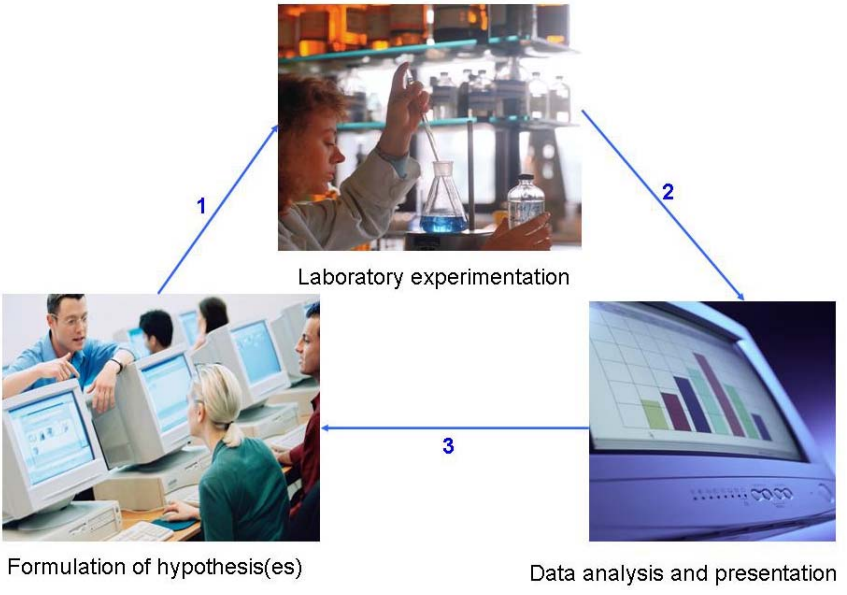


Fig. 1.3 Use of computers in biomedical domain

increasing amount of information. Also, computers are frequently used for data analysis performed after (large-scale) experiments. Various sciences are experiencing this kind of paradigm shift in the way that they conduct research. As a result, the new generations of scientists have to be highly computer literate in advanced computational techniques.

We show an example from the biomedical domain in Figure 1.3. The researchers formulate specific hypotheses through the study of contents of huge databases (arrow1). Laboratory experiments need to be performed in order to test the hypothesis (arrow2). This results in a collection of data. Computers are used to organize the data, perform the necessary calculation and present the data in meaningful format for the users (arrow3). The computer will also store this data and let the users access them when needed.

There is a need to create systems that can apply all available knowledge to scientific data. For any knowledge domain, the size of the existing domain knowledge base is too large for any human to assimilate. Computers are able to capture and store all this information but are unable to currently link this information and derive some useful knowledge from it. Better predictions can be made using a larger, more complex knowledge base for the domain. But the predictions are still being made on a relatively small subset of the available knowledge due to:

- the size of the available information;
- the autonomous, distributed and heterogeneous nature of the information resources; and
- the lack of tools to analyse the available information and derive useful knowledge from it.

As a result, some important information that could be used to provide knowledge is being neglected. Sometimes, this can have a significant impact on the outcomes. We need a system that can effectively and efficiently use all the available information.

Scientists would be able to progress much faster if they were able to use all the available knowledge efficiently. Usually, one particular problem can be solved by various research teams. Only when considered in the context of the other available information, does information provided by each individual team show its real value. Each research team provides a piece of information, a part of the jigsaw puzzle that, together with the information provided by other research teams, forms a complete picture.

Cooperation between different groups and professionals is also important. Different groups have different capabilities, skills and roles. Even if they all have common goals, they may be operating and executing their tasks on different knowledge levels. In most cases, the solution to a problem involves coordination of the efforts of different individuals and/or groups. There is an obvious need for these people to complement and coordinate with each other in order to provide the definition and/or solution to their common problem.

The complexity of the information retrieval process is due to:

- the huge body of available information;
- the nature and characteristics of information resources; and
- the fact that many users still use collections of stand-alone resources to formulate and execute queries. This places a burden on the user and limits the use that can be made of the available information. Users need mechanisms to capture, store, and diffuse domain-specific knowledge.

Use of the internet for information retrieval is still a complex task. Most problems need to be decomposed into smaller, interdependent sub-problems. A sequence of small tasks needs to be performed in order to solve the overall problem where each task may require querying of different information resources. Following the problem decomposition, the user needs to (Stevens et al. 2002):

- identify and locate appropriate information resources
- identify the content of those resources
- target components of a query to appropriate resources and in the optimal order
- communicate with the information resources
- transform data between different formats
- merge results from different information resources

One of the biggest issues in information integration is the heterogeneity of different information resources. Usually, a number of different information resources need to be queried in order to solve the problem effectively. Each information resource has its unique data structure, organization, management etc. This is represented by different colors in Figure 1.4.

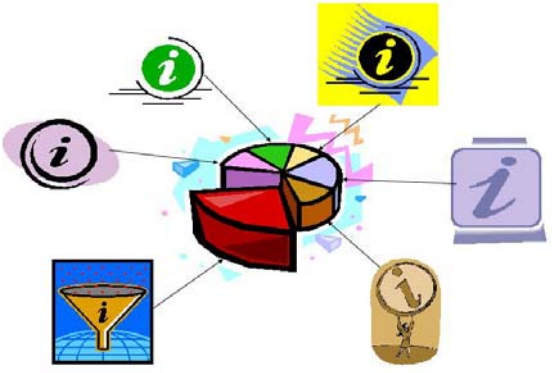


Fig. 1.4 Integration of heterogeneous information

Additional difficulties encountered by the users are:

- Extensive sifting through a multitude of information is necessary. For example, in 2007, a researcher setting a genetic human trial for "bipolar disorder" would have needed to sift through a multitude of information, from various sources (for example, Entrez Gene) to have found that loci 2p13-16 are potential positive sites for this disorder (the information originating from the research reported by Liu (Liu et al. 2003).
- As the research continues, new papers or journals are frequently published and added to the databases and more and more of this published information is available via the internet. Problematically, no collaborative framework currently exists to help inform researchers of the latest research and where and when it will become available.
- Huge amounts of information exist in many different databases. These databases mostly have their own structure. This is equivalent to the situation in libraries before the advent of Libraries Australia (http://librariesaustralia.nla.gov.au/apps/kss) and the development of universal cataloguing standards. No tool is yet available to help manage, search, interpret, categorise and index the information in these disparate databases.
- Most published results cover one specific topic. Usually, one needs to analyse and combine information regarding various topics in order to obtain an overall picture Currently, no tool exists that allows the simultaneous examination and analysis of different factors.
- Portions of the information or data on the internet may be related to each other, portions of the information may overlap, and portions of the information may be semi-complementary. No knowledge-based middleware is available to help identify these issues. 'Common knowledge', for example, may reduce the possibility of undertaking the same experiments, such as examining the same region of the DNA sequence, by different research groups, thus saving time and resources. This helps to create a cooperative environment making big research tasks coherent between different research teams. Gap analysis may also be easier.

1.5 Search Engines

In Section 1.4, we mentioned a number of factors responsible for the complexity of the information retrieval process. It is possible to design a search engine which will not be affected by the three issues described there. The sheer volume of the information and its continual increase is an unnecessary problem that can be solved by putting the semantics underlying this information into a universally agreed and shared standardized format which can be 'understood' by computers and their applications. The same is true for the nature and characteristics of information resources. If the semantics underlying the information content are stored according to the universally agreed and shared structure, the nature and characteristics of these information sources will be transformed to support efficient and effective information retrieval. In this situation, the user will not need to use a collection of stand-alone resources; all information resources together will 'act' as one big virtual information resource that contains uniformly formatted information from various knowledge domains.

Currently, the fourth and main problem associated with complex and inefficient information retrieval is the nature and characteristics of the current search engines. At present, Google-like search engines based on syntactic keywords are fundamental for retrieving information, but the search results are still often meaningless. For example, consider the concept "plate". This concept has different meanings in different domains such as architecture, biology, geology, photography, sports etc (http://www.answers.com/topic/plate) (Figure 1.5). Doing the search for "plate" using current search engines, you may get various results from various knowledge domains. If you try to specify search by typing in "plate, sports", you may still get something like: "Everyone needs to bring a plate with food on Sunday night".

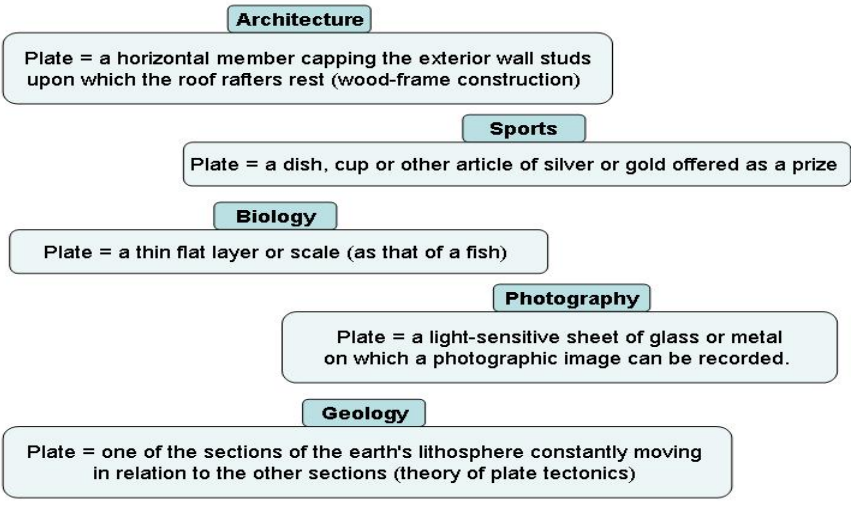


Fig. 1.5 The term "plate" has different meanings within different knowledge domains

We need search engines to look for the meanings of words within specific contexts.

The content of the web pages needs to be described in such a way that search engines look for the meaning of the word within a specific context rather than the word itself. The content of the information resources is currently described using 'presentation-oriented' tags and the associated search engines work on this principle. In order to bring about a positive transformation regarding this issue, the content of the information resources need to be described using 'meaning-oriented' tags so that the search engines can be designed to search for the meaning of the information rather than simply its appearance in the text. This is one of the major concerns and goals in creating and increasing Web Semantics which concentrates on meaning.

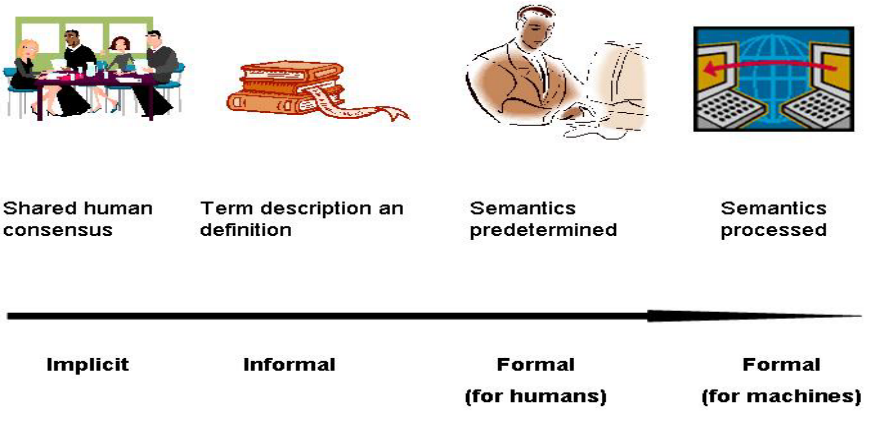


Fig. 1.6 Semantic Continuum

1.6 Web Semantics

The Semantic Continuum (Uschold 2001) (see Figure 1.6) moves from the semantically poor which is easily understood by humans toward semantically rich and easily 'understood' by computers. Semantics that exists only in the minds of the humans who communicate and build web applications is implicit. Semantics may also be explicit and informal such as those found in thesauruses and dictionaries where terms are described and defined. Semantics may also be explicit and formal for humans such as in computer programs created by people. There is less ambiguity the further we move along the semantic continuum, and it is more likely to have robust, correctly functioning web applications. Here, the semantics is explicit and formal for machines.

The current web can reach its full potential through enabling machines to be more actively involved in the web 'life-processes'. Machines need to be able to 'understand' the web content, access it, retrieve and process the data automatically rather than only displaying them. They need to be able to meaningfully collect

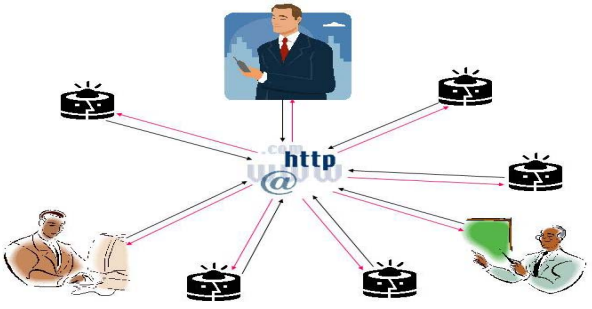


Fig. 1.7 Machines and humans exchange information

information from various information resources, process this information and exchange the results with other machines and human users (Figure 1.7). The web information that is semantically rich and easily 'understood' by computers can facilitate this. Namely, the current web can reach its full potential through increasing the semantics of its content which is commonly referred to as Web Semantics.

The main and the biggest project currently associated with establishing and increasing Web Semantics is the Semantic Web project. Tim Berners-Lee, World Wide Web Consortium, is leading this project. He aims to transform the current web into a semantic one, the content of which will be described in a meaningful computer-understandable and computer-processable way.

> "*The Semantic Web is an extension of the current Web in which information is given well-defined* meaning*, better enabling computers and people to work in cooperation.*" (Berners-Lee et al. 2001).

> "*The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be* used by machines *not just for display purposes, but for automation, integration and reuse of data across various applications.*" (www.w3c.org)

These definitions emphasize:

- well-defined meaning of the information
- machines use the information (automation)
- cooperation through data integration and reuse

The Semantic Web project aims to enable machines to access the available information in a more efficient and effective way. The researchers and developers are working on two different options towards the realization of their vision. In the first option, the machine reads and processes a machine-sensible specification of the semantics of the information. In the second option, web application developers implant the domain knowledge into the software which will enable the machines to perform assigned tasks correctly.

The first two important languages that played a significant role in the Web Semantics are: the eXtensible Markup Language (XML) and the Resource Description Framework. XML defines structure in the documents. However, it says nothing about the meaning of this structure. On the contrary, meaning is expressed by RDF. The meaning is encoded in sets of triples: things, its properties and values. These triples can be written using XML tags and form webs of information about related things.

1.7 Agents for Dynamic Information Retrieval

Agents are intelligent software objects used for decision-making. They are capable of autonomous actions and are sociable, reactive and proactive in an information environment (Wooldridge 2002). Agents can answer queries, retrieve information from servers, make decisions and communicate with systems, other agents or with users. Use of agent-based systems enables us to model, design and build complex information systems.

The Semantic Web project aims to transform the current web into a semantic one where agents can understand web content. The idea is presented in Figure 1.8. In this environment, an information resource will contain data as well as metadata which describe what the data are about. This allows agents to access, recognize, retrieve and process relevant information. They are also free to exchange results and to communicate with each other. In such an environment, agents are able to access and process the information effectively and efficiently.

In this situation, the web is more than just a collection of web pages. Agents can 'understand' and 'interpret' the meaning of the available information, perform complex tasks and communicate with each other and with humans. Humans can use one or several such agents simultaneously.

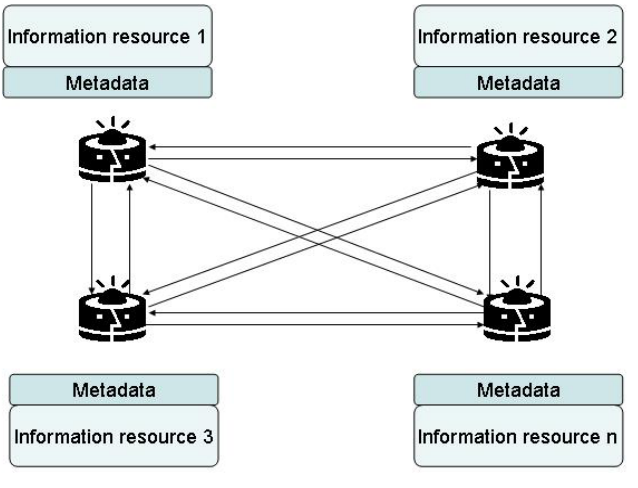


Fig. 1.8 Annotation of information resources enables agents to work cooperatively

1.8 Ontologies for Intelligent Information Retrieval

An ontology is an enriched contextual form for representing domain knowledge. An ontology provides a shared common understanding of a domain and the means to facilitate knowledge reuse by different applications, software systems and human resources (Gómez-Pérez 1996, 1998).

Hendler emphasizes the interdependence of agent technology and ontologies (Hendler 2001). Intelligent agents must be able to locate meaningful information on the web and understand the meaning of this information, to advertise their capabilities and to know the capabilities of other agents. All this can be archived through the use of ontologies: firstly, through ontology sharing between the agents; and secondly, through the annotation of information resources.

Agents need to communicate and interact with other agents. This requires them to share a common understanding of terms used in communication. The specification of the used terms and exact meaning of those terms is commonly referred to as the agent's ontology (Gruber 1993). Because this meaning must be available to other software agents, it needs to be encoded in a formal language. An agent can then use automated reasoning and accurately determine the meaning of other agents' terms.

An ontology is "a formal and explicit specification of a conceptualization" and allows the information providers to annotate their information (Berners-Lee et al. 2001). The annotation phase is a crucial step in creating a semantically rich environment which can be intelligently explored by users' agents. The annotation provides background knowledge and the meaning of the information contained within this information resource.

1.9 Conclusion

In this chapter, we discussed the issues associated with the current use and management of information on the Web and discussed proposed solutions to these problems. We explained Web Semantics, introduced the Semantic Web project and discussed its vision. We briefly introduced two major technologies that may help realize this vision: ontology and agent technology. We will discuss the agent and ontology concepts in greater detail in the following two chapters.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284, 34–43 (2001)
2. Brodie, M.L.: Computer Science 2.0: A New World of Data Management. In: 33rd International Conference on Very Large Data Bases Vienna, Austria (2007)
3. Goble, C.: The Grid Needs you. In: International Conference on Cooperative Object Oriented Information Systems, pp. 589–600 (2003)
4. Gómez-Pérez, A.: Towards a Framework to Verify Knowledge Sharing Technology. Expert Systems with Applications 11, 519–529 (1996)

5. Gómez-Pérez, A.: Knowledge Sharing and Reuse. The Handbook on Applied Expert Systems, 1–36 (1998)
6. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5, 199–220 (1993)
7. Hendler, J.: Agents on the Semantic Web. IEEE Intelligent Systems 16(2) (2001)
8. Liu, J., Juo, S.H., Dewan, A., Grunn, A., Tong, X., Brito, M., Park, N., Loth, J.E., Kanyas, K., Lerer, B., Endicott, J., Penchaszadeh, G., Knowles, J.A., Ott, J., Gilliam, T.C., Baron, M.: Evidence for a putative bipolar disorder locus on 2p13-16 and other potential loci on 4q31, 7q34, 8q13, 9q31, 10q21-24, 13q32, 14q21 and 17q11-12. Mol. Psychiatry 8, 333–342 (2003)
9. Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A., Brass, A.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. Bioinformatics 16, 184–186 (2002)
10. Uschold, M.: Where are the Semantics in the Semantic Web? In: Proceedings of the Ontologies in Agent Systems Workshop, held at the Autonomous Agents Conference, Canada (2001)
11. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley and Sons, Chichester (2002)

Chapter 2
Introduction to Multi-Agent Systems

2.1 Introduction

The body of available information is rapidly increasing every day. It is becoming impossible for humans to peruse all the available information and extract the data related to a specific topic. A great deal of important information is being neglected primarily for two major reasons. Firstly, highly relevant information is losing its true meaning within a pool of insignificant and unauthorized information. A lot of available information is published which is of no significance and sometimes may even be destructive and corruptive in nature. Secondly, new knowledge and discoveries are emerging quickly as we continue to progress and new technologies emerge. It is becoming more and more difficult each day to find specific information even within a body of authorized information.

As a solution to this problem, humans are designing software agents that will do the search for them. In this chapter, we will discuss agent features such as environment, internal structure, mobility, communication, and cooperative work with other agents.

2.2 Agent Definition

Various attributes associated with agents are of differing importance for different domains. As a result, the term 'agent' is being defined differently by different domains and no universally accepted definition for this term exists.

The definition presented here is taken from the book "An Introduction to Multi-agent Systems" (Wooldridge 2002):

"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 design objectives."

An agent is autonomous, meaning that it decides and acts independently of its designer. The agent is constantly aware of the design objectives and is on an everlasting mission to complete them in the most efficient and effective way.

Agent architectures are actually software architectures embedded in an environment as shown in Figure 2.1. The agent reacts to the inputs from the environment. An agent can produce a range of outputs. The production of a specific output may have preconditions associated with it. The preconditions are usually specified as conditions on the states of the inputs and/or environment. Depending on the input, the agent decides what kind of output it needs to produce. This output results in actions that affect the environment. The set of available actions, together with associated preconditions, determine the ability of the agent to modify its environment, and this is implemented by the way an agent is programmed.

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 15–35.
springerlink.com

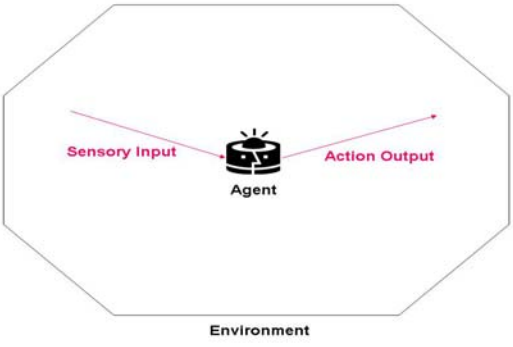


Fig. 2.1 Agent and its environment

2.3 Agent's Environment

An agent's environment is characterized, as shown in Figure 2.2, by the following four main features (Russel and Norvig 1995):

- non-deterministic
- inaccessible
- dynamic
- continuous

Since an agent does not have complete control over its environment, its influence is only partial. It is possible for an agent to generate two different outputs on the basis of two identical inputs. Namely, the same action performed twice in apparently identical circumstances might appear to produce entirely different effects. Moreover, these actions may fail to have the desired effect (Wooldridge 2002). An action performed by an agent within an environment does not have a single guaranteed effect and the resulting state of this action is uncertain. For this reason, the agent's environments are assumed to be non-deterministic.

The internet environment is becoming less accessible for agents as more information becomes available online. It is impossible for agents to obtain complete, accurate and up-to-date information within this environment. It is becoming more difficult to build agents that can operate effectively within this pool of heterogeneous information. For this reason, most real-world environments are inaccessible.

A dynamic environment is an environment that has other processes operating on it, thereby causing it to change state. As a result, this environment is changing beyond the agent's control. In contrast, a static environment is assumed to be changed only by the agent's performances. Most of the agents are situated in a highly dynamic and continuously changing environment such as the Internet.

As an agent environment is dynamic and constantly changing, the number of states within this environment is neither fixed nor finite. For this reason, we say that the agent's environment is continuous rather than discrete.

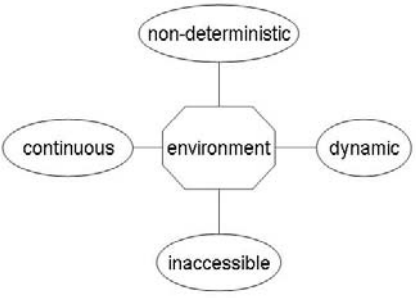


Fig. 2.2 Characteristics of a typical agent's environment

2.4 Agent's Characteristics

An agency is concerned with assigning concepts and attributes to agents in order to define their nature. This makes it possible to predict an agent's behaviour when found in a specific situation. A well-defined agent whose behaviour is predictable is more likely to be trusted and used. Some important features of agents (Wooldridge 2002) are shown in Figure 2.3. These include:

- autonomy
- social ability
- reactivity
- pro-activeness

An agent needs to be autonomous; it needs to be independent when making decisions and performing actions. It needs to be able to make appropriate choices and satisfy its design objectives. An agent needs to be able to make rational decisions based upon the available information, but also to determine what to do after an action either succeeds or fails.

An agent needs to show social abilities. An agent needs to interact and communicate with other agents especially in multi-agent systems where the agents are collaboratively working towards a shared goal. An agent needs to interact and communicate with the local environment for various purposes such as the maintenance, discovery, retrieval and exchange of information. This is achieved by engaging in social activities like cooperation, coordination and negotiation. An agent also needs to interact and communicate with users in order to take their requests, keep them informed of the agent's progress and provide them with the requested information.

Agents need to be reactive. They need to be able to perceive their environment, and recognize and understand changes in that environment. Agents need to know how to react to these changes and respond to them in a timely manner.

Agents need to be pro-active. They need to be able to initiate actions that will help them achieve their goals. This means that an agent needs to appreciate the state of its environment and decide how best to fulfill its target mission.

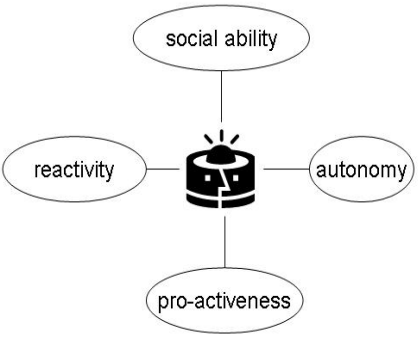


Fig. 2.3 Agent's characteristics

2.5 Internal Data Structure of Agents

The internal data structure enables agents to act autonomously and make independent decisions. Internal data structure is a major factor in the decision-making process. The internal data structure (Wooldridge 2002) contains:

- domain knowledge description
- knowledge about the agent's environment
- the agent's history files

An agent needs to know the meaning of the information in order to search for, retrieve and manipulate this information. Usually, ontologies are used to represent the domain knowledge and enable the agent to function optimally. This ontology makes up part of the agent's internal data structure.

An agent needs to be familiar with its environment in order to function efficiently. For example, an agent responsible for retrieval of information from a number of databases needs information describing the structure and content of these databases. Just as a city map for a driver makes it easier for him to find a particular street, this information about the database structure and content makes it easier for an agent to find the requested information. The agent's knowledge about its environment is embedded in its internal data structure.

An agent can use information about its previous experiences for the purpose of improving itself and increasing efficiency. For example, information about the agent's previous actions enables an agent to learn through experience and to promote self-correction. Similarly, if a system has been queried for the same information previously, an agent can compare new results with the old ones and update the old file for that specific query if needed. Such information about the previous experiences of an agent is kept in the agent's internal data structure.

Note that the representation of domain knowledge may differ from one agent to another. It is not that the domain knowledge changes; rather, it is the function of different agents that changes. Because the functions of different agents are different, the knowledge needed to enable them to perform the function successfully also needs to be different. Also, environmental states may differ from one agent to another; for example, different databases may be assigned to different agents.

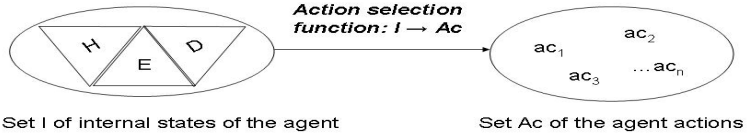


Fig. 2.4 Action selection function

The idea is represented in Figure 2.4. E is the set of possible environmental states of the agent, O is the set of different knowledge domain descriptions and H is the set of history files. I is the set of internal states of the agent so that $I = E \cup O \cup H$.

Ac is a set of agent actions. Action selection function is defined as:
action : $I \rightarrow Ac$.

The internal value of the data structure of an active agent is constantly changing. On the basis of an input and the current value of an agent's internal data structure, an agent may decide to execute an action. This action will then in turn affect the value of the agent's internal data structure. For example, history files may need to be updated. Sometimes, through execution of an action, an agent may gain more information regarding the environment or knowledge domain. In this case, the internal data structure needs to be updated. This is shown in Figure 2.5. Some agents are able to perform self-correction and update this information within their internal data structure. For other agents, the information about the nature of the environment and knowledge domain information needs to be updated by software engineers. Because of an agent's dynamic nature and its dynamic environment, the agent's internal data structure needs to be constantly updated.

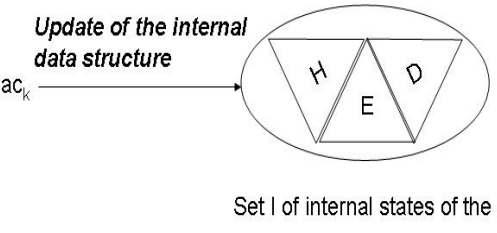


Fig. 2.5 Agent's action may affect its internal data structure

2.6 Mobile Agents

Mobile agents are able to migrate between nodes on a network or between nodes across networks. They carry their program code and their current state with them to the new node. This allows them to carry out execution at the new node. The

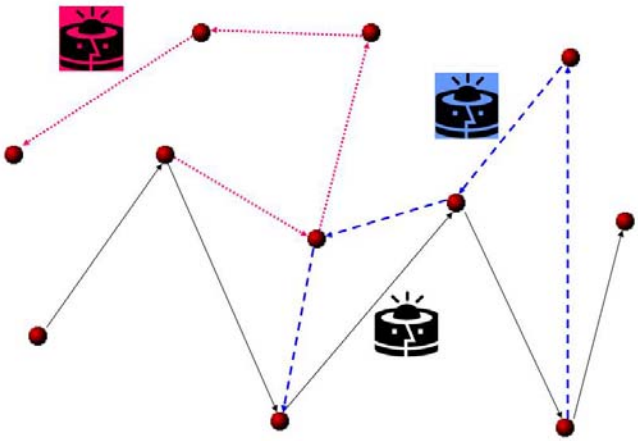


Fig. 2.6 Agent's migration between modes

agent's migration between nodes is pictorially represented in Figure 2.6. Typical characteristics of mobile agents are their ability to migrate, autonomy in performing their actions, a peer-to-peer personality, and processing independence and network independence from their original location.

It is important to keep in mind that a mobile agent and agent migration is fundamentally different from a process and process migration (Smith 1988). The system forces the process to move in process migration, while the mobile agent decides when and where to move in agent migration. The underlying infrastructure must support and execute this agent's request. A mobile agent can be forced to move only in extreme circumstances, such as to avoid malicious agents at the current execution site.

The two approaches to moving agents between network nodes are (Dale 1997):

- state-oriented
- stateless

In state-oriented migration, the agent is able to decide to move at any point in its execution. The current state of the agent is encapsulated and transferred across the network to the receiving network node. In Figure 2.7, this is represented as state "123" which is encapsulated and transferred to the second node. The agent itself or the infrastructure captures the agent's state. At the receiving network (second) node, execution of the agent resumes. In Figure 2.7, continuum of the execution is represented as generating the state "456". When the execution is completed at the second node, the agent's state "123456" is encapsulated and transferred by the agent itself or by the infrastructure to the third node. At the third node, execution continues until state "123456789" is reached and the agent decides to move to the fourth node etc. This kind of migrations is typical in situations when an agent needs to collect information on a specific topic; the agent is building a body of information through addition of newly found information to the information accumulated previously.

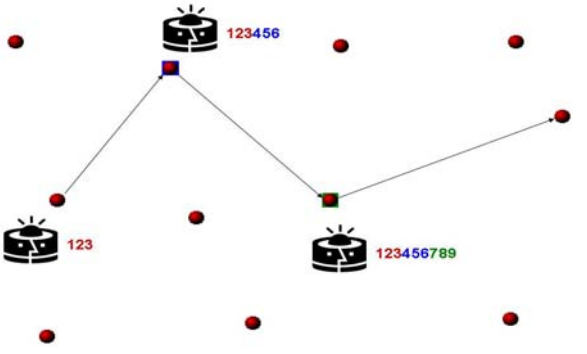


Fig. 2.7 State-oriented agent's migration

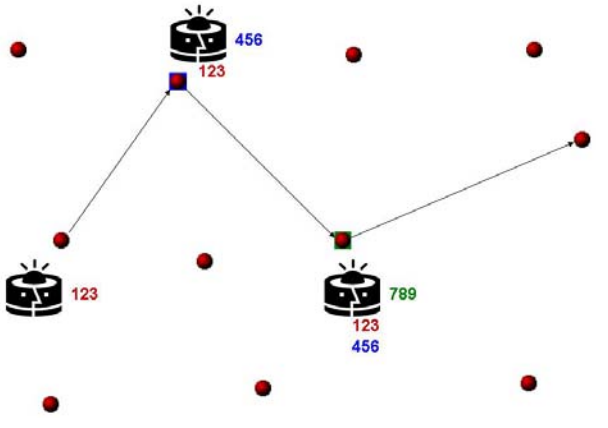


Fig. 2.8 Stateless agent's migration

In Figure 2.7, the agent is 'building up' its state on the basis of its previous activity; first node corresponds with "123" state, second node corresponds with the "123456" state, third node corresponds with "123456789' state etc. An example of such an agent occurs in certain e-commerce applications such as in the travel industry where an agent might visit various sites to collect information on the cheapest airfare.

In stateless migration, an agent needs to restart its execution from the beginning at the receiving node. Before the agent can move, all state information must be written to a data repository or knowledge base. This information is encapsulated and transferred with the agent. At the receiving network node, the agent needs to restore its state from the repository if needed. In Figure 2.8, we represent that each node is associated with an agent's state; the first node corresponds with "123"

state, the second node corresponds with the "456" state, the third node corresponds with "789' state etc.

Once an agent has finished with a network node, it must decide where to move to next. The three methods for making this decision are (Dale 1997):

- predetermination
- dynamic determination
- hybrid determination

Predetermination is the situation where the agent is given a set of destinations that it must visit once it is launched. This approach is implemented in situations where the activities of an agent need to be carefully controlled such as when the order in which an agent visits different nodes is important.

In dynamic determination, the agent has complete freedom in choosing which network nodes to visit. The mobile agent may make a random choice or it may make a decision based upon its own knowledge such as its own history files or information gained via other agents. This type of freedom is useful for agents which mine through a wide range of information in order to derive new knowledge and/or derive new data patterns (data mining agents).

Hybrid determination is a method whereby a mobile agent is given a list of destination nodes that it must (or must not) visit. Alternatively, the agent is equipped with knowledge and a set of criteria that will support and guide its decisions in regard to nodes selection. This method is used in situations where the speed of data return is more important that the depth of data covered.

2.7 Dependency Relationships between Agents

Most current systems are composed of a number of sub-systems that must interact in order to successfully carry out and complete the overall task. Multi-agent systems are composed of various agents that interact with each other and complement each other.

The multi-agent system can be observed from two perspectives. The first one is concerned with the individual agent, while the second one is concerned with collections of those agents which communicate and interact with one another.

The agents act in an environment; agents have control over and/or are able to influence different parts of the environment. Here, we are talking about 'sphere of influence' associated with each agent. Overlap of the spheres of influence gives rise to dependency relationships between the associated agents (Wooldridge 2002). This principle is shown in Figure 2.9.

If one agent requires the other agent in order to achieve its goals, then we say that a dependence relation exists between those agents. A number of possible dependency relations may exist between different agents as represented in Figure 2.10. These are:

- Independence; there is no dependency between the agents.
- Unilateral; one agent depends on the other but not vice versa.

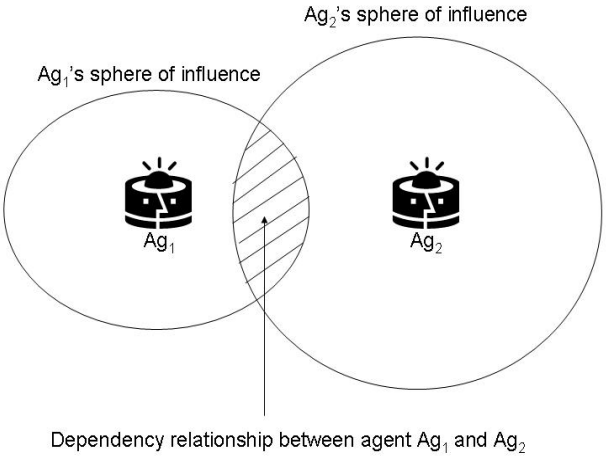


Fig. 2.9 Dependency relationships between agents

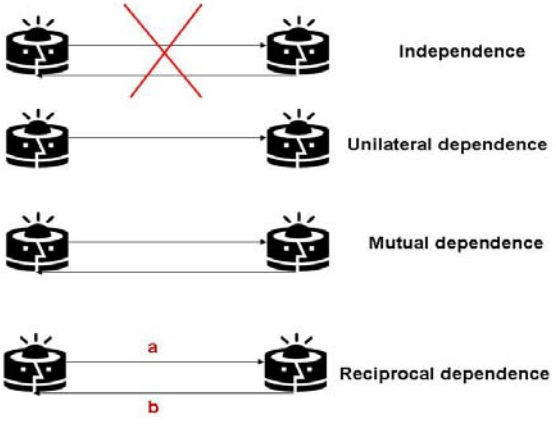


Fig. 2.10 Possible dependency relationships between agents

- Mutual; both agents depend on each other with respect to the same goal.
- Reciprocal dependence; one agent depends on the other for some goal, while the second agent also depends on the first agent for another goal. Here, the two goals are not necessarily the same.

These dependencies between agents may be:

- locally believed; one agent believes that the (a) dependence exists and (b) the other agent does not believe it exists;
- Mutually believed; one agent believes that the (a) dependence exists and (b) the other agent also believes it exists.

2.8 Agent's Communication

The communication between different agents is specified by:

- parties involved in the communication
- place where the communication parties are situated
- time when the communication takes place
- language used in the

2.8.1 Communication Parties

The parties with which an agent may wish to communicate include:

- its environment
- other agents
- users

Agents communicate with their environment for various purposes such as to determine existing resources or services, to access data, to transfer the results, to determine the location of another agent, and other similar reasons.

Communication between agents is of high importance in various situations, such as to allow agents to share information, to query other agents and to coordinate the activities between agents, thereby enabling them to work collaboratively.

Agents need to take requests from and communicate the results back to their user. This is one of the main tasks of interface agents. Other agents may log their data with the interface agent that presents the information `as is' or pre-processes it. This information may need to be collected from various agents and organized into a format which is suitable for the user to view.

2.8.2 Communication Place

The communication can take place:

- within the same node (node-oriented), or
- between different nodes (network-oriented)

When the communicating parties are located on the same network node, they communicate with each other through some local inter-process communication mechanism.

In network-oriented communication, the communicating parties do not have to be situated on the same node or even the same network. They can communicate with each other through some network-based mechanism.

2.8.3 Communication Time

The communication between different communicating parties can take place:

- synchronously, or
- asynchronously

In synchronous communication, communicating parties must arrange a time for communication. In some situations, it is very important for the agents to be synchronized such as in the case of data transfer that needs to be confirmed, in interactive dialogues, in risky interactions which can bring destructive results, and so on.

In a synchronous communication, communicating parties communicate with each other as they desire and when it is convenient for them to do so. This type of communication is used in more flexible situations and lower risk situations such as transfer of informative data.

2.8.4 Communication Languages

If two agents are to communicate with each other, they must share a common language. The language that the agents use in communications is called the Agent Communication Language (ACL).

2.8.4.1 Desired Features of Agent Communication Languages

The desired features of agent communication languages are (Finin et al. 1994) related to its:

1) form
2) content
3) semantics
4) implementation
5) networking
6) environment
7) reliability

The form of the language should be kept simple and linear (or should be easily translated into a linear form) because of the underlying transport mechanism. An agent communication language should be readable by people and declarative. Its syntax should be simple and extensible, as it needs to be integrated into various systems.

The communication language that expresses communicative acts needs to be separated from the content language that expresses facts about the domain. A layering approach that separates the communication language from the content language helps achieve this as it (1) enables successful integration of the language to applications; and (2) provides a conceptual framework for the understanding of the language. Irrespective of the application, e.g. database, object-oriented system or knowledge base, a core of primitives that capture most of communicative acts needs to be specified. This set of communicative acts primitives needs to be extensible to enable the language to be used by a variety of systems.

As in any other language, the semantics of a communication language should be grounded in theory, be unambiguous and exhibit canonical form (meaning similarity should lead to representation similarity). As the communication

language enables interaction between spatially dispersed applications, and this interaction extends over time, location and time need to be addressed by the semantics of a communication language. Communication parties should have a shared understanding of the language, its primitives and the protocols associated with their use and they need to be formally described.

The implementation of a communication language should fit well with existing software technology. The implementation of the language should also be efficient. The interface should be easy to use with the networking layers that lie below the primitive communicative acts hidden from the user. As some agents may require only a subset of the primitive communicative acts, the partial implementation of the language should be possible.

A communication language should also fit well with modern networking technology. The basic connections (point-to-point, multicast and broadcast) as well as synchronous and asynchronous connections should be supported by the language. The language should contain a set of primitives that is rich enough to serve as a basis upon which higher-level languages and interaction protocols can be built. The design of these higher-level protocols should not be dependent on the transport mechanisms used.

An agent communication language must function effectively in an environment that is highly distributed, heterogeneous and dynamic. Moreover, it should support functions such as interoperability with other languages and protocols and knowledge discovery in large networks. The language needs to be easily attachable to existing legacy systems.

Reliable and secure communication needs to be supported by the communication language. The language needs to support authentication of agents as well as identify and signal errors and warnings.

2.8.4.2 Agent Communication Languages

The communicating agents need to know with whom to talk, how to find them, how to initiate and how to maintain the communication. Agent Communication Languages (ACLs) are concerned primarily with these aspects.

The two important characteristics of ACLs are:

- handling propositions, rules, and actions instead of simple objects with no semantics associated with them; and
- Describing a desired state in a declarative language, rather than a procedure or method.

The ACLs are still limited; they do not cover the entire spectrum of what applications might want to exchange. In some situations, agents need to exchange complex objects such as shared plans, goals, experiences or long-term strategies. When using an ACL, agents transport messages over the network using a lower-level protocol. The types of these messages and their meanings are defined by ACL. Agents are not only engaged in single-message exchanges, but also in more complex task-oriented conversations such as negotiation or auction. The agent's

communications are driven by a higher-level conceptualization of the agent's strategies and behaviours.

We will discuss two major research streams associated with the design and development of ACLs: Knowledge Sharing Effort (KSE) and Foundation for Intelligent Physical Agents (FIPA) research effort.

Knowledge Sharing Effort (KSE) (Finin et al. 1994) aims to develop a technical infrastructure that will support knowledge sharing among systems. The KSE is organized around the following three working groups:

- Interlingua
- Shared Reusable Knowledge Bases, and
- External Interfaces

The Interlingua Group developed the Knowledge Interchange Format (KIF). KIF is a computer-oriented language for expressing the content of a knowledge base and interchanging the knowledge between different applications. KIF can be used as a common content language between two agents which use different representation languages or to support translation from one content language to another. In this latter case, KIF has the mediating role of an interlingua. The language description includes both syntax and semantics specifications. The main advantages of KIF are:

- logically comprehensive, i.e. arbitrary sentences can be expressed in first-order predicate calculus
- declarative semantics, i.e. understandable meaning of the expressions
- specification of class hierarchies
- representation of knowledge about knowledge
- description of procedures

The Shared Reusable Knowledge Bases (SRKB) Group is concerned with the content of sharable knowledge bases. SRKB Group developed Ontolingua (Farquhar et al. 1997) which is a language that enables the description, development and maintenance of ontologies. KIF was extended with additional syntax to capture the intuitive bundling of axioms (Gruber 1993) and a Frame Ontology was designed to define terms including class, subclass-of, slot, slot-value-type, slot-cardinality, facet, and so on. Also, a set of tools and services are built around Ontolingua to support the geographically distributed groups in the ontology sharing. Ontologies are stored on an ontology server and the users are able to publish, browse, edit those ontologies, assemble a new ontology, and similar undertakings.

The scope of the External Interfaces Group developed KQML language. KQML (Labrou et al. 1999; Finin et al. 1994) is a message-oriented communication language and protocol for information exchange. KQML is independent of the (1) transport mechanism, (2) content language, and (3) associated ontology. KQML messages do not only communicate sentences; they also articulate an action such as assertion, request or query. The following three layers can be identified in a KQML message:

- Content layer: it bears the content of the message in the program's own representation language. KQML can carry any representation language.
- Communication layer: it describes the lower-level communication parameters, e.g. sender's and recipient's identity, and a unique identifier associated with the communication.
- Message layer: it is the core of KQML and encodes a message that needs to be transferred. Its primary function is to determine the network protocol used to deliver the message.

Foundation for Intelligent Physical Agents (FIPA) (Labrou et al. 1999, Aparicio et al. 199, Bellifemine et al. 1999 and 2001) is based on the international collaboration of companies and universities active in the field. Its goal is to maximize interoperability across agent-based systems and to promote the success of emerging agent-based applications, services and equipment. FIPA ACL is superficially similar to KQML. As in KQML, FIPA's ACL messages can be viewed as actions or communicative acts (Labrou et al. 1999). FIPA's ACL syntax is identical to KQML's except for different names for some reserved primitives. FIPA ACL maintains the KQML approach of separating the outer language from the inner language. FIPA's ACL communicative acts correspond to the KQML communication primitives and are the same kind of entity. SL is the formal language used to define FIPA ACL's semantics. SL is a quantified, multimodal logic with modal operators for beliefs (B), desires (D), uncertain beliefs (U), and intentions (persistent goals, PG). SL can represent propositions, objects and actions. The semantics of each communicative act is specified as sets of SL formulae that describe the act's feasibility preconditions and its rational effect, including the conditions that should hold true for the recipient.

2.8.4.3 Ontologies for Agent Communication

It is necessary for the communicating parties to agree on the terminology they use in the communication. Ontologies can be used to formally define the body of knowledge and specify the set of terms. In this way, ontologies permit coherent communication between communicating parties. Ontologies enable cooperatively working agents to communicate with each other. Shared ontologies are undergoing development in many important application domains.

The DARPA (Defence Advanced Research Project Agency) Agent Markup Language (DAML) initiative aims to support the development of the semantic web. In DAML, semantic relations are captured in machine-readable form through expressive term descriptions and precise semantics. The expressive power of DAML is greater than the expressive power of RDF and RDF Schema. This is one of the main factors to enable intercommunication. Work on the DAML ontology languages was inspired (McGuinness et al. 2002) by:

- Web integration
- frame-based systems, and
- description logics

The DARPA Agent Markup Language (DAML) ontology language integrated with RDF and RDF Schema results in a language that is compatible with existing Web standards. The language also permits interoperability with the other contents and language tools.

DAML was also influenced by systems with a frame-based look, e.g. SHOE (Simple HTML Ontology Extensions) (McGuinness et al. 2002). SHOE was specifically designed for Web use and was the first frame-based knowledge-representation language. It extended HTML (and later XML) to permit the description of classes, their properties and the inference rules.

Description logics provide a formal foundation for frame-based systems. It provides complete denotational semantics and tractable reasoning algorithms to support clear and unambiguous languages.

2.9 Collaborative Problem-Solving Process in Multi-Agent Systems

The multi-agent system goes through different stages during the problem-solving process. The major steps are represented in Figure 2.11 and include:

1) problem decomposition
2) problem solution
3) result sharing
4) result analysis
5) solution synthesis

In the problem decomposition stage, the overall problem to be solved is decomposed into smaller subproblems. The agent that performs the problem decomposition must have knowledge of the task structure and must know how the task is put together. The subproblems are further decomposed into smaller sub-subproblems and so on. This kind of decomposition is usually hierarchical and each of these different levels in the problem solving hierarchy represents the problem at a progressively lower level of abstraction. The goal of this problem decomposition is to reach a stage where the subproblems are of an appropriate granularity to be solved by individual agents.

In the problem solution stage, the subproblems identified during the problem decomposition phase are individually solved by individual agents.

In the result sharing phase, agents share information relevant to their subproblems. One information agent may be aware of the tasks assigned to the other information agents. Sharing of the information between different information agents is important because different information resources and tasks are assigned to different agents.

In the result analysis phase, the shared information is analyzed for its relevance and significance. The goal is to select relevant and appropriate information to be assembled together into a final solution. For example, if one of the resources contains more precise information than the other, then the more precise information needs to be selected to be assembled into the final solution.

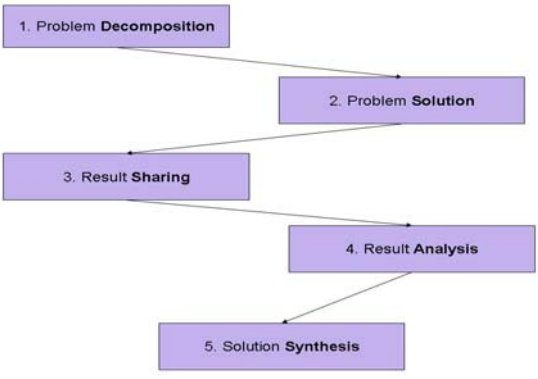


Fig. 2.11 Collaborative problem-solving process in multi-agent system

In the stage of solution synthesis, solutions to atomic problems are integrated into an overall solution. The selected information covering different topics is assembled together. A solution is developed through cooperative information exchange between agents. Solutions to small problems are progressively refined and assembled at different levels of abstraction into larger and more abstract solutions. This stage is usually hierarchical and corresponds with the opposite direction to the problem decomposition phase.

2.10 Different Types of Agents

Agents can be divided into different types according to several different criteria. In this section, we discuss several of these classifications. In particular, we look at:

1) Haag's classification
2) Dillenbourg et al.'s classifiaction
3) Maes's classification
4) Classification according to agent's functions

2.10.1 Haag's Classification

According to Haag (Haag et al. 2002), there are four essential types of agents. These are represented in Figure 2.12.

1. Buyer Agent
Buyer agents travel around a network retrieving information about goods and services. *Amazon.com* is an example of a buyer agent. This agent may offer you a list of books you may be interested in on the basis of what you're buying now and what you have bought in the past. Buyer agents are usually very efficient with products such as CDs, books, electronic components etc.

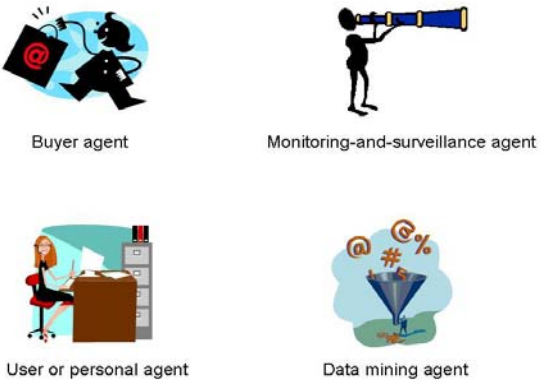


Fig. 2.12 Agent classifications according to Haag

2. User or Personal Agents
User agents are agents who take action on your behalf. These agents can perform a variety of actions such as searching for specific information, highlighting the significant information, assembling this information, storing the information for future reference, filling out various forms for you, checking and sorting your e-mails, playing computer games with you, discussing various topics with you etc.

3. Monitoring-and-Surveillance Agents
Monitoring-and-surveillance agents observe and report on equipment and/or processes. They usually monitor complex processes, so that computer networks and configurations of each computer connected to the network can be tracked.

4. Data Mining Agents
Information from various sources is usually stored in a data warehouse. Data mining agents operate in a data warehouse discovering information that can be used to support and/or take action. For example, data mining agents can derive information that can be used to help increase the sales.

2.10.2 Dillenbourg et al.'s Agents Classification

Empirical study of multimodal human collaboration helps in the understanding of agents' interactions in task-based collaboration. Dillenbourg have identified several different types of internet agents (Dillenbourg et al 1997):

1. Sub-agents
Sub-agents are autonomous software entities which carry out tasks for the user. These are user or personal agents from Haag's classification. Many of these systems are based on machine learning techniques. Due to the limitations of machine-learning techniques, distributed multi-agent systems are also used.

2. Co-agents
Symmetrical systems allow human and artificial agents to perform the same actions. This approach is used within various learning paradigms such as

collaborative learning, competitive activities, reciprocal tutoring, learning by teaching and teacher training.

3. Super-agents
Super-agents are situated in intelligent and/or multi-user learning environments. They monitor the user's actions and provide various solutions for them. They also monitor the interactions among the users. For example, these agents may be teachers analyzing interactions and intervening when needed.

4. Observers
Observers calculate statistics regarding collaboration. They collect information regarding users' interactions and aggregate these results into high level indicators. These indicators are displayed in order to be applied to collaborative settings which guarantee effective collaboration.

2.10.3 Maes's Classification

Patty Maes (Maes et al. 1997) distinguishes between three different types of agents which are represented in Figure 2.13.

1. User programmed agents
The users provide rules for the agents. The agents then perform actions based on these rules.

2. Artificial Intelligence engineered agents
The design of these agents is based on traditional Artificial Intelligence techniques.

3. Learning agents
They detect the actions and behaviours of their users and use this knowledge to program themselves. They can also learn from other agents.

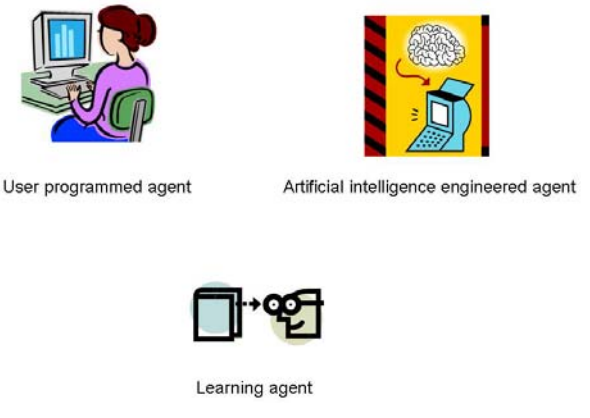


Fig. 2.13 Maes' agent classification

2.10.4 Classification According Agent's Functions

In our work, we classify agents according to their function within the system. We distinguish the following types of agents:

1. Interface agent
Interface agent is involved in tasks such as translation of human queries into computer-understandable language, translation of the information retrieved by other agents into human-understandable language, presentation of this information to the users in an appealing and interesting way etc.

2. Manager agent
Manager agent has the task of assigning tasks to other agents of the systems. In order to do this, it needs to know:

(1) the task structure, so that it can decompose the overall problem into smaller problems; and
(2) the functions, capabilities and resources of all agents under its management, so that it can assign tasks to these agents accordingly.

Problems are broken down into smaller problems that are assigned to individual agents. Those agents cooperate and coordinate their activities to ensure that the problems are solved efficiently.

3. Executive agent
Executive agents perform tasks specific to the system in which they operate. For example, in the information retrieval system, an information agent will play the role of the executive agent. Such a system is represented in Figure 2.14. Here, information agents provide access to information repositories, such as Web sites

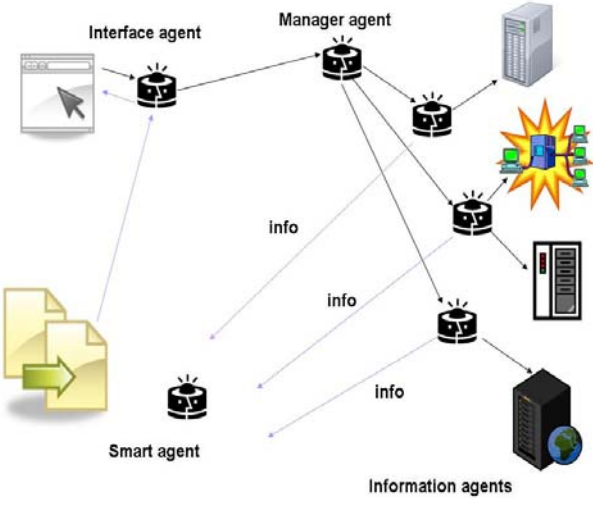


Fig. 2.14 Classification according to agents' tasks

or databases, and answer queries about their content. Information agents collect and manipulate the information obtained from those sources in order to answer queries posed by users and other information agents. An information agent has access to at least one and potentially many information sources. The information agent needs to filter the information from those information sources before it stores or forwards this information to some receiving destination.

Another example of executive agents are Product Specification agents that could accompany a product during its production and assembly so that a specially customized products rules and specifications are available to each stage of the production process.

4. Smart agent
Smart agents analyse the target information using various techniques. Data Mining and Learning agents from the previous classification fall under this category. In Figure 2.14, the Smart agent integrates the relevant information into an overall solution and sends this information to the interface agent to present it to the user.

2.11 Conclusion

In this chapter we introduced agents and described their main features. The agents are situated in an environment. For the most currently designed agents, this environment is non-deterministic, inaccessible, dynamic and continuous.

The most important characteristics of agents are their autonomy, social ability, reactivity and pro-activeness. These features form a strong basis for the design of highly effective and efficient systems.

The internal data structure of agents consists of descriptions of domain knowledge, its environment and its history files. The content of these components is the major factor in the decision-making process.

We discuss the mobility of agents and different types of migrations. The migration can be state-oriented or stateless, and predetermined, dynamically determined and hybrid determined.

Agents may be dependent on each other in regard to their common goal. Dependency relationships between agents take the following forms: independence, unilateral, mutual and reciprocal dependence.

If the agents are to collaborate with each other, they need to understand each other and speak the same language. We gave examples of some Agent Communication Languages and briefly mentioned that ontologies can be used in agents' communications.

The problem-solving within a multi-agent system is a complex process consisting of the following stages: problem decomposition, atomic problems solution, result sharing, result analysis and solution synthesis.

We also discuss different ways of classifying agents: Haag's, Dillenbourg et al.'s, Maes' and classification of agents according to agents' functions.

Agents need to be intelligent in order to carry out their tasks efficiently and effectively. We did not discuss any details regarding agents' intelligence in this

chapter. In the following chapter, we will introduce ontologies. Ontologies take the role of an agent's brain; they support the design of intelligent agents by providing the knowledge an agent needs to carry out its tasks intelligently.

References

1. Aparicio, M., Chiariglione, L., Mamdani, E., McCabe, F., Nicol, R., Steiner, D.H.S.: FIPA - Intelligent agents from theory to practice (1999), http://www.chiariglione.org/leonardo/publications/telecom9 9/telecom99.htm (retrieved: March 16, 2005)
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE- A FIPA compliant agent framework. In: Proceedings of Practical Applications of Intelligent Agents and Multi- Agents (PAAM 1999), pp. 97–108 (1999)
3. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA compliant agent framework. Software Practice and Experience 31, 103–128 (2001)
4. Dale, J.: A Mobile Agent Architecture to Support Distributed Resource Information Management. University of Southampton, Faculty of Engineering (1997), http://eprints.ecs.soton.ac.uk/825/04/dale1996a.pdf (retrieved: March 14, 2005)
5. Dillenbourg, P., Jermann, P., Schneider, D., Traum, D., Buiu, C.: The design of moo agents: Implications from a study on multi-modal collaborative problem solving. In: 8th World Conference on Artificial Intelligence in Education (AI-ED 1997), pp. 15–22 (1997)
6. Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: A tool for collaborative ontology construction. International Journal of Human–Computer Studies 46, 707–727 (1997)
7. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Proceedings of the third international Conference on Information and Knowledge Management (CIKM 1994), pp. 456–463 (1994)
8. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5, 199–220 (1993)
9. Haag, S., Cummings, M., McCubbre, D.J.: Management Information Systems for the Information Age. McGraw-Hill, New York (2002)
10. Labrou, Y., Finin, T., Peng, Y.: Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems 14, 45–52 (1999)
11. Maes, P., Shneiderman, B., Miller, J.: Intelligent Software Agents vs. User-Controlled Direct Manipulation: A Debate. In: Conference on Human Factors in Computing Systems, USA (1997)
12. McGuinness, D.L., Fikes, R., Hendler, J., Stein, L.A.: DAML+OIL: An Ontology Language for the SemanticWeb. IEEE Intelligent Systems 17, 72–80 (2002)
13. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
14. Smith, J.: A Survey of Process Migration Mechanisms. ACM Special Interest Group on Operating Systems 22, 28–40 (1988)
15. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley and Sons, Chichester (2002)

Chapter 3
Introduction to Ontology

3.1 Introduction

It is important for agents to communicate and interact with each other, especially if they are part of the same multi-agent system. In most cases, different agents are working collaboratively towards the same goal. They need to talk to each other, share tasks, exchange results etc. Here, it is important that agents understand each other; for example, they need to speak the same language or be able to translate and understand the language spoken by other agents.

Ontologies are used to establish effective communication between different agents. Ontologies specify the terms used in agents' communication and provide the exact meaning of those terms relative to other ontology terms and within a specific context. Ontologies provide the agent with the domain knowledge and enable it to function intelligently.

In this chapter, we will introduce ontologies. We will provide a definition of ontology and explain associated terminology such as ontology commitments, ontology representation, ontology classification; we will give a formal description of ontologies and ontology design criteria.

3.2 Ontology Origins

The term "ontology" has its origins in metaphysics and the philosophical sciences. Aristotle first introduced ontology as a philosophical discipline that investigates existence or being (Corazzon 2000). Ontology explains the nature and essential properties and relations between all beings. Ontology is based on the truth; its nature being independent of one's background, present understanding, perspective and knowledge of the world.

Artificial Intelligence researchers first borrowed the concept of ontology from Philosophy (Russel and Norvig 1995) (Figure 3.1). Since then, the notion of ontology has become a matter of interest to information and computer scientists in general. In information and computer science literature, the term "ontology" takes on a new meaning that is partly related to its philosophical counterpart.

Usually, a team of researchers who need to share information in a specific domain propose an ontology. The researchers may be inspired to design a common ontology for various reasons (Gruber 1995) such as:

- common understanding of the domain knowledge (knowledge within a community of people committed to a common goal)

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 37–60.
springerlink.com

Fig. 3.1 Ontology originates from Philosophy and Metaphysics

- sharing of the domain knowledge
- reusing of the domain knowledge
- analyzing domain knowledge
- separation of domain knowledge from operational knowledge
- making domain assumptions explicit

3.3 Ontology Definition

Different ontology definitions can be found in the computer and information science literature. But all researchers agree on the importance of ontology in the representation, sharing and reuse of existing domain knowledge (Gómez–Pérez 1998).
The most prevalent ontology definition is the following (Gruber 1993).

> "*A body of formally represented knowledge is based on a conceptualization… A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. An ontology is an explicit specification of a conceptualization*."

We analyze Gruber's ontology definition in more detail. Three main concepts are central to this definition: "formal", "domain conceptualization" and "explicit". "Formal" refers to knowledge representation that is mathematically described and machine readable. A "domain conceptualization" is an abstract model of a phenomenon, i.e. an abstract view of domain concepts and relationships among them. "Explicit" expresses clear and precise definitions of concepts and their relationships.

Gruber puts further emphasis on ontology sharing. There exists an agreement between ontology users that the ontology represents consensual knowledge. This knowledge is not related to an individual, but is accepted by a group (Fensel 2001).

In the same article, Gruber states further:

> "*the set of objects that can be represented is called the universe of discourse… definitions associate the names of entities in the universe of*

discourse (e.g., classes, relations, functions, or other objects) with human-readable text"

Here, terms "objects" and "entities" are mentioned within an ontology context. This can be quite confusing for people who are being introduced to the concept of ontology for the first time and have a background in entity-relationship and object-oriented modelling. In this book, we will avoid using terms associated with other modelling techniques.

We adopt the following definitions of concept and term:

Concept is a unit of thought.

Term is a lexical representation of a concept.

We suggest the use of "concept" instead of "object" and of "term" instead of " entity". We also prefer and agree to use "relationships" between concepts rather than "relations" between concepts.

Ontologies are used to describe and represent the domain knowledge. Ontology represents a shared understanding of domain knowledge and consists of concepts used to describe the domain knowledge and the relationships between those concepts. Formal axioms are used to constrain the interpretation and well-formed use of the defined concepts. Terms are used to designate the concepts and the relationships in the universe of discourse (Gruber 1993). Every branch of science has its own ontology.

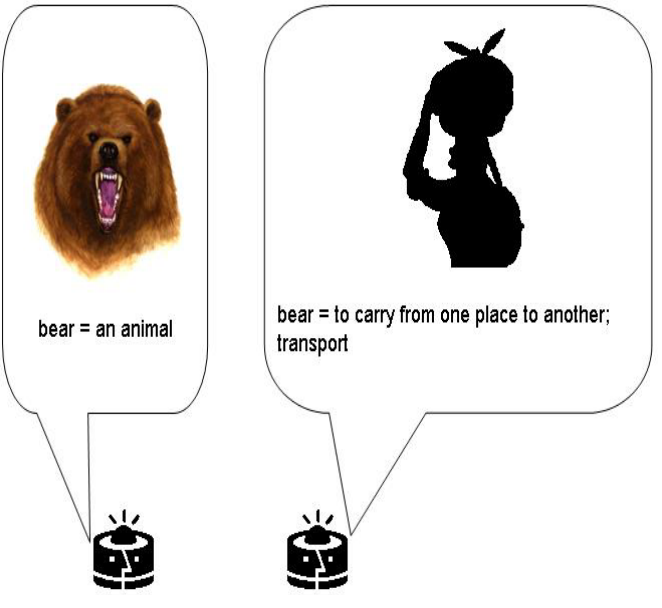


Fig. 3.2 Two different meanings of the same term- i.e. representing two different concepts

3.4 Ontology Commitments

In order to communicate effectively, users, applications and agents need to have a common and shared understanding of the terms used in communication. The terms need to have the same meaning for all parties involved in communication. For example, consider the word "bear". For one agent, this term is associated with the concept of bear being an animal, while for the other agent this term is associated with the concept of the action of carrying something (Figure 3.2). How is it possible for these two different agents to reach an agreement if, for each of them, the same term has a different meaning? We need to ensure that all parties involved in the communication share the same ontology and have a common understanding of all terms used in conversation.

Ontological commitment is described as the agreement about the concepts and relationships between those concepts within the ontology (Gruber 1993). When a user, application or agent commit to an ontology, there is an agreement with respect to the meaning of the concepts and relationships represented. Furthermore, there is agreement to use the shared vocabulary coherently and consistently. Ontological commitment to the semantics (meaning) of the concepts and their relationships in the common ontology includes commitments to the axioms, rules, constraints etc. stated in the ontology.

Even though the Agent Communication Language (ACL) is generally used for agents' communication (Labrou et al. 1999; Finin et al. 1994), different agents can communicate with each other using the same ontology. When different agents commit to an ontology, this common ontology defines the vocabulary and enables the different agents to exchange the information efficiently (Van Aart et al. 2002). The agents can meaningfully communicate about a domain even though they may be designed and used by different applications (Stuckenschmidt 2002).

It is also possible for an agent to commit to several ontologies. In some situations, an agent can commit to several ontologies at the same time where each ontology is used in communication with a particular agent. In other situations, different agents' ontologies may contain knowledge complementary to each other. Those ontologies can then be merged into a large ontology that will serve as a common ontology for those agents. Of course, here it is very important to have these different ontologies formatted in the same way so that the agents can be designed in correspondence with this format and be able to easily accept and process the new ontologies.

Ontology commitments play an important role in the ontology design. We will give some examples to illustrate this. As we mentioned in the first chapter, the same term may have different meanings in different domains (Figure 1.5). Also, the same term can be defined differently within different ontologies from the same domain even though its meaning is the same; i.e. they refer to the same concept. For example, within Genetics the concept "gene" can be defined in different ways (Figure 3.3). For this reason, it is crucial that ontologies be designed by a number of domain experts together with ontology engineers. Ontology commitments need to be carefully considered and evaluated against the project objectives.

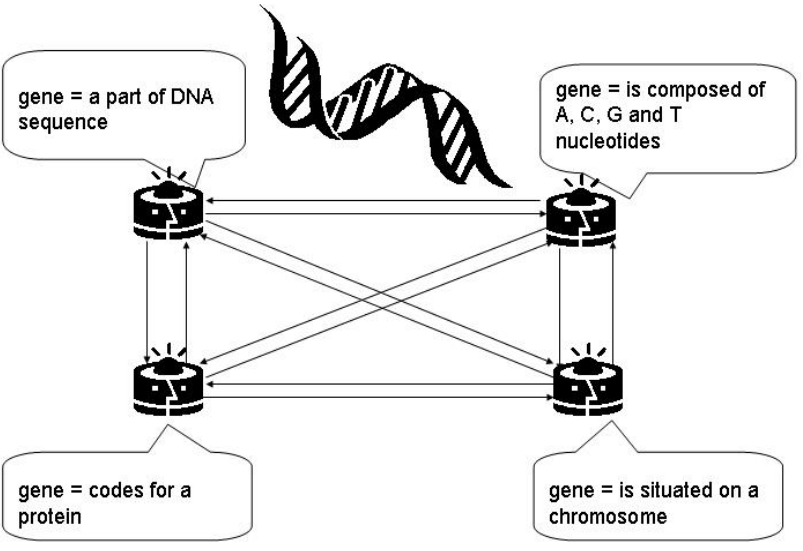


Fig. 3.3 Different definitions of the same concept

3.5 Ontology Community

Ontologies represent how a specific community understands part of the world. It is important for all members of the community to reach an agreement on (1) their understanding and definition of domain knowledge; and (2) representation of this knowledge through an ontology. A definition may be added to an ontology only with the consensus of all community members. No member can alter or override the definitions in an ontology according to his/her preferences. Community members, when committed to their common ontology, are able to use this ontology for various purposes such as the sharing and using of the available knowledge.

Different members of a community can share and reuse existing ontologies. Many ontologies are already available in electronic format. These ontologies can be imported into a new ontological development environment and can be reused by another application.

Any community can commit to ontologies developed by other communities. A group of communities can accept and commit to an ontology with more general terms and fewer constraints. These communities can use such general ontologies to develop their own specialized ontologies.

3.6 Generalization/Specialization of Ontologies

Generic ontologies contain less detailed information than do specialized ontologies. Usually, more detailed specialized ontologies are derived from the generic ontologies (Figure 3.4). These new specialized ontologies are more detailed as they refine and extend the general descriptions present in generic ontologies.

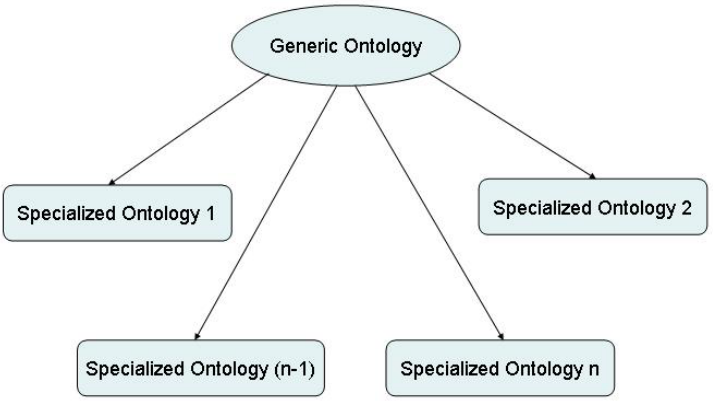


Fig. 3.4 Ontology specialization

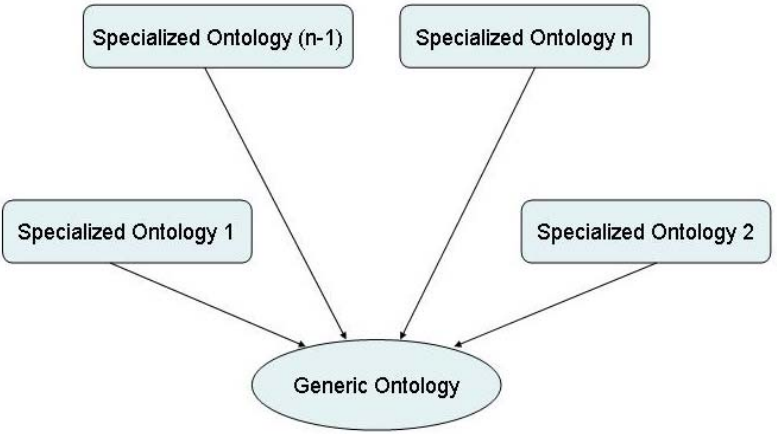


Fig. 3.5 Ontology generalization

Generic ontologies are more likely to be shareable by a large number of communities. Usually, specialized ontologies are for a smaller community within a larger community. A generic ontology consists of a minimal number of axioms, while a specialized ontology has a large number of axioms and needs a very expressive language. It is possible to progress gradually from generic to specialized ontologies through an incremental increase of the number of axioms.

Adding constraints can specialize an ontology to be used by a specific community. In the "Ontolingua", a library of ontologies, there are ontologies with general terms and minimum constraints that can be used by many communities to develop specialized ontologies (Farquhar et al. 1997). One can add new terms along with their definitions to specialize an ontology for a subcommunity (Visser et al. 1998).

We can also build a generic ontology which is based on a number of specialized ontologies. Such a process is referred to as "ontology generalization" and is shown in Figure 3.5. This process aims to resolve the conflicts between definitions of terms in the ontologies. The result of such a process is an ontology that can be accepted by the participating communities.

3.7 Properties of Ontologies

Some essential properties of ontologies are:

- ontologies describe a specific domain
- ontology users agree to use the ontology terms consistently
- ontology concepts and relations are unambiguously defined in a formal language by axioms and definitions
- relationships between ontology concepts determine the ontology structure e.g. hierarchical or non-hierarchical. The generalization/specialization relationship (i.e. "is-a" relationship) between two concepts is an example of a hierarchical relationship between concepts.
- ontologies can be understood by computers

Ontology has different meanings within different contexts (Figure 3.6). In Philosophy and Metaphysics, the ontology encompasses nature and existence, beings and relations between beings. The resulting knowledge is explicit, shared and easily understandable by humans. In Computer Science and Artificial Intelligence, an ontology is used to formally and explicitly represent shared domain knowledge through definitions and axioms of concepts and relationships between the concepts. The main difference is that in Computer Science and Artificial Intelligence literature, ontology is designed to be understandable by machines.

Two important functions of ontologies are that they:

1) enable agents to work cooperatively to communicate with each other
2) make the available information more accessible to automated agents

Agents working cooperatively can communicate with each other by means of a common ontology. Ontology is machine readable, and supports agent communication by defining and providing a shared vocabulary to be used in the course of communication. Ontologies not only provide a definition of the terms that can be used in communication; ontologies also provide the definition of the world in which an agent grounds its actions. Different agents of a system can reach a shared understanding by committing to the same ontology. This enables them to make statements, communicate knowledge and make different queries. Use of ontology permits coherent communication and easier information sharing between different agents, enabling agents to cooperate and coordinate their actions.

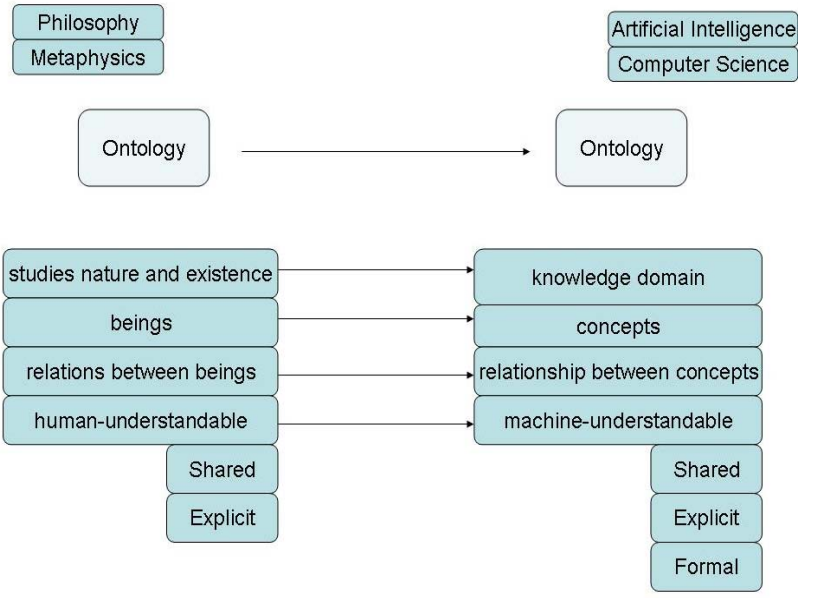


Fig. 3.6 Meaning of ontology in two different contexts

If the available information is to be more accessible to automated agents, it is essential that the meaning of this information be understood by such agents. People and software systems would be able to extract, analyze and apply information from different organizational sites if those sites operate on the basis of the same underlying ontology. Adding information that describes Web content in a machine-understandable way enables Web information resources to become more accessible to automated processes (Horrocks et al. 2002). Typically, an ontology has a hierarchical structure, and describes domain concepts and their properties. As such, an ontology is a collection of shared and precisely defined terms that can be used in meta-data to describe Web content in a machine-understandable manner. Through this, ontologies play an important role in efficient and effective information access and retrieval.

3.8 Characteristics of Ontology Models

Analogous to the characteristics of the data models (Chandrasekaran et al. 1999), we developed a similar scheme for the ontology models:

- Concept is a unit of thought. Concepts may have abstract meaning (e.g. human diseases) and physical meaning (e.g. gene).
- Term is a lexical representation of a concept (or name of a concept).

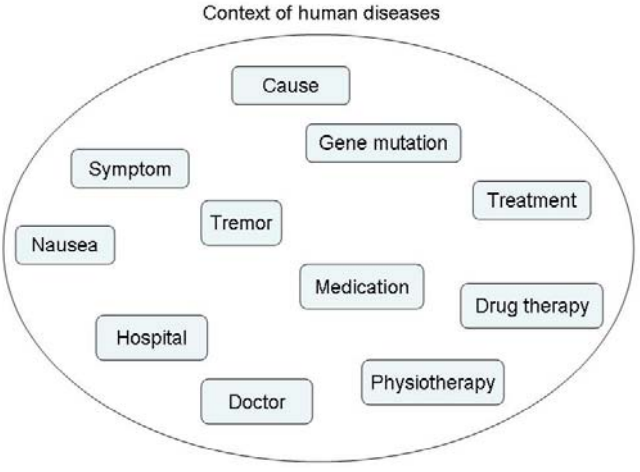


Fig. 3.7 Several concepts in the context of human diseases

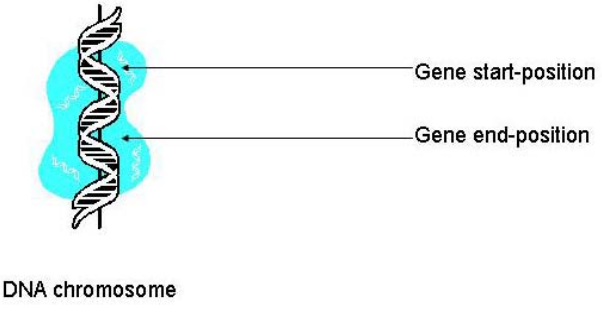


Fig. 3.8 Gene is situated on a DNA chromosome

- Context is used to group terms in an ontology. For example, in the context of human diseases, we group terms used to describe and represent the knowledge of human diseases (Figure 3.7).
- Terms exist in various relationships with each other. For instance, the term 'gene' is related to the term 'DNA chromosome' because a gene is situated on a DNA chromosome (Figure 3.8).
- The knowledge representation changes over time. The level of detail of the ontologies is related to the level of detail of the information. It is often necessary to filter and omit details of information in order to make the situation clearer by representing only information relevant to the concept within the context. At other times, this is inadvisable because the requested knowledge can be found only in the detailed information (Langacker 1987). For example, when determining the exact position of a gene within human DNA, we firstly

locate a chromosome (out of 23 different human chromosomes) that contains this gene. At this stage, the ontology associates only a chromosome with the term "gene position". Secondly, we locate an arm of this chromosome (p or q arm) where the gene of interest is situated. At this stage, the ontology associates p or q chromosome arm with the term "gene position". Finally, we locate the exact region within the chromosome arm, such as p11. At this final stage, the ontology associates the exact region of a chromosome with the term "gene position". The knowledge model that describes the position of a gene within human DNA needs to be updated as new knowledge becomes available. Generally, the knowledge model needs to be updated when new knowledge emerges.

- In processes such as information retrieval, information needs to be integrated at different levels of detail. It may be necessary to navigate at different levels in the ontology structure, select the relevant information and integrate this information in a controlled way. Information may need to be combined at different levels of detail but merged on a specific level. A mechanism for integrating ontologies should provide these services.
- Incorporation of a new concept within an ontology may require the presence of an additional concept. Similarly, the deletion of an ontology concept may make the presence of another ontology concept redundant. For example, gene mutation is a change of the gene structure which may cause a disease. If we introduce the term 'gene mutation' into an ontology, we may also need to introduce the term 'human disease'. Because the ontological model is a semantic network of concepts and their relationships, the model extension may change the ontology structure significantly.

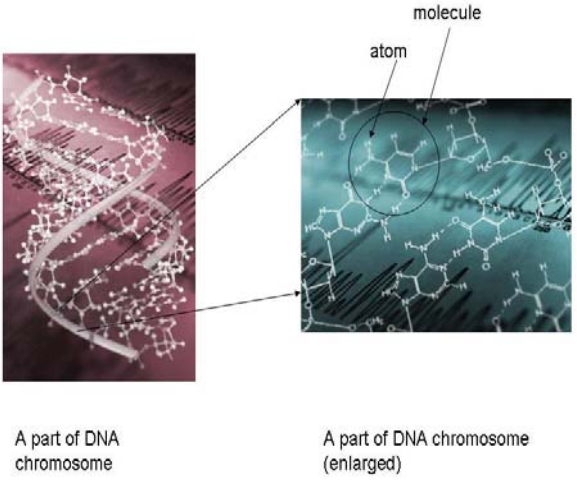


Fig. 3.9 Double-helix DNA is composed of molecules which are composed of atoms

- Different terms may be incorporated into the ontology to enable the definition of a new term. For example, as we show in Figure 3.9, human DNA is a double-stranded helix composed of four different nucleotides that consist of a base (Adenine (A), Cytosine (C), Guanine (G) and Thymine (T)), a phosphate molecule and a sugar molecule (deoxyribose). The nucleotides are grouped together in a helix forming human DNA. Different atoms are grouped together to form a molecule and different molecules are grouped together to form DNA. The level of detail described by an ontology will depend on the purpose of the ontology design.

3.9 Representation of Ontology Domain

Hierarchies have been predominantly used to represent ontologies. Two main reasons for this are: (1) hierarchies are similar to the way we organize the mental models of the world in our minds; and (2) hierarchies allow for the generalization/specialization mechanism in information processing.

In the example given in Figure 3.10, we represent the top-level of the 'Nutrients' ontology that describes the basic classification of nutrients as carbohydrates, proteins and fats. At the top of the hierarchy, we have the concept "nutrients". On the next level down, we have three concepts: "carbohydrates", "proteins" and "fats". The reason for the choice of these particular three different groups to classify the concept "nutrients" depends on many factors such as the purpose and use of the ontology. We will discuss this in Chapter 7. The ontology has further branches as follows: "carbohydrates" can be "starch", "sugar" or "fibre", "proteins" can be "incomplete protein" or "complete protein"of vegetable or animal protein respectively. "Fats" can be "saturated fats", "unsaturated fats", "dietary cholesterol" and "trans fatty acids". The knowledge in the 'Nutrients' ontology is represented through hierarchical relationships between different concepts.

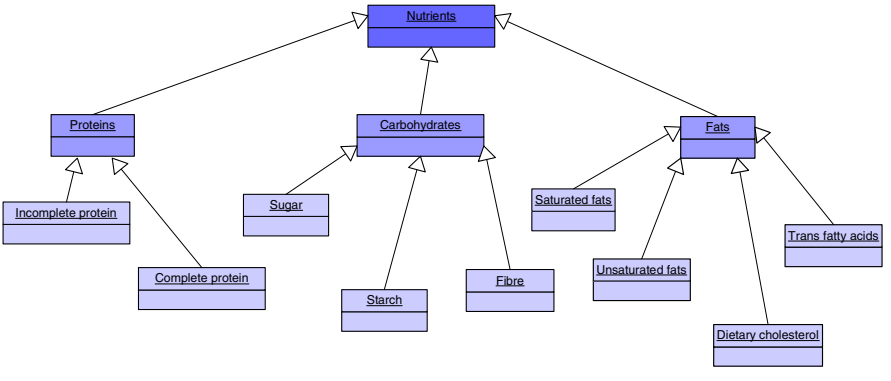


Fig. 3.10 Top-level of 'Nutrients' ontology

Although ontologies often assume the form of a taxonomic class hierarchy, they are not restricted to hierarchies in any way. In fact, ontologies may take on much more general and complex structures (Gruber 1995). The ontology concepts may be grouped together to form layers of related hierarchy trees, networks, cubical structures or any other shapes. The way the ontology concepts are grouped together depends on the purpose of the ontology design and on the complexity of the knowledge that it represents.

3.10 Ontology Design Versus Data Modelling

Ontologies and data models both represent domain knowledge but exhibit some fundamental differences. Useful points of reference when making the comparison are (Spyns et al. 2002):

1) Application dependencies
2) Knowledge coverage
3) Expressive power
4) Operation levels

Ontology and data models differ from each other in their dependencies on the application(s) they are designed for. Data models are designed to fulfil specific needs and therefore depend on the specific tasks that need to be performed. Users, goals, purposes and intended use of the model influence the modelling process and the level of detail described by this model. In contrast, ontologies are as generic and task-independent as possible. Minimum of application requirements are being considered during the ontology design. For example, an important property of a gene is the fact that gene codes for a protein. If the application does not require this knowledge, the data model will be designed without it. On the other hand, ontologies represent agreed and shared knowledge. So, the fact that gene codes for a protein will need to be included in the gene ontology. Ontology consists of relatively generic knowledge that can be reused by different kinds of applications across a community of users.

An ontology and data models differ from each other in the amount and kind of knowledge they cover. Data models focus on the establishing correspondence between organization of the data in databases and concepts for which the data is being stored. On the contrary, ontologies are concerned with the understanding of the knowledge by community members. Ontologies are a more complete representation of concepts and relationships. Ontology knowledge is considered to be the hard coded part of a computer system (Greiner et al. 2001). Knowledge understood to be true within a certain domain does not change relatively to the way this knowledge is organized and stored within a computer system. The data represent a dynamic part of a computer system which may change even during run-time (Greiner et al. 2001).

Ontologies have more expressive power compared to the data models. Data modelling languages use typical language constructs. In contrast, ontology languages are more expressive languages as they include constructs that express other kinds of meaningful constraints such as taxonomy or inferencing. Ontology languages make the domain conceptualization more correct and precise.

Ontology and data models operate on different levels. Ontologies are generic, task- and implementation-independent and as such operate on a higher level of abstraction. In contrast, data models are at the lower level of abstraction. Thus a 'new car database' would model a car using attributes such as model and maker of car, pricing, colour, engine size, accessories. In contrast, a 'repair database' might model spare parts associated with a particular model and make of a car, their price, the price for the labour to fit in the spare part, result ranges for certain tests, etc. Note each of these data modelling have a narrow view of the concepts of a car. As the ontology is the knowledge shared by different applications across a community, we can design a Car Ontology which will represent the knowledge common and shared by both 'new car database' and 'repair database'. This knowledge will operate on a higher level of abstraction and will be designed independently from the applications running on each of these databases.

Differences between the ontologies and data models, the characteristics of application dependencies, knowledge coverage, expressive power and operation levels, can be used by data modellers and ontology designers as points of reference. These guidelines will help them stay focused on their own goal.

Because ontologies and data model operate on the different levels but are oriented in the same direction, it is possible to:

1) use ontologies to support data model design
2) use data models to elicit knowledge that could be useful for ontology design
3) integrate ontologies with data models
4) use ontologies as means of working with and querying databases with different data models in the same domain (see Figure 1)

Ontologies can be used to build, support and clarify data models. An ontology can play an important role during the data model design, especially in the requirements analysis. A wide range of ontologies is available via ontological libraries and can be used to assist in the modelling of a specific application domain.

Also, data models may support the ontology design process. Data models may help to provide knowledge that will help designers make decision regarding organization and structuring of the concepts within an ontology. The ontologies designed through support of data models are usually specialized ontologies and not general ontologies. The number of applications for which these specialized ontologies can be used is limited.

The integration of ontologies with data models into a unified meta-model can result in an effective combination of data, documents and formal knowledge (Kuhn and Abecker 1997). The main goal of this integration is to enrich the data models and increase the expressiveness of the domain under analysis. The application designer needs to identify ontologies from the ontological libraries that are suitable to be used for a portion of the domain which needs to be formally described.

3.11 Ontology Versus Knowledge Base

The difference between an ontology and a knowledge base can be seen in their different objectives (Maedche 2003). An ontology captures and represents the

conceptual structure of a domain in a form that is shared by the community of users in that domain. In contrast, a knowledge base (Dillon and Tan 1993) seeks to model knowledge in that domain in a form suitable for a particular Knowledge-Based System (KBS) to carry out problem solving. Knowledge base models the problem solver's (expert's) approach to problem solving for the particular set of problems being addressed within this domain. An ontology has terminology and conceptualization agreed by the community to be correct and to be consistently used across the community. Whilst in the KBS, the terminology can be specific to the particular system and consistency is understood to mean logical consistency within the narrow focus of the problems being solved. The knowledge base enables an inference engine or agent to reason in order to solve the particular set of problems that the KBS is meant to address.

The main differences between ontology and knowledge bases are summarized in Table 3.1.

Table 3.1 Ontology versus knowledge base

	Ontology	**Knowledge base**
objectives	conceptual structures of a domain	particular states of a domain
consistency	facts always true	facts true for a particular state of affairs
actions	communication	problem solving
knowledge	domain knowledge	operational knowledge & restricted domain knowledge
applicability	shared by community	specific to system

The purpose of an ontology is to describe facts assumed to be always true by the community of users. It aims to capture the conceptual structures of a domain. In contrast, a knowledge base aims to specify the portion of the domain that is useful for solving a particular set of problems. It may also describe facts related to a particular state of affairs.

For an agent, a shared ontology describes a vocabulary for communicating about a domain. In contrast, a knowledge base contains the knowledge needed to solve problems or answer queries about such a domain. Domain knowledge is described by ontologies while operational knowledge is described in the knowledge base. An ontology specifies knowledge about a certain domain while a knowledge base specifies knowledge used by the agent and/or an inference engine to perform its actions.

Recently, the use of ontologies in preference to knowledge bases for the agents' operational knowledge has been proposed. This would result in two different kinds of ontologies: domain ontology and operational ontology (which replaces the knowledge base). Let us consider an example to clarify the difference between domain and operational knowledge. We may be assigned the task of delivering some books to a certain address. We may have the domain knowledge about the

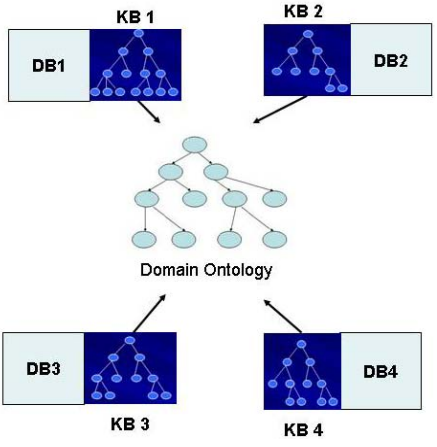


Fig. 3.11 Different databases share same ontology

street locations in the city. Still, we may not able to complete this task if we lack the operational knowledge (i.e. if we do not know how to perform the action of locating the address and delivering the books). Domain and operational knowledge are both needed by an agent if it is to function optimally.

In Figure 3.11, we show an example where each database (DB) of the four different databases has specific applications associated with it and uses its own knowledge base (KB) to perform the specific actions. The knowledge base of one database is independent from the knowledge base of another database. A new application is proposed where the four different databases need to collaborate with each other. A domain ontology needs to be designed for this purpose and each of the four databases needs to commit to this common ontology. This ontology would enable effective and efficient cooperation and collaboration of the four databases. The databases would be able to communicate with each other and share the information, tasks and results efficiently.

3.12 Classification of Ontologies

Different types of interpretation are associated with the concept of ontology. The knowledge expressed by different ontologies might be the same, but the ontologies may be different from each other in the way in which this knowledge is expressed. The ontologies may differ from each other in regard to their:

1) degree of formality
2) degree of granularity
3) level of generality
4) amount, type and subject of conceptualization
5) expressiveness

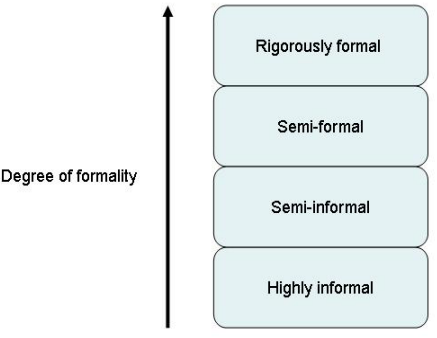


Fig. 3.12 Classification of ontologies according to degree of formality

3.12.1 Degree of Formality

Ontologies differ in the degree of formality with which the terms and their meaning are expressed (Uschold et al. 1996). As shown in Figure 3.12, ontologies are classified into following four categories:

1) highly informal ontologies
2) semi-informal ontologies
3) semi-formal ontologies
4) rigorously formal ontologies

- Ontologies expressed in natural language are "highly informal" ontologies. Due to the ambiguity of natural language, term definitions in "highly informal" ontologies may be ambiguous, and therefore difficult for a software agent to work with.
- Ontologies expressed in a restricted and structured form of natural language are "semi-informal" ontologies. Restricting and structuring of natural language helps to reduce the ambiguity and improve the clarity.
- Ontologies expressed in artificial languages which are formally defined are "semi-formal" ontologies.
- Ontologies whose terms are precisely defined through formal semantics and theorems, and have desired ontology properties (such as soundness and completeness) are "rigorously formal" ontologies.

Having an informal ontology makes it easier to understand the motivation and idea behind the ontology design, and to discuss possible modifications that need to be introduced into the ontology. The computers cannot use this informal ontology. In order for an ontology to be used by computers and by various applications, it needs to reach a stage where it is formally described. Only when formally described, can an ontology reach its ultimate goal of being used by some program/application(s).

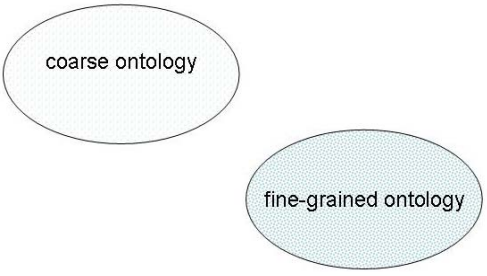


Fig. 3.13 Classification of ontologies according to degree of granularity

3.12.2 Degree of Granularity

The ontologies can be classified according to their granularity (see Figure 3.13) in the representation of domain concepts into:

1) coarse ontologies
2) fine-grained ontologies

A coarse ontology is shared among users who already agree on the underlying conceptualization of the domain. It consists of a minimal set of axioms written in a language of minimal expressivity and supports only a limited set of specific services.

A fine-grained ontology is used to establish agreement on the underlying conceptualization of the domain between the ontology users. In contrast, fine-grained ontology consists of a large number of axioms written in a language of extensive expressivity, to support a variety of services.

3.12.3 Level of Generality

The ontology types, shown in Figure 3.14, are classified into four groups according to the level of generality used in the description of a domain (Guarino 1998):

1) application ontologies
2) task ontologies
3) domain ontologies
4) top-level ontologies

- Vocabulary related to both a particular domain and a particular task is described by "application" ontologies. They are often a specialization of both domain and task ontologies.
- Vocabulary related to a specific task or activity is described by "task" ontologies.
- Vocabulary related to a particular domain is described by "domain" ontologies.
- General concepts or common-sense knowledge is described by "top-level" ontologies. Those ontologies are independent of a particular problem or domain.

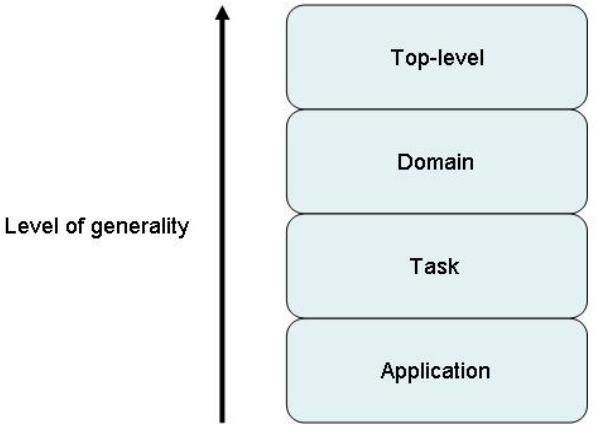


Fig. 3.14 Classification of ontologies according to level of generality

This kind of classification could confuse someone who is new to the ontology field. By the ontology definition, an ontology is used to represent the knowledge of a particular domain and all ontologies are "domain" ontologies as they describe the domain knowledge. It could also be confusing to say that the "top-level" ontologies are independent of a particular domain as this contradicts the ontology definition. Also, an ontology is designed for the purpose of ultimately being used by an application. In order to avoid any confusion associated with this ontology classification, we need to remain aware of the context within which this ontology classification occurs.

3.12.4 Amount, Type and Subject of Conceptualization

Ontologies can be classified according to amount and type of conceptualization structure and subject of the conceptualization (Van Heijst et al. 1997).

1. The amount and type of structure of the conceptualization is mainly concerned with the level of granularity of the conceptualization. In Figure 3.15, we show ontology types associated with this classification:

1) terminological ontologies
2) information ontologies
3) knowledge modelling ontologies

- "Terminological" ontologies are not really ontologies but lexicons used to specify terminology for representation of the domain knowledge. But terminological ontologies do not capture semantics of the terms.
- "Information" ontologies specify the record structure of databases and enable their controlled use and management. But information ontologies do not define the concepts that are instantiated by database instances.

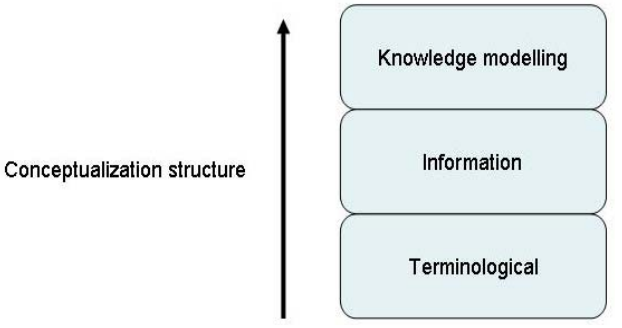


Fig. 3.15 Classification of ontologies according to conceptualization structure

- "Knowledge modelling" ontologies specify conceptualizations of knowledge. They are often specified according to a particular use of the knowledge they describe. Knowledge modelling ontologies are structurally richer than information ontologies.

In the early stages of ontology acceptance by the computer and information science community, the definition of an ontology and the purpose of its design should be clearly stated. Why do we compromise the ontology definition? If something is really not an ontology (like "terminological ontologies" mentioned above), why do we support the existence of these kinds of "ontologies" if they are not ontologies? In doing so, we only dilute and hide the real importance of an ontology. The concept of ontology needs to stand independently from other similar concepts (such as lexicons) in order for its true and full significance to be appreciated. If we hold back on what we already know, we cannot fully progress towards the point where the ontology is taking us. Moreover, why do we need to define a class of "information ontologies" if the same can be done by UML or ORM modelling techniques? By using ontology for purposes other than those for which it was introduced, the full significance of the ontology technology is obscured and prevented from achieving its full potential. We need to embrace the true ontology definition, know the ontology's purpose and reason for its existence.

2. The subject of the conceptualization concerns the type of knowledge that is modelled in the ontologies, and is further subdivided into:

1) application ontologies
2) representation ontologies
3) domain ontologies
4) generic ontologies

- "Application" ontologies contain concepts necessary for knowledge modelling required for specific applications. Usually, these ontologies take terms from more general ontologies and extend this knowledge by representing method- and task-specific components.

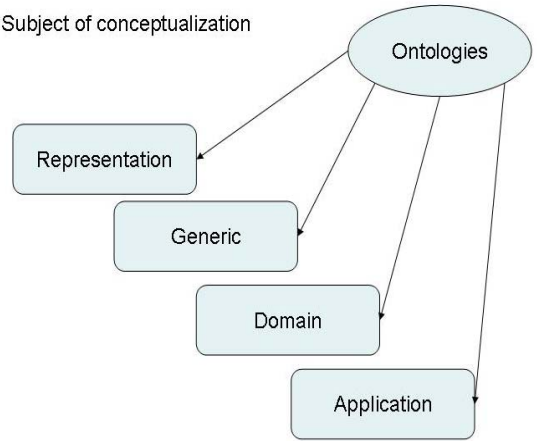


Fig. 3.16 Classification of ontologies according to subject of classification

- "Domain" ontologies contain concepts specific to a particular domain. Domain ontologies specify constraints to be applied on the structure and the content of the domain knowledge. The design of domain ontologies is based, and depends on, the domain knowledge.
- "Generic" ontologies specify concepts that are generic across many fields. Specialization of "generic" ontologies for a particular domain results in " domain" ontologies.
- "Representation" ontologies make no claims about the world and are neutral with respect to the world. They provide a representational framework; " domain" and "generic" ontologies can be described using the primitives given in the representation ontologies.

This classification is shown in Figure 3.16. This classification according to the subject of the conceptualization is similar to the classification according to the level of generality which is shown in Figure 3.14.

3.12.5 Expressiveness of Ontologies

Ontologies can be classified on the grounds of their expressiveness (Lassila and McGuinness 2001) into the following categories:

1) controlled vocabularies
2) glossaries
3) thesauri
4) informal is-a hierarchies
5) formal is-a hierarchies
6) formal instances relations ontologies
7) frames ontologies
8) value restriction ontologies
9) general logical constraints ontologies

- "Controlled vocabularies" are the simplest possible notion of ontology. They are nothing more than a finite list of terms with an unambiguous interpretation.
- "Glossaries" are also lists of terms, but their meanings are ambiguously defined. "Glossaries" are usually expressed in natural language statements; they are aimed at humans and cannot be used by computer agents.
- "Thesauri" define the relationships between terms and, in this way, add semantics to glossaries. "Thesauri" do not provide an explicit hierarchical structure, but broader or narrower term specifications can be used to deduce this kind of structure. A computer agent can often interpret the relationships defined in a thesaurus.
- "Informal Is-a hierarchies" provide a general notion of generalization and specialization. But the inheritance cannot be assumed here because concepts of "Informal Is-a hierarchies" are not organized according to "strict subclass hierarchies". This means that an instance of a more specific class is not necessarily an instance of a more general class.
- "Formal Is-a hierarchies" are ontologies where concepts form "strict subclass hierarchies" and the inheritance is always applicable; namely, an instance of a more specific class is also an instance of a more general class.
- "Formal instances relations" ontologies enforce also a strict hierarchical structure. Formal instance relations hold when the ontology content describes ground individuals and their relationships with the concepts they instantiate.

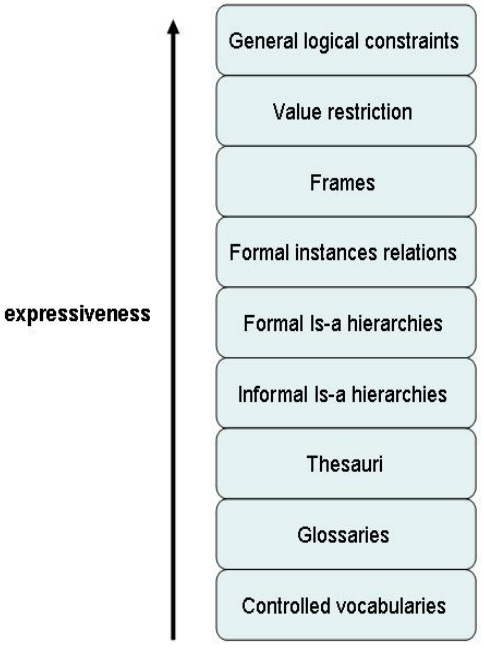


Fig. 3.17 Classification of ontologies on the grounds of their expressiveness

- "Frames" ontologies describe concepts together with their properties. Inheritance can be applied to these properties; more specific concepts down the hierarchy can inherit properties specified for a more general concept.
- "Value restriction" ontologies apply restrictions on the values associated with properties. These restrictions are usually inherited by more specific concepts down the hierarchy.
- "General logical constraints" ontologies are written in very expressive ontology languages permitting a thorough specification of concepts and their properties. Of all the abovementioned ontology types, "general logical constraints" ontologies have the richest expressiveness.

The classification of ontologies according to their expressiveness is shown in Figure 3.17. In this book, classes that cannot be understood by computers such as "controlled vocabularies", "glossaries", some "thesauri" will not be classified as ontologies.

3.13 Conclusion

In this chapter, we focused on the following:

- ontology origins; we compared the meaning of an ontology in Philosophy and Computer Science, and showed main similarities and differences;
- ontology definition; we emphasized the importance of (1) formal representation of the knowledge in order to make the ontology understandable by computers, and (2) agreement in order to make the ontology sharable within the ontology community;
- ontology commitments are another important factors in ontology sharing; users, agents and applications need to commit to the common ontology and agree to share this ontology in a coherent manner;
- generalization versus specialization of ontologies; specialized ontologies contain more details then the generalized ontologies. As a consequence, only a limited number of users can commit to specialized ontologies while the generalized ontologies are designed for a broader community;
- main ontology properties and characteristics of ontology models in terms of its basic elements (e.g. concepts, terms, their relationships, ontology context) as well as more complex ontology features (e.g. knowledge representation, level of detail and ontology extension);
- representation of an ontology domain through hierarchies and other structures;
- distinction of (1) ontology from knowledge base and (2) ontology model from data model; we summarized the main points of reference when doing the comparisons;
- four different types of ontology classification according to the degree of formality, degree of granularity, generality level, expressiveness, and amount, type and subject of conceptualization; we discussed the relevance of this classification in regard to the ontology definition.

In the following chapter, we will discuss some major agent design methodologies to cover different aspects of the design process.

References

1. Greiner, R., Darken, C., Santoso, N.I.: Efficient reasoning. ACM Computing Surveys 33, 1–30 (2001)
2. Chandrasekaran, B., Josephson, J., Benjamins, V.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems 14, 20–26 (1999)
3. Corazzon, R.: Ontology. A resource guide for philosophers (2000), `http://www.formalontology.it` (retrieved: May 20, 2003)
4. Dillon, T., Tan, P.: Object-Oriented Conceptual Modeling. Prentice Hall, Australia (1993)
5. Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: A tool for collaborative ontology construction. International Journal of Human–Computer Studies 46, 707–727 (1997)
6. Fensel, D.: Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, Heidelberg (2001)
7. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: The third international Conference on Information and Knowledge Management (CIKM 1994), pp. 456–463 (1994)
8. Gómez-Pérez, A.: Knowledge Sharing and Reuse. The Handbook on Applied Expert Systems (1998)
9. Gruber, T.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Computer Studies 43, 907–928 (1995)
10. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5, 199–220 (1993)
11. Guarino, N.: Formal Ontology in Information Systems. In: The conference on Formal Ontology in Information Systems (FOIS 1998), pp. 3–15 (1998)
12. Horrocks, I., Patel-Schneider, P.F., Van Harmelen, F.: Reviewing the Design of DAML+OIL: an Ontology Language for the Semantic Web. In: The Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2002), pp. 792–797 (2002)
13. Kuhn, O., Abecker, A.: Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges. Journal of Universal Computer Science 3, 929–954 (1997)
14. Labrou, Y., Finin, T., Peng, Y.: Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems 14, 45–52 (1999)
15. Langacker, R.W.: Foundations of Cognitive Grammar: Theoretical Prerequisites. Stanford University Press (1987)
16. Lassila, O., McGuinness, D.: The role of frame-based representation on the semantic Web: area The Semantic Web. Electronic Transactions on Artificial Intelligence (ETAI) Journal 6(5) (2001)
17. Maedche, A.D.: Ontology Learning for the Semantic Web. Kluwer Academic Publishers, Dordrecht (2003)
18. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
19. Spyns, P., Meersman, R., Jarrar, M.: Data modelling versus Ontology engineering. SIGMOD Record 31, 7–12 (2002)

20. Stuckenschmidt, H.: Exploiting Partially Shared Ontologies for Multi-Agent Communication. In: Klusch, M., Ossowski, S., Shehory, O. (eds.) CIA 2002. LNCS, vol. 2446, pp. 249–263. Springer, Heidelberg (2002)
21. Uschold, M., Gruninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review 2(11), 93–155 (1996)
22. Van Aart, C., Pels, R., Caire, G., Bergenti, F.: Creating and Using Ontologies in Agent Communication. In: The first International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS 2003) (2002)
23. Van Heijst, G., Schreiber, A.T., Wielinga, B.J.: Using explicit ontologies in kbs development. International Journal of Human-Computer Studies 46, 183–292 (1997)
24. Visser, P.R.S., Jones, D.M., Bench-Capon, T.J.M., Shave, M.J.R.: Assessing heterogeneity by classifying ontology mismatches. In: Formal Ontology in Information Systems (FOIS 1998), pp. 148–162 (1998)

Chapter 4
Design Approaches for Multi-Agent-Based Systems

4.1 Introduction

In Chapter 2, we discussed agent-based systems, their characteristics and numerous advantages. We may feel inspired to design such agent-based systems but we need a methodology to guide us through the design process. There are many different multi-agent system design methodologies. In this chapter, we will discuss some of these design methodologies. We will give a brief overview of their different approaches and their important advantages. The chosen methodologies are different from each other and together cover various aspects of the multi-agent system design process.

4.2 Agent Design Criteria

Autonomy, social ability, reactivity and pro-activeness of agents (Wooldridge 2002) (see Section 2.4) can be used as a set of design criteria for multi-agent systems. Each agent needs to be able to perform its tasks autonomously. Moreover, it also needs to be pro-active and perform required actions on its own initiatives. An agent needs to show social abilities, i.e. to interact, cooperate and act in coordination with other agents of the system. Most environments of the agent-based systems are dynamic and the agents needs to timely react on the changes within its environment.

Another three important characteristics of agents are their flexibility, their purpose and ability to learn. The agent needs to be flexible to easily adapt to its changing environment. A rigid agent cannot survive and/or efficiently perform within a dynamic environment. The agent must have a clear purpose defined by its designer which will guide it in its actions. This is a very important factor as some agents may start to function destructively within its own system. And finally, the agent needs to be able to learn, to perfect itself and to strive towards the most optimal performance.

Effectiveness and efficiency of an agent-based system can be evaluated against each of these criteria and each of them needs to be effectively addressed. Failure to adequately address one of these criteria may result in a system whose performance is not satisfactory. These four characteristics of an agent-based system work together in synergy and enable the system to function successfully.

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 61–74.
springerlink.com

4.3 Agent Design Methodologies

Agent-based systems design methodologies can be broadly divided into two groups (Wooldridge 2002), namely those that:

1) extend or adapt existing object-oriented (OO) methodologies
2) adapt knowledge engineering and similar techniques

4.3.1 Belief, Desire and Intention (BDI) Approach

This methodology and the modelling technique is referred to as the Belief, Desire and Intention paradigm (Kinny et al. 1996). This approach is based on the existing object-oriented methodologies but with some important differences. Firstly, unlike the OO methodologies, the BDI approach primarily emphasises the roles, responsibilities, services and goals of different agents. These are the key abstractions enabling the management of complex multi-agent systems. Secondly, unlike the OO methodologies, the BDI approach emphasises the end-point that needs to be reached, rather than the types of behaviours that will lead to the end-point. The focus is on what needs to be achieved and in what context.

The system is approached from two different perspectives:

- From the external viewpoint, agents are modelled as complex objects characterized by their external interaction, the information they require and maintain, the services they perform, and their responsibilities.
- From the internal viewpoint, the elements required by a particular agent, such as an agent's beliefs, goals and plans, are modelled for each agent.

The external viewpoint is captured in two models, namely:

1) Hierarchical relationships between abstract and concrete agent classes are described and the agent instances are identified by an Agent Model.
2) Responsibilities of an agent class, the services it provides, its interactions and relationships between agent classes are specified by an Interaction Model.

For each agent class, the internal viewpoint is captured in three models:

1) The information about an agent's environment, its internal states and the actions it may perform are described by a Belief Model. It consists of a belief set which describes the possible beliefs of an agent and their properties. Particular instances of this belief set are described in the form of belief states.
2) The events to which an agent can respond and the goals that an agent may adopt are described by a Goal Model. The goal and event domain are specified by a goal set. One or more goal states are particular instances of this set.
3) The plans that an agent may employ to achieve its goals are described by a Plan Model. A plan set describes the properties and control structure of individual plans.

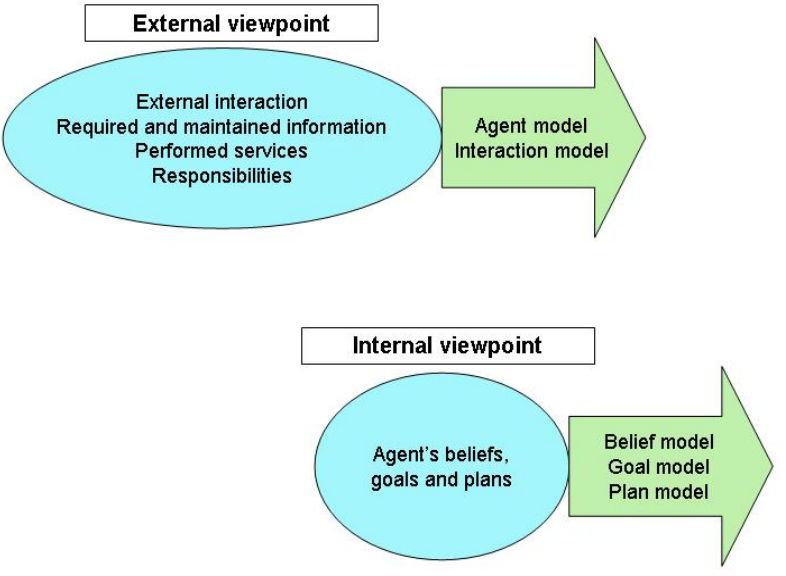


Fig. 4.1 BDI approach

The overview of this methodology is presented in Figure 4.1.

4.3.2 Gaia

The ideas behind the Gaia methodology are inspired by object-oriented analysis and design. The Gaia approach (see Figure 4.2) takes the analyst from abstract to increasingly concrete concepts (Wooldridge 2000).

The main Gaian concepts can be divided into two categories:

- Abstract entities which are used during the analysis phase to conceptualize the system. These entities do not necessarily have any direct realization within the system.
- Concrete entities which are used during the design process and have direct counterparts in the run-time system.

The Gaia methodology puts the emphasis on the analogy between agent-based systems and an organization, and views them as a collection of roles. Those roles stand in a certain relationship to one another and interact with each other in a systematic way. A role is defined by four characteristics:

1) Responsibilities determine functionality of a role and are further divided into two types: liveness properties and safety properties. Liveness properties, in contrast to the safety properties, are variant and are dependent on environmental conditions.

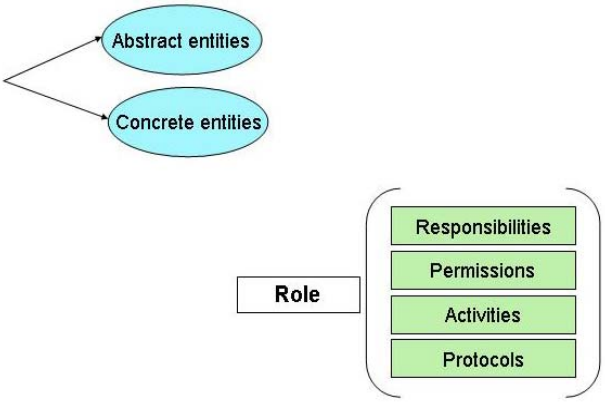


Fig. 4.2 Gaian approach

2) Permissions identify the resources available to that role. The permissions tend to be information resources.
3) Activities are the computations that are specific to that role and are carried out by the agent without interacting with other agents.
4) Protocols define the way in which a role interacts with other roles.

4.3.3 Agent UML

AUML is the Agent UML (Unified Modelling Language). It is a set of UML idioms and extensions that can be used to model multi-agent systems (Odell et al. 2001). AUML adopts a three-layer representation:

1) agent interaction protocols
2) interactions among agents
3) internal agent processing

The three layers of the AUML approach are shown in Figure 4.3

The agent interaction protocols that provide reusable solutions are found in the first layer of AUML. The two UML techniques used to express protocol solutions that can be reused are packages and templates where the modelling elements are conceptually grouped and aggregated in accordance with the purpose. The AUML protocol solutions can be reused in various situations and allow various kinds of message sequencing between agents.

The interaction diagrams are used in the second AUML layer to capture the structural patterns of agents' interactions. Sequence diagrams and collaboration diagrams are two different types of interaction diagrams but they contain similar information. The sequence diagram represents the chronological sequence of the processes that need to be carried out within the multi-agent system, while the collaboration diagram represents the associations between different agents during these processes. The flow of processing in the agent community is captured by activity diagrams and state charts.

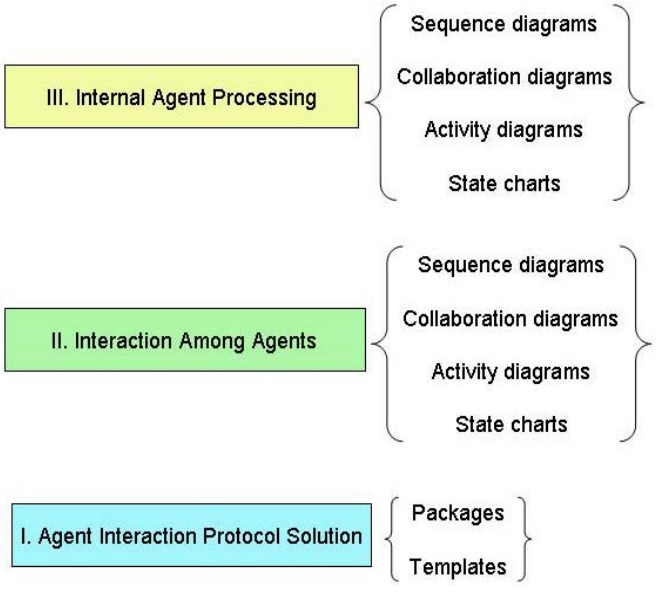


Fig. 4.3 The layered AUML approach

The interaction diagrams are also used in the third AUML layer to specify the detailed processing that takes place within an agent. Sequence diagrams, collaboration diagrams, state charts and activity diagrams can all be used to specify the internal processing of agents.

4.3.4 Agents in Z

The standard properties of the Z specification language are used to specify the computational models (Luck et al. 1997). The abstract definitions are refined to include the relevant systems constraints. The four-tiered hierarchy that comprises entities, objects, agents and autonomous agents is based on the idea that all components of the world are entities. Of these entities, some are objects, of which some, in turn, are agents and of these, some are autonomous agents.

Entities are viewed as being distinct from the remainder of the environment. They can organize perception and serve as a useful abstraction mechanism. The various attributes are grouped together through entities without the need to add a layer of functionality on top.

An object has abilities and attributes but has no further defining characteristics.

An agent is an object associated with a set of goals. Different instantiations of an agent, together with agents created in response to another agent, may result from one object.

Agents that can generate their own goals from motivation are defined as autonomous agents.

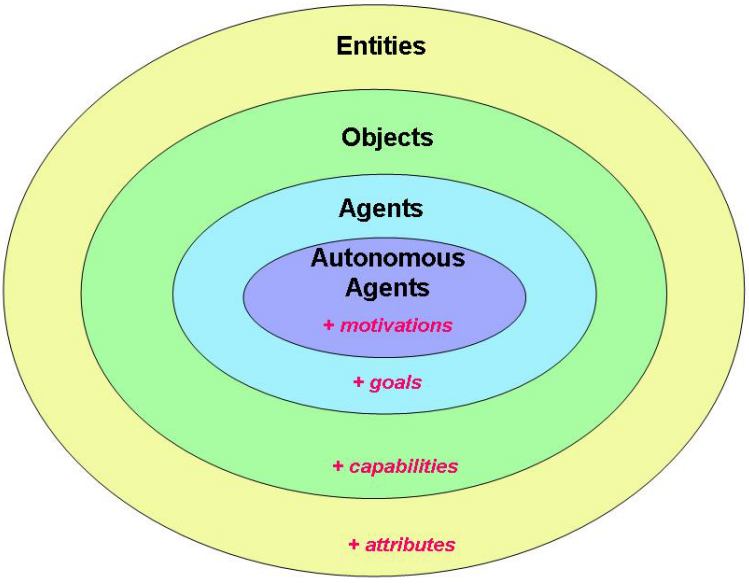


Fig. 4.4 The four-tiered hierarchy approach

In summary, if the attributes are grouped without capabilities, then an entity is identified. If an entity is characterized by attributes and abilities but there are no goals, then it is an object. If an object has goals but no motivations, then it is an agent. Finally, if an agent is under motivation, then it is an autonomous agent. This four-tiered hierarchy is presented in Figure 4.4.

The implementation of the system reflects this structural framework and is inspired by object-oriented methods in C++ . The container data structure contains all entities of the simulation environment. Objects and agents are uniformly situated within this environment and are linked together in the container class.

The user specifies the number of iterations within the simulation run. The granularity of the iterations determines the simulation time. Each object or agent, together with its internal levels of motivation, is updated on each iteration. An individual world view for each agent is obtained through the perception functions. The initial conditions are checked and active behaviours are performed. This step results in a change to the environment. The agent itself is responsible for the maintenance of its current state and configuration.

4.3.5 DESIRE

DESIRE (framework for DEsign and Specification of Interacting Reasoning components) was originally designed for formal specification of complex reasoning systems and was extended to enable formal specification of multi-agent systems (Brazier et al. 1995). The special feature of DESIRE is that it specifies the dynamics of reasoning and acting behaviour. In Figure 4.5, we gave an overview of the DESIRE methodology extensions.

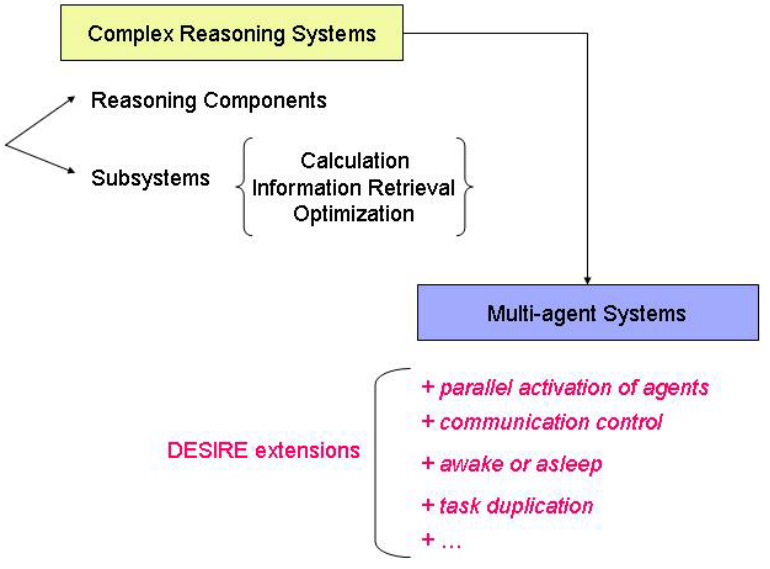


Fig. 4.5 DESIRE methodology extensions

Complex reasoning systems are designed as hierarchically structured components. These components can be reasoning components as well as subsystems capable of performing tasks such as calculation, information retrieval and optimization. The components interact with each other, with the external world, and with the users. These task-based interactions are formally specified. The formal semantics of such compositional reasoning systems are defined using temporal logic.

The basic assumptions behind DESIRE are used as a foundation for the extension of the formal specification framework to suit the definition of multi-agent systems. The extended framework enables (1) formal specification of both static and dynamic aspects of the interactions within the system behaviour; and (2) definition of the semantics of task performance.

Agents can be viewed as being composed of components within the DESIRE framework and their interactions can be viewed as interactions between composed components. For the multi-agent case, the DESIRE framework has been extended to allow parallel activation of agents and to enable them to directly control their communications. In addition to being active or idle, agents as well as information links, can be either awake (capable of processing incoming information as it arrives) or asleep. Duplication of tasks across agents can also be easily implemented.

4.3.6 Cassiopeia

The MICROB project (Making Intelligent Collective Robotics) aims to investigate collective organization within societies of robots. Experiments conducted using a

simple testbed support the examination and better understanding of the organizational features of agents.

The Cassiopeia method (Collinot et al. 1996) focuses on the collective behaviours within a set of agents. It provides a methodological framework for better understanding of a computational organization and supports its design. According to Cassiopeia, a multi-agent system is characterized by three levels of agent behaviours: (1) elementary, (2) relational, and (3) organizational.

Firstly, elementary behaviours are required for the achievement of the collective task needs listed. A functional or object-oriented analysis step is used to identify these behaviours. Based on the elementary behaviours, different types of agents are defined.

Secondly, dependencies between the elementary behaviours determine the coupling graph underlying the collective task. This coupling graph is projected onto the different types of agents and the inconsistent dependencies are removed. The dependencies between the different types of agents are called influences, and the refined projected graph is called the influence graph. The designer needs to specify the relational behaviours that enable the agents to identify and handle the influences. The paths and the elementary circuits of the influence graph determine a global representation of the organization structure.

The third step addresses the dynamics of the organization. The organizational behaviours that will enable the agents to manage the formation, durability and dissolution of groups are specified in this step. When an agent needs to form a group, it produces the relevant influence signs. This trigger agent needs to evaluate all potential groups in order to decide which one is the most appropriate in the current context. The designer should:

1) identify the trigger agents. For this purpose, the grouping potentialities identified in the influence graph are used.
2) for each of the identified trigger agents, determine the selection methods to introduce control over the formation of groups. A variety of techniques can be used for this purpose e.g. task announcement, agent consensus and negotiation, agent priorities, use of a supervisor or a hierarchy etc. These techniques define the group formation behaviours which are the first type of organizational behaviour.
3) specify the commitment signs which are produced by the trigger agents to inform other agents that a group has been formed to meet their need. These signs enable agents that are not members of the formed groups to control their relational behaviours. The joining behaviours, which are the second type of organizational behaviour, are defined for each type of agent.
4) revise an already-formed group if the need arises; e.g. a group can be replaced by another more efficient group. For each type of agent, the designer needs to define the group dissolution behaviour which is the third type of organizational behaviour.

An overview of Cassiopeia's three levels of agent behaviours is presented in Figure 4.6.

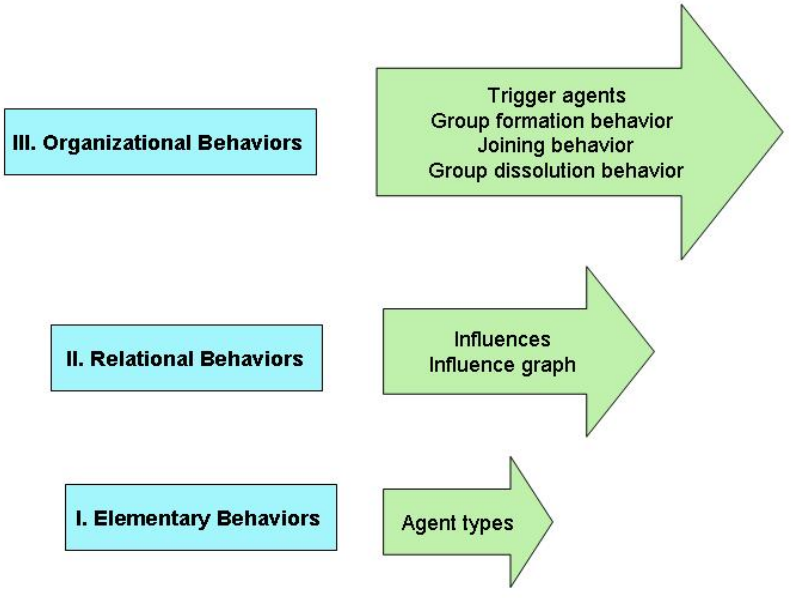


Fig. 4.6 The three levels according to the Cassiopeia method

4.3.7 Multi-Agent System Engineering

Multi-agent Systems Engineering (MaSE) (Deloach et al. 2001) is a methodology for developing heterogeneous multi-agent systems. In the MaSE process, system goals, behaviours, agent types and agent communication interfaces, as well as internal agent structure, are defined through graphically-based models. The set of these abstract models is transformed into more concrete representations that specify and describe the system in greater detail.

MaSE consists of two phases: Analysis phase and Design phase (see Figure 4.7).

The MaSE analysis phase consists of three steps: Capturing Goals, Applying Use Cases and Sequence Diagrams, and Refining Roles. Typically, a system has an overall goal that must be achieved which consists of a number of sub-goals. Capturing Goals is concerned with abstracting the functional requirements of the system into a set of system goals. Use Cases and Sequence Diagrams are used to validate the system goals, identify an initial set of roles and communications paths. Here, a role is seen as an entity responsible for achieving specific system (sub-)goals. In the Refining Roles step, goals are transformed into a set of roles. Role Model that captures roles and their associated tasks, and/or a Concurrent Task Model that defines role behaviour for each task, are created.

During the MaSE Design phase, the roles and tasks defined during the Analysis phase are transformed into agent types and conversations. This determines the organizational structure of the system. The agent's internal structure is defined as well. The Design phase has four steps: Creating Agent Classes, Constructing

Conversations, Assembling Agent Classes, and System Design. In the Creating Agent Classes step, the designer assigns roles to specific agent classes and identifies required conversations. The actual conversations between agent classes are defined in the Constructing Conversations step. For each communication path, the messages and states are extracted and added. The internal architecture and reasoning processes of the agent classes are designed in the Assembling Agents Classes step. Components and connectors are used to define the architecture of each agent. In the final System Design step, the final system structure including the actual number and location of agents in the system is defined using Deployment Diagrams.

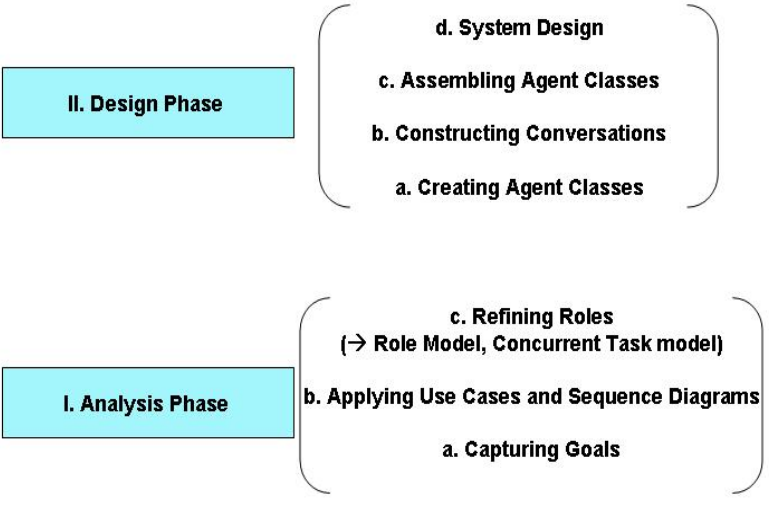


Fig. 4.7 The two phases of Multi-agent Systems Engineering methodology

4.3.8 TROPOS

The five main development phases of the TROPOS methodology (Bresciani et al. 2002) are:

1) Early Requirements
2) Late Requirements
3) Architectural Design
4) Detailed Design
5) Implementation

Early Requirements analysis identifies and analyses the actors and their goals. The actors depend on one another for goals, plans and resources. A goal-oriented analysis is used to decompose the overall goal into finer sub-goals.

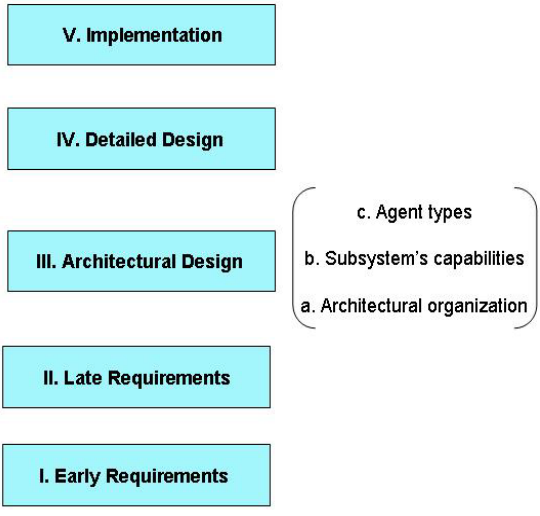


Fig. 4.8 TROPOS methodology

Late Requirements analysis examines the functions and qualities of the proposed subsystems. The interdependencies between subsystems determine the system's functional and non-functional requirements.

The Architectural Design phase defines the system's global architecture through specification of subsystems that are interconnected through data and control flows. This phase consists of the definition of:

1) an overall architectural organization
2) the subsystem's capabilities required to fulfil its goals and plans
3) the set of agent types and assigning capabilities to each of them

The agents are specified at the micro level in the Detailed Design phase. The focus is on agent's goals, beliefs and capabilities, as well as agent's communication. Development platforms and features of the agent programming language determine the practical implementation of these specifications.

JACK Intelligent Agents is an agent-oriented development environment that is built on top and is fully integrated with Java. It is used in the Implementation phase to design agents as autonomous software components capable of achieving their goals. The agents in JACK are usually programmed with a set of plans.

An overview of the TROPOS methodology is given in Figure 4.8.

4.3.9 Prometheus

The Prometheus methodology (Padgham and Winikoff, 2002) consists of three phases (see Figure 4.9):

1) System specification phase
2) Architectural design phase
3) Detailed design phase

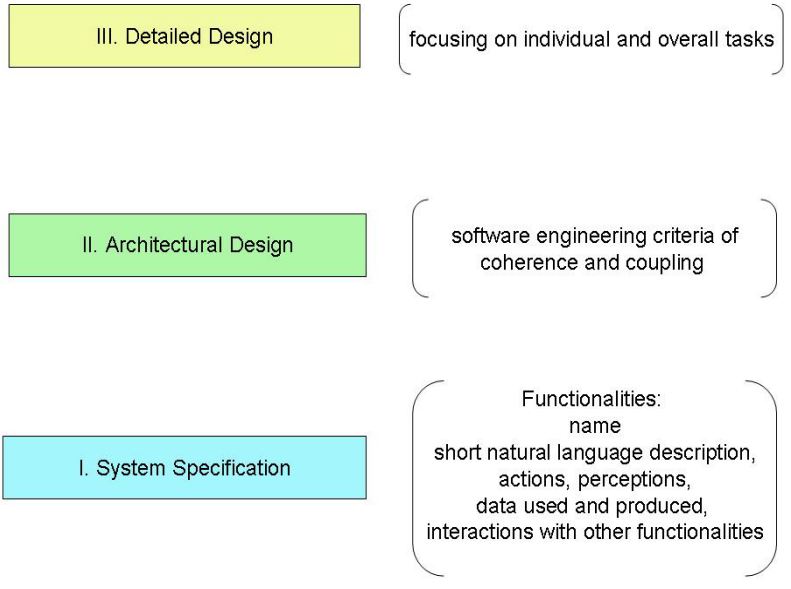


Fig. 4.9 Prometheus methodology

The basic functionalities of the system are identified during the System Specification phase. In this phase, the information that is required (inputs), as well as the information produced by the system (outputs), needs to be identified. The resulting functionality descriptor contains a name, a short natural language description, a list of actions, a list of relevant perceptions, data used and produced, and a brief description of interactions with other functionalities.

The defined functionalities are used during the Architectural Design phase to determine necessary agents and their interactions. The traditional software engineering criteria of coherence and coupling are used to evaluate assignments of functionalities to agents. For example, different functionalities that use the same data should be grouped with relevant interaction established between them. The design of agents with strong coherence and loose coupling is recommended.

The internal structure of each agent is explored in great detail during the Detailed Design phase. The designer focuses specifically on an agent's abilities to accomplish its individual tasks and the integration of these tasks with others into an overall task.

4.4 Conclusion

In this chapter, we have discussed a number of different multi-agent system design methodologies. These include:

1) Belief, Desire and Intention (BDI) approach
   The BDI system is approached from two different perspectives: the external viewpoint and the internal viewpoint. The characteristics of external viewpoint are captured in two models: Agent Model and Interaction Model. From the internal viewpoint, for each agent class, the details are captured in three models: Belief Model, Goal Model and Plan Model.
2) Gaia
   The main Gaian concepts can be divided into two categories: Abstract entities and Concrete entities. The Gaia methodology views agent-based systems as a collection of roles in certain relationships to one another. A role is defined by four attributes: Responsibilities, Permissions, Activities and Protocols.
3) Agent UML
   Agent UML is a set of UML idioms and extensions that can be used to model multi-agent systems. It adopts a three-layer representation: agent interaction protocols, interactions among agents and internal agent processing.
4) Agents in Z
   In Agents in Z methodology, the standard properties of the Z specification language and the four-tiered hierarchy approach that comprises entities, objects, agents and autonomous agents, is used to specify the computational models.
5) DESIRE
   DESIRE specifies the dynamics of reasoning and acting behaviour. Some main features of DESIRE are parallel activation of agents, task duplication, communications control and different agent states (active or idle and awake or asleep).
6) Cassiopeia
   The Cassiopeia method focuses on the collective behaviours within a set of agents. According to Cassiopeia, a multi-agent system is characterized by three levels of agent behaviours: elementary, relational and organizational.
7) Multi-agent System Engineering
   MaSE consists of two phases: Analysis phase and Design phase. The MaSE Analysis phase consists of three steps: Capturing Goals, Applying Use Cases and Sequence Diagrams, and Refining Roles. During the MaSE Design phase, the roles and tasks defined during the Analysis phase are transformed into agent types and conversations. The Design phase has four steps: Creating Agent Classes, Constructing Conversations, Assembling Agent Classes, and System Design.
8) TROPOS
   The five main development phases of the TROPOS methodology are: Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation.
9) Prometheus
   The Prometheus methodology consists of three phases: System Specification, Architectural Design and Detailed Design.

In the following chapter, we will discuss different ontology design methodologies.

References

1. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-agent Systems 3, 285–312 (2000)
2. Brazier, F., Keplicz, B.D., Jennings, N.R., Treur, J.: Formal Specification of Multi-agent systems: a Real-World Case. In: The first International Conference on Multi-agent systems (ICMAS 1995), pp. 103–112 (1995)
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: an Agent-oriented Software Development. Technical Report # DIT-02-0015 (2002)
4. Collinot, A., Drogoul, A., Benhamou, P.: Agent Oriented Design of a Soccer Robot Team. In: ICMAS 1996, pp. 41–47 (1996)
5. Deloach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent system engineering. International Journal of Software Engineering and Knowledge Engineering 11, 231–258 (2001)
6. Kinny, D., Georgeff, M., Rao, A.: A methodology and modeling technique for systems of BDI agents. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 56–71. Springer, Heidelberg (1996)
7. Luck, M., Griffiths, N., d'Invernoy, M.: From Agent Theory to Agent Construction: A Case Study. In: Jennings, N.R., Wooldridge, M.J., Müller, J.P. (eds.) ECAI-WS 1996 and ATAL 1996. LNCS (LNAI), vol. 1193, pp. 49–63. Springer, Heidelberg (1997)
8. Odell, J., Van Dyke, H.P., Bauer, B.: Representing Agent Interaction Protocols in UML. Springer, Heidelberg (2001)
9. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In: Giunchiglia, F., Odell, J.J., Weiss, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 174–185. Springer, Heidelberg (2003)
10. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley and Sons, Chichester (2002)

Chapter 5
Ontology Design Approaches

5.1 Introduction

In Chapter 3 we discussed the concept of ontology and its characteristics, and the numerous advantages of ontologies. We may feel inspired to design an ontology for our information system but we still do not know: where to start, how to prepare ourselves, how many people are needed for the ontology design, how we ascertain that we are on the right track, whether similar ontologies already exist, how to determine that we are designing a valid ontology, what tool we can use to assist us with our design, and other similar issues.

There is not a single, consensual ontology-design methodology. Many different ontology design methodologies have been proposed. We will consider some of them in order to illustrate different approaches to ontology design. The methodologies presented are different from each other and together cover various aspects of the ontology design process.

5.2 Ontology Design Criteria

A set of criteria has proven to be useful as a guiding tool during the ontology design process. These principles have also been used to evaluate otology design. Gruber suggests five design criteria for ontologies (Gruber 1995):

1) Clarity
2) Coherence
3) Extendibility
4) Minimal Encoding Bias
5) Minimal Ontological Commitment

Clarity. We need to be clear about our reasons for choosing a particular set of ontology terms and the intended meanings of those terms. The number of possible interpretations of a term needs to be restricted. Complete definitions of ontology terms, which are defined by necessary and sufficient conditions, are preferred over partial definitions of ontology terms, which are defined by necessary or sufficient conditions. The definitions need to be objective, documented in natural language but stated in formal axioms within the ontology.

Clear definitions of ontology terms will contribute to the effectiveness of communication between agents. Take the example presented in Figure 3.3 where we gave four different definitions of a gene. The definition that "a gene is situated on a chromosome" is incomplete because there are some other parts of chromosomes

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 75–91.
springerlink.com

that are not genes but have a regulatory function; for instance, promoters are situated on a chromosome and are responsible for the activation or repression of a particular gene. The condition that "a gene is situated on a chromosome" is necessary but not sufficient. If agent A gives information about promoters to agent B who thinks that everything situated on DNA is a gene, the agent B will conclude that the promoter is a new gene. It is difficult and sometimes impossible for different agents to reach an agreement if they have different understandings of the terms used in conversation.

Coherence. Propositions that can be inferred from the ontology definitions and axioms may not contradict other ontology definitions and axioms. Inferences need to be consistent with the existing definitions and axioms, and should be clear and logical. In the definition "gene is situated on a chromosome", it has not been clearly stated if the gene is part of the chromosome sequence or whether this chromosome is used to carry genes on its surface. An agent could 'infer' from this definition that "a gene is superimposed on a chromosome structure". We know that a gene is a part of a chromosome sequence and for domain experts this inferred statement may appear to be strange. But the knowledge that an agent holds is limited to the knowledge given to it and we need to be aware of this fact when defining ontology terms.

Extendibility. As new knowledge emerges each day, this information may need to be added to an already existing ontology. For this reason, the ontology needs to be easily extendable. The new ontology terms should be easily defined based on the existing vocabulary and/or preferably without the need to alter the existing ontology definitions.

Extending of an ontology may result in a new ontology that will contain more precise and correct definitions. Let us say that an ontology contains the definition "a gene is part of chromosome sequence". When the new knowledge emerges that implies the presence of promoters within the chromosome sequence and when this knowledge needs to be added to the existing ontology, it will point to the mistakes in the definition of the term "gene". The existing definition needs to be corrected in order to allow the new definition of the term "promoter" to be incorporated in the ontology. The resulting ontology will not only be extended but it will also be more precise.

Minimal Encoding Bias. Conceptualizations need to be specified at a knowledge-level and not at a symbol-level. The goal of the ontology is to represent true facts, even when it seems to be more convenient to compromise on ontology correctness. In our example of gene definition, it may appear to be more convenient to use the expression "positioned on" instead of "part of", but this may be at the cost of much time and energy when we come to the implementation phase and our agent needs to communicate with other agents. We should not compromise the correctness of an ontology for the convenience of its notation or implementation.

Minimal Ontological Commitment. More commitments in the ontology make the ontology structure more rigid and limit the number of ontology users. Ontological commitment needs to be minimal but sufficient to support the intended knowledge sharing activities. For this purpose, it may be needed to specify an

ontology which is necessary for effective communication consistent with the domain conceptualization and easily extendable by individual agents.

All the agents can communicate effectively and efficiently between each other on the basis of the 'necessary' ontology. In our example of gene definitions, two agents need to find a shared understanding of the term "gene" and the 'necessary' ontology may contain the definition "gene codes for a protein". Agents talk to each other on a higher conceptual level and for this they need the 'necessary' ontology. Each of these agents has certain tasks assigned to it. For this reason, the agents need to extend the 'necessary' ontology to a 'specialized' ontology. Suppose that the tasks of agent A are more gene-oriented and those of agent B are more protein-oriented. Agent A may need to extend this definition with the statement that "a gene is part of chromosome" and agent B with "protein is composed of amino acids". The agents need to be able to extend the common ontology where needed for their individual purposes in order to perform their individual tasks efficiently and effectively.

Cooperative work between knowledge and software engineers, ontology designers and domain experts is required for the design and development of ontology-based information systems.

5.3 Ontology Design Methodologies

In this section, we will discuss six different Ontology Design Methodologies, namely:

1) Knowledge Engineering Methodology
2) DOGMA Methodology
3) TOVE Methodology
4) Methontology
5) SENSUS Methodology
6) DILIGENT Methodology

5.3.1 Knowledge Engineering Methodology

Knowledge engineering methodology (Uschold and Gruninger 1996) for designing and maintaining ontologies is identified as a six-stage process, as shown in Figure 5.1, namely:

1) definition of the domain, purpose and scope of ontology
2) acquisition and conceptualization of the domain knowledge
3) reusing of existing ontologies
4) formal specification of the ontology
5) population of the ontology with individual instances
6) evaluation and documentation

1. Definition of the Domain, Purpose and Scope of Ontology. Domain problem, purpose and scope of the ontology is defined in this stage. This stage involves description of the ontology based on the use of a motivating scenario.

A set of questions is used in order to facilitate the tasks associated with this phase (Noy and McGuinnes 2001), such as:

1) What domain will the ontology cover?
2) For what will the ontology be used?
3) For what types of questions should the ontology provide answers?
4) Who will use and maintain the ontology?

The output of this phase is a natural language ontology specification document which supports the ontology design process. This document contains information such as the purpose of the ontology within a community, ontology scope, use case scenarios, degree of formality used to codify the ontology and other similar matters. An ontology specification document must include relevant terms without duplication, and the meanings of these terms and their relationships need to make sense in the domain, and coincide with their meaning in the real world.

For example, a genetics institute wants to design an intelligent information retrieval system that will look for functions and characteristics of a particular DNA sequence. A DNA ontology needs to be designed to support the intelligent searches performed by the information retrieval system. Does the system need the full DNA sequence in order to perform its searches, or does it only look for the numerically represented positions of the sequence? Those and similar kinds of ontology details and ontology scope will depend on how the ontology is intended to be used within the system and on the system requirements.

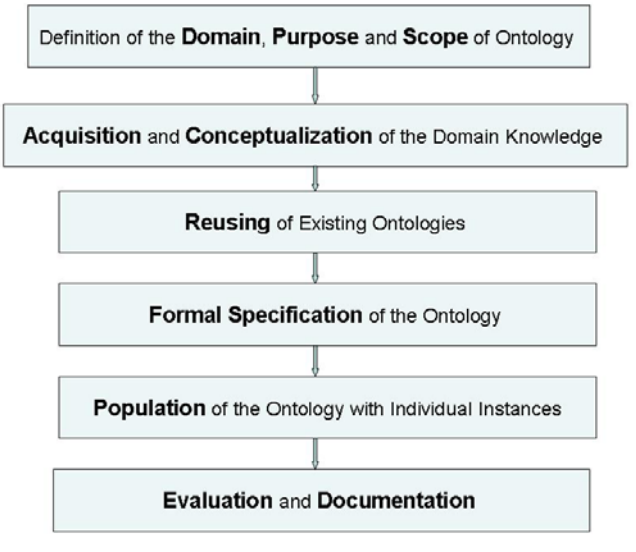


Fig. 5.1 Knowledge engineering methodology

2. Acquisition and Conceptualization of the Domain Knowledge. This stage involves the process of acquiring knowledge from a given domain. Here, it is important to establish cooperation between the ontology engineer and the domain experts. Different methods for collecting and analyzing the data may be applied here such as domain expert interviews, questionnaires and text analysis of relevant documents.

The output of this step is an informal ontology. The informal ontology is a conceptualized domain model that represents domain concepts and the relationships between them. The informal ontology contains a glossary of terms (concepts), definition of the relationships between the terms and constraints on their usage.

Different conceptual modelling techniques can be used to represent the ontology. A suggested conceptual modelling approach consists of the following two steps:

(1) make a list of domain terms and their meaning

The domain terms need to be identified and their properties described. A set of tables can be created to support the collection of information about domain terms. Domain concepts can be identified from the collected information sources and related data interpreted and analyzed for their significance in the ontology design. The identification of the relevant domain terms is crucial for the ontology design process.

In our example, the DNA ontology may consist of the following terms: DNA, chromosome, sequence, gene, nucleotide, nucleotide base, sugar, phosphate group, protein, amino acid.

(2) classify the concepts

Concept classification includes hierarchical conceptual structures, conceptual graphs, semantic nets and ontologies. The quality of the applied classification schema affects the effectiveness of the ontology design. Top-down and bottom-up approaches can be applied for the categorization and classification of domain concepts (Nakata 2001).

A top-down concept classification includes specification of all possible subconcepts of a concept. This approach is usually applied with the help of domain experts. Depending on the richness of the domain knowledge required in the analysis, specific limits on the granularity for this classification need to be introduced. For example, for our DNA ontology, do we need to say that the DNA consists of a sequence of nucleotides, or are we going to go a step further and add that each nucleotide consists of a sugar molecule, phosphate group and one of the four nucleotide bases (Adenine, Cytosine, Guanine and Thymine)? The choice will depend on the richness required by the ontology model.

The bottom-up concept classification includes identifying concepts in the documents. This includes analyzing terms, phrases, and segments of text and assigning them to a concept or group of concepts. The bottom-up concept classification is applied in situations such as analysis of the created and collected documents, interview results and questionnaire responses. Collected concepts are then grouped in a hierarchy, from which the ontology may begin to emerge.

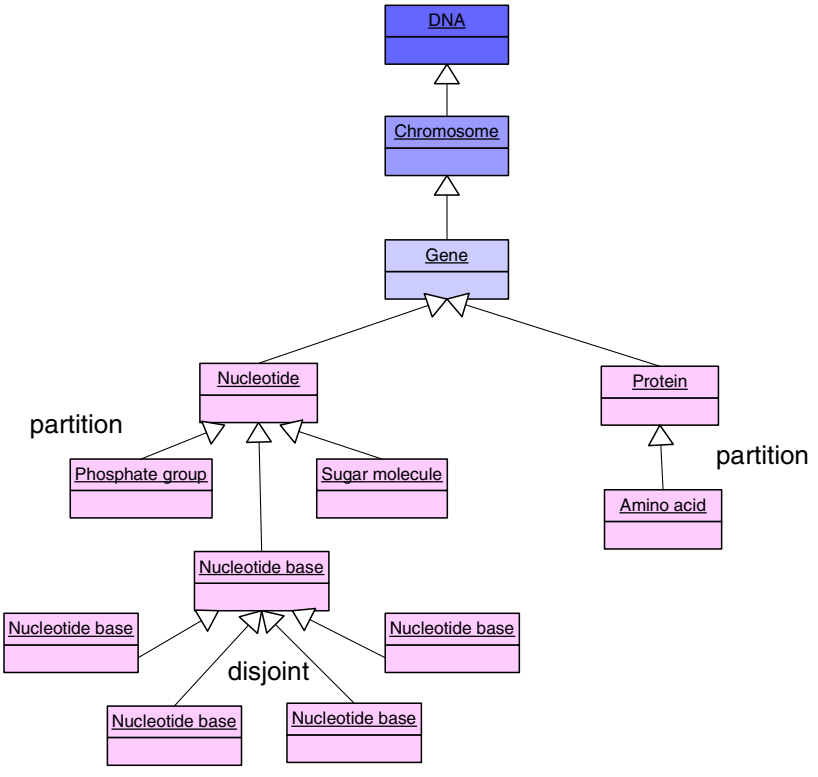


Fig. 5.2 Hierarchy of the DNA ontology

The abovementioned concepts can be represented by the hierarchy shown in Figure 5.2. DNA, chromosomes and genes are all sequences of nucleotides. Nucleotides consist of a phosphate group, a sugar molecule and a nucleotide base (Adenine, Cytosine, Guanine or Thymine). A gene is a part of a chromosome and a chromosome is part of a DNA. A gene codes for a protein which is a sequence of amino acids.

3. Reuse of Existing Ontologies. This stage focuses on the process of building a new ontology by reusing and adapting ontological terms from other published and consensual ontologies. Usually, a set of sharable and reusable ontologies is used during this process. The goal is to obtain some uniformity across different ontologies.

Two main processes associated with the reuse of existing ontologies are ontology merging and ontology alignment. The difference between these two processes is that the merging of different ontologies results in a single coherent ontology, while the aligning of different ontologies establishes links between the concepts of the aligned ontologies, allowing them to reuse information from one another. These processes both take as input two (or more) source ontologies and return, in

the case of ontology merging, a single merged ontology or, in the case of ontology alignment, a cluster of ontologies where the two (or more) original ontologies persist with links established between them. Usually, in ontology merging, the original ontologies cover similar or overlapping domains while in ontology alignment the original ontologies cover domains that are complementary to each other.

In our example from Figure 5.2, we may have two different ontologies: the first one is on the left hand side of the figure which consists of the terms: “DNA”, “chromosome”, “gene”, “nucleotide”, ”phosphate group”, “sugar molecule”, “nucleotide base”, “Adenine”, “Cytosine”, “Guanine” and “Thymine” while the other ontology is on the right hand side of the figure and consist of the terms “protein” and “amino acid”. In this case, the joining point is between the terms “gene” and “protein: gene codes for a protein”. We can merge those two ontologies into a single ontology, or we can align them but still represent them as two separate ontologies with an indication of links between them.

4. Formal Specification of the Ontology. Ontologies need to be formally specified using a formal representation language. The output of the two previous steps is an informal ontology written in natural language that can be understood by humans, but not by computers. The information represented by this ontology needs to be translated into a formal language. This enables machines to effectively use the ontologies; the machines can ‘read’, ‘understand’ and ‘act’ upon this information. This is one of the main reasons that the concept of ontology was introduced into the domain of computer and information science. The choice of the formal representation language depends on the way the system needs to be implemented and/or integrated with the existing system. Currently, most of the ontologies are written in OWL.

5. Population of the Ontology with Individual Instances. Ontology consistency is experimentally analyzed in this step. An ontology design tool, such as Protégé, can be used in this step. Defining an individual instance of a class requires:

1) Identification of a class to which this instance belongs
2) Creation of an individual instance of this class
3) Determination of slot attributes with the proper values for this instance

6. Evaluation and Documentation. The ontology is evaluated in respect to ontology consistency and completeness (Goméz-Péréz et al. 1996), as mentioned in Section 3.2. The terms in the ontology need to be clearly defined, the terminology needs to be coherent, and the relationships between the terms logically consistent. Some inconsistencies and redundancies may emerge during the implementation process. For this reason, ontology (re)modification follows the implementation process in most cases.

Each of the abovementioned methodology steps must be documented and explicitly written.

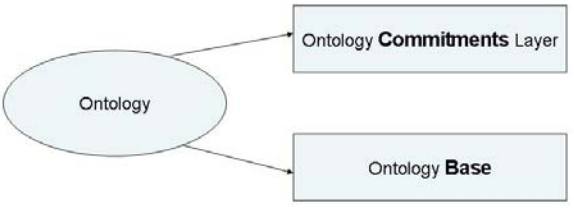


Fig. 5.3 DOGMA methodology

5.3.2 DOGMA

Whether or not axioms should form part of an ontology is one of the main discussion topics in regard to ontology definition (Pisanelli et al. 2002). For some, an ontology is a domain description that organizes the concepts into a specific structure e.g. hierarchy; for others, an ontology is a complete theory consisting of a formal vocabulary as well as well-defined axioms that allow further deductions or inferences to be made in regard to the knowledge the ontology represents.

One way to address the above issue is to adopt an approach that neatly separates the specification of ontology concepts from their axioms. The DOGMA (Developing Ontology-Guided Mediations of Agents) framework has been developed at the VUB STARLab (Vrije Universiteit Brussel Semantic Technology and Applications Research Laboratory). The DOGMA approach is based on the principle of a double articulation (De Bo et al. 2003). An ontology designed using the DOGMA double articulation approach consists of two layers: ontology base and commitment layer, shown in Figure 5.3.

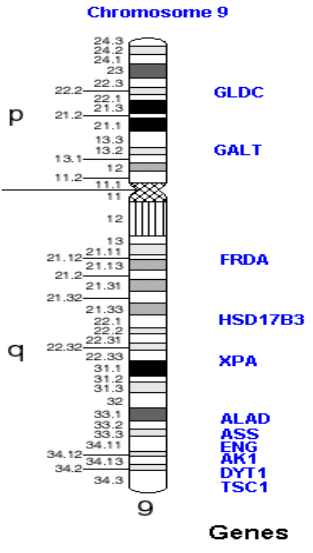
1. Ontology Base. The ontology base holds the conceptualization of a domain. Namely, ontology concepts and relationships between these concepts are formally defined in the ontology base. The ontology base is defined in such a way that it enables correct contextual identification of these concepts and the associated relationships.

According to the DOGMA methodology, an ontology base is a set of context-specific binary fact types, called lexons (notation: <μ, t1, r, cr, t2>). Here $\mu \in \Delta$ is an abstract context identifier chosen from a set Δ; t1, r, cr, t2 are term1, role, co-role and term2 respectively. The lexical terms (t1, r, cr, t2) are constructed from a given alphabet. For each $\mu \in \Delta$ and each term t occurring in a lexon, the pair (μ, t) specifies exactly a unique concept.

In our example, we may have the following lexon of the form <μ, t1, r, cr, t2>:

<DNA, gene, is part of, contain, chromosome>. Here, "DNA" corresponds to context identifier μ, "gene" corresponds to term t1, "is part of" corresponds to role r, "contain" corresponds to co-role cr and "chromosome" corresponds to term t2. This statement means that in the DNA context, "gene is a part of a chromosome" and "chromosome contain(s) gene(s)". An example shown in Figure 5.4, specifies that genes GLDC, GALT, FRDA, HSD17B3 etc. are found on chromosome 9 (out of 22 human chromosomes, and X and Y sex chromosomes).

Fig. 5.4 Genes on chromosome 9



2. Commitment Layer. Each commitment within the commitment layer contains:

1) a set of constraints, derivation and domain rules applied to a specified subset of the ontology base (Jarrar et al.2003), and
2) a set of mappings between ontological elements and application elements (Deray and Verheyden 2003).

In our example, a commitment is "each chromosome contains at least one gene". The corresponding statement from the ontology base, "chromosome contain(s) gene(s)", has been more precisely described by this commitment.

An agent or application that operates in the ontology domain is able to further specialize the ontology base through modifications to its commitment layer. An agent may, or may not, require the presence of a specific commitment within its ontology. This depends on the agent's function within the system. The ontology base plus the agent's commitment to a part of it, is the real world to the application agents.

5.3.3 TOVE Methodology

The TOVE (Toronto Virtual Enterprise) (Gruninger and Fox 1995) project required the design of an ontology for the business processes and activities domain. Experiences gained during the development of this ontology were used as the basis for the development of a novel ontology design methodology, called the TOVE methodology. The TOVE methodology consists of the following six steps, shown in Figure 5.5.

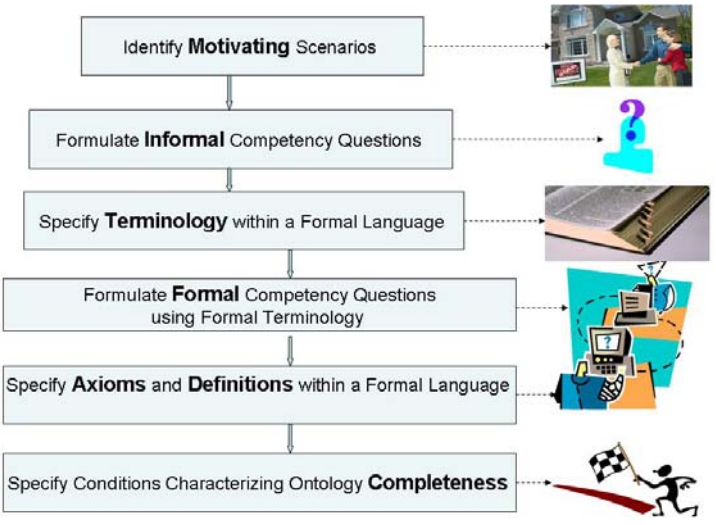


Fig. 5.5 TOVE methodology

1) Identify motivating scenarios
2) Formulate informal competency questions
3) Specify terminology within a formal language
4) Formulate formal competency questions using formal terminology
5) Specify axioms and definitions within a formal language
6) Specify conditions characterizing ontology completeness

1. Identify motivating scenarios. Various scenarios that may arise during the application process motivate the development of ontologies. The motivating scenarios are story problems or examples that cannot be solved using the existing knowledge base, ontologies or similar techniques. They require a new ontology or a current ontology to be extended in order to enable the system to solve the problems from motivating scenarios. This is the reason that the motivating scenario will usually also provide a set of possible solutions to the scenario problems that can be solved only once the proposed ontology has been developed. The motivating scenario, together with a set of possible solutions to the scenario problems, provide an informal semantics for the concepts and relationships between those concepts which later need to be included in the ontology. When designing a new ontology or extending an existing one, one or more motivating scenarios need to be described as well as the set of possible solutions for the problems presented in the scenarios.

In our example of DNA ontology, we may have the following scenarios where we need to:

1) determine the exact position of a specific gene within a chromosome
2) determine the number of genes that are found on a specific chromosome;

3) determine the length of a specific gene in nucleotide base pairs (length of a gene is usually expressed in nucleotide base pairs because DNA is a double helix; nucleotide bases of the one helix bind to the nucleotide bases from the other helix).

2. Formulate informal competency questions. The informal competency questions are written in natural language and are used to evaluate the expressiveness of the ontology. Once the ontology has been expressed in a formal language, it needs to be able to answer these questions. The ontology to be developed must be able to use its terminology, definitions and axioms to represent these questions and characterize the answers to these questions.

Examples of competency questions relating to the abovementioned motivating scenarios are:

3) Given the name of the gene (e.g. GLDC, GALT, FRDA or HSD17B3 from Figure 5.4) and constraints in regard to gene positions on a specific chromosome (in this example, constraints in regard to gene positions on chromosome 9), can the exact position of this gene within the specific chromosome be determined?
4) Given the name of the chromosome (e.g. chromosome 9) and constraint in regard to the number of genes on a specific chromosome (e.g. greater than or equal to 11), can the number of genes found on this specific chromosome be determined?
5) Given the name of the gene (e.g. GLDC, GALT, FRDA or HSD17B3 from Figure 5.4) and constraints in regard to gene length (e.g. greater that or equal to 114 nucleotide base pairs), can the exact length of this chromosome in nucleotide base pairs be determined?

3. Specify terminology within a formal language. This step consists of two parts: extraction of informal terminology and formal specification of the terminology.

Extraction of informal terminology. The set of terms used within informal competency questions is extracted. They serve as a basis for specification of the terminology in a formal language. In our example, we can extract terms such as GLDC, GALT, FRDA, D17B3, gene name, gene position, gene length, nucleotide base, nucleotide base pair, chromosome, and similar.

Formal specification of the terminology. A formalism, such as first-order logic, can be used to specify the ontology terminology. This will enable the use of axioms to express ontology definitions and constraints. If we aim to use first order logic to specify the ontology terminology, firstly we need to identify objects in the universe of discourse. Examples of objects are: GLDC, GALT, FRDA, HSD17B3, chromosome 9 etc. Secondly, we need to identify unary predicates which are used to represent concepts (such as gene, chromosome, nucleotide base), binary predicates for attributes (such as gene-name, gene-length, gene-position, chromosome-name) and predicates that represent binary relations (has-length, has-position).

4. Formulate formal competency questions using formal terminology. In this step, the competency questions identified in step 2 are defined formally according to the specifications in step 3.

5. Specify axioms and definitions within a formal language. An ontology consists of a set of terms and a set of axioms that define the semantics (meaning) of ontology terms. Ontology axioms define the ontology terms and specify the constraints on their interpretation. If the existing axioms are insufficient to represent and define the solutions to the competency questions in a formal language, additional axioms need to be added to the ontology. Therefore, development of the ontology with respect to the competency questions is an iterative process.

6. Specify conditions characterizing ontology completeness. As a final step in the ontology design, we need to define the conditions under which the solutions to the formal competency questions are complete. These conditions serve then as a basis for development of ontology completeness theorems.

5.3.4 METHONTOLOGY

The METHONTOLOGY (Fernandez et al. 1997) framework enables the construction of ontologies at the knowledge level and includes:

1) An ontology development process
2) An ontology life cycle based on evolving prototypes

1. The Ontology Development Process. The ontology development process refers to the activities that need to be carried out when building ontologies. Three different categories of these activities have been identified: ontology management activities, ontology development-oriented activities and ontology support activities. The diagram is shown in Figure 5.6.

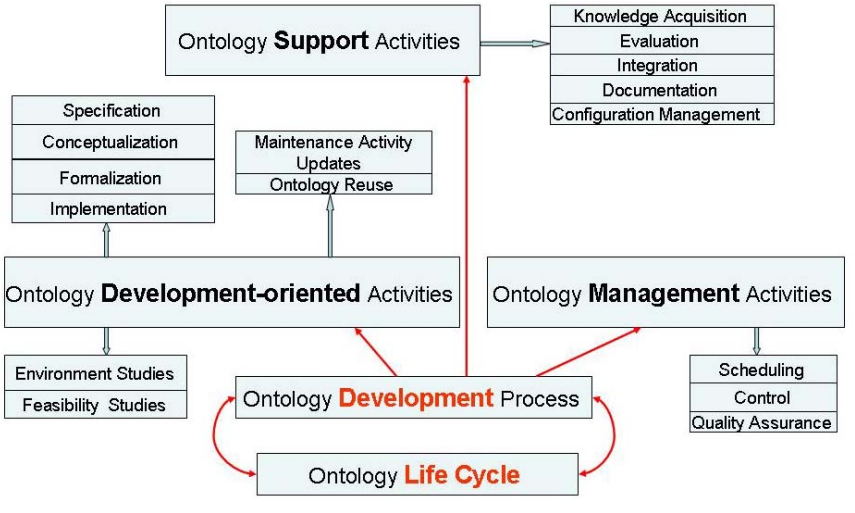


Fig. 5.6 METHONTOLOGY methodology

a. Ontology Management Activities include scheduling, control and quality assurance. Scheduling helps in the identification of tasks that need to be performed, their arrangement, and the time and resources needed for their completion. Control monitors the scheduled tasks and ensures that they are performed as planned. Quality assurance is responsible for the satisfactory quality of every product output (e.g. ontology, software or documentation).

b. Ontology Development-Oriented Activities are grouped into pre-development, development and post-development activities. Environment and feasibility studies are carried out during the pre-development phase. Environment study involves analysis of the platforms and applications where the ontology will be used and integrated, while the feasibility study analyzes possibility and suitability of ontology design. Specification, conceptualization, formalization and implementation activities that are carried out during the development. The purpose of the ontology, its intended uses and end-users are analyzed during the specification phase. Domain knowledge is represented as a meaningful model during the conceptualization phase. This conceptual model is transformed into a formal or semi-computable model during formalization. During implementation, computable models are built in a computational language. In the post-development phase, the ontology is updated and corrected on the basis of the maintenance activity. Also, other ontologies and applications can use the ontology.

c. Ontology Support Activities include knowledge acquisition, evaluation, integration, documentation and configuration management. During the knowledge acquisition phase, domain experts or some kind of (semi)automatic process is used in order to acquire knowledge of a given domain. Ontologies, their associated software environments and documentation need to be technically evaluated. Integration of ontologies is required when designing a new ontology by reusing existing ontologies. Each completed stage, as well as any generated products, needs to be documented. All the versions of the documentation, software and ontology code are recorded by configuration management.

2. The Ontology Life Cycle. The ontology building life cycle is based on evolving prototypes. The ontology life cycle identifies the set of phases through which the ontology moves during its lifetime, describes activities that need to be performed in each phase, and relationships between the different phases. Ontology changes correspond to the improvements required in the evolving prototypes.

5.3.5 SENSUS Methodology

This approach is based on the assumption that the knowledge between two ontologies can be easily shared if they have a common underlying structure i.e. if they are based on a common ontology.

SENSUS (Swartout et al. 1997) is an ontology based on natural language and is composed of upper, middle and lower regions. It provides a broad coverage conceptual structure and contains more than 70.000 nodes. PENMAN Upper Model, ONTOS, WordNet and semantic categories from electronic dictionaries (English, Spanish and Japanese) are used for development of the SENSUS ontology.

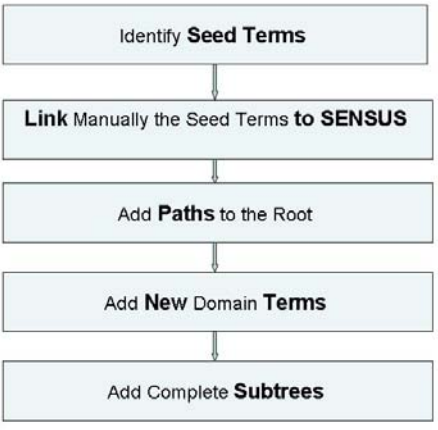


Fig. 5.7 SENSUS methodology

According to the SENSUS methodology, the steps shown in Figure 5.7 need to be followed when building an ontology for a particular domain:

1) Identify seed terms
2) Manually link the seed terms to SENSUS
3) Add paths to the root
4) Add new domain terms
5) Add complete subtrees

Seed terms act as representatives of relevant domain-specific concepts. They need to be identified and manually linked to the SENSUS ontology. All of the concepts from the seed terms to the linking node within the SENSUS ontology are included in the final ontology. Irrelevant SENSUS terms are pruned from the domain-specific ontology. Relevant domain terms absent from the SENSUS ontology need to be added to the final ontology. Some nodes may have a large number of paths through them. If many of the nodes in a subtree have been found to be relevant for the final ontology, then the other nodes in the subtree are likely to be relevant as well. For this reason, the entire subtree under the node may need to be added to the final ontology structure. Because this decision requires understanding of the domain, this step is best done manually.

In our example of the DNA ontology from Figure 5.2, “gene” is a seed term. This term needs to be linked to a node of a SENSUS ontology. The choice of the linking node depends on the intended purpose and use of the ontology. This linking node from the SENSUS ontology is called the “root”. We will add paths to the root by introducing terms “chromosome” and “DNA”. It may be necessary to add new domain terms such as “protein” and “amino acid”. Terms “Adenine”, “Guanine”, “Cytosine” and “Thymine” appear to have some relevance but the ontology designer is not sure whether the terms “nucleotide”, “nucleotide base”, “phosphate group”, “sugar molecule” are relevant as well. If these terms are found to be relevant, then the whole subtree under the term “nucleotide” needs to be added to

the final ontology. A domain expert will approve this because gene is a sequence of nucleotides that are composed of a nucleotide base (Adenine, Guanine, Cytosine or Thymine), a sugar molecule and a phosphate group.

5.3.6 DILIGENT Methodology

DILIGENT (Pinto et al. 2004) is a methodology for Distributed, Loosely-controlled and evolvInG Engineering of oNTologies. This methodology may play an important role in the development of shared ontologies in distributed settings, such as the Semantic Web. This methodology was developed to support domain experts in a distributed setting to engineer and evolve ontologies using a fine-grained methodological approach based on Rhetorical Structure Theory (RST). RST is used to analyze the arguments exchanged when consensus is sought in evolving distributed ontology engineering processes. RST assumes a function for every part of a text and shows the evident role of this part of the text.

General process, roles and functions in the DILIGENT process are shown in Figure 5.8 and include five main activities:

1) build
2) local adaptation
3) analysis
4) revision
5) local update

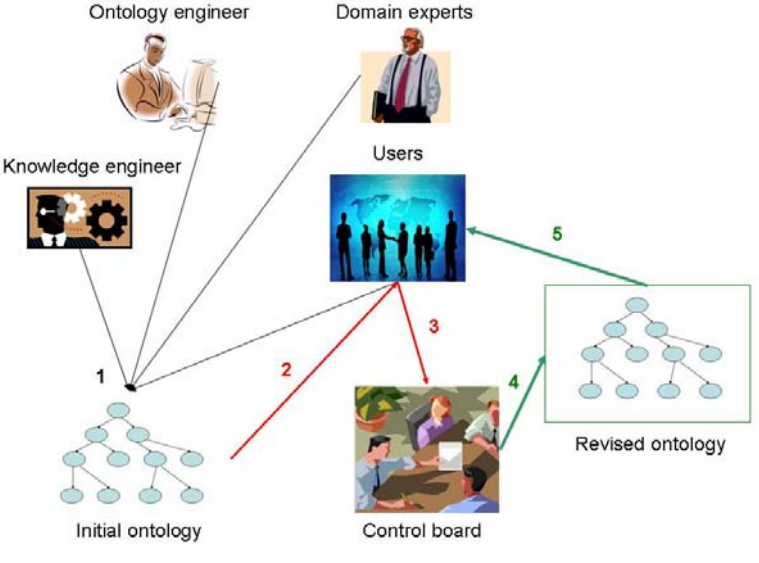


Fig. 5.8 DILIGENT methodology

Different stakeholders usually have different purposes, needs and locations. Such distributed ontology development requires online ontology engineering support. Domain experts, users, knowledge engineers and ontology engineers build an initial ontology (step 1 in Figure 5.8). This initial shared ontology is made available and users can start using it. The users need to adapt the initial ontology needs locally for their own purposes (step 2 in Figure 5.8). They can change the ontology in their local environment, but the original ontology that is shared by all users may not be changed. The control board collects and analyzes change requests to the shared ontology (step 3 in Figure 5.8). The board must decide which changes need to be introduced in the next version of the shared ontology. A balanced decision must be made that takes into account the different users' needs and meets their evolving requirements. The control board revises the shared ontology accordingly. Once a new version of the shared ontology has been released (step 4 in Figure 5.8), users can locally update their own ontologies (step 5 in Figure 5.8).

5.4 Conclusion

In this chapter, we introduced different ontology design methodologies: Knowledge Engineering, DOGMA, TOVE, MENTHONTOLOGY and SENSUS methodology. We took a simple DNA ontology as an example and illustrated the design of this ontology using each of the different methodologies. We have chosen these methodologies in particular in order to illustrate the diversity in ontology design methodologies. None of the abovementioned methodologies is similar to another. On the other hand, all of these different methodologies solve the same problem. It is very interesting that such diversity exists in the different approaches to the same problem.

In the following chapter, we will discuss the advantages of ontology- and agent-based systems, and the benefits of integrating these two complementary technologies.

References

1. De Bo, J., Spyns, P., Meersman, R.: Creating a "DOGMAtic" multilingual ontology infrastructure to support a semantic portal. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 253–266. Springer, Heidelberg (2003)
2. Deray, T., Verheyden, P.: Towards a semantic integration of medical relational databases by using ontologies: a case study. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 137–150. Springer, Heidelberg (2003)
3. Fernandez, M., Gomez-Perez, A., Juristo, N.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In: AAAI 1997 Spring Symposium on Ontological Engineering, USA (1997)
4. Gómez-Pérez, A.: Towards a Framework to Verify Knowledge Sharing Technology. Expert Systems with Applications 11, 519–529 (1996)
5. Gruber, T.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Computer Studies 43, 907–928 (1995)

6. Gruninger, M., Fox, M.S.: Methodology for the Design and Evaluation of Ontologies. In: IJCAI 1995 Workshop on Basic Ontological Issues in Knowledge Sharing, Canada (1995)
7. Jarrar, M., Demey, J., Meersman, R.: On Reusing Conceptual Data Modeling for Ontology Engineering. In: Spaccapietra, S., March, S., Aberer, K. (eds.) Journal on Data Semantics I. LNCS, vol. 2800, pp. 185–207. Springer, Heidelberg (2003)
8. Nakata, K.: A Grounded and Participatory Approach to Collaborative Information Exploration and Management. In: Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34) (2001)
9. Noy, N.F., McGuinnes, D.L.: Ontology Development 101: A Guide to Creating Your First Ontology (2001), http://protege.stanford.edu/publications/ontology_development/ontology101.pdf (retrieved: May 20, 2004)
10. Pinto, H.S., Staab, S., Tempich, C.: DILIGENT: Towards a fine-grained methodol-ogy for DIstributed, Loosely-controlled and evolving Engineering of oNTologies. In: The 16th European Conference on Artificial Intelligence (ECAI 2004), Spain, pp. 393–397 (2004)
11. Pisanelli, D.M., Gangemi, A., Steve, G.: Ontologies and Information Systems: The Marriage of the Century? In: International workshop on LYEE methodology (2002), http://www.loa-cnr.it/Papers/lyee.pdf (retrieved: January 18, 2005)
12. Swartout, W.R., Patil, R., Knight, K., Russ, T.: Towards Distributed Use of Large-Scale Ontologies. In: AAAI 1997 Spring Symposium on Ontological Engineering, USA (1997)
13. Uschold, M., Gruninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review 2, 93–155 (1996)

Chapter 6
Significance of Ontologies, Agents and Their Integration

6.1 Introduction

In this chapter, we will discuss some advantages of ontology- and agent-based systems. Ontologies provide machine readable and understandable domain knowledge and play an important role in knowledge representation, sharing and management, data semantics, intelligent information retrieval, mediation, natural language applications, and the like. Two important characteristics of agents are their autonomous and collaborative behaviour and, as such, agent-based systems are used to support distributed computing, dynamic information retrieval, automated service discovery, computational intelligence etc. We also describe the advantages of the integrated approach i.e. ontology-based multi-agent systems. Ontology- and agent-based computing are two different but complementary technologies; ontologies give intelligence to the system while the agents provide the system dynamics.

6.2 Advantages of Ontologies

The Artificial Intelligence community envisaged using ontologies to describe a variety of knowledge domains and providing this knowledge to applications. Since then, a conceptual bridge has been established between Information Systems and Artificial Intelligence (Guarino 1998).

Various communities have recognized the importance of ontologies and a large number of domain ontologies have already been developed.

The importance of ontologies in information sharing has been emphasized from the early days of ontology adoption by the information and computer communities. One important advantage of using ontologies is that information can be shared and reused. Moreover, ontologies support intelligent information retrieval and enable cooperation (Pretorius 2004).

In this section, we discuss and illustrate the various advantages and uses of ontologies.

6.2.1 Ontologies for Data Semantics

The underlying semantic framework supports effective and efficient data storage, retrieval and analysis tasks. A domain ontology can be used as the semantic

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 93–110.
springerlink.com

framework to underpin a variety of tasks such as systematic annotation of Web pages or the querying of heterogeneous information sources.

Ontologies can be used to develop an effective methodology to identify, classify, represent and reuse the existing knowledge within different information resources. Large amounts of heterogeneous and distributed information spread over various information resources can be given structure, and be accessed and maintained through the use of ontologies.

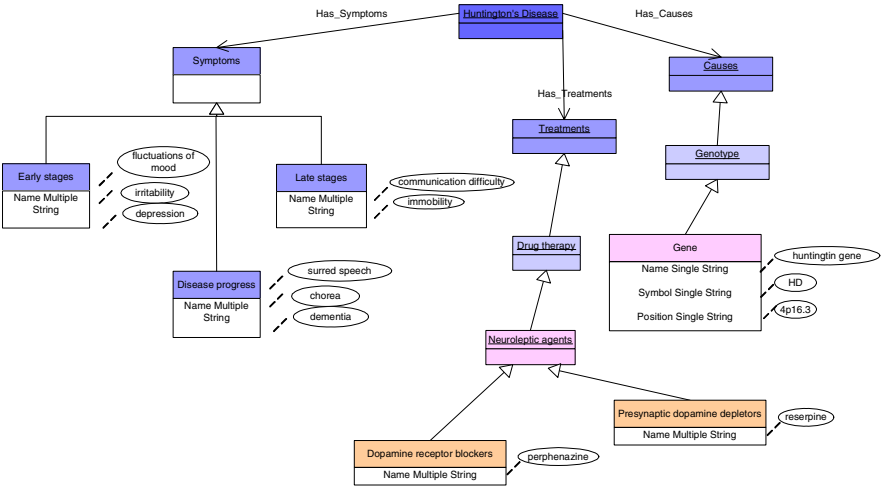


Fig. 6.1 Agents needs to share the same ontology to work together

As an example, let us assume that Agent D needs to query seven different websites. The agent has to have appropriate knowledge to do this. Let us assume that the Disease Ontology is used for this purpose. Agent D uses the Disease Ontology to assist with performing its actions, e.g. to understand the content of the websites and to communicate with agents associated with these websites. As we can see in our example from Figure 6.1, Agent I cannot understand the query of Agent D as it operates on the ontology in which the term 'disease' is expressed by the term 'illness'. The agents do not know that the terms 'disease' and 'illness' are being used to represent the same concept. In this case, the relevant information will escape their notice and will not be retrieved by Agent D. Two agents must share the same ontology in order to work together effectively.

6.2.2 Ontologies as Basis for Knowledge Sharing

Increased demand for data and knowledge sharing has resulted in increasing interest in ontologies. All applications that need to share knowledge need to have common understanding and representation of the fundamental concepts used to describe this knowledge. Specialized systems may organize the basic concepts differently for their own purposes but the fundamental concepts need to be

common to all applications. A common and shared vocabulary to represent the shared knowledge needs to be defined. Ontologies can be used for this purpose as they provide a shared and common understanding of a knowledge domain (Gómez-Pérez 1996). Ontologies enable knowledge sharing and reuse by several different applications and across organizational agents, including humans and software systems (Gómez–Pérez 1998). The application of ontologies plays an important role in establishing a framework that embraces all the knowledge shared by a certain community.

Information sharing can significantly reduce the costs of a project and significantly increase the success of the project. It is possible to rely on previous labour and experience, and use it for another application. Common knowledge is going to alert one to the possibility that one is undertaking the same experiments, by different research groups, thus saving their time and their resources. Moreover, ontologies can be used to help create a world-wide cooperative environment making it possible to take on a big task by dividing it among different research teams.

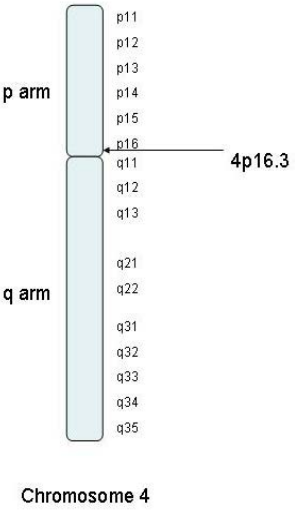


Fig. 6.2 Position of Huntington's gene on chromosome 4

For example, let us assume that various groups of medical researchers are looking for a gene mutation that causes Hungtington's disease. Here it is very important to reuse previously gained knowledge and experience, and further build up on this knowledge.To identify the correct position of the mutated gene, various medical research teams performed many different experiments. As human DNA is composed of 23 chromosomes, firstly the chromosome related to this disease needed to be identified. In this case, it was chromosome 4. As each chromosome has a short (p) and a long (q) arm, then secondly, the chromosome arm needed to be identified. In this example, it was p arm. As there are 191 million base pairs on chromosome 4, the exact gene position needed to be determined. It was 16.3, from base pair 3,113,411 to base pair 3,282,655. In the shared ontology, the meaning of

terms ‘human DNA’, ‘gene’, ‘chromosome’, ‘gene position’, ‘p arm’, ‘q arm’, ‘gene mutation’, ‘protein’ etc. needs to be clearly defined so that it is interpreted in the same way by the different teams that are collaboratively working towards the same goal of identifying the gene responsible for Huntington’s disease.

Huntington’s disease is a genetic disorder that develops in people who inherited a larger than normal Huntington (HD) gene. This larger gene produces an abnormal protein that begins to kill brain cells in middle age. The HD gene is located on the short (p) arm of chromosome 4 at position 16.3, from base pair 3,113,411 to base pair 3,282,655. This is shown in Figure 6.2.

Not only can the ontology be used to facilitate knowledge sharing, but an existing ontology itself can be shared and reused by other applications and/or domains if relevant. The ontological discipline promotes the reuse of ontologies through the use of ontology libraries. Knowledge related to various knowledge domains can be found within an ontology library. Different ontologies, each describing a particular knowledge domain, are available to the users. The ontology terms may be selected and restructured into a new ontology to be used by several different applications and/or in several different domains. Generic domain ontologies can be shared, reused and integrated for specific applications.

Three main uses of the available ontologies and similar resources have been identified:

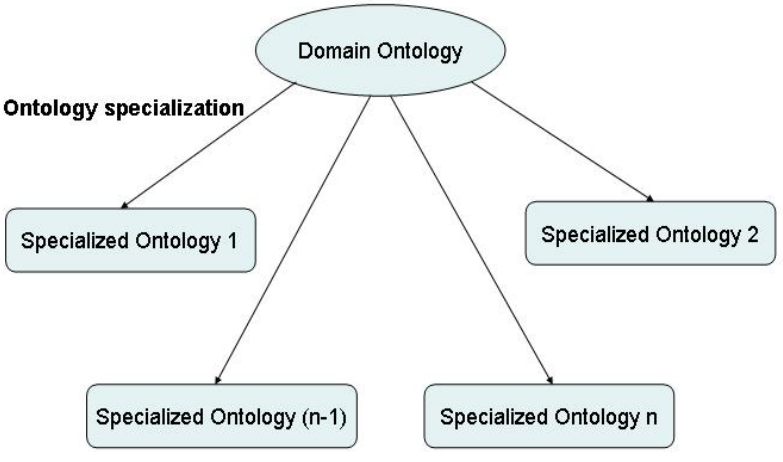


Fig. 6.3 Ontology specialization

(1) Ontology specialization. Domain ontologies can be specialized to be used by specific applications. This may require adding more detail to the original ontology. Generally, specialized ontologies have to commit to the concepts and relationships of the generic ontology from which they are derived. This idea is represented by Figure 6.3.

For example, we may have a Disease Ontology as a Domain Ontology, and A rthritis Ontology as Specialized Ontology 1, Depression Ontology as Specialized Ontology 2, Diabetes Ontology as Specialized Ontology 3 etc.

(2) Ontology integration. Sometimes, integration of various ontologies can be of great benefit for the ontology engineering community and computer society in general as it enables the sharing and reusing of the available knowledge. This knowledge may cover different knowledge areas that are often complementary to each other. This principle is shown in Figure 6.4.

For example, Symptoms Ontology that specifies information about disease symptoms, Treatments Ontology that specifies information about disease treatments and Causes Ontology that specifies information about disease causes can be unified into a Disease Ontology to specify information about disease symptoms, treatments and causes.

(3) Ontology design. A large number of distributed computerized resources such as lexicons, thesauri and glossaries are available (Meersman 2001) which can

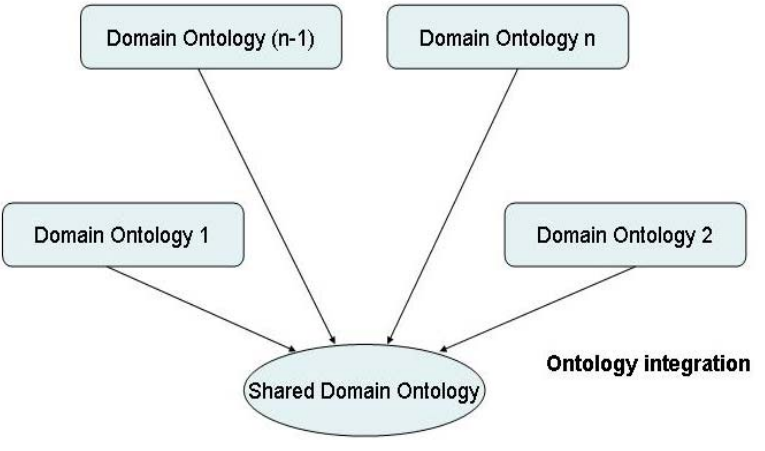
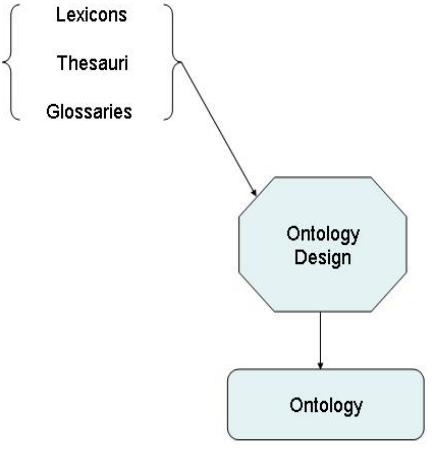


Fig. 6.4 Ontology integration

Fig. 6.5 Lexicons, thesauri and glossaries can be used to design ontologies



be used as a basis to design ontologies. For example, UMLS Metathesaurus can be used to design more specific medical ontologies. The idea is shown in Figure 6.5.

6.2.3 Ontologies as Basis for Knowledge Representation

Ontologies are rich and highly expressive knowledge models. Moreover, the inherited organisation of ontologies makes it easier to spot conceptual relationships in data. Use of ontologies enables the users to fully understand the knowledge domain and to identify possible links between different ontology concepts.

When all information on a specific topic is put together, it is also much easier to identify open or unresolved issues where more research is needed. In Figure 6.2, let us assume that the following information was found regarding the position of the Huntington gene. The test was positive (which means that this DNA region contains the Huntington gene) for region p15-q12. This is the chromosome region shown in red on Figure 6.6. However, some other research teams may provide information that will show the p15-p16 and q11-q12 regions to be negative (two chromosome regions shown in blue on Figure 6.6). These results imply that the gene of interest is located in the p16-q11 region. This chromosome region needs to be examined further in order to precisely locate the gene of interest (Huntington gene).

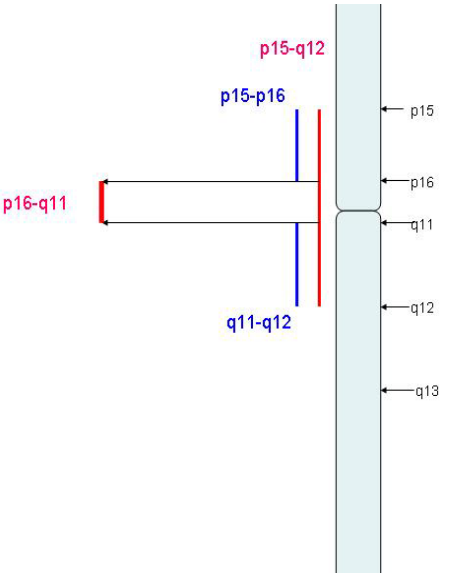


Fig. 6.6 Identification of links within the available information

6.2.4 Ontologies as Basis for Knowledge Management

Systematic management of the domain knowledge enables effective use of this knowledge. In order to structure and maintain large sources of heterogeneous and spatially dispersed information within the target community, expressive

knowledge descriptions need to be defined. Ontologies can be designed to provide the needed knowledge descriptions (Guarino and Welty 2000). This will enable efficient access and management of the available information. In this way, the value of the available information will be greatly increased. Use of ontologies for knowledge management would facilitate effective structuring and maintenance of the existing knowledge and addition of new knowledge.

Use of ontologies can help organize information within the target community. For example, teams can use ontologies as common support to classify the knowledge within an organization. This may allow users on a world-wide basis to access the available information much more quickly. In our example of Hungtington disease research, a common ontology can be used to represent the common knowledge about Hungtington disease. This ontology can be used by each research team to organize the knowledge available through them. This research community has a common goal of identifying the genetic cause of Huntington's disease. Because these research teams are sharing a common ontology, the knowledge made available by one research team is understandable by all other research teams of this research community. In this way, all available knowledge is effectively used as it is accessible and understandable by each member of the community when needed. Moreover, efficient organization of available knowledge enables the easy addition of new knowledge.

6.2.5 Ontologies for Intelligent Information Retrieval

Ontologies represent the domain knowledge and can be used for systematic annotation of information available through various information resources. The annotation supports intelligent information retrieval and enables the retrieval of highly specific information. It is much easier to understand the content of the information resource and retrieve relevant information when the available information comes with the meaning of this information. Most of the current search engines do not operate in this way. As they search for a particular string of letters, they often retrieve results that are completely outside the domain in which the user is interested. In our example from Figure 6.6, we may want to find information regarding p or q arm of a chromosome. If the search engine does not operate on the basis of a domain ontology (i.e. genetics ontology), it may retrieve information about arms of a body, military branches, company branches and other irrelevant information outside the domain in which we are interested.

6.2.6 Ontologies for Mediation

Two or more resources of an integrated system can be heterogeneous in their structure, content, organization etc. A mediator bridges these heterogeneities between different resources. A mediator uses a particular terminology and maintains the unified knowledge, but it does not maintain a database of objects. Instead, it projects a set of articulations to the underlying resources. An

articulation to a source is a set of relationships between the terms used by the mediator and the terms of that source (Tzitzikas et al. 2001).

Ontologies have been proposed as mediators as they can offer unified access to the heterogeneous resources (Weiderhold 1992). Uniformity is achieved through specification of the relationships between the ontology terms and the terms of the underlying resource.

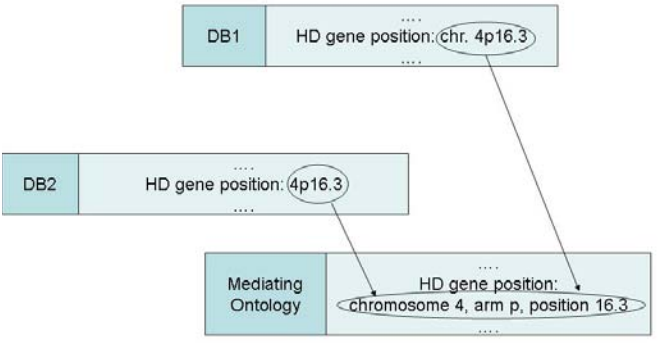


Fig. 6.7 Role of mediating ontology

Of special importance is that mediation recognizes the autonomy and diversity of data resources, information services and the user applications utilizing them. The mediator allows new participants to be easily inserted, thereby enabling the overall system to grow.

For our example using the search for the genetic cause of Huntington's disease, let us assume that database (DB) 1 provides information regarding gene position in the following form: 'chr. 4p16.3'. DB2 may represent the same information in a different way: '4p16.3'. We can then use an ontology to mediate between the different databases. In this ontology, the gene position is represented in the following way: 'chromosome 4, arm p, position 16.3'. We map 'chr. 4p16.3' to 'chromosome 4, arm p, position 16.3' for DB1 and '4p16.3' to 'chromosome 4, arm p, position 16.3' for DB2. The different databases and the mediating ontology represent the same information albeit in a different way. The mediating ontology enables an agent to interpret the information available via DB1 and DB2 in the same way. This example is illustrated by Figure 6.7.

6.2.7 Ontologies for Natural Language Applications

Natural language applications can involve various knowledge acquisition techniques. As an ontology provides factual and contextual information about different knowledge sources, it can be used by natural language applications. The ontology will enable machines to understand, analyze and extract the knowledge available through the sources.

In our example from Figure 6.6, let us say that three different electronic documents provide information regarding the position of the Huntington gene. This information needs to be recognized within the text and extracted from the text. This is possible only if the structure of the ontology used by the machine to extract the knowledge corresponds to the structure of the ontology used to annotate the text available through the electronic documents. The information extracted may then be positive (means this DNA region contains the gene) for region p15-q12 and negative for regions p15-p16 and q11-q12. This example is illustrated in Figure 6.8. Following this, analysis of this data needs to take place. The results imply that the gene is located in the region p16-q11.

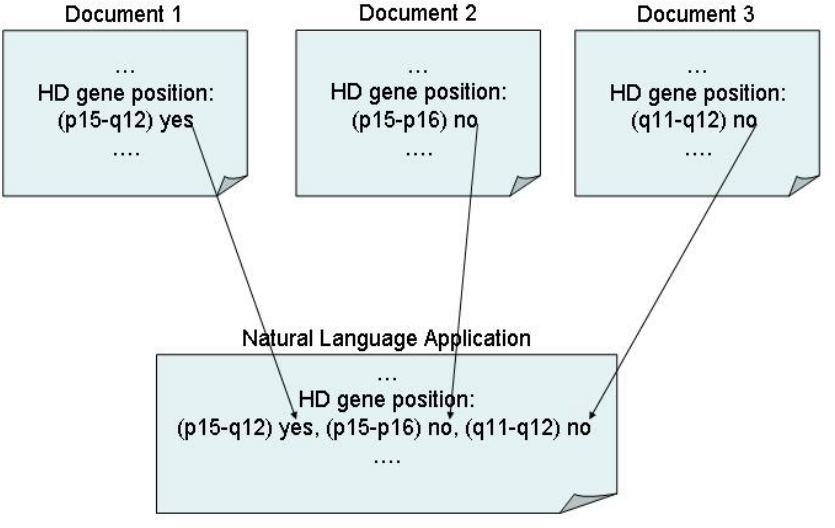


Fig. 6.8 Using ontologies in knowledge acquisition with Natural Language Applications

6.3 Advantages of Agent-Based Systems

Multi-agent systems support efficient information retrieval and design of flexible, adaptive and distributed systems and have the ability to cooperatively work with other agents. Moreover, mobile agents have the ability to migrate between various information resources and cooperatively work with other agents. Use of agent systems enables us to model, design and build complex information systems. Multi-agent systems are being increasingly used in various domains.

Multi-agent systems that are based on a common and shared ontology can have even more advantages than simple multi-agent systems. Ontology-based, multi-agent systems support the design of efficient information systems. For example, a multi-agent system may be designed for information retrieval. Here it is important to retrieve appropriate information from reliable sources without having to devote much time and effort to look for, analyze, evaluate and filter it. Ontologies can play an important role here; agents can use the common ontology to function

efficiently within the system and the information available through the information resources may be annotated using the same ontology so that the agents will be able to understand this information and retrieve it if it is found to be relevant.

In the rest of this section, we describe some advantages of multi-agents systems (Nealon and Moreno 2002).

6.3.1 Agents are Autonomous

Agents are able to autonomously act and perform their tasks. For this reason, they are particularly suitable to be used in information systems in which each component can keep its autonomy and independence from the rest of the system.

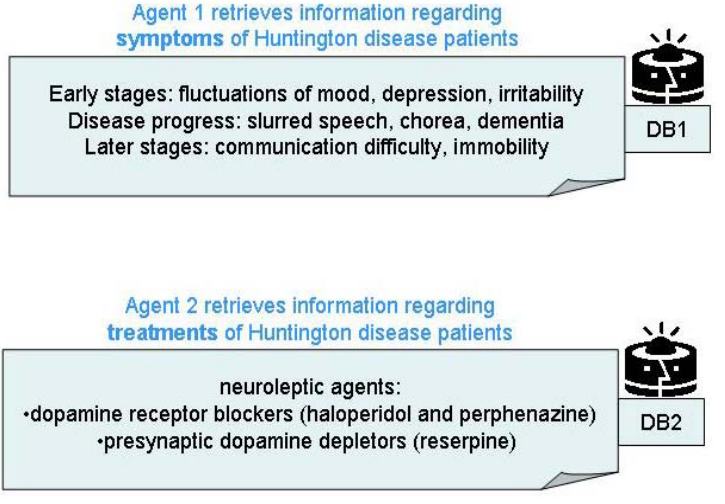


Fig. 6.9 Autonomous agents work together towards the shared goal

For our example of Huntington's disease, we may want to design a multi-agent system for the purpose of intelligent and dynamic information retrieval regarding Huntington's disease. This multi-agent system would be composed of a number of autonomous agents. One particular agent can have a function of, for example, gathering information about drug treatments for this disease. At the same time, another agent within the same multi-agent system may have the function of information retrieval regarding disease symptoms. Those two agents are shown in Figure 6.9. The information retrieval task of the first agent is independent of the information retrieval task of the second agent as well as of the information retrieval tasks of other agents within the system. However, they are designed to work together towards the same goal, namely, efficiently retrieving information about Huntington's disease.

6.3.2 Agents Support Computational Intelligence

Intelligent multi-agent information systems can be designed so that its components may be located and running in different information resources. Agents can negotiate, coordinate their actions and collaborate during the process of solving their problem. A problem can be partitioned, different subtasks to be solved can be distributed among different agents, information can be exchanged, and the partial results can be combined for the solution of the original problem. Each of the agents is responsible for a part of the knowledge required to solve the problem.

In our example of a multi-agent system for information retrieval about Huntington's disease, the task can be partitioned so that, for example, one agent is responsible for the information retrieval regarding symptoms of this disease, while another agent needs to retrieve information regarding disease causes, the following agent about disease treatments, and so on. This idea is shown in Figure 6.10. The information agents are specialized in analyzing the information from different information resources, filtering redundant or irrelevant information and retrieving specific information. After information retrieval, information integration needs to take place and the retrieved information can be presented to the user in a meaningful way.

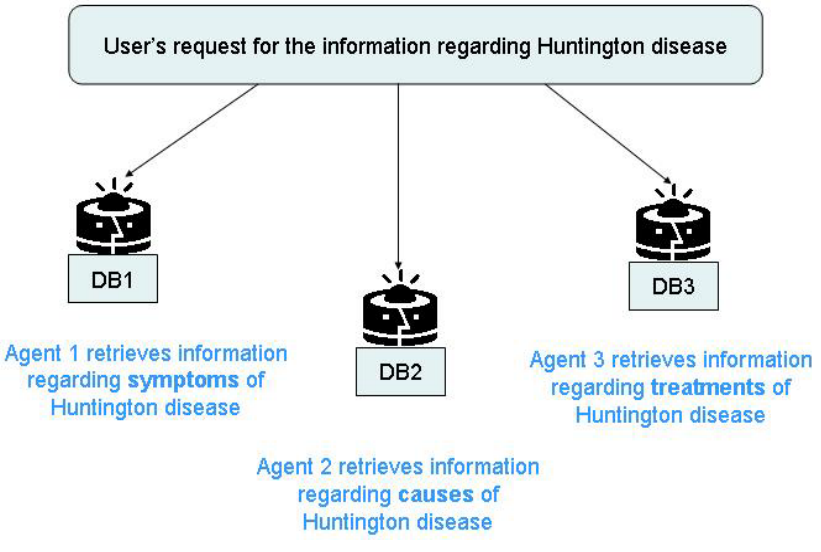


Fig. 6.10 Task sharing among various information agents

6.3.3 Distributed Mobile Agent-Based Computing

Mobile agent-based computing can sometimes significantly increase the effectiveness and efficiency of information retrieval process. A mobile agent migrates to and

collects information from various distributed information resources. Mobile agents are capable of moving across different databases, analyzing available data and selecting up-to-date information. The data are analyzed and manipulated in order to transfer the required information. In this way, mobile agents also diminish network traffic caused by large amounts of data transfer, as only aggregated or analysed information is transferred, not all of it.

The implementation of mobile agents can be of great importance within communities where examination, experiments and gaining of new knowledge are still in progress. In our example, an agent specialized in information retrieval regarding symptoms of Huntington's disease may need to visit various information resources, each corresponding to a medical research team that examined a specific disease stage (e.g. early, later). For this reason, our agent needs to migrate from one resource to another and at each resource it needs to retrieve the relevant information. The route of migration is shown in Figure 6.11. In the beginning, this disease is characterized by unexplained fluctuations of mood, with patients becoming more depressed or irritable than usual. There is a slurring and slowing of speech. As the disease progresses, patients develop slurred speech, writhing movements known as chorea, and dementia. Eventually, the patients have difficulty communicating and are confined to a wheelchair or bed. All this information is available via DB1, DB2 and DB3 and needs to be retrieved and integrated in order to provide a satisfactory answer.

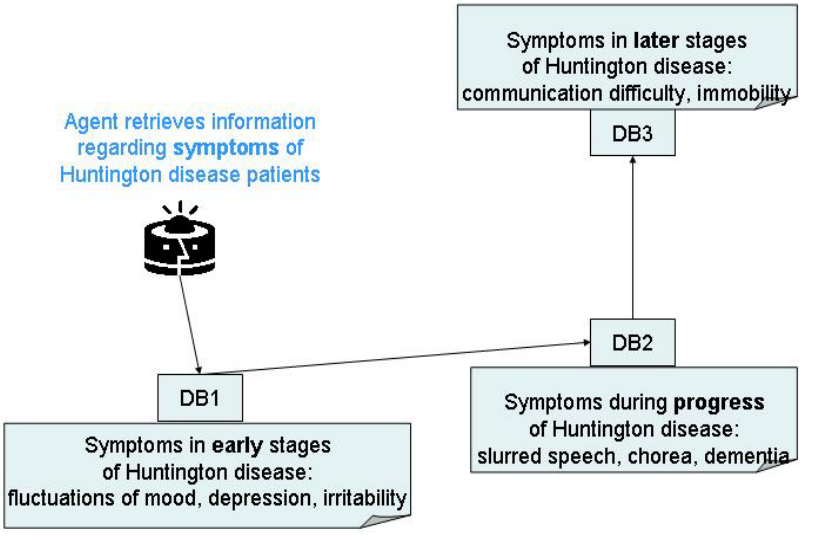


Fig. 6.11 Migration of information agent in order to retrieve information regarding symptoms of Huntington's disease

6.3.4 Agents Collaboration and Cooperation in Their Activities

An ontology provides agents with a vocabulary to represent and communicate knowledge, make statements, and ask queries about a subject domain. In this way, ontologies enable agents to share their tasks, cooperate, and coordinate their actions.

Agents of a multi-agent system need to share, commit to and use a common language (ontology) in order to understand each other, share their tasks, collaborate and cooperate with each other. In the example of a multi-agent system designed for the purpose of information retrieval regarding Huntington's disease, all agents of the system need to share and commit to a common ontology. This ontology is designed to capture the knowledge about Huntington's disease. Commitment of the agents to this common ontology enables them to understand and communicate with each other, effectively share tasks and integrate their results. Another agent, for example, from a multi-agent system designed to retrieve the information about various cars is committed to, and operates on, a different domain ontology (in this case, ontology of cars) and will not be able to communicate with the agents of our system. The lack of communication is illustrated in Figure 6.12.

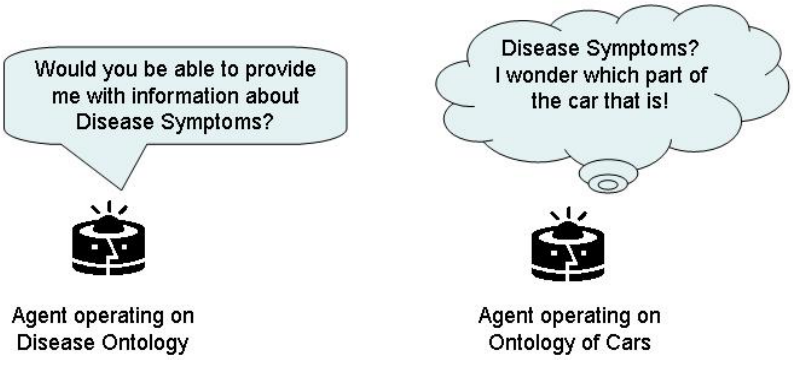


Fig. 6.12 Agents committed to different domain ontologies are unable to communicate with each other

6.3.5 Agents Support Automated Service Discovery

Agents can use an ontology to describe information resource content or a service in a way that other agents 'understand'. This enables the other agents to locate relevant information or a service which usually results in mutually beneficial actions.

For example, an agent located within an information resource may describe the content of this information resource using an ontology. Another agent that shares and is committed to the same ontology is then able to 'understand' the content of this information resource and can retrieve relevant information from this information resource.

6.4 Ontology and Agent-Based Systems Complementing Each Other

In previous sections, we discussed the advantages of the integrated ontology/multi-agent systems such as the use of an ontology to allow agents to communicate and collaborate with each other, to share tasks effectively, to support automated service discovery etc. In this section, we will mention some additional advantages of ontology-based, multi-agent systems.

The multi-agent system uses an ontology for intelligent and dynamic information retrieval. For example, an ontology-based, multi-agent system can extract and manipulate information from various information resources. Within such a multi-agent system, the ontology can be used at different levels including problem decomposition, locating relevant information and retrieval of this information, agent's communication, information analysis and manipulation, and information presentation.

6.4.1 Problem Decomposition

An ontology can provide the agents with knowledge of the task structure. The overall problem usually needs to be decomposed into smaller subproblems to facilitate solution. This kind of decomposition is hierarchical so that subproblems are further decomposed into smaller sub-subproblems, and so on. The goal of the problem decomposition is to reach a stage where the subproblems are of an appropriate granularity so that they may be solved by individual Information agents. This process of problem decomposition and task assignment to different agents assumes that the agents must have knowledge of the task structure and must know how the task is put together.

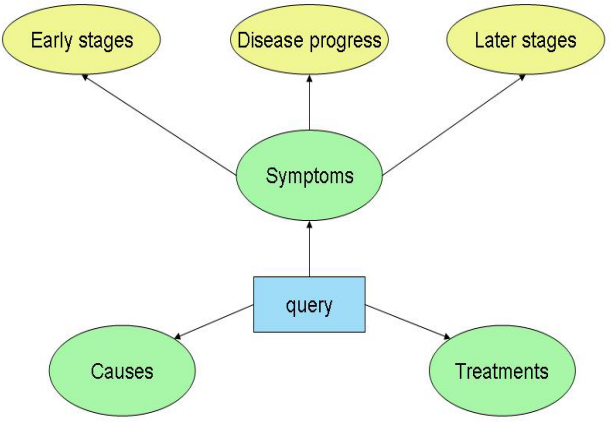


Fig. 6.13 Hierarchical problem decomposition according to the ontology branches

In our example, the Huntington's Disease Ontology can be designed to have hierarchical structure branching out in the following Subontologies: Symptoms, Causes and Treatments. The query of a user who is interested in information about Huntington's Disease, can be firstly partitioned according to the three Subontologies (subproblems). The three Subontologies are shown in green on Figure 6.13. These Subontologies are further subdivided into Sub-Subontologies. For example, Symptoms Subontology can be divided into Early stages, Disease progress or Later stages Sub-Subontology, shown in yellow in Figure 6.13. The user's query is partitioned according to the Sub-Subontologies, and these sub-subproblems are assigned to individual Information agents.

6.4.2 Locating and Retrieving of Information

An ontology is used to locate and retrieve the requested information. Information content within an information resource can be described using an ontology. This provides a more controlled and systematic way to look for the requested information. An agent that commits to this ontology is able to 'understand' the information contained within these resources and to exactly locate and retrieve the requested information.

In our example, agents are faced with the following question: In which of the available resources is it possible to find the requested information, and where exactly within the resource? Different databases are assigned to different agent according to their functions within the system. For example, information resources containing information about disease symptoms are assigned to agents responsible for retrieval of information about symptoms of the disease in question. To make use of all the available information, the information within each of the information resources needs to be annotated using a shared ontology. This enables the agents to 'understand' the information available in these information resources, and retrieve the requested information.

6.4.3 Agent Communication

A shared ontology enables agents working together cooperatively to communicate with each other. Use of the ontology permits coherent communication between the different agents and enables them to effectively and efficiently perform the complementary actions. The ability of agents to effectively and efficiently communicate with each other is of the greatest importance during complex actions such as task and result sharing.

In our example of retrieval of information about Huntington's Disease, all the agents of the system need to share, commit to and use the common ontology (that represents the knowledge about Huntington's disease) when communicating with each other. Concepts such as 'symptoms', 'causes' and 'treatments' need to be precisely defined by this ontology and these concepts should be interpreted in the

same way by different agents of the system. This will facilitate the effective sharing of tasks and results the multi-agent system.

6.4.4 Information Analysis and Manipulation

As the ontology represents the domain knowledge, it can be used by the agents to analyze and manipulate information. The ontology provides agents with the domain knowledge and enables them to reason about the information, detect any redundancy and/or inconsistency, and remove irrelevant information.

In our example, the Huntington's Disease Ontology needs to be designed so that it represents the knowledge about Huntington's Disease and enables reasoning about this knowledge. This ontology needs to contain information about p and q chromosome arms enabling the agents to conclude that DNA region p16-q11 is smaller then p15-q12 (see Figure 6.6) and select the smaller DNA region to be incorporated into the overall answer to the user's query. As stated previously, a smaller DNA region of interest means that a smaller DNA region needs to be examined and searched for the gene of interest (which causes the disease in question when mutated). This results in quicker identification and location of the mutated gene.

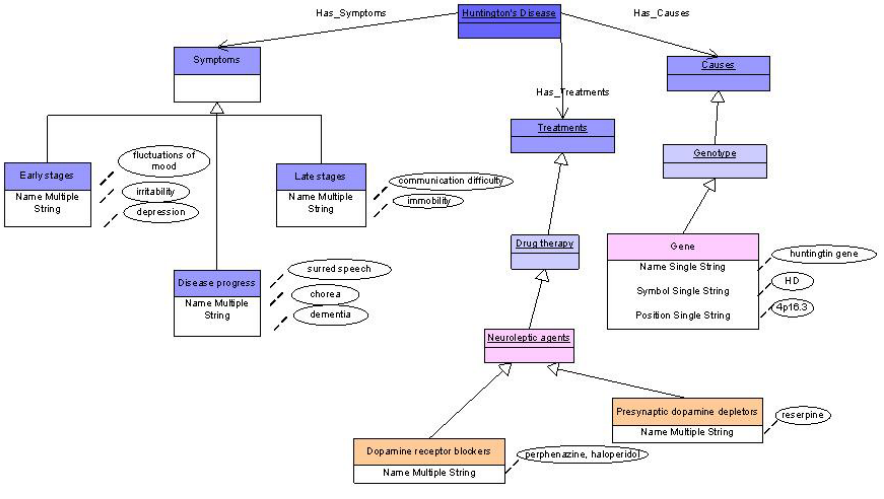


Fig. 6.14 Presentation of the retrieved information regarding Huntington's Disease according to the three Ontology branches: Symptoms, Causes and Treatments

The ontology is used to structure information and give an overview of the domain knowledge, and allows one to present information in a meaningful way. Moreover, the hierarchical organization of ontologies adds a taxonomical context to this information and makes it easier to spot conceptual relationships in data.

In our example of Huntington's Disease, the retrieved information can be represented according to the tree different subontologies: Symptoms, Causes and Treatments Subontologies. This is shown in Figure 6.14. Within the multi-agent system, the three Huntington's Disease Ontology branches (Subontologies) are used as the basis for problem and task sharing, and the assembly of results

We gave an illustrative example of an ontology-based multi-agent model for the information retrieval of knowledge related to Huntington's Disease. The Huntington's Disease Ontology can be integrated with a multi-agent system designed to aid medical researchers, physicians and patients in retrieving relevant information regarding Huntington's Disease. We believe that a similar approach is also applicable to other knowledge domains.

6.5 Conclusion

In this chapter, we have discussed various advantages of ontology-based systems and the role of ontologies in data semantics; knowledge sharing, representation and management; intelligent information retrieval; mediation and natural language processing techniques. We have also discussed the autonomous, collaborative and mobile nature of agents which forms a strong basis for the design of distributed and mobile multi-agents systems. As the ontology- and agent-based systems address different and complementary aspects of the same problem, we believe that the integration of the two approaches would result in a highly effective and efficient system. We discussed different stages within the multi-agent system that could benefit form the integrated approach including problem decomposition; location and retrieval of the information; communication between different agents; information analysis and manipulation; and information presentation.

In the following two chapters we will present a ten-step methodology for the design of such integrated systems.

References

1. Gómez-Pérez, A.: Towards a Framework to Verify Knowledge Sharing Technology. Expert Systems with Applications 11, 519–529 (1996)
2. Gómez-Pérez, A.: Knowledge Sharing and Reuse. In: The Handbook on Applied Expert Systems, pp. 1–36 (1998)
3. Guarino, N.: Formal Ontology in Information Systems. In: Conference on Formal Ontology in Information Systems (FOIS 1998), pp. 3–15 (1998)
4. Guarino, N., Welty, C.: A Formal Ontology of Properties. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS, vol. 1937, pp. 97–112. Springer, Heidelberg (2000)
5. Nealon, J.L., Moreno, A.: The Application of Agent Technology to Health Care. In: AgentCities: research in large scale open agent environments Wotkshp at the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), pp. 169–173 (2002)

6. Meersman, R.: Ontologies and Databases: More than a Fleeting Resemblance. In: International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organisations, OES-SEO2001 (2001), http://www.starlab.vub.ac.be/staff/robert/Research%20Papers/Ontologies%20and%20Databases-OES-SEO%20paper.pdf (retrieved: December 10, 2003)
7. Pretorius, A.J.: Ontologies - Introduction and Overview (2004), http://www.starlab.vub.ac.be/teaching/Ontologies_Intr_Overv.pdf (retrieved: January 19, 2004)
8. Tzitzikas, Y., Spyratos, N., Constantopoulos, P.: Mediators over ontology-based information sources. In: Second international conference on Web Information Systems Engineering (WISE 2001), pp. 31–40 (2001)
9. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25, 38–49 (1992)

Chapter 7
Design Methodology for Integrated Systems - Part I (Ontology Design)

7.1 Introduction

Ontologies are high expressive knowledge models and as such increase the expressiveness and intelligence of a system. Ontologies were introduced into the computer and information community to be used by various agents and for different applications addressing the problems of various knowledge domains.

There is not a single widely accepted ontology design methodology. We discussed some of the currently available methodologies in Chapter 5. We propose an Onto-Agent Methodology as the first methodology that unifies the different approaches of the existing ontology and multi-agent systems design methodologies. The five steps of the first part of the Onto-Agent Methodology (Onto Methodology) are described in this chapter and are accompanied by some illustrative examples. The second part of the Onto-Agent Methodology (Agent Methodology) also consists of five steps which are described with illustrative examples in the next chapter.

7.2 Generalization and Conceptualization of the Domain

The aims of this step of the ontology design process are to:

- Identify the main ontology concepts and relationships
- Establish the ontology framework
- Determine organization of the concepts within the ontology

For this reason, four important points need to be taken into account:

- Community for which the ontology is being designed
- Purpose of the ontology
- Knowledge domain that will be represented by the ontology
- Applications based on the designed ontology

7.2.1 Ontology Communities

Within the community for which the ontology is being designed, smaller communities can be further differentiated. Within the ontology context, we have identified three groups that may be associated with the term “community” (see Figure 7.1).

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 111–126.
springerlink.com

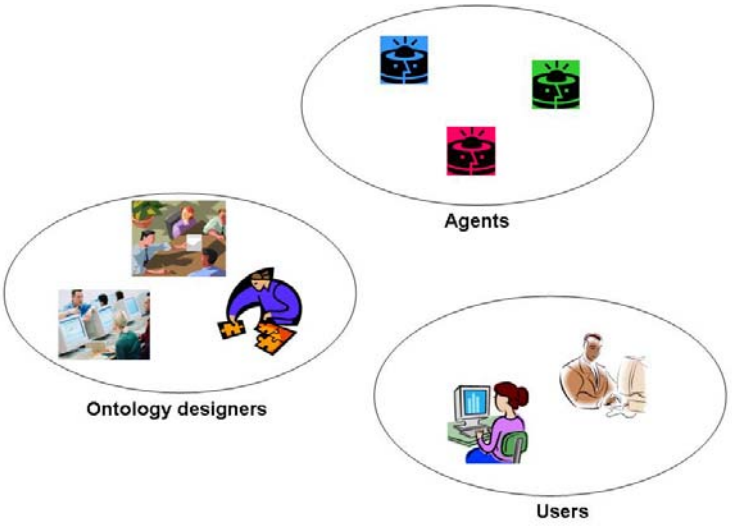


Fig. 7.1 Communities associated with ontology design, use and application

Firstly, we are talking about the community of ontology users. For example, we may have a broad community of researchers examining different aspects of Huntington's Disease. Within that community, we may have smaller communities working on specific topics such as specifying disease symptoms, identifying disease causes, looking for disease treatments etc.

Secondly, we have the community of agents. The ontology may be designed for a community of agents that are cooperatively working towards the same goal. For example, we may have a multi-agent systems designed for retrieval of information about Huntington's Disease.

Thirdly, we have the community of ontology designers. An ontology is best designed by a group of researchers with different and complementary capabilities because of the complexity of the ontology design process.

The three identified types of communities are actually a part of one big community as they are working towards the same goal but on different levels. All the members within the community need to agree and commit to the designed ontology.

7.2.2 Purpose of the Ontology

The main purpose of the ontology is to represent the knowledge of a specific knowledge domain. The following questions also need to be considered:

- why does this knowledge need to be represented?
- what is to be expected from the proposed ontology-based application?

- who are the end users and what advantages are offered to them through this application?

Ontologies are usually proposed as a solution to some problems. For this reason, the purpose for which an ontology is designed is strongly related to the issues present within a certain community. Automatic, problem-solving information systems are facilitated through the use of ontologies.

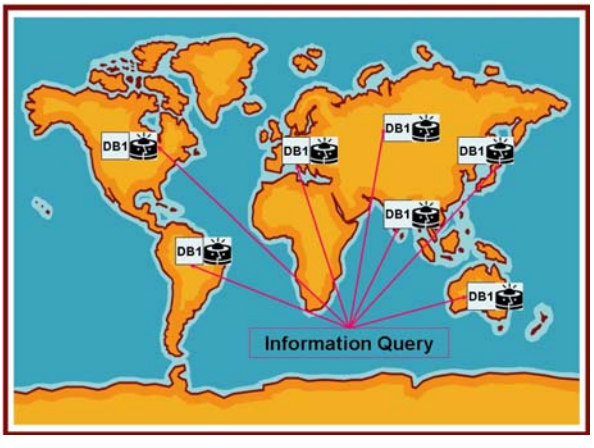


Fig. 7.2 Agents support information retrieval from various databases

In our example, we have the issue of distributed, heterogeneous and autonomous information resources spread around the world that contain information regarding Huntington's Disease. This situation is represented in Figure 7.2. If the use of the available information is to be maximized, the information must be unified. A multi-agent system, together with an ontology (which will capture the knowledge about Huntington's Disease and support the intelligent actions of the multi-agent system), will need to be designed in order to efficiently and effectively retrieve the information from various information resources.

7.2.3 Ontology Domain

Ontologies are domain knowledge representations. The level of detail represented by the ontology depends mainly on the application for which the ontology is being developed. For some applications, it may be necessary to capture a small part of the domain but in greater detail. Other ontologies may cover a large portion of domain knowledge not including detailed descriptions.

As each knowledge domain is being constantly researched and new knowledge is continuously emerging, a huge ontology may be needed to cover all the knowledge within a domain. Due to disciplinary interconnections, one knowledge domain may be closely related to another knowledge domain.

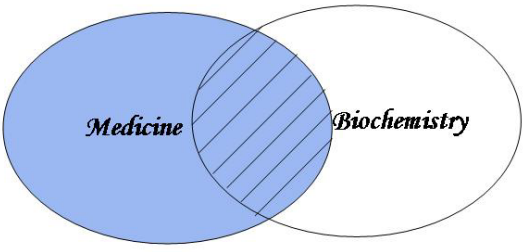


Fig. 7.3 Intersection of medical and biochemistry knowledge domains

In our example, the Huntington's Disease knowledge domain is at the intersection of the medical domain and the biochemistry domain. This idea is illustrated in Figure 7.3. The medical domain deals with human diseases but some biochemistry experiments may need to be performed in order to identify the causes of the disease or to design effective drugs. Such biochemistry experiments may include DNA and protein sequencing and manipulation.

7.2.4 Application of the Ontology

During the design process, we need to be aware of the community of people for whom the ontology is being developed, and the purpose of the ontology and applications for which the ontology is being designed. This will allow us to establish the borders within which the ontology will be designed.

Even though the ontology needs to be designed with a specific application in mind, ontology dependence on any one application needs to be kept to a minimum. Separation of the design of the ontology base from the design of ontology commitments may support this objective. Here, the ontology base will remain application-independent and the application dependency will be incorporated in the ontology commitment layer. This makes it possible for the different applications within the same domain to use the same ontology base and include the variations in the ontology commitment layer.

As an example, consider the task of constructing an intelligent retrieval system for the information about Huntington's Disease. This is the ontology application. To define the ontology community, user categories will be considered. The two main user categories in this domain may be medical researchers who are mainly interested in the causes of a disease, and physicians and patients who are faced with a situation of a disease and are mainly interested in symptoms and treatments of a disease. Once the purpose of the ontology is revealed, a decision can be made on how to represent the domain knowledge. Huntington's Disease Ontology (HDO), that represents the general knowledge regarding Huntington's Disease, can be represented through three subontologies: Causes, Symptoms and Treatments (shown in Figure 7.4). Each of these subontologies will be developed using the concepts from the existing ontologies, where possible. In the next section, we discuss the aligning and merging of existing ontologies.

Fig. 7.4 Two main user categories of the intelligent retrieval system regarding knowledge about Huntington's disease

7.3 Aligning and Merging Ontologies

The goal of this step in the ontology design process is to:

- identify available ontologies suitable for reuse
- choose a suitable ontology merging and alignment tool
- import the source ontologies
- identify correspondence among these ontologies
- align and/or merge ontologies

7.3.1 Identify Suitable Ontologies for Reuse

Researchers in the ontology-design field have developed different ontologies for various knowledge domains. A large number of the existing ontologies cover overlapping domains and/or are expressing the same domain in different ways. Many ontologies are already available in electronic format in the library of ontologies. The need for sharing and reusing this body of knowledge becomes increasingly critical.

It is possible to build a new ontology by reusing and adapting terms from other ontologies. An ontology is designed to be agreed upon and shared within a community. It would be contradictory to the definition of ontology if each domain application used its own ontology. Rather, consistency across the existing ontologies in the domain needs to be achieved.

Rather than creating a completely new ontology, concepts from the existing ontologies can be used where available. The way that these concepts are structured within the existing ontologies may not be suitable for use by some other application. In that case, the concepts from the existing ontologies may be used but must be organized in a way that can be used for that particular application.

In our example of retrieval of information about Huntington's Disease, other medical ontologies such as LinkBase (Ceusters et al. 2001), UMLS (Bodenreider

2004) and SNOMED (Ceusters et al. 2004) can be used. The idea is illustrated in Figure 7.5. It is possible to use concepts from the existing ontologies but these concepts must be organized according to the three branches of the Huntington's Disease Ontology (symptoms, causes and treatments). The resulting ontology can then be used for the purpose illustrated in Figure 7.2.

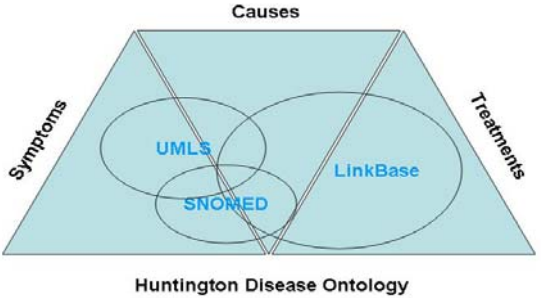


Fig. 7.5 LinkBase, UMLS and SNOMED ontologies can be used to cover a portion of knowledge required by the Huntington's Disease Ontology

7.3.2 Define Merging and Alignment Tool

Manual ontology merging and alignment using conventional editing tools is difficult, labour intensive and error prone (Noy and Musen 2000).

There are many tools available for ontology merging and alignment such as Chimaera (McGuinness et al. 2000), FCA-Merge (Stumme and Maedche 2001), PROMPT (Noy and Musen 2000). Each tool has some advantages and disadvantages. The tool characteristics, together with the nature of the application, need to be analyzed carefully in order to identify the most appropriate tool for the task at hand.

7.3.3 Import the Source Ontologies

The available ontologies are imported into a new ontological development environment, modified and adapted to be reused by another application. A set of sharable and reusable ontologies can be used during this process.

As different ontologies may be designed using different tools and these may have different formats, it may be necessary to convert all these ontologies into one format in order to use them efficiently. In Figure 7.6, different ontology formats are represented by different colours and the standardized format is represented by green. This standardized format needs to be supported by the chosen ontology alignment and merge tool. The result of alignment and merging of the four different ontologies is represented as 'Ontology 1, 2, 3, 4'.

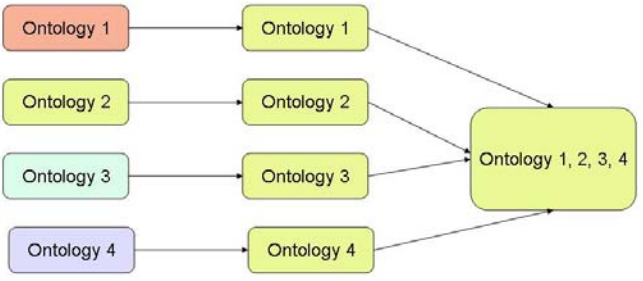


Fig. 7.6 Ontologies originally found in heterogeneous formats are converted to the same format to be aligned and merged

7.3.4 Identify Ontology Correspondences

It is possible for two different ontologies to describe the same knowledge domain and represent the same facts in two different ways. Also, different ontologies which need to be aligned/merged may cover overlapping domains. The way in which these ontologies express the knowledge in the overlapping domain needs to be clarified.

For these reasons, it is important to recognize correspondences among the source ontologies. This step involves identification of overlapping concepts, concepts with similar meaning but different names i.e. synonyms, and concepts unique to each of the source ontologies.

This work must be done in both ontology merging and ontology alignment. Ontology merging combines all the information of the source ontologies into one big ontology, whilst in ontology alignment, the sequences of the source ontologies are aligned but kept separate (Noy and Musen 2000).

7.3.5 Align and/or Merge Ontologies

In the aligning of the different ontologies, the two original ontologies persist but links are established between these ontologies. Alignment is usually performed when the source ontologies cover complementary domains.

In the merging of the different ontologies, a single coherent ontology is created as a merged version of the source ontologies.

Ontology alignment and merging processes are illustrated in Figure 7.7. The three different ontologies (shown in black, blue and red) are merged into a single ontology on the top of the figure (ontology merging) while the correspondences between the different ontologies are identified at the bottom of the figure (ontology alignment).

For example, LinkBase ontology (Montyne 2001) is the largest medical ontology and consists of 1.5 million language-independent medical and general-purpose concepts. We can use a part of the LinkBase ontology to construct the abovementioned Huntington's Disease ontology and its three subontologies

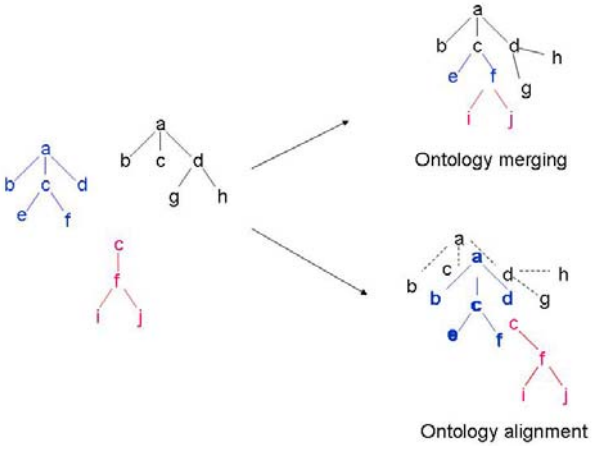


Fig. 7.7 Merging and alignment of three different ontologies

(symptoms, causes, treatments). If all the required information cannot be represented by the concepts of the LinkBase ontology, UMLS (Bodenreider 2004) or other medical ontologies can be used until the ontology content is completely developed.

7.4 Formal Specification of Conceptualization

Generally, the preliminary idea of the ontology is firstly represented in an informal way. Usually, group discussions expose some minor issues which need to be addressed. Upon the agreement and modification, the ontology is represented in a formal way. Formal ontology representation enables the ontologies to be understood and used by computers.

Different formalization approaches can be used to formalize ontologies. The choice of a particular formalization approach depends on the purpose for which the ontology is being built. For some purposes, an ontology may be formalized as a simple hierarchy of concepts, while for other purposes it may be necessary to choose a more expressive way to formalize an ontology.

The formalism for specifying an ontology should be as simple as possible. We adopted the two-layer approach for formalizing ontologies (De Bo et al. 2003; Jarrar and Meersman 2002): specifying of ontology conceptualization and specifying of ontology commitments. The ontology conceptualisation is defined in the ontology base, the first ontology layer in which a set of (binary, even) conceptual relationships are defined. The other domain knowledge and its formal semantics are specified in the second ontology layer or commitment layer.

When specifying ontology conceptualization, the goal is to define:

- ontology concepts
- relationships between these concepts
- identify groups of related concepts

7.4.1 Ontology Concepts

A concept is a proxy formed in our minds. "Term" is the name given to a concept. Reaching agreement about the definition of a concept is a difficult task. As every concept definition is dependent on the definition of other concepts, one needs to rely on a commonly accepted understanding of some basic terms.

For example, Wikipedia defines symptom as "a sensation or change in health function experienced by a patient". Wordnet defines symptom as "any sensation or change in bodily function that is experienced by a patient and is associated with a particular disease". We understand that Wikipedia and Wordnet are talking about the same thing (disease symptom) but they are defining it in a different way. If we are to make programs and applications of Wikipedia and Wordnet work together, they need to commit to one of the two definitions or to a third definition which can be a merged version of the existing definitions. Only then would the applications based on Wikipedia and Wordnet be able to effectively function as one.

7.4.2 Relationships between Concepts

Relationships between the ontology concepts need to be precisely defined. This step involves identifying intentional and extensional relationships, as explained in the previous chapter. Intentional relationships (conceptual relationships) map every concept from the domain to the ontology structure. Extensional relationships between the chosen domain concepts within the ontology structure represent the domain knowledge that we want to describe and represent.

The purpose and scope of the ontology will determine the domain concepts that will be mapped by intentional relationships and the kind of extensional relationships that will exist among the mapped concepts. In the abovementioned definition of symptoms according to Wordnet, we can identify the following related concepts: sensation, change, function, body, patient, disease. These concepts are related to each other as defined by binary relationships A, B1, B2, C1, C2, D1 and D2 shown in Figure 7.8. Note that relationship A becomes a concept in relationship B1 and relationship B1 becomes concept in relationship C1, and similar.

7.4.3 Groups of Related Concepts

A context can be defined as a mapping from Δ to a collection of sources; for example, mapping to a corpus of journals on the same topic. In some cases, related concepts may be grouped into a subontology of the ontology.

In our example shown in Figure 7.8, we could identify a group of related concepts. As these concepts represent the knowledge about disease symptoms, they can be grouped together in the disease symptoms subontology of the disease ontology. Other subontologies of the disease ontology may be disease causes or disease treatments subontologies, and similar.

Wordnet: *symptom is any sensation or change in bodily function that is experienced by a patient and is associated with a particular disease*

concepts: sensation, change, function, body, patient, disease

A = a function may belong to a body (bodily function)

B1 = a change may occur in A (bodily function)

B2 = a sensations may occur in A (bodily function)

C1 = B1 (a change in bodily function) may be experienced by a patient

C2 = B2 (a sensation in bodily function) may be experienced by a patient

D1 = C1 (a change in bodily function experienced by a patient) may be associated with a particular disease

D2 = C2 (a sensation in bodily function experienced by a patient) may be associated with a particular disease

Fig. 7.8 Relationships between related concepts

7.5 Formal Specification of Ontology Commitments

The commitment layer contains ontological commitments that logically connect, instantiate and constrain lexons of the associated ontology base. Each commitment defines a set of rules or axioms in a given syntax providing a specific interpretation of a lexon or a subset of lexons of the ontology base.

In this way, the commitment layer adds the semantics to the ontology model. An agent or application can further extend or specialize the ontology base by modifying its commitment layer. For this agent, the ontology base together with the commitment layer represents the real world in which it functions and operates. When defining the commitment layer, the goal is to:

- identify the ontology commitments
- formalize the commitments
- identify reusable knowledge components

7.5.1 Identify the Ontology Commitments

It is important to precisely define ontology commitments. As an example, consider two concepts: ‘disease’ and ‘cause’, and relationships between those two concepts ‘has/is-of’. The commitment associated with this relationship is ‘each disease has at least one cause’(if a disease had no cause this will contradict existence of this disease). But it may be possible that a cause of a specific disease cannot be found within the information available to a specific application. If a system operates using the ‘each disease has at least one cause’ commitment, it will be blocked if it

cannot find the information specifying the cause of a disease. For this reason, the commitment specific to this application needs to be encoded as 'disease has zero or more cause'.

7.5.2 Formalize the Commitments

Ontology commitments need to be formalized to make them computer readable and understandable. The way ontology commitments will be formalized depends on the ontology tool used.

7.5.3 Identify Reusable Knowledge Components

An ontology base as a set of related concepts together with ontological commitments that logically connect, instantiate and constrain those relationships may be seen as a set of reusable knowledge components. Usually, similar or related applications share or reuse the same ontology components.

For example, agents designed to cooperatively work together share a common domain ontology but those agents show minor differences in their individual ontologies as they have different functions within the multi-agent system. We can use the same ontology base (or part of a common ontology base) together with the different combinations of ontology commitments to design different ontologies needed for different agents of a multi-agent system. This principle is shown in Figure 7.9, adopted from (Jarrar and Meersman 2002).

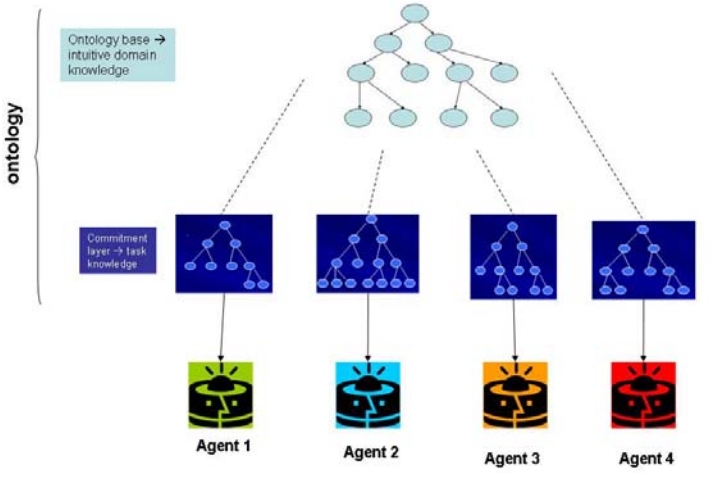


Fig. 7.9 Different agents of a multi-agent system share the same ontology base

When designing an ontology for two different agents of a multi-agent system, the following concepts can be identified as reusable knowledge components: ‘ disease’ and ‘cause’ with relationship ‘has/is-of’. One agent may be a specialist in general knowledge regarding human diseases and the ontology commitment for this agent needs to be ‘each disease has at least one cause’. Another agent may be assigned the task of information retrieval. It may be possible for this agent that in the body of the available information no information can be found that will describe causes of a specific disease. For this reason, the ontology commitment for this agent would be ‘each disease has zero or more causes’.

7.6 Ontology Evaluation

We may consider the ontology evaluation process from different viewpoints:

1) technical point of view (quality of the designed ontology)
2) practical view (usability of the designed ontology)
3) mathematical point of view (evaluation of the ontology design through use of mathematical models)

For the purpose of evaluation of quality of the designed ontology, we adopted the five criteria suggested by Gruber (Gruber 1995) against which the ontology needs to be evaluated:

1) clarity
2) coherence
3) extendibility
4) minimal encoding bias
5) minimal ontological commitment

Secondly, we focus on the evaluation of the usability of the designed ontology. For this purpose, we focus on the following:

1) matching of originally stated ontology purpose and scope against the designed ontology
2) evaluation of conceptual coverage
3) evaluation of practical usefulness

Ontology models can be evaluated using mathematical theories. Ontologies can be viewed as sets of related concepts. In most cases, Set Theory is used for constructing mathematical models to define and reason about ontological models (Wouters et al. 2004).

7.6.1 Ontology Design Quality Evaluation

A set of criteria has been proven to be useful as a guiding tool during the ontology design process. Those principles have also been used to evaluate ontology design. Gruber suggests five design criteria for ontologies (Gruber 1995):

1) Clarity
2) Coherence
3) Extendibility
4) Minimal Encoding Bias
5) Minimal Ontological Commitment

See Section 5.2 for More detail.

7.6.2 Ontology Usability Evaluation

In this section, we discuss the evaluation of the ontology's usability by taking into account the following points:

Matching the originally stated ontology purpose and scope against the designed ontology. By matching the originally defined purpose and scope of the ontology against the resulting ontology, it is possible to assess how well the originally defined expectations have been met. Some gaps may be found during this process. In this situation, the ontology design processes need to be continued to meet the originally defined expectations where possible. In other situations, the mismatches may be the results of the originally inaccurately defined ontology purpose and scope. This is possible as it is difficult to have a clear overview of the ontology design process at the very beginning of the ontology design process. In this case, the mismatches found need to be clearly explained and well documented.

Evaluation of conceptual coverage. A test set is used to evaluate conceptual coverage of the designed ontology. For example, this test set may consist of 20 abstracts of published papers randomly chosen from publication databases. The developed ontology is used to encode the knowledge from this test set. The evaluation of conceptual coverage includes calculation of the percentage of sentences that can be fully represented by the designed ontology.

Evaluation of practical usefulness. In order to evaluate the practical usefulness of the designed ontology, an application prototype based on the designed ontology needs to be developed and evaluated by the users of the ontology.

7.7 Overview of the Ontology Development

The Onto Methodology comprises the following five stages:

1) Generalization and conceptualization of the domain
2) Aligning and merging ontologies
3) Formal specification of conceptualization
4) Specifying of ontology commitments
5) Ontology evaluation

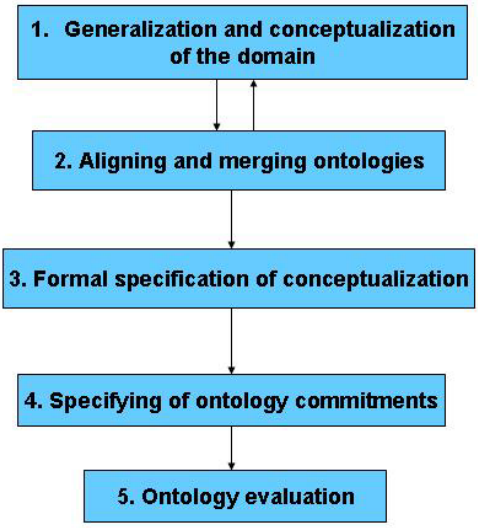


Fig. 7.10 Ontology design methodology

This new ontology design methodology combines the advantages of the existing ontology design methodologies, and introduces some new features into the design process. The first and the second stage of KE methodology are included and extended in the first stage of the Onto Methodology. We believe that two additional aspects need to be taken into account: (1) the community for which the ontology is being designed; and (2) the application(s) which will operate on the basis of the designed ontology. This is the first step of the TOVE methodology. The third stage of the KE methodology is the second stage of the Onto Methodology. We strongly agree that existing ontologies should be reused where possible. The fourth stage of the KE methodology corresponds to the third and fourth stages of Onto Methodology as we adopted the DOGMA principle of separation of the ontology base from the ontology commitments. In the TOVE methodology, the step of terminology specification is also separated from the step of specification of axioms and definitions (ontology commitments). The population of the ontology with individual instances needs to be carefully considered. In some cases, an excessive number of ontology instances may affect ontology functionality, flexibility and reusability. Highly specialized ontologies with a large number of instances have less chance of being used by a large number of users. In these or similar situations, access of the instances from external information resources (e.g. TAMBIS ontology (Stevens et al. 2002) through the use of agents may be more appropriate. This gives the ontology a dynamic nature and enables it to be used by a larger range of users. All the concepts needed for the design of a new ontology are chosen in the second stage of Onto Methodology, and are formally specified in the third and the fourth stages. We agree with the KE methodology that the designed ontology should be evaluated. This is the fifth and the last phase of the Onto Methodology. One of the ways to evaluate the ontology is through the use of evolving prototypes as mentioned by METHONTOLOGY.

The five stages of the ontology design methodology are shown in Figure 7.10. The first and second stages may need to be alternated until the ontology borders and conceptual coverage of the domain have been clearly established and agreed upon.

We recommend a combination of top-down and bottom-up approaches during the ontology design process. In the first stages of the ontology design, a top-down ontology design approach may be preferred as it gives a better overview and control over the ontology design process. The amount of detail to be provided by the ontology will influence the starting point of the ontology design using the bottom-up approach. We believe that a combination of both approaches is the best option. Eventually, the ontology concepts incorporated during the bottom-up approach will meet those incorporated during the top-down approach.

7.8 Conclusion

In this chapter, we have discussed the first part of the design methodology for the ontology-based, multi-agent systems. This first part consists of five steps and addresses the ontology design.

In the first step, the domain of interest is generalized and conceptualized. The goal of this step is to clearly identify the knowledge domain, community associated with the ontology design, purpose of the ontology and application(s) based on the designed ontology.

In the second step, existing ontologies from the same domain are aligned and merged and the existing knowledge reused where possible. Here, it is necessary to identify available ontologies suitable to be reused; choose a suitable ontology merging and alignment tool; import the source ontologies; identify correspondence among these ontologies; and align and/or merge them.

It may be necessary to alternate the first two steps of the design methodology until ontology boundaries have been established which correspond with the ontology design objectives.

In the third step, the resulting conceptualization of the domain is formally specified in the ontology base. It is necessary to define ontology concepts, relationships between these concepts, lexons, relationships between different lexons and form groups of related lexons.

The layer of ontology commitments operates on the ontology base and is defined in the fourth step of the design methodology. When specifying ontology commitments, it is necessary to identify intra-commitments; identify inter-commitments; formalize the commitments; and identify reusable knowledge components.

In the fifth step, the resulting ontology is evaluated from technical, practical, and mathematical perspectives. From the technical point of view, the ontology is evaluated for its clarity, coherence, extendibility, minimal encoding bias, and minimal ontological commitment. From the practical point of view, the originally stated ontology purpose and scope is matched against the designed ontology; the

conceptual coverage is evaluated; and its practical usefulness is evaluated. For the evaluation from the mathematical point of view, Set Theory is predominantly used.

In the following chapter, we will discuss the second part of the design methodology for the ontology-based, multi-agent systems. The second part also consists of five steps and addresses the design of the multi-agent system which is based on the previously designed ontology.

References

1. Bodenreider, O.: The Unified Medical language System (UMLS): Integrating Biomedical Terminology. Nucleic Acids Res. 32, 267–270 (2004)
2. Ceusters, W., Martens, P., Dhaen, C., Terzic, B.: LinkFactory: an Advanced Formal Ontology Management System. In: Proceedings of interactive tools for Knowledge Capture, KCAP 2001 (2001), http://www.isi.edu/~blythe/kcap-interaction/papers/LinKFactoryXWhiteXPaperXfinal.doc (retrieved: May 26, 2004)
3. Ceusters, W., Smith, B., Kumar, A., Dhaen, C.: Ontology-Based Error Detection in SNOMED-CT. In: MEDINFO 2004, pp. 482–486 (2004)
4. De Bo, J., Spyns, P., Meersman, R.: Creating a "DOGMAtic" multilingual ontology infrastructure to support a semantic portal. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 253–266. Springer, Heidelberg (2003)
5. Gruber, T.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Computer Studies 43, 907–928 (1995)
6. Jarrar, M., Meersman, R.: Formal Ontology Engineering in the DOGMA Approach. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 1238–1254. Springer, Heidelberg (2002)
7. McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: An Environment for Merging and Testing Large Ontologies. In: Proceedings of the seventh international conference on Principles of Knowledge Representation and Reasoning, KR 2000 (2000), http://www.ksl.stanford.edu/people/dlm/papers/kr2000-camera-ready-copy.doc (retrieved: April 28, 2004)
8. Montyne, F.: The importance of formal ontologies: a case study in occupational health. In: Proceedings of the international workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations, OES-SEO2001 (2001), http://cersi.luiss.it/oesseo2001/papers/28.pdf (retrieved: December 11, 2003)
9. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proceedings of the seventeenth national conference on Artificial Intelligence (AAAI 2000), pp. 450–455 (2000)
10. Stevens, R., Baker, P., Bechhofer, S.G., Ng, A.J., Paton, N.W., Goble, C.A., Brass, A.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. Bioinformatics 16, 184–186 (2002)
11. Stumme, G., Maedche, A.: Ontology Merging for Federated Ontologies on the Semantic Web. In: Proceedings of the IJCAI 2001 Workshop on Ontologies and Information Sharing, pp. 91–99 (2001)
12. Wouters, C., Dillon, T.S., Rahayu, J.W., Chang, E., Meersman, R.: Ontologies on the MOVE. In: 9th International Conference Database Systems for Advances Applications (DAS FAA 2004), pp. 812–823 (2004)

Chapter 8
Design Methodology for Integrated Systems - Part II (Multi-Agent System Design)

8.1 Introduction

A multi-agent system provides a distributed collaborative platform and as such determines the system dynamics. Ontologies represent the domain knowledge and can be used to support some important processes within a multi-agent system such as: problem decomposition and task sharing among different agents, result sharing and analysis, information retrieval, selection and integration etc.

There is not a single and consensual multi-agent system design methodology. We discussed some of these methodologies in Chapters 4. We introduce an Onto-Agent Methodology as the first methodology that unifies the different approaches of the existing ontology and multi-agent systems design methodologies. This methodology is composed of two interconnected processes. The five steps of the first part of the Onto-Agent Methodology (Onto Methodology) were presented in the previous chapter where we described how to design an ontology on which the multi-agent system will be based. The second part of the Onto-Agent Methodology (Agent Methodology) also consists of five steps and will be presented in this chapter. Using the Agent Methodology, we will define the multi-agent system which functions and operates using the ontology designed with the Onto Methodology as described in the previous chapter.

8.2 Overview of the Agent Methodology

The Agent Methodology consists of the following steps:

1) Classify agents according to their responsibilities
2) Identify the need for an ontology to support agents intelligence
3) Define agents' collaborations
4) Construct individual agents
5) Protect the system by implementing security requirements

The five steps of the Agent Methodology are shown in Figure 8.1. The approaches of the existing methodologies are unified, and some new characteristics are introduced into the multi-agents systems design process.

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 127–142.
springerlink.com

According to Cassiopeia, a multi-agent system should be designed in terms of agents provided with three levels of behaviour: elementary, relational and organizational. Based on the elementary behaviours, the designer defines the different roles of agents. We identify different roles of agents in the first stage of the Agent Methodology. In the second stage of the Agent Methodology, we focus on making the agents intelligent through the use of ontologies. The third phase of the Agent Methodology addresses the dynamics of the organization where relational and organizational behaviours (as defined by the Cassiopeia method) are specified. The importance of the organization of agents within a multi-agent system is also emphasized in the Gaia methodology. Here, we define agents' collaborations using an approach similar to the BDI methodology. We firstly define the organization of agents within the multi-agent system and then we specify communication routes between different agents. We also emphasize the importance of the internal structure of an agent and this is the fourth stage of the Agent Methodology. The fifth stage of the Agent Methodology focuses on the implementation of the security requirements within the system.

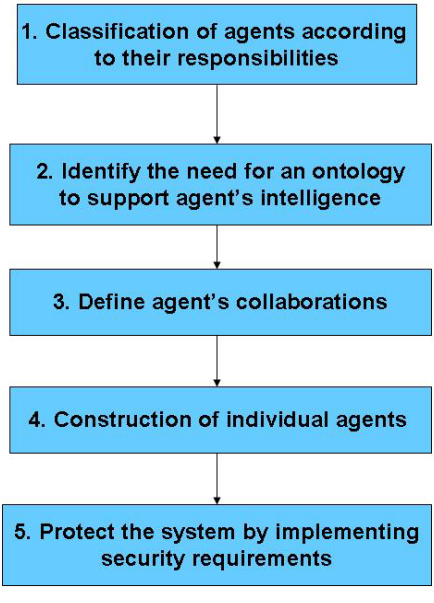


Fig. 8.1 Multi-agent system design methodology

8.3 Classification of Agents According to Their Responsibilities

A multi-agent system is a community of agents with different capabilities that complement each other and are cooperatively working towards the same goal. The different agents need to share the overall task, coordinate their actions and integrate their results. When classifying agents according to their responsibilities, it is important to:

- establish intuitive flow of problem solving, task and result sharing
- identify different agent functions needed to establish this kind of flow
- identify different agent roles according to these different functions

8.3.1 Establish Intuitive Flow of Problem Solving, Task and Result Sharing

As a multi-agent system is composed of cooperatively working agents, an intuitive flow of problem solving, task and result sharing has to be established. In most cases, the overall problem can be decomposed into smaller subproblems so that the tasks can be shared among different agents. The whole problem solving process needs to be systematically followed in our minds from the beginning to the end, and different aspects of solutions to the problems progressively identified. Information agents have access and need to prove their identity to the outside agents of the information resources (authentication). After their identification validation, they are able to access the information resource or a part of it (availability). The integrity property needs to be guaranteed by the agents of the information resources to the Information agents. Namely, a part of the information retrieved from an information resource should not lose its meaning once found outside the context of the information contained within the information resource. Only agents that can prove their authority over the information have access to the information (confidentiality). This is true for inside as well as for outside the system. For example, a Smart agent needs to prove its identity to the Information agents in order to access the information retrieved by the Information agents. Furthermore, Information agents need to prove to the Smart agents that the information they provide is from the correct source (non-repudiation).

For example, we may want to design a multi-agent system for the purpose of information retrieval regarding knowledge about Huntington's Disease. Intuitive flow of problem solving, task and result sharing could be established through the following sequence:

query formulation → task sharing → information retrieval → result sharing → result analysis → result assembly and result presentation

8.3.2 Identify Corresponding Agent Functions

The corresponding functions required to solve the identified problems have to be clearly defined. It may be necessary to assign exclusive tasks, responsibilities and functions to different agents. All agents associated with these functions need to be clearly identified.

In the above example of information retrieval system for the knowledge about Huntington's Disease, we may need an agent to help query formulation, another agent to decompose the overall problem and assign task to various agents that need to retrieve the needed information. We will also need agents to analyze the retrieved information and assemble the selected information to be presented to the user.

8.3.3 Identify Corresponding Agent Types

It is common for agents within a multi-agent system to have diverse functions. Different agent types are identified according to their functions within the system. In Chapter 2, Section 10, we mentioned several different types of agents. Additional agent types and subtypes can be further identified.

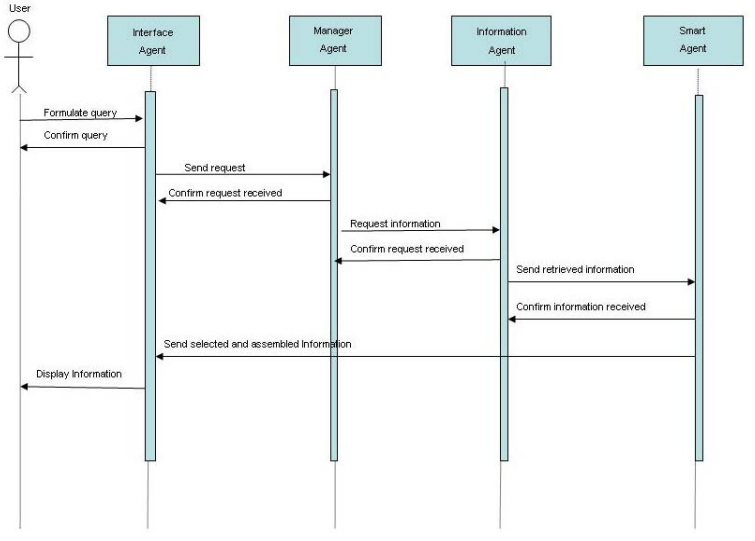


Fig. 8.2 Interface, Manager, Information and Smart agent

As shown in Figure 8.2, the circle may start with the Interface agent that is helping a user in the querying process. Once a query has been formulated, the Interface agent sends the request to a Manager agent. According to this query, the Manager agent requests information from different Information agents. The Information agents search for, retrieve and send the requested information to the Smart agent. The Smart agent selects relevant information and assembles this information. The resulting assembled information is returned to the user via the Interface agent. In this case, we have four different roles of agents: Interface, Manager, Information and Smart agents.

In complex, multi-agent systems it may be necessary to design various Smart agents so that the results are assembled at the different levels of complexity. Other complex multi-agent systems may require different Manager agents that have the ability to decompose the problems at the different levels of complexity. One group of agents may be designed for decomposing the problems (and/or assembling the results) on the higher and more generic knowledge level, while the other group of agents may be needed to decompose the problems (and/or assemble the results) on the lower and more detailed knowledge level.

8.4 Identify the Need for an Ontology to Support Agent's Intelligence

The ontology is used to represent the knowledge domain. Use of the ontology enables the multi-agent system to retrieve the information intelligently. The ontology may be used at different stages within a multi-agent system, so as to:

- enable decomposition of the overall problem
- support the process of information retrieval
- enable communication between the cooperatively working agents
- support the process of analyzing and manipulating information
- enable presentation of the information in a meaningful way

8.4.1 Problem Decomposition

In most cases, the overall problem needs to be decomposed into smaller subproblems. The ontology provides the agents with knowledge of the task structure and can be used during the process of problem decomposition. In this case, the subproblems of the problem correspond to and are represented by subontology of the domain ontology. The subproblems may further be decomposed into smaller sub-subproblems and so on. The goal of the problem decomposition is to reach a stage where the subproblems can be solved by individual Information agents.

This process of problem decomposition and task assignment to different agents assumes that the agents must have knowledge of the task structure and must know which agents are suitable for solving specific subproblems. The ontology is used to provide the agents with this knowledge.

If the multi-agent system is designed for the information retrieval regarding knowledge about Huntington's Disease, the query may needed to be decomposed according to the three different subontologies (disease symptoms, causes and treatments) as shown in Figure 8.3. The first group of Information agents will need to retrieve information regarding disease symptoms, the second group regarding disease causes and the third group of Information agents will need to retrieve information regarding disease treatments.

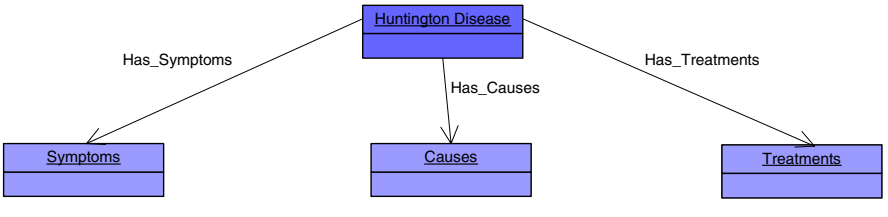


Fig. 8.3 Huntington's Disease Ontology

8.4.2 Information Retrieval

The ontology can be used to locate and retrieve requested information. Machine-readable information content within an information resource can be described using an ontology. This provides agents with a more controlled and systematic way to look for the requested information. Use of the ontology enables the agent (that commits to this ontology) to 'understand' the meaning of the information contained within these resources and to exactly locate and retrieve the requested information.

In our example, different information resources are assigned to different Information agents. One group of agents retrieves information regarding disease symptoms, a second group of agents retrieves information regarding disease causes and a third group of agents retrieves the information regarding disease treatments. The content of the information resources needs to be annotated for those agents to enable them to recognize and retrieve the specific information from the pool of available information.

8.4.3 Agent Communication

The ontology is used to enable cooperatively working agents to communicate with each other during the process of information retrieval. The ontology provides the vocabulary needed for communication. The different agents of a system can reach a shared understanding by committing to the same ontology. Agents are then able to communicate knowledge, make statements, and ask questions about a subject domain. Use of the ontology permits agents to communicate with each other and share information, enabling the agents to cooperate and coordinate their actions.

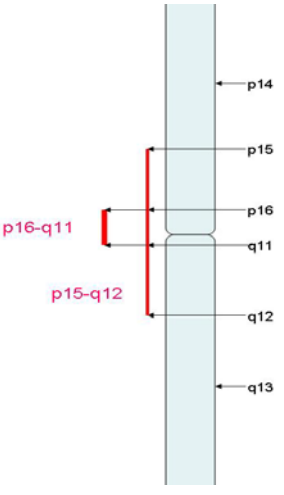
In our example shown in Figure 8.2, the Manager agent needs to communicate with and assign various tasks to different Information agents. After information retrieval, various Information agents need to communicate with and hand over the results to the Smart agent. These kinds of communications are very important if the agents are designed to work together and cooperate with each other.

8.4.4 Information Analysis and Manipulation

The ontology can be used to analyze and manipulate retrieved information. As the ontology represents the domain knowledge, it allows agents to reason about the retrieved information. Any redundant and/or inconsistent information can be easily recognized and removed. Only relevant information is selected and integrated to be presented to the user.

According to the definition, ontology represents domain knowledge and through this enables reasoning about this knowledge. For example, in genetics "DNA region of interest" represents a part of human DNA that contains a gene that, when mutated, causes a specific disease. Because ontology represents the domain knowledge, it is coded by the ontology that smaller DNA region of interest means being closer to the gene of interest. In Figure 8.4, p16-q11 is a

Fig. 8.4 DNA region of interest



smaller DNA region than p15-q12 and it is easier to find the gene of interest on this smaller DNA region. So, when two different Information agents provide information regarding overlapping DNA regions of interest, the Smart agent intelligently chooses the most helpful answer.

8.4.5 Meaningful Information Presentation

The ontology can be used to present the retrieved information to the user in a meaningful way. Structured and organized presentation of the requested information becomes possible through the use of ontologies. Moreover, the inherited organization of ontologies adds a taxonomical context to search results, making it easier for the researcher to spot conceptual relationships in data.

In the example of Hungtington's Disease Ontology shown in Figure 8.3, the overall problem to be solved can be decomposed according to the three subontologies (Symptoms, Causes and Treatments subontologies), and further decomposed according to the sub-subontologies. The same hierarchical ontology structure can also be used to present the results in a meaningful way.

8.5 Define Agent's Collaborations

In the first stage of the Agent Methodology, we described how to identify different roles of agents according to their different functions within the multi-agent system. In this stage, we emphasize the importance of structural organization of the agents within a system. These two stages may sound similar, but the difference is that in the first stage we defined activities of agents within a multi-agent system, while in this stage we define structural organization of agents within the multi-agent system. The aim of this step is to:

- organize agents within the multi-agent system so that the tasks are performed in the most efficient way
- establish correspondence between different agent roles and positions of these agents within the multi-agent system

8.5.1 Establish Efficient Organization of Agents

The agents need to be organized in such a way that the problem solving process can easily flow into its completion and that the communication between different agents of the system can be established at any point. The overall task needs to be efficiently and effectively performed. Each agent needs to be able to perceive and evaluate changes that occur in the environment, interact with other agents of the system, and reach an optimal decision. Agents need to be able to respond timely and efficiently to unexpected changes in their environment and continuously coordinate their activities.

Sometimes, a system structured in a simple way functions the best. In other cases, a complex system may be more appropriate for the task at hand. As an example, we show triangle structure in Figure 8.5. In other cases, a complex system may be needed due to the complexity of the task. As a more complex example, we may have a holonic multi-agent system (Ulieru 2003; Unland 2003). Holonic multi-agent systems consist of autonomous agents with holonic properties that group together forming holons and holarchies. Such hierarchically structured, multi-agent architecture provides a framework where each agent represents a node in the hierarchic tree.

8.5.2 Establish Correspondence between Agent Types and Their Organization

It is necessary to establish correspondence between different agent roles (defined in the first stage) and the positions of these agents within the multi-agent system. We do not recommend overloading the agents with responsibilities. It is possible for one agent to correspond to a number of different agent roles. For example, an agent can be both a Manager and Smart agent as it may have the responsibility of problem decomposition in the first stages and the responsibility of result assembly in the last stages of the problem solving process. On the other hand, many different agents may correspond to a similar role. This can be the case with Information agents which are situated over different databases. They all have the one function of retrieving the information and correspond to the same agent role (Information agents) but are situated over different databases and thus differ in their position within the multi-agent system.

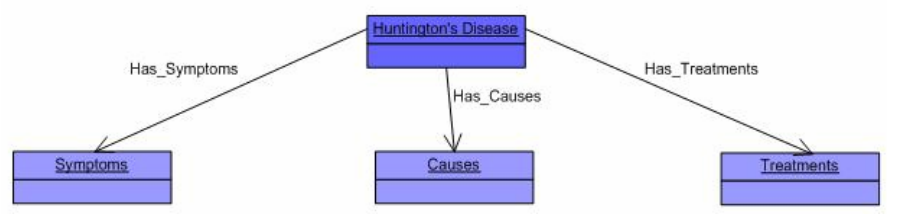


Fig. 8.5 An example of organization of different agents

We may have a simple structure such as that shown in Figure 8.5. The Interface agent helps the user to formulate queries. Once the query has been formulated, the Interface agent requests the information from the Information agent. The Information agent searches, retrieves and sends the requested information to the Smart agent. The Smart agent selects the relevant information and assembles this information. The resulting assembled information is presented back to the user via the Interface agent.

Usually, a multi-agent system is more complex than the one shown in Figure 8.5. There may be more than one Information agent situated over different databases. The different Information agents may form a hierarchy of agents. There may be more than one Smart agent so that the information can be assembled at different levels of complexity.

8.6 Construction of Individual Agents

Agents need to be carefully constructed so that they meet the requirements specified by their function and role. In this stage of the Agent Methodology, the aim is to:

- identify required agent components
- assemble the components for the purpose of constructing various agents

8.6.1 Identify Required Agent Components

Different agent components may include the Human interface, Agent interface, Communication component, Cooperative, Procedural, Task, Domain and Environment knowledge, History files, and so on.

The Human interface is needed in agents that interact with users e.g. in Interface agents.

The Agent interface enables agents of the system to interact and communicate with each other.

The Communication component processes the messages which can be written in an agent communication language and ontology. All agents within the system need to agree and commit to this common language and ontology.

The Cooperative knowledge is required by an agent in order to negotiate with other agents of the system, and cooperate and coordinate its actions with their actions.

The Procedural knowledge contains information about the procedures involved in the problem solving and goal prioritization method.

The Task knowledge is the knowledge regarding tasks assigned to that particular agent. It enables agents to decompose the task into smaller tasks where needed, and perform atomic actions.

The Domain knowledge describes the domain of interest of that particular multi-agent system.

The Environment knowledge contains information regarding the environment in which the multi-agent system is situated. This component may, for example, contain the information regarding content and structure of each of the different information resources assigned to different Information agents.

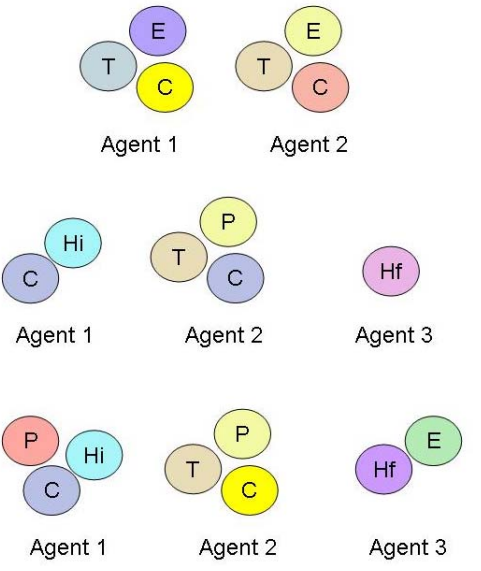
The History files contain information about the past experiences of the agent.

8.6.2 Construct Various Agents

The variety of agents within a multi-agent system can be achieved in three different ways:

- Different components that are used to construct different agents are the same, but the content of these components is different for different agents. For example, all the agents within a system may have Task knowledge, Environmental knowledge and Cooperative knowledge components. These components are represented by letters T, E and C in Figure 8.6. The component describing task(s) of an agent will differ for different agents as they have different functions within the multi-agent system. In Figure 8.6, the different contents of the same components are represented by different colours. As all the agents of the system are not placed into the same environment, the Environmental knowledge components will be different for different agents. Not all agents will cooperate in the same way because they have different functions and are situated in different environments. The Cooperative knowledge module will also be different for different agents.
- The content of the components used to construct different agents are the same, but different agents are constructed by a different combination of used components. For example, the first agent may have only a Human interface and Cooperative knowledge component (represented by Hi and C in Figure 8.6), while the second agent may contain Procedural knowledge, Task knowledge and Cooperative knowledge component (represented by P, T and C in Figure 8.6). The third agent may have only a History files module (represented by Hf in Figure 8.6), and so on.
- The third and most common option is that different agents differ in the combination of the components used to construct them, and in the content of these components, as illustrated in the third row of agents on Figure 8.6.

Fig. 8.6 Design of various agents



8.7 Protect the System by Implementing Security Requirements

Security plays an important role in the development of multi-agent systems. The risks of jeopardizing the system security must be minimized by providing as much security as possible. The aim of this Agent Methodology stage is to:

- identify critical security requirements within the multi-agent system
- implement the requirements within the system

8.7.1 Identify Security Requirements

Security is usually defined in terms of the following five properties (Mouratidis et al. 2003):

1) Authentication: proving the identity of an agent.
2) Availability: guaranteeing the accessibility and usability of information and resources to authorized agents.
3) Confidentiality: information is accessible only to authorized agents and inaccessible to others.
4) Non repudiation: confirming the involvement of an agent in a certain communication.
5) Integrity: assuring that the information remains unmodified from source to destination.

The abovementioned properties are critical inside the multi-agent system as well as outside the multi-agent system, such as during the agent interaction with the environment. We highlight some additional security requirements:

6) Compliance: acting in accordance with the given set of regulations and standards. The designer must give the agents clear instructions regarding their roles and tasks within the systems that will enable them to be successful in performing their actions for their own benefit and for the benefit of the whole multi-agent system.
7) Service: agents need to be designed to serve one another for mutually beneficial purposes. Agent needs to fulfil its role within the multi-agent system and perform tasks assigned to it, but it also needs to serve other agents when needed. If some agents are under malicious attack, other members of the multi-agent system need to make their best efforts to protect those agents and the whole multi-agent system.
8) Dedication: complete commitment of the agents to the multi-agent system goal and purpose. Every agent has a unique role within multi-agent system. Some agent roles are more important than others but, generally speaking, all roles are important in jointly achieving the overall goal of this multi-agent system. Different agents are working on different aspects of the overall goal and they act like different organs in a body. Heart, lungs, liver, stomach, eyes etc., all have unique functions and, in unity, enable the body to live and function optimally. Analogously, all agents have unique functions within the multi-agent system and, in unity, enable the multi-agent system to live and function optimally. All agents must commit to the overall goal and purpose of the multi-agent system and function in unity and as one in order to jointly achieve this goal.

8.7.2 Implement the Requirements

After the identification of required security properties, it is necessary to implement them within the multi-agent system. As different agents have different functions within the system, some agents will be more critical than others in regard to the security of the system. As a consequence, the critical agents will be assigned more security requirements than the others.

When considering security of a multi-agent system according to the abovementioned requirements, it is necessary to identify the:

1) role of each individual agent in the security of the system
2) most critical agents with respect to security
3) role of each environmental factor in the security of the system
4) most critical environmental factor with respect to security
5) most critical actions that operate within the system with respect to security
6) most critical actions that operate during the interaction of the system with the environment with respect to security
7) part of the system that is most susceptible to attack from the outside

A designer needs to create a security network from two perspectives:

1) the security of the individual agent within the multi-agent system; and
2) the security of the whole multi-agent system

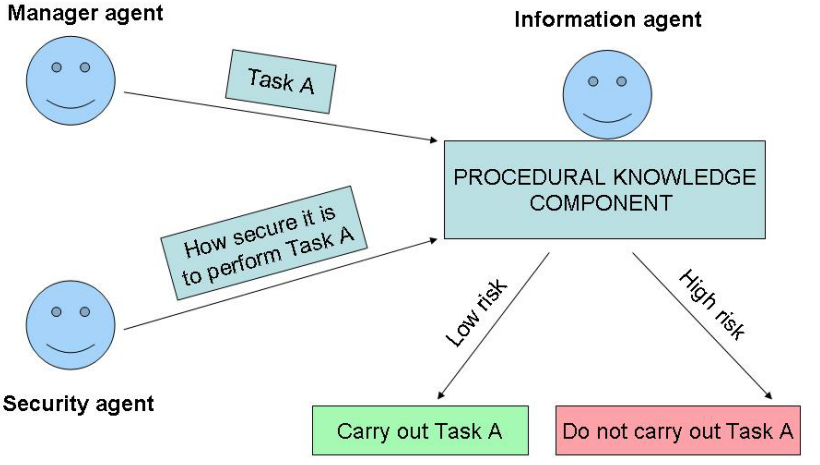


Fig. 8.7 Information agent decides whether to perform an action using inputs provided by Manager and Security agents

Designers should avoid overloading the agents with security requirements. Sometimes, it is necessary to incorporate new agents into the system that function as Security agents. In other cases, it is enough to modify and adapt existing agents' components or to build additional ones.

For example, in Section 8.6.1 we mentioned that the Procedural knowledge of an agent contains information regarding the procedures involved in the problem solving and goal prioritization method. This component may be designed to take two inputs. This is illustrated in Figure 8.7. The first input will contain information about the action this agent needs to perform, while the second input will describe how secure it is to performing this action. The first input is provided by the Manager agent, while the second input may be provided by a Security agent or it may have a constant value and be contained in the internal structure of this agent. On the basis of the two inputs, the agent decides whether or not to perform the action.

In the example shown in Figure 8.5, the whole system operates as a sequence of actions of different agents: Interface agent → Information agent → Smart agent → Interface agent. Interface and Information agents operate inside as well as outside the system and are more critical with respect to security than Smart agents. The Smart agent operates only with the agents within the multi-agent system.

Information agents have access and need to prove their identity to the outside agents of the information resources (authentication). After their identification validation, they are able to access the information resource or a part of it (availability). The integrity property needs to be guaranteed by the agents of the information resources to the Information agents. Namely, a part of the information retrieved from an information resource should not lose its meaning once found outside the context of the information contained within the information resource. Only agents that can prove their authority over the information have access to the

information (confidentiality). This is true for inside as well as for outside the system. For example, a Smart agent needs to prove its identity to the Information agents in order to access the information retrieved by the Information agents. Furthermore, Information agents need to prove to the Smart agents that the information they provide is from the correct source (non-repudiation).

8.8 Conclusion

In this chapter, we have discussed the second part of the design methodology for the ontology-based multi-agent systems (Agent Methodology). This second part consists of five steps and addresses the design of a multi-agent system.

In the first stage of the Agent Methodology, different agents are classified according to their responsibilities. Here, it is important to: define the intuitive flow of problem solving, and task and result sharing; identify different agent functions needed to establish this kind of flow; and, identify different agent roles according to these different functions.

In the second stage of the Agent Methodology, we identify the need for ontology to support the agents' intelligence. Ontology may be used for different purposes within a multi-agent system such as during the process of problem decomposition; information retrieval; analyzing and manipulating information; enabling communication between cooperatively working agents; and presenting of information in a meaningful way.

In the third stage of the Agent Methodology, we define agents' collaborations. The way that the different agents are organized within the system needs to be defined in such a way that the system operates at optimum efficiency. Here, it is also important to establish correspondence between different agent roles defined in the first stage and the positions of agents within the multi-agent system.

In the fourth stage of the Agent Methodology, we construct individual agents. Each agent and its components need to be carefully constructed to satisfy the defined objectives. Agent's components may include Human interface, Agent interface, Communication component, Cooperative knowledge, Procedural knowledge, Task knowledge, History files etc.

In the fifth phase of the Agent Methodology, we protect the system by implementing security requirements of authentication; availability; confidentiality; non repudiation; and integrity (Mouratidis et al. 2003) as well as the requirements of compliance, service and dedication. The security of individual agents, as well as that of the complete multi-agent system, needs to be taken into account when designing multi-agent systems.

8.9 Advantages of the Onto-Agents Methodology

The principle of triangulation is often mentioned in the context of research methods (Neuman 2000). In the study of the same phenomenon or construct, the triangulation method uses a combination of different methodologies. Accuracy of information is established by comparing three or more types of independent points

of view on data sources. A problem is studied using several different methods for the purpose of seeing it from different angles, thereby enabling the researcher to find the "real truth".

Analogously is a multi-agent system best defined when observed and analyzed from different perspectives: intelligence, functionality and security. We use ontologies to support agents' intelligence, we define the responsibilities of different agents within the system and organize agents so that their actions are performed in the most efficient way, and we focus on the security requirements of authentication, availability, confidentiality, non repudiation, integrity, compliance, service and dedication.

The use of ontologies to make agent systems intelligent is the most important reason for importing ontologies into the computer and information society. Various ontologies describing different knowledge domains are yet to be used by some applications. We believe that our application-driven approach to the ontology and agent design methodology will increase the usage of existing ontologies, and effectiveness and efficiency of the designed multi-agent systems.

The five-step approach in the development of multi-agent systems requires undertaking the system design process as if one were designing a single, large functioning component constructed from individually functioning parts. We can compare a multi-agent system with a human body:

1) In the first step, we identify different groups of agents according to their functions and responsibilities within the system. This step is analogous to identification of different human organs within a human body according to functions they have within the body.
2) In the second step, we develop ontologies to make agents intelligent. This step is analogous to the development of the brain of a human body, connecting it to other organs of the body, enabling them to function properly.
3) In the third step, we determine the structural organization and networks of agents within the multi-agent system. This step is analogous to finding out a way to organize and put together different human organs in a human body so that each of them functions optimally and to its full potential.
4) In the fourth step, we carefully construct each individual agent and its individual components. This step is analogous to examining in detail each individual human organ, identifying and constructing its individual parts so that this organ can function properly.
5) The fifth section, where we establish a secure multi-agent system, is analogous to protecting the human body from unwanted threats from both inside and outside the body.

We believe that the process of building multi-agent systems needs to be done step by step and at each step we should be aware of the overall system objectives. Providing structure to the design methodology gives the designer more control and a better overview of the design process.

The Onto-Agents Methodology provides a more concise understanding of ontology-based multi-agent systems and their design, and increase the control over the design process. This methodology will no doubt be improved upon and refined as more experience is gained with the design of such systems.

References

1. Mouratidis, H., Giorgini, P., Manson, G.A.: Modelling secure multiagent systems. In: The second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), pp. 859–866 (2003)
2. Neuman, W.L.: Social Research Methods: Qualitative and Quantitative Approaches, 4th edn. Allyn and Bacon,
3. Ulieru, M.: Internet-Enabled Soft Computing Holarchies for e-Health Applications. In: New Directions in Enhancing the Power of the Internet, pp. 131–166 (2003)
4. Unland, R.: A Holonic Multi-agent System for Robust, Flexible, and Reliable Medical Diagnosis. In: On The Move to Meaningful Internet Systems 2003 Workshops, pp. 1017–1030 (2003)

Chapter 9
Notations for the Integrated Ontology and Multi-Agent System Design

9.1 Introduction

In this chapter, we present the modelling of the integrated ontology and multi-agent system using well designed notations. Modelling is an important aspect of all methodologies. The modelling process analyses the range of problems that describe various aspects of the system being considered. With a precise definition of modelling, design can take place and the development process can then be transformed. The life cycle spans everything from modelling to evolution built up of possibly many autonomous, rational, proactive, and semantic entities. Particular modelling languages and notations have to be created in order to capture particular features of systems. It is insufficient and ineffective to use traditional software engineering modelling languages in the development of integrated ontology and multi-agent systems due to the specific characteristics of semantic and autonomous agents.

There are several agent-oriented modelling languages and notations such as Agent UML (AUML), Agent Modelling Language (AML), Multi-Agent System Modelling (MAS-ML), Agent-Object-Relationship (AOR). However, they have not yet been accepted for use in the wider community. The community is not yet aware of the standard modelling approach. AUML, as its name suggests, exploits an extension of UML providing specific modelling mechanisms over object-oriented modelling. In AUML, some researchers have been modelling Agent protocols (Odell et al. 2001) using a non-standard version of sequence diagrams where each rectangle represents an agent playing a different role. However, the rectangles at the head of lifelines are actually meant to represent classes, not agents. Having multiple rectangles each representing the same Agent is also non-standard, where each rectangle represents the Agent playing a particular role (da Silva and Nova 2005). AML is also based on the UML; however, it has been claimed that it augments with several modelling concepts appropriate to capture typical features of MAS.

On the ontology side, there are various formalisms for modelling ontologies notably, Knowledge Interchange Format (KIF) and knowledge representation

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 143–163.
springerlink.com

languages descended from KL-ONE. However, these representations have had little success outside Artificial Intelligent (AI) research laboratories (MacGregor 1991; Farquhar et al. 1996) and require a steep learning curve. KIF provides a Lisp-like syntax to express sentences of first order predicate logic and descendants of KL-ONE include description logics or terminological logics that provide a formal characterisation of the representation (Genesereth and Fikes 1992). Traditionally, AI knowledge representation has a linear syntax. Recent efforts, reflected in the literature (Kogut et al. 2002; Duric 2004; Evermann 2008), use UML for ontology modelling. In UML, ontology information is modelled in class diagrams and Object Constraint Language (OCL) constraint (Kogut et al. 2002). However, there are some debates stating that because ontology goes beyond the standard UML modelling, standard UML cannot express advanced ontology features such as constraints or restrictions (Gašević et al. 2006) cannot totally conclude whether the same property was attached to more than one class, and does not support the creation of a hierarchy of properties. Therefore, additional notations need to be defined in order to leverage expressiveness in the ontology. Note that the models underlying ontology should be distinguished from its use in software development to model the application domain model. This kind of agile modelling method for ontology design has some benefits in using the same paradigm for modelling ontology and knowledge.

In this chapter, we divide modelling structure into two parts: notations and metamodels / diagrams. We define the abstract syntax, semantics, and notations. The notations further structure into metamodels / diagrams which are logically structured. The integrated ontology and multi-agent system modelling is a semi-formal graphical representation for specifying and modelling systems in terms of concepts drawn from Ontology and MAS concepts and principles. We design the modelling to encompass a broad set of both ontology and multi-agent system theories and approaches inclusively.

9.2 Modelling Notations

Formation of the integrated ontology and multi-agent system modelling expressed by notations is given in this section.

9.2.1 Ontology Modelling Notations

In this section, graphical notations are presented to facilitate the ontology modelling process. Some concepts or terms represented by the notation have multiple presentations. Table 9.1 shows a list of notations and their potential use.

Table 9.1 A list of modelling notations

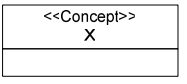
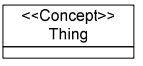
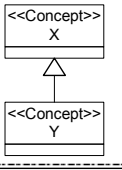
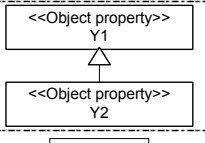
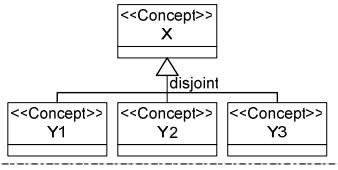
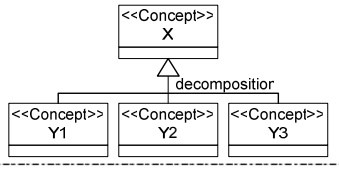
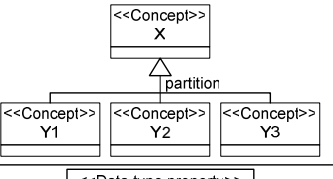
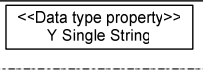
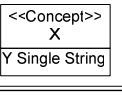
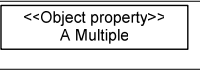
Concept/Term	Notation	Semantics
Class		Ontology class *X*.
Thing Class		All classes in ontology are subclasses of class *Thing*.
Generalisation		To express either class-subclass or property-sub property.
		Class *Y* is subclass of class *X*.
		Object property *Y2* is sub property of object property *Y1*.
		Disjoint classes *Y1*, *Y2* and *Y3*.
		Decomposed classes *Y1*, *Y2* and *Y3*.
		Partition classes *Y1*, *Y2* and *Y3*.
Data type property		Data type property *Y* which is functional and its type is string.
		Class *X* has data type property *Y* which is functional and its type is string.
Object property		Object property *A* which is non-functional.

Table 9.1 (*continued*)

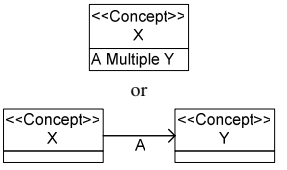
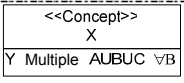
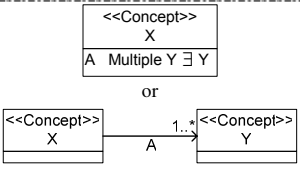
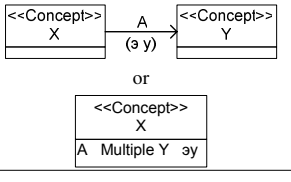
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property.
Property characteristic		
- Functional property		A functional property *X*.
- Non-functional property		A non-functional property *X*.
- Inverse functional property		An inverse functional property *X*.
- Symmetric property		A symmetric property *X*.
- Transitive property		A transitive property *X*.
Property restriction		
- allValueFrom	∀	All value from
		Class *X* has relations with the union of class *A*, *B* and *C*. The relation considers as an object property named *Y* which is non-functional property. The allValueFrom of property *Y* defines that those relations relate to instances from the class *B* only.
- someValueFrom	∃	Some value from
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. The someValueFrom of property *A* defines at least one relation related to an instance of class *Y*.
- hasValue	∍	Has value
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional

Table 9.1 (*continued*)

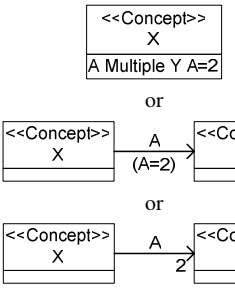
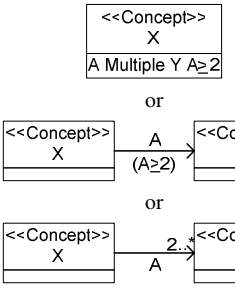
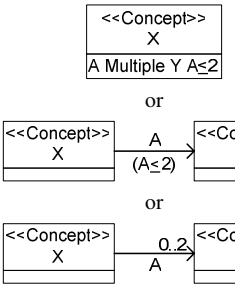
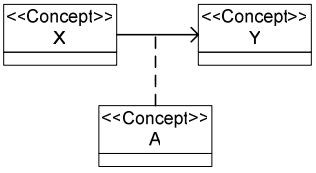
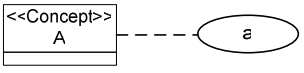
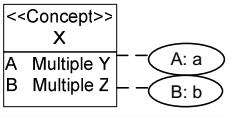
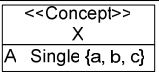
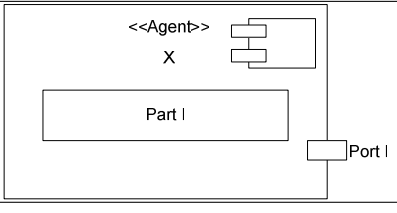
		property. hasValue restriction is used to the property *A* implied that instances of class *X* have at least one relation to specific instance y of class *Y*.
- Cardinality	=	To specify the exact number
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are exactly two relations from instances of class *X* related to instances of class *Y*.
Minimum cardinality	≥	To specify the minimum number
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are from at least two of relations from instances of class *X* related to instances of class *Y*.
- Maximum cardinality	≤	To specify the maximum number
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are from at most two of relations from instances of class *X* related to instances of class *Y*.

Table 9.1 (*continued*)

Association class		Class *A* is association class of object property related class *X* to class *Y*.
Class instance		*a* is instance of class *A*.
Property instance		*a* is instance of property *A* and *b* is an instance of property *B*.
oneOf		Functional data type property *A* relates to a set of data value of 'a', 'b' and 'c'.
Agent		Agent X with part I and Port I

9.2.1.1 Class Notation

Ontologies consist of instances, properties, and classes. Ontology instances represent concrete information specified as an instance of one or several ontology classes. Links between ontology instances represent their interactions. Ontology properties represent relations held among ontology classes/ontology instances. Ontology classes represent the concepts defined as the specification of a group of instances that belong together because they share some properties. The notation of ontology class is represented as a rectangle with two compartments. The top compartment is for labelling the class and the second compartment is used for presenting properties related to the class or to an XML schema data type value as shown in Figure 9.1 (a). It is mandatory to specify the word '<<Concept>>' above the class label in the top compartment. Figure 9.1 (b) shows an example of ontology class X.

Ontology class Thing is the special class that represents the set containing all instances in the ontology. All classes in the ontology are subclasses of class Thing. Its notation description is the same as an ontology class notation, but the top compartment contains the word '<<Concept>>' and 'Thing' as its label and the second compartment is empty as shown in Figure 9.1 (c).

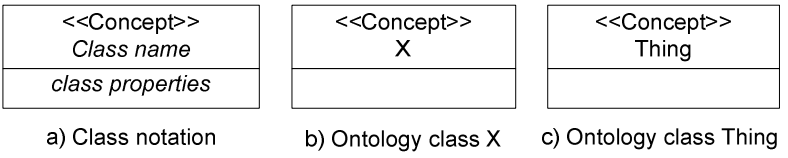


Fig. 9.1 Ontology class notation

9.2.1.2 Generalisation Notation

The ontology classes are organised in a hierarchy so that each main ontology class has its subclasses. The subclasses can either be disjoint or decomposition or partition. Disjoint ontology classes: their ontology instance cannot be an ontology instance of more than one of these disjoint ontology classes. This means that an ontology instance that has been asserted to be a member of one of the ontology classes in the group cannot be a member of any other ontology classes in that group. Decomposition ontology classes: the ontology classes are overlapped. An ontology instance can be a member of a combination of ontology classes in the group. Partition ontology classes: the ontology classes form the covering. An ontology instance of an ontology class must be either an ontology instance of any one of the ontology classes in the group.

Because the concept of generalisation in the ontology modelling is similar to the object-oriented model, we use the same generalisation symbol as in UML. The generalisation symbol appears as a line with one end empty and the other with a hollow triangle arrowhead as shown in Figure 9.2 (a). The empty end is always connected to the class being subsumed, whereas the hollow arrowhead connects to the class that subsumes. An example shown in Figure 9.2 (b) indicates that class Y is a subclass of class X. Properties of class Y are then properties inherited from its superclass plus its own properties. Subclasses properties contain properties inherited from superclass plus its own properties.

This symbol is also used to indicate generalisation of ontology properties. For example, as shown in Figure 9.3, object property Y2 is a sub-property of object property Y1.

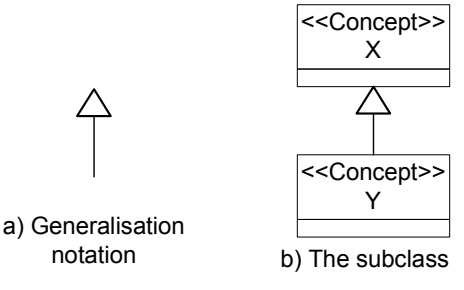
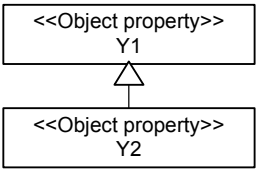


Fig. 9.2 Ontology generalisation notation

Fig. 9.3 An example of object property generalisation



Where subclasses of super class are disjoint or decomposed or partition, the words 'disjoint' or 'decompose' or 'partition' are attached to generalisation symbol respectively. Figure 9.4 (a), (b), and (c) show that all the subclasses Y1, Y2, and Y3 of class X are disjoint, decomposed, and partition respectively.

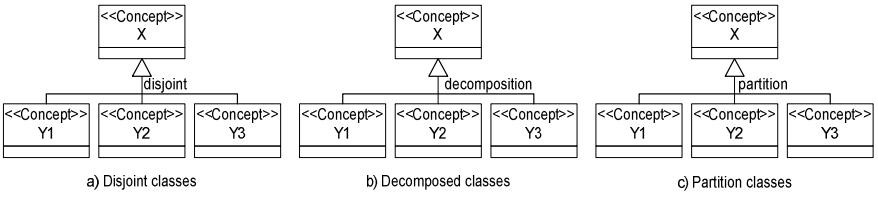


Fig. 9.4 Disjoint, decomposition, and partition ontology classes

9.2.1.3 Property Notation

Ontology properties represent binary relations between ontology classes. Note that ontology properties are not necessarily tied to ontology classes. There are two types of ontology properties which come from two different sources. Data type property associates ontology classes to an XML schema data type value or an RDF literal. Object property associates an ontology class to an ontology class.

The notation of ontology property can be represented as a rectangle. In the rectangle, it is mandatory to specify the word either '<<Object property>>' or '<<Data type property>>' to distinguish the object property or the data type property respectively above the property label as shown in Figure 9.5.

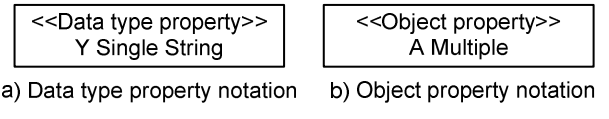


Fig. 9.5 Property notation

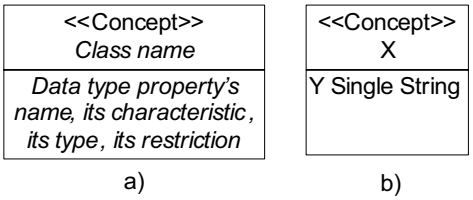


Fig. 9.6 Data type property notation

Just as class has subclasses, property can have sub-properties as well. Symbols of generalisation are the same for both class and property. Because they are applied to different entities, they will not create any confusion.

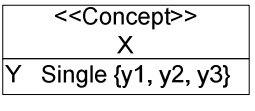
Data type property of the ontology class can be expressed in the bottle compartment of class notation as shown in Figure 9.6 (a). This is an alternative design for data type properties. That means the top compartment is called its domain. In the bottle compartment, notation formats as in the order of data type property name, its characteristic, its type (e.g. String, Integer) and its restriction. The type of data type property is considered as a range. The characteristics of data type property can be functional and non-functional. An example shown in Figure 9.6 (b) expresses that class X has data type property Y which is functional and its type is String.

A oneOf is used to fix defined data values of data range for a data type property. The set of data values of the data ranged can be specified. The oneOf is represented in curly brackets ({}) as shown in Figure 9.7. For example, Figure 9.7 defines functional data type property Y to relate a set of data value of 'y1', 'y2' and 'y3'.

The object property of the ontology class can be expressed in the bottle compartment of class notation like data type properties as shown in Figure 9.8 (a). This is an alternative design for object property. In this manner, the top compartment is still called its domain. Notation format in the bottle compartment is the same as the order of the object property's name, its characteristics, class name (its range), and its restriction. Class name expression as a range can assert the complex class description e.g. union, intersection. A union ontology class combines two or more ontology classes, while an intersection ontology class intersects two or more ontology classes.

In addition, an object property can be expressed as an arrow with an open arrowhead and with the object property name, its characteristics and its restriction as shown in Figure 9.8 (b). This is another alternative design for object properties.

Fig. 9.7 oneOf notation



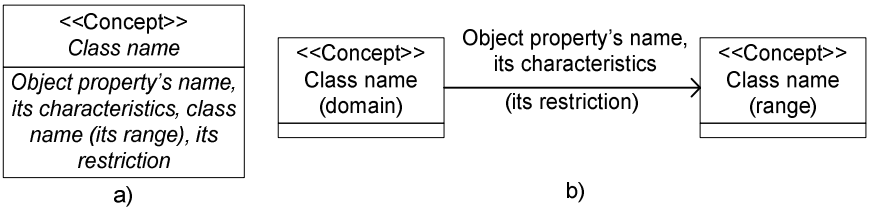
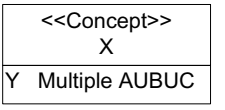


Fig. 9.8 Object property notation

Fig. 9.9 An example of object property Y



The arrow points from the domain of property to the range of property. Its restrictions can alternatively be expressed in the bracket. The characteristics of object property can be functional, inverse functional, non-functional, symmetric and transitive. The notation of the characteristics and the restrictions is discussed later in the section dealing with property characteristic notation and property restriction notation respectively.

An example, shown in Figure 9.9, shows that class X has relations with the union of classes A, B and C through the property Y. The domain of the property is then class X, whereas the union of classes A, B and C is the range of the property. The term 'Multiple' is the property's non-functional characteristic. In this example, there is no restriction on property.

Characteristics of the ontology property are classified into five categories which are functional, inverse functional, non-functional, symmetric and transitive properties. A functional property has a maximum cardinality of 1 in its range. A non-functional property has a minimum cardinality of 1 in its range. An inverse functional property has a maximum cardinality of 1 in its domain. Object properties can also be declared symmetric or transitive.

A functional property notation is represented as an open arrowhead with text label of property name and symbol number 1 at near the arrowhead as shown in Figure 9.10 (a). A non-functional property notation is represented simply as an open arrowhead with the text label of the property name which is similar to a functional property notation as shown in Figure 9.10 (b). The difference is that there is no symbol number 1 near the arrowhead. However, at the arrow head, the cardinality restriction can be presented near the arrowhead (details in the part of property restriction notation). An inverse functional property notation is represented as an open arrowhead with the arrowhead pointing in the opposite direction to its inverse property notation and a dot line placed in the middle links the inverse functional property symbol and its inverse as shown in Figure 9.10 (c). As usual, the text label represents the property name and there is symbol number 1 near the arrowhead. A symmetric property notation is represented simply as a solid line with a text label of the property name as shown in Figure 9.10 (d). Lastly, a transitive property notation is represented as a dotted arrow with an open arrowhead and with a text label of its property name as shown in Figure 9.10 (e).

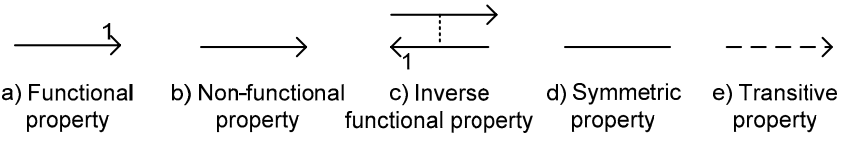


Fig. 9.10 Characteristics notation

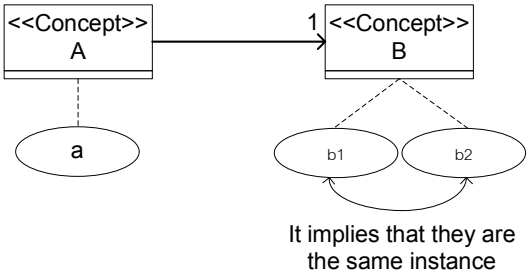


Fig. 9.11 An example of functional property

In the case of property being expressed in the bottle compartment of class notation, a functional property is represented by the word 'Single', whereas a non-functional property is represented by the word 'Multiple'. An inverse functional property and a symmetric property are represented by the words 'Inverse' and 'Symmetric' respectively.

For example, object property y shown in Figure 9.11 represents functional property related class A to class B. If instance 'a' of class A and instances 'b1' and 'b2' of class B, because property y is functional, then it can be inferred that instances 'b1' and 'b2' must be the same instance.

An example of a non-functional property is illustrated in Figure 9.12 representing object property y related class A to class B. The term 'Multiple' represents the non-functional property. Figure 9.12 (b) is an alternative representation way of Figure 9.12 (a).

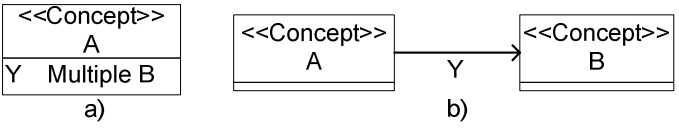


Fig. 9.12 Examples of non-functional property

Basically, in ontology modelling, the multiplicity of a relation has to be specified in pairs because it can be indicated at the end of the relation line, one indicator for each relation only. Hence, in order to specify multiplicity 'one to one' then one needs to specify a functional property and an inverse functional property shown in Figure 9.13 (a). Likewise, the multiplicity 'one to many' can specify a non-functional property and an inverse functional property as shown in Figure 9.13 (b).

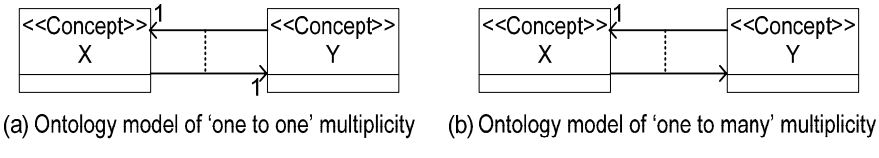


Fig. 9.13 Multiplicity presentation

9.2.1.4 Restriction Notation

Ontology property can have its range restricted when the property is applied to the domain class; either that the range is limited to a class only (allValueFrom) or that the range is one part of a class (someValueFrom). A hasValue restriction describes the set of instances that have at least one relation along a specified property to a specific instance. An ontology property can be constrained by cardinality restrictions on the domain, giving the minimum (minCardinality), maximum (maxCardinality), or exact (cardinality) specified number of instances which can participate in the relation.

An upside down A symbol ∀ represents restriction allValueFrom. A backwards facing E symbol ∃ represents restriction someValueFrom. A symbol denoted by ∋ represents restriction hasValue. For the cardinality restriction, symbols equal (=), greater than and equal (>) and less than and equal (<) represent respectively cardinality specifying the exact number, minimum cardinality specifying the minimum number and maximum cardinality specifying the maximum number. Cardinality constraints can be placed with the format of x..y where 'x' is the value of minimum cardinality and 'y' is the value of maximum cardinality. The asterisk (*) is used as part of the specification to indicate the unlimited upper bound. For example, specifying 'a > 2' is equivalent to '2..*', specifying 'a < 2' is equivalent to '0..2', specifying 'a = 2' is equivalent to '2..2' or just '2'.

Table 9.2 summarises the potential indicators matching with ontology property characteristics or restriction.

For example object properties, shown in Figure 9.14 (a), xy1 and xy2 restrict those instances of class X relating only to instances of class Y. An example of restriction someValueFrom is shown in Figure 9.14 (b) indicating that along the object property xy at least one relation relates instance of class X to instance of class Y. The hasValue restriction is, for example, applied to property xy depicted in Figure 9.14 (c) relating class X to a particular instance y that is instance of class Y.

Table 9.2 A list of indicators

Indicator	Property Characteristic/ Restriction	Meaning
1	Functional property	One only
0..*	Non-functional property	Zero or more
1..*	someValueFrom restriction	At least one
n	Cardinality restriction	Only n where n > 1
0..n	Maximum cardinality restriction	At most n where n > 1
n..*	Minimum cardinality restriction	At least n where n > 1

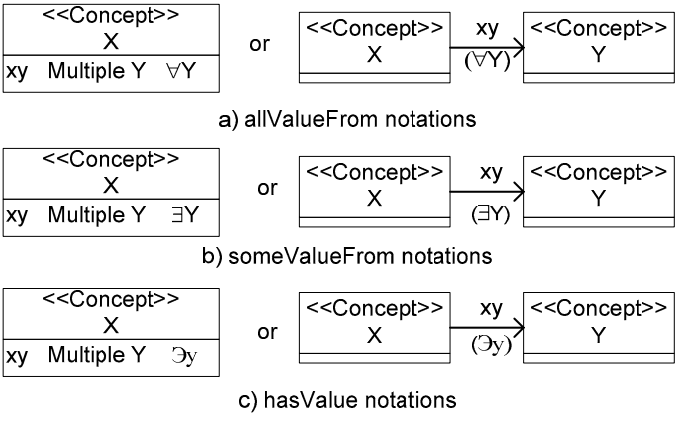


Fig. 9.14 Example of restriction notation

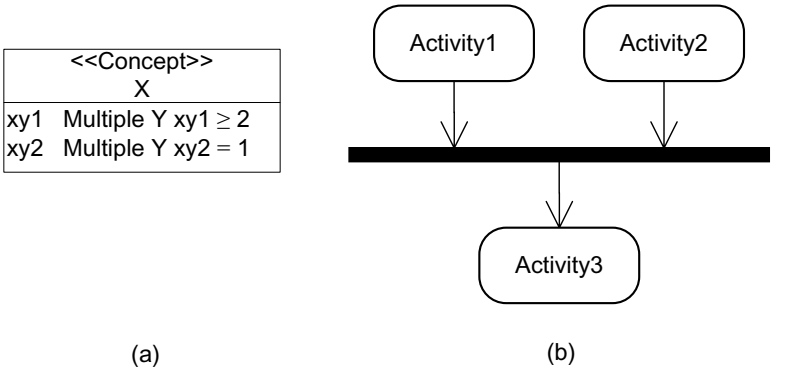


Fig. 9.15 Examples of restriction notation

An example of the cardinality restriction is shown in Figure 9.15. From Figure 9.15 (a), it can be inferred that at least two xy1 relations relate instances from class X to class Y. For the property xy2 restricts instance from class X to exactly one instance of class Y. An example of this case is in UML, join transition (from activity diagram); there are at least two related activities transited into one related activity as shown in Figure 9.15 (b).

9.2.1.5 Associated Class Notation

Association class can be used to participate in further associations for property in ontology. To specify association classes, a dotted line is used to attach the property notation to the class notation as shown in Figure 9.16(a).

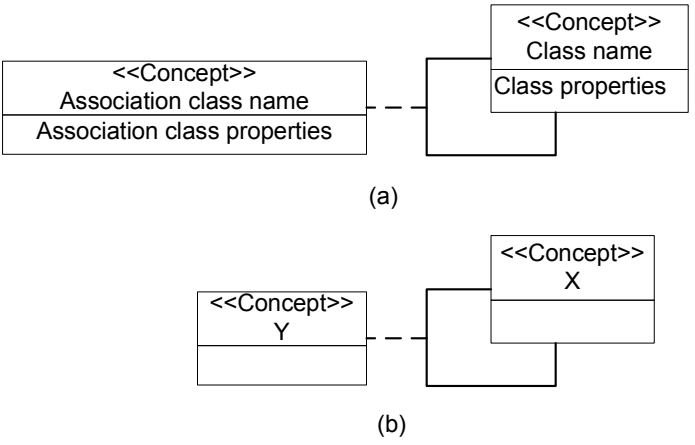


Fig. 9.16 Associated class notation and example

The example shown in Figure 9.16 (b) illustrates that class Y is an association class of object property that relates class X together. Association class Y can participate in further associations with bundles of properties.

9.2.1.6 Ontology Instances Notation

An instance notation is represented as an ellipse with a dot line attached to its class or property as shown in Figure 9.17 (a). If it is an instance of property, then in the ellipse it contains the property name followed by a colon and then the instance name as shown in Figure 9.17 (b). Unlike an instance of class, in the ellipse there is only the instance name. To make it easy to read, a dotted line is attached

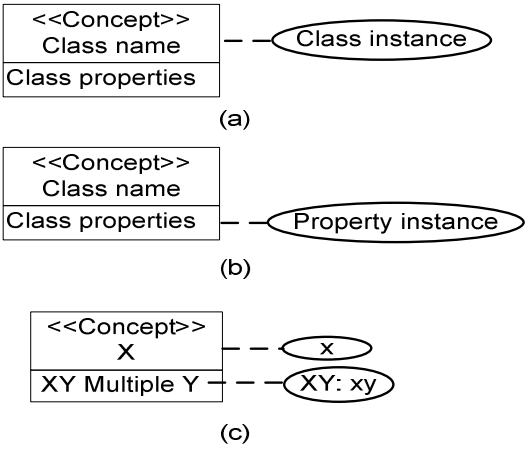


Fig. 9.17 Instance notation and example

to most of the class name or property name of its class instance or property instance. For example, x is an instance of class X and xy is an instance of property XY as shown in Figure 9.17 (c).

9.2.2 Agent Modelling Notations

One of their key characteristics of modelling Agents is that they are sociable. This means that they are able to interact with each other in order to co-operate, collaborate and negotiate with respect to information, knowledge and services. Very often each agent will have only part of the full picture needed to solve the problem at hand. The ability to subdivide the tasks in order to reduce the complexity of the problem, have individual agents work only on their aspect of the problem, and then combine sub-solutions into a final solution is extremely helpful and productive.

Being goal driven is a feature of many different agents. In order to consider what this means we can re-examine the concept of search space. Forward-chaining begins with data which drives the reasoning toward goals. Backward-chaining goes backwards decomposing goals into sub-goals and then checking to see if any of them is true. If so, the ultimate goal is considered to be true. If not, then the process of decomposition is continued.

Most traditional software is not goal driven as such, but is a black box. That is, specific combinations of inputs lead to specific outputs. The fact that an agent has an overriding goal, regardless of the specifics of its processing, endows it with many other features. Specifically, it will be pro-active. Even if there are no events generated by human users that trigger the agent, it will perform actions on its own to try and meet its goals. It will also be intelligent in trying to make use of its environments. For example, if the goal of the agent is to find certain data, it may migrate to another site once it has exhausted all possibilities at the current site. The decision to migrate may come from within the agent, rather than being triggered by an external event.

In this case, we use the composite structure diagram, and extend it by using a stereotype in order to define the constructs necessary to define the goal-driven aspect of an Agent.

The basic definition for a composite structure diagram in UML 2.1 is as follows. A composite structure diagram is a diagram that shows the internal structure of a classifier, including its interaction points to other parts of the system (Wongthongtham et al. 2008). It shows the configuration and relationship of parts, which together, perform the behaviour of the containing classifier (Wongthongtham et al. 2008). Class elements have been described in great detail in the class diagrams. The classes can be displayed as composite elements exposing interfaces and containing ports and parts.

9.2.2.1 Role Notation

Each agent is defined by specifying a specific set of roles that it plays. Each role could be associated with a distinct interface. These interfaces could be specified by a technique called "method lifting" outlined below. Method lifting defines a composite class. Firstly, we consider a hierarchy of component classes, each of which has an interface. Then we relate these component classes to a composite class that also have an interface, and which is formed by taking a selection of methods from the interfaces of the component classes. This process of relating the interface of component classes to the interface of a composite class is known as method lifting. In the example shown in Figure 9.18, the methods A, B & C are individually chosen from different component classes and combined in the composite class at the head of the hierarchy.

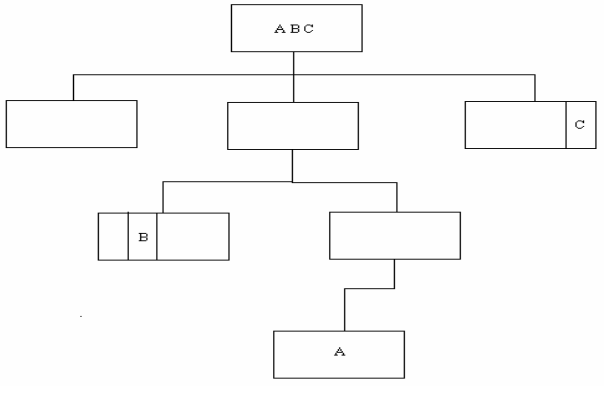


Fig. 9.18 An example of Method Lifting (Wongthongtham et al. 2008)

Secondly, a particular class may have more than one lifeline. For example, a particular class may have many ports, each one with its own lifeline. This is invaluable in the case of modelling Agents since we need the facility to be able to represent an Agent in a sequence diagram, where it plays more than one role concurrently. Refer to Figure 9.19 for an example of this. The agent may be represented by a rectangle, and have many ports, each with its own lifeline. The method lifting paradigm above the composite class may have many interfaces, each of which chooses a selection of methods from a hierarchy of component classes used as the source for method lifting. A sequence diagram where composite classes have more than one port is shown below in Figure 9.19.

9.2.2.2 Communication Notation

Generally, communication between objects is done in UML in a sequence diagram, or a communication diagram. They are semantically similar although a sequence diagram can generally be made to contain additional information. A sequence

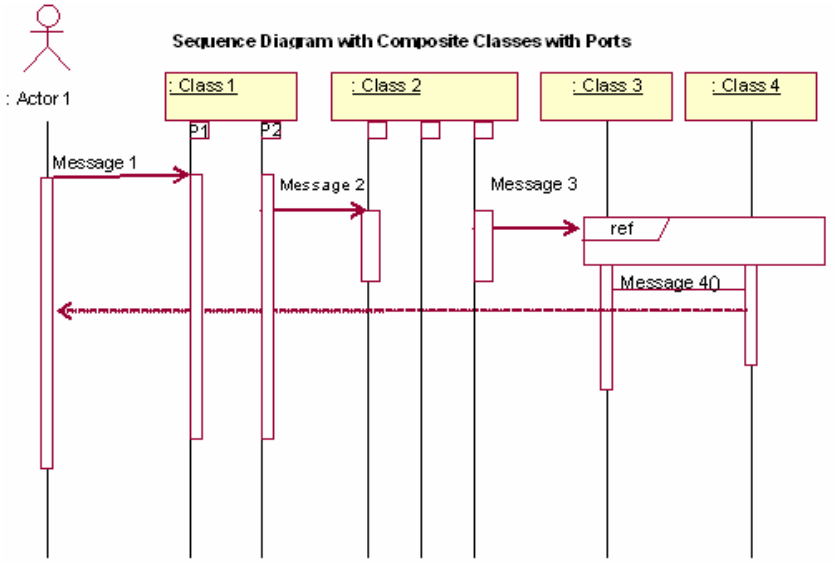


Fig. 9.19 Sequence diagram giving an example of a composite class with ports

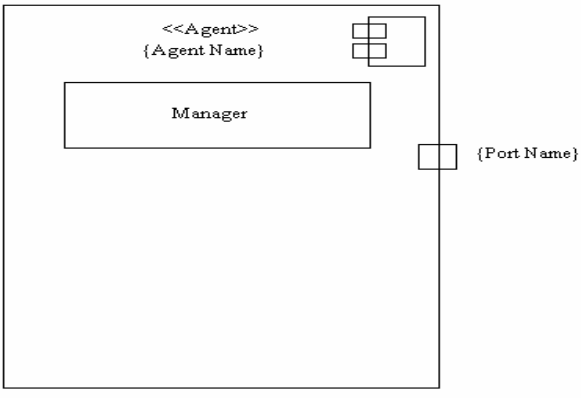


Fig. 9.20 Agent notation

diagram is generally defined across the page by a series of rectangles, each of which represents a class. Each of these rectangles has a dotted line running vertically down the page. These dotted lines are known as lifelines. As you go down the page, time passes as messages flow between objects.

9.2.2.3 Agent Notation

Figure 9.20 shows the agent notation based on the composite structure. From the definition it must have a name, at least a Manager part which controls the efforts of

the Agent to achieve a goal, and at least one port, which relates to its playing a role.

9.3 Diagrams

This section precedes illustrative examples of the integrated ontology and multi-agent system design. An example of ontology modelling for representing a part of trust ontologies is shown. Figure 9.21 shows ontology representation of human relationships. Basically, a human relationship is a relationship between human entities. The human relationship can be specific to the relationship between provider and customer and we call this particular relationship a business relationship. Ontologically, the business relationship is a sub-ontology of a human relationship so it has to commit to the upper ontology's specification. Properties of trusted entities, trusting entities, and trust value in upper ontologies are inherited by sub-ontologies. Properties of trusted entities and trusting entities in the business relationship can be made specific to the human entities of provider and customer respectively.

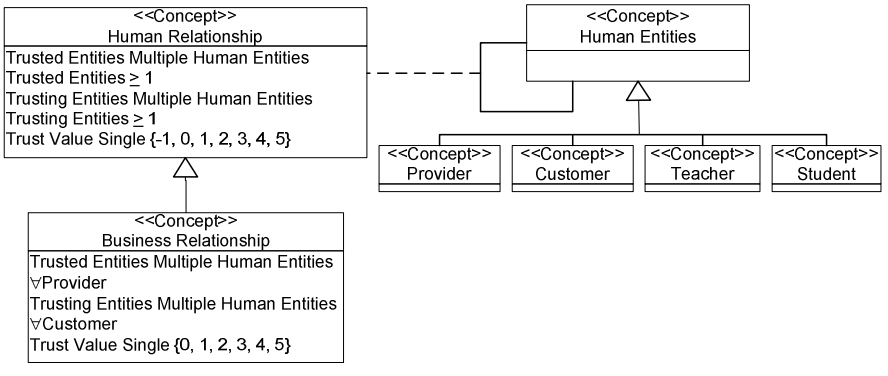


Fig. 9.21 A part of trust ontologies

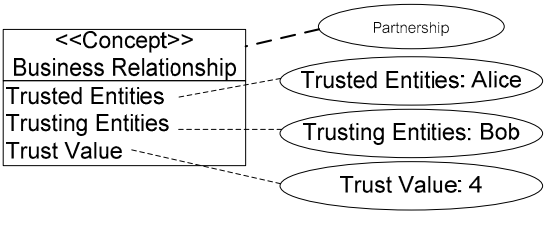


Fig. 9.22 Business relationship instances

Figure 9.22 shows ontology instances of the business relationship. Ontologically we can say that the partnership between Alice and Bob is a kind of business relationship and the trust value of this particular relationship is worth 4.

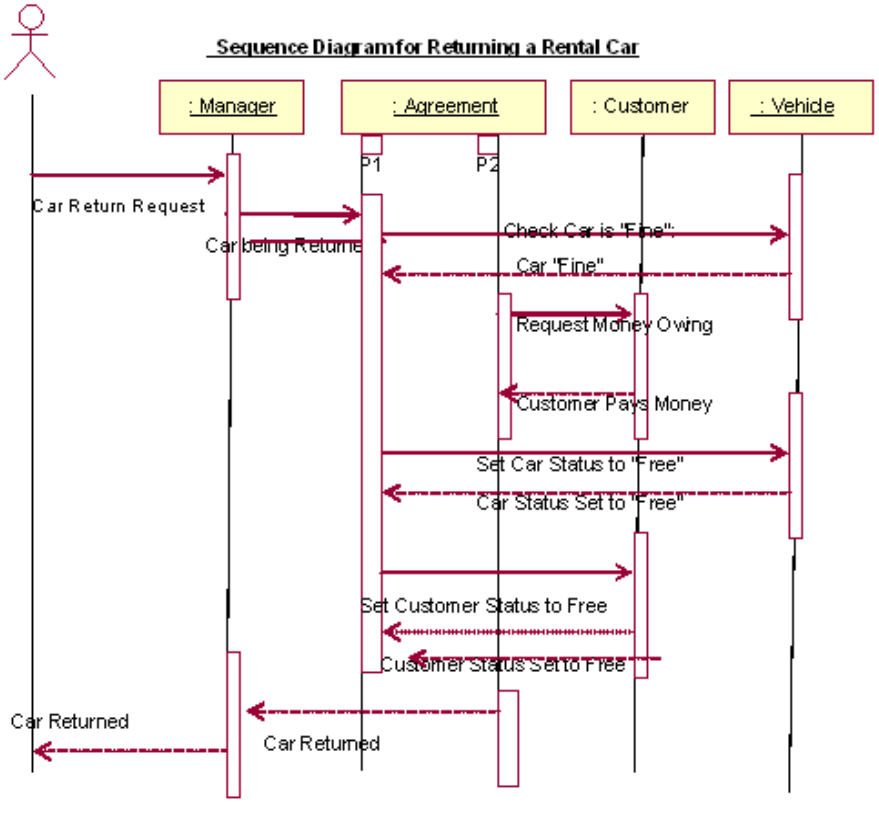


Fig. 9.23 Sequence Diagram illustrating Sociability of Agents

We now proceed to an illustrative example involving a set of Agents, one of which (Agreement Agent) plays two roles concurrently represented by P1 and P2. Depending on which role the agent is acting in when it sends/receives messages, the sequence diagram will show arrows to/from a particular lifeline for the agent. The corresponding sequence diagram of a rental car being returned to a depot (for a car rental system) and payment being made by the customer is shown below in Figure 9.23.

If we follow the sequence across and down, we note a number of points. Firstly, P1 (or port 1) represents the :Agreement agent in its role to establish status. P2 represents the :Agreement agent in its role to perform transactions. Note that each port has its own lifeline. If there are two ports, this signifies two roles that are played by the agent from which the ports come. Initially, the request is made to

return a car. Secondly, the Agreement agent checks that the car is fine, and receives a message back that this is so. The same agent then performs a transaction to request the money owing on the car and the customer agent pays the money. Note that the Agreement agent plays two different roles here. First is the role to check the status of the car, and second is to perform the transaction. Then, the Agreement agent sets the status of the car to “free”, and receives a message back from the Vehicle agent that the car status is “free”. Again, the Agreement agent is acting in its role to compute the status of the car. The Agreement agent makes a request to the Customer agent to set it to “free”. The Customer agent sets the status to “free” and returns the message to the Agreement agent that the Customer is free. Finally, the Agreement agent sends a message to the Manager agent that the car has been returned, and the Manager agent sends a message that the car has been returned to the Employee.

All the interaction between different Agents is shown on this sequence diagram. Importantly, an agent (Agreement) is shown playing two different roles on the same sequence diagram (Status and Transaction) within the same time frame.

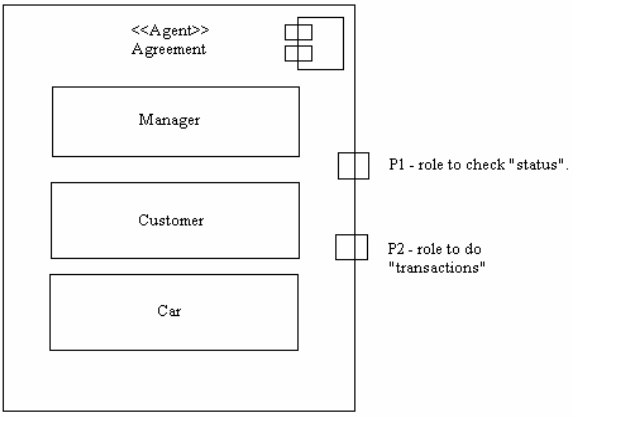


Fig. 9.24 Composite Structure Diagram representing Goal Driven characteristic of Agent

We can then model the goal-driven aspect of the agent by a Composite Structure Diagram with Parts, and Ports. Each part represents a distinct area of processing within the agent. Each port represents a different role played by the agent. The diagram encapsulating this information is shown in Figure 9.24.

Note that the same two ports that were present in the sequence diagram are also present here. Each of the ports is a construct which enables the Agent to interact with the environment and with other Agents. For example, if the goal of the Agent is to close out processing with respect to a specific Rental Agreement, then the Agent will have to consult the Goal Driven part of the Agent to decide to check the car and the customer processing part of the Agent to finalize return and payment, and the goal driven part itself to see whether the necessary checklist of items has been finalized for the return of the car.

9.4 Conclusion

In this chapter, we have defined modelling structure for integrated ontology and multi-agent system modelling. The abstract syntax, semantics, and notations are defined. The notations further structure into metamodels/diagrams which are logically structured. It is a semi-formal graphical representation for specifying and modelling an integrated ontology and multi-agent system. We designed the model to encompass a broad set of both ontology and multi-agent system theories and approaches inclusively.

References

1. da Silva, V.T., Noya, R.C.: Using the MAS-ML to model a multi-agent system. In: Software engineering for multi-agent systems II: research issues and practical applications, pp. 129–148 (2004)
2. Duric, D.: MDA-based Ontology Infrastructure. Computer Science and Information Systems (2004)
3. Evermann, J.: A UML and OWL description of Bunge's upper-level ontology model Software and Systems Modeling (2008)
4. Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: A Tool for Collaborative Ontology Construction. In: 10th Knowledge Acquisition for Knowledge-Based Systems Workshop Banff, Canada (1996)
5. Gašević, D., Djurić, D., Devedžić, V.: Model Driven Architecture and Ontology Development. Springer, Heidelberg (2006)
6. Genesereth, M.R.: Knowledge Interchange Format - draft proposed. American National Standard (1998)
7. Genesereth, M.R., Fikes, R.E.: Knowledge Interchange Format Version 3 Reference Manual. Stanford University Logic Group (1992)
8. Genesereth, M.R., Gikes, R.E.: Knowledge Interchange Format Version 3.0 Reference Manual (1992)
9. Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J.: UML for ontology development. The Knowledge Engineering Review 17, 61–64 (2002)
10. MacGregor, R.: Inside the LOOM classifier. SIGART bulletin 2, 70–76 (1991)
11. Odell, J., Parunak, H.V.D., Bauer, B.: Representing Agent Interaction Protocols in UML. In: Agent-Oriented Software Engineering, pp. 121–140. Springer, Heidelberg (2001)
12. Wongthongtham, P., Dillon, D., Dillon, T.S., Chang, E.J.: Use of UML 2.1 to model multi-agent systems based on a goal-driven software engineering ontology. In: Proceedings of the 4th international conference on semantics, knowledge and grid (SKG 2008), China (2008)

Chapter 10
Architecture of the Integrated Ontology and Multi-Agent System

10.1 Introduction

In this chapter, we explore the architecture of the integrated ontology and multi-agent system. The ontology provides an important mechanism to facilitate producing semantic information. Since the ontology has been used to express formally a shared understanding of information, it enables the sharing of an agreement among users by making assumptions explicit. The key idea is to have agreement explicitly interpreted by software tools rather than just being implicitly interpreted by a human. The representation of knowledge including ideas, tasks, models, processes as well as documentation using an ontology and sub-ontology, will provide intuitive, clear, precise concepts and ideas, knowledge and classified issues. Knowledge is organised into the ontology and used as the basis for classifying knowledge enabling questions and problem solving. Additionally, knowledge can be shared among users in community. A framework showing the use of ontologies in conjunction with multi-agent system is given in Figure 10.1.

As the figure 10.1 indicates, resources consist of a generic ontology, a specific ontology, and a knowledge base repository that are for software agents to consult or refer to. Generic ontology represents general knowledge in the domain. A subset of ontological knowledge can be identified. Specific ontology represents the specific knowledge and that specifically meets a particular information resource need. The specific ontology is to be used for a particular purpose. Note that the specific ontology can also be tailored and customised, and once it has been created, it should be available to share among the community. All users are encouraged to obtain knowledge from it though multi-agent system and studying, and obtaining answers, classifying knowledge and using it as the basis of any conceptual discussion and questions arising, including documentation. The knowledge base stores the ontologies along with actual data in an ontology repository. Set in the environment, these resources can be used for sharing of intra- and inter-communications of both common knowledge in the domain knowledge that have been established by consensus agreement by different users. In particular, the specific ontology fosters a seamless and virtual environment for internal browsing, searching, sharing, and authoring of ontological information.

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 165–177.
springerlink.com

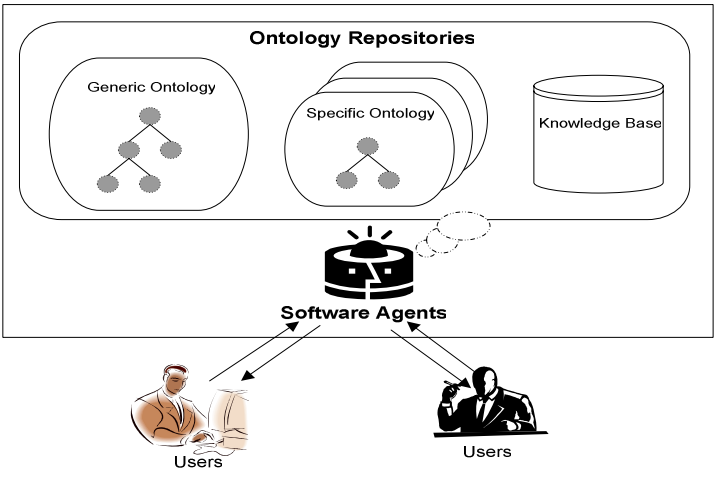


Fig. 10.1 Framework of the use of ontologies in conjunction with multi-agent system

Software agents in a multi-agent system are intelligent, autonomous problem solvers capable of getting answers from user queries, making decisions based on appropriateness, communicating with other agents and conveying results to the system or the users. They have their own goals, capabilities and beliefs what allow them to act intelligently within their field of expertise. Ontologies coupled with a multi-agents system allow greater ease of communication by aggregating the agreed knowledge and the domain knowledge into a shared information resource platform and allow them to be shared among users and enable the intelligent agents to use the ontology to carry out initial communication with users when the problem is raised in the first instance. The system utilises software agents-based computing in the sense that the agent has knowledge through consultation with ontologies in the ontology repository. Due to agent capacities in reading and reasoning published knowledge with the guidance of the ontology, the shared ontology enables agents to have meaningful communications.

10.2 Ontology Server

The ontology is made available to any application or software agents to deploy or reason the knowledge. The ability to make use of the knowledge described in the ontology enables applications or agents in the systems to have capabilities in managing or reasoning knowledge stored in the ontology repository.

As an example, we design a set of agents cooperating with each other within the multi-agent system and interacting with users. These are: (i) user agents which represent each user being provided with services; (ii) safeguard agents which represent system authentication for user authorisation and determination of the access level; (iii) ontology agents which represent manipulation and maintenance of the ontology; and (iv) decision maker agents which represent decision making

on the matter of updating the ontology. The safeguard agents communicate with the ontology agents if the user (through user agent) wants to query or update the ontology. The ontology agents manipulate and maintain the ontology repository. The functionalities of safeguard agents include system authentication, access level allocation, solution proposal management and monitoring of the user activities. Functionalities for the ontology agents include navigating, querying and manipulating knowledge formed in the ontology.

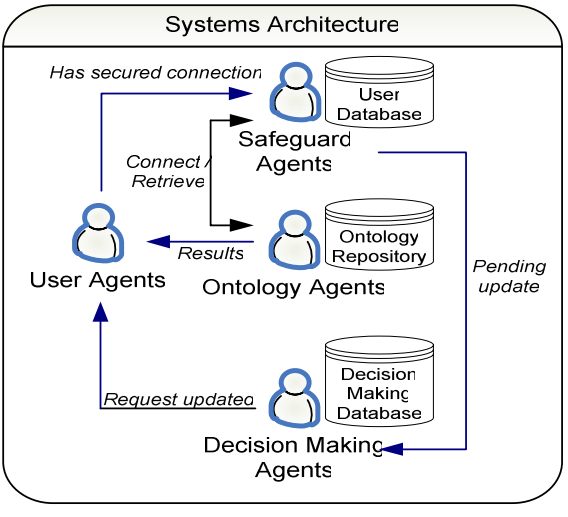


Fig. 10.2 Overview example architecture of the integrated ontology and multi-agent system

The example architecture of the integrated ontology and multi-agent system is shown in Figure 10.2. Each user is assigned to an individual user agent serving as the communication media. It uniquely identifies each user on the system. This allows direct communication between different users using a messaging system and allows monitoring of the users' activities. The user agents represent each user being provided with services, e.g., to query knowledge formed in the ontology and with a particular set of access privileges that are dependent on the role of that user in community. The user agents communicate with the safeguard agents that represent system authentication for user authorisation and determination of the access level. The safeguard agents communicate with the ontology agents if the user agents want to query or update the ontology. The ontology agents manipulate and maintain the ontology repository. The ontology agents contact the decision maker agents if an operation needs to be certified. The decision maker agents are responsible for decision making on the matter of updating the ontology including acknowledgement of the decision made to all involved users. As can be seen from the model of the architecture, only the safeguard agents have a connection with the user database. This means that the safeguard agents record all user activities as

well. All agents call the safeguard agents and pass information to log all the events that the agents carried out. Thus, tracking can be accomplished by the safeguard agents if needed. Not only do they allow tracing, but the safeguard agents can also determine bottlenecks if there is any occurrence with the use of the timestamp. The ontology agents are the only ones manipulating the ontology. Thus, it is the only one that has access to the ontology repository. All agents contact the ontology agents in the cases of wanting to view, query, or update the ontology. The decision maker agents have their own database to store data for decision making occurring in the system.

In detail, the functionalities of the integrated ontology and multi-agent system can be observed in Figure 10.3. The safeguard functionalities include system authentication, access level allocation, solution proposal management and monitoring of the user activities. Functionalities in the ontology agents include navigating, querying and manipulating the ontology.

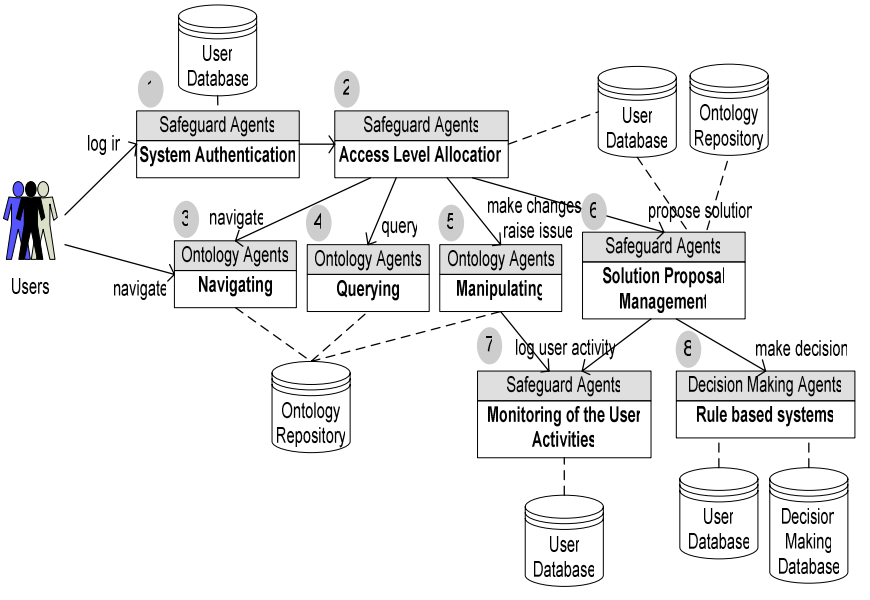


Fig. 10.3 The functionalities of the integrated ontology and multi-agent system

Every user can navigate the knowledge formed in the ontology but no changes are allowed. A user can log into systems. Once logged in, system authentication verifies user access from user database. Once authorised, the user will be provided the access privileges. The user can now navigate, query, make changes, raise issue, or propose solution. Navigation, query, and manipulation of ontology that are functioned by the ontology agents are recorded into user database. Solution proposal is managed by safeguard agents. Any decision is made by decision making agents.

10.2.1 Ontology Repositories

The architecture of the ontology repositories can be consisting of three packages: generic package, specific package, and ontology package. The generic package defines the interfaces of the data structures of generic ontology and generic ontology objects. Likewise, the specific package defines the data structures of specific ontology interfaces and specific ontology objects, such as class and its instances. Both generic and specific packages can provide an in-memory or data persistent implementation of the data models of generic and specific ontologies respectively. The ontology package provides the utilities for the ontologies defined in generic and specific packages.

Generic ontology can be accessed by anyone without system authentication. It can be used for a search of concepts relating to the domain. Unlike specific ontology, it is meant to be used for particular purposes, and therefore system authentication is required. Generic search allows searching of any concept within the domain. Search results display the contents of the concept the user specifies including its subclasses, its properties and restrictions. The output can be made in XML format in order for it to be displayed in user-friendly mode on the web browser. Basically, the display of the hierarchy of subclasses can be accomplished using a recursive function. The function will find out the entire sub-concepts of a concept by recursively calling the function itself over and over again until no more sub-concepts can be found.

A specific package provides a set of functionalities that helps users to have a mutual understanding through the use of specific ontology. The user can update as well as withdraw relevant knowledge from the ontology. The set of functionalities includes: view, query, add, delete and modify. To retrieve instances of a concept or ontology class, a function retrieves all direct instances related to the class or the concept. From here, users can browse this instance information. All information associated with the instance is like its properties (both object properties and data type properties) or its relationships, value inside those properties and its restriction. The main purpose of retrieving its relationships is to help the user understand its underlying concept and to help update to be completed according to its domain concept. It is easy to become confused if discussion takes place while ambiguity exists in the domain. Therefore, by retrieving the relationships associated with instances, it can help users illustrate what they truly mean. This is done even better with some automation function. Manipulating specific ontology is an essential tool to help maintain ontology. In reality, there are always updates from time to time.

The ontology package is a compilation of functions that provides the utilities for the ontologies defined in ‘generic’ and ‘specific’ packages. It does not belong to any category and does not have enough information to create its own category either. The functions in the package are mainly like (i) getting ontology name space, (ii) search engine for the ontology system, (iii) ontology class or concept restrictions and property characteristics checker to specify the range or restrict the values of input, (iv) converting information into XML format for output and (v) parsing and serialising ontologies in OWL language. Firstly, obtaining ontology namespace is necessary in order to extract the namespace of the ontology. The OWL file stores all the information of the different URLs and namespace in the

header of the file. When the OWL file is loaded into the ontology model by calling the Jena modelfactory (McCarthy 2004), all the URLs and namespace for the ontology can be retrieved. Since the ontology model loads all URLs first then loads the namespace of the ontology, the namespace is always the last element. By using Java's StringTokeniser (McCarthy 2004) the namespace of the ontology can be retrieved with ease. Secondly, a function in the package serves as a search engine for the ontology system. It especially includes finding any close match of the ontology class or concept. This is useful when the search does not return any exact match to the user. Ontology restrictions include quantifier restrictions and cardinality restrictions, and ontology property characteristics include functional, inverse functional, symmetric and transitive properties. Functions of checking all ontology restrictions and property characteristics are all in the package to restrict the conditions. A function checks whether there is a minimum cardinality restriction implemented on the concept or ontology class. Minimum cardinality restriction refers to the minimum number of properties that must be input in order to satisfy the condition. For maximum cardinality restriction, a function checks for a maximum number before a new property value is added. If the maximum cardinality number has been reached, the addition of a new property value is disallowed, or else the addition of a new property value is allowed. If there is a cardinality restriction present, it means there can be no more and no less cardinality than the cardinality specified. The quantifier restrictions of allValueFrom means that all the values of this property to whom this restriction applies, must have all values falling within its range. Likewise, someValueFrom restrictions, some values of the property that this restriction applies to, must have some values falling within its range. Therefore, the aim of a function for this is to check whether the new value which is going to be added falls into the category (if so return true; if it does not fall into the category, return false). These kinds of restrictions are only for adding new object properties. Fourthly, a function is to convert information retrieved into XML format for output. This function is used often to display instance information including its restrictions information retrieved. All restrictions follow the same output format with XML tag as the name of the restriction and its value. Lastly, parsing and serialising ontologies in OWL language are needed for format translation. A format or syntax translator requires the ability to parse, represent the results of the parsing into an in-memory or persistent ontology model, and then serialise. Manipulation capabilities for example would also be required, in between the parsing and serialising processes, in order to allow construction and editing of ontologies. In the implementation view, parsing is taking the OWL file and converting it to an in-memory or a persistent ontology model. Conversely, analogous to the parsing, serialising produces an OWL concrete syntax in the form of a syntactic OWL file from an in-memory or a persistent ontology model.

10.2.2 Ontology Evolution

This section focuses on ontology evolution and factors that allow the ontology to evolve. Ontology evolution is defined as "the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent

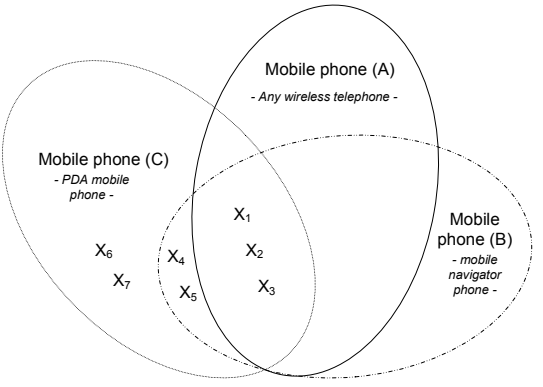


Fig. 10.4 Definition of mobile phone

artefacts (Stojanovic, 2004)". An example of ontology evolution is illustrated in Figure 10.4, where the concept 'mobile phone' defined in the ontology evolves as technology develops to improve the human condition in a society.

In this example, mobile phone (A) (i.e. any wireless telephone) could evolve in time. Based on the ontology, a mobile phone is formally defined by the set of primary attributes – x1 (to transmit speech at a distance), x2 (to transmit a message at a distance), and x3 (wireless operation). However, with timely improvement, these three attributes may not be sufficient to describe the technology of a mobile phone. For instance, two more attributes, x4 (to provide precise positional and velocity data for air, sea, and land travel) and x5 (to get directions and track progress along the route) should be further defined for a mobile navigator phone. The introduction of x6 and x7 has given rise to the concept mobile phone (B). In a similar vein, for a PDA mobile phone which combines the mobile organisational features of a personal digital assistant with the connectivity of a mobile phone, the secondary attributes x8 (email capabilities), x9 (high-speed downloads), etc. led to the formation of the concept mobile phone (C) for a PDA mobile phone. Note that x4 (to provide precise positional and velocity data for air, sea and land travel) and x5 (to obtain directions and track progress along the route) are also included in the PDA phone and navigator phone because a PDA phone can include the features of a GPS phone (e.g. by installing GPS software). From this example, one can see that the evolution of ontology plays a key role in a developing and changing world.

There are three types of important architectural data elements. The first element represents unresolved queries. The second element represents the latest/different understanding of related information. The ontology agents will check whether these comply with the ontology. By referring the ontology, the agent is able to resolve these. Otherwise, the ontology agents record them in the third data element representing different concepts, which include all information (requests) that do not comply with the semantics, relations, and concepts defined in the ontology and notify the requestors of this non-compliance. These different concepts form the basis for ontology evolution which can be facilitated by social software such as Wiki and RSS/Atom.

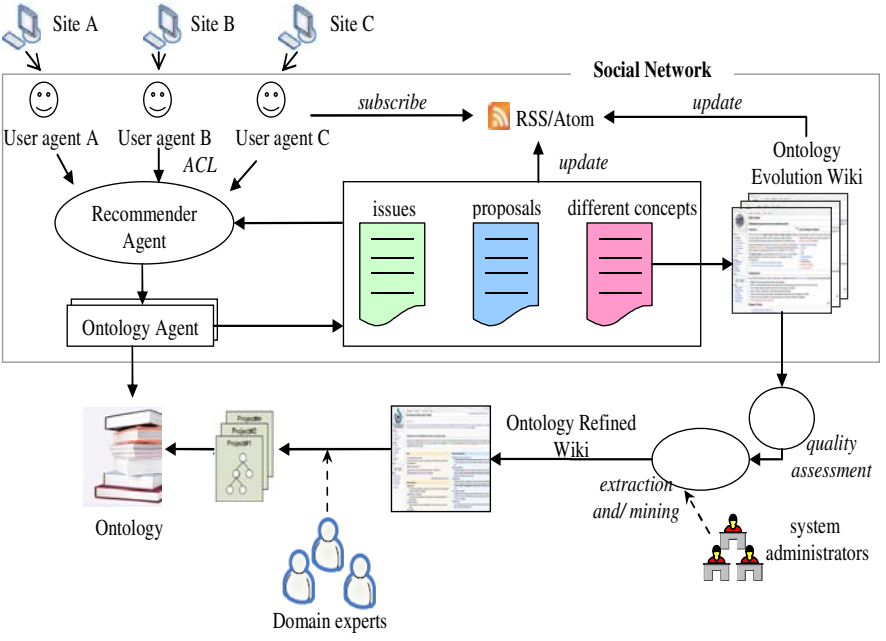


Fig. 10.5 Ontology evolution system architecture

These different concepts are stored in the database and represented as an Ontology Evolution Wiki (OEW) as shown in Figure 10.5. OEW allows each user agent to express its own understanding of certain concepts and interpret them in ways different from ones defined in the ontology. More importantly, the complete sets of revision histories stored in a Wiki server provide a preliminary tracking mechanism that facilitates "the quality assessment" extremely essential to ontology evolution. For example, (Lim et al. 2006) have suggested that Wiki articles' qualities and contributions can be effectively measured using these revision editing data. However, (Lim et al. 2006) focus on the quality of each article, whereas we talk more about the quality of each concept/term edited/defined on the Wiki. This requires not only distinct quality measurement models but also effective information extraction techniques that can capture important concepts from collaboratively edited Wiki articles. Mining semi-structured information and limited forms of partial structured natural language can be leveraged to facilitate the "extraction and mining" process. Once the quality assessment, data mining and information extraction has been done, the remaining selected concepts will be used to generate the Ontology Refined Wiki (ORW). Unlike OEW, ORW is used by domain experts, who will then collaboratively discuss, edit, and decide the final version of each concept that needs to be updated in the ontology. Once these experts have reached agreement on the changes to be made, final versions of these

concepts are merged with the existing ontology with the help of ontology editor tools. Thus, the ontology evolution approach considers both users' opinions (e.g. OEW) and experts' decisions (ORW).

10.2.3 Ontology Management

In this section, we illustrate how ontology can be managed to facilitate knowledge sharing. It is how the man-machine system interfaces particularly works. The man-machine interactions could be designed and developed into four platforms: Knowledge Navigation Platform, Question and Issue Platform, Suggestion and Proposal Platform and Solution Decision Platform.

The Knowledge Navigation Platform, or Navigation Platform for short, is basically for all users to navigate or query shared domain knowledge and instance knowledge. Anyone can view domain knowledge for classification of certain concepts and can view instance knowledge for clarification of agreements or understandings. However, permission is needed in the Navigation Platform if users want to make changes to the knowledge. Authorised users can make changes from the Navigation Platform. Unauthorised users can propose changes as an issue. The issue will then be processed through the Question Platform, Suggestion Platform, and Solution Platform depending on how substantial the changes are. The Question and Issue Platform, or Question Platform for short, is for raising difficulties encountered or issues which occurred within. A user logs a matter through the Question Platform. Basically, issues raised could be comments or debates over data and agreement or different opinions on knowledge representation itself. If it is about data or agreement, a question posed needs to be elucidated by clarifying the specific data he/she wants to discuss. As a user progresses towards clarifying the issue, the retrieval of relevant knowledge by the platform assists other users to have a clear understanding of the issue as well as assisting them to recognise the involved knowledge they have been discussing. The Suggestion and Proposal Platform, or Suggestion Platform for short, is as its name suggests, for all users to propose the potential solutions of the discussion or issue proposed in the Question Platform. Thus, this platform mainly involves potential modification of knowledge. In the Suggestion Platform, all proposed solutions are initially pending. After a certain time of gathering some feedback from users, the necessary action or solution is determined through the Solution Decision Platform, or Solution Platform for short. The actual updating process is also carried out through the Solution Decision platform. A final solution resulting from the discussion is revealed by the Solution Platform. The ontology gets updated in the ontology repository at the stage.

10.3 Agent Platform

JADE (Bellifemine et al. 2001) can be utilised as the platform for building intelligent agents in compliance with the Foundation for Intelligent Physical Agent

(FIPA) allowing agents to communicate with the other agents in the same standard. JADE implemented in Java simplifies implementation of multi-agent systems through a middleware. Monitoring and controlling the status of agents while implementing can be utilised as provided by the JADE platform. Agent communication intentions e.g. inform, propose, query, etc. are available in Agent Communication Language (ACL). The agents themselves not only listen but can decide on a course of action on their own. Some functions that JADE provides to deploy a multi-agent system are as follows (Bellifemine 2001):

- Yellow page service – provides a directory service for agents to publish their service as well as search the directory for the services they want
- Agent life-cycle management – from the creation of a particular agent to its termination
- Message transport service – formats the message with a certain tag and allows easier filtering of different types of messages and easier searching of messages.
- Security – JADE provides a set of security protocols that can be implemented if a secure JADE platform is required.

An agent can be assigned to every user who logs into the system after authentication and provides different services to different access levels for each user. All users are provided with the service to query the ontology. Each user has a different access level: Querying level – only the querying service is allowed no modification, Modifying level – restricted access to add and modify service of the ontology, and Full access level – unrestricted access to all services provided.

These access levels are given according to the different status of the users. Agents' lifecycles reside on the server, and a user is assigned to a user agent when a login is made. Each user agent is an initiator based on the user actions; the agent will carry out the specific operations accordingly. All the operations have different logic involved, but the structure of creating a user agent is the same. In the agent creation process, an agent object is created when a user login is carried out. In the agent listener process, the agent listens to any events based on the user actions. After a fixed waiting time expires, with no any (other) actions from the user, the agent will terminate itself. The agent will set up the required communication in the service creation process setting based on user operation e.g., setting the service type. Then the agent needs to find the identical service type in the Directory Facilitator (DF). The DF contains all the services published by agents that provide their services. The user agent will look up the DF for any services that match the one it is looking for. If the user agent cannot find the service, it deems that the service is unavailable for now and the operation will be terminated. During the process of ACL message setup, the user agent creates a message object and sets up the content to be passed to the other agent. Normally it is a request for service. If the particular operation needs to communicate multiple times, or additional tasks need to be carried out, the ACL message will then be set up again in a loop action. Once the goal has been achieved, the user agent will call the logger to log all the

activities. The user agent will then kill itself if the user decides to log off the system or the user is idle for too long; otherwise, the agent will go back to the listener process.

This is typical behaviour of goal-specific agents, which exhibit one-shot behaviour. In other words, the agent is created for a purpose, and once the purpose has been achieved, the agent will be terminated. The life-cycle of the user agent listed above applies to all the operations that the user performs. And if the user decides to use another operation, a new user agent will be created for that specific operation.

To implement security features, another agent could be appointed named the "safeguard agent". The safeguard agent implements and enforces the system's authentication, the access control policies, and user activity log. The user identification will be verified with the user database as well as access level allocation. There will be a few security levels, mainly users who have no access to updating, and can only do look- up, users who have an access to update data, and users who have unlimited access to all functions such as updates, backups, ontology maintenance and database access. The safeguard agent is created when the platform is initialised and running. It cannot be terminated by normal means, and only the leader/manager has access to undertake platform maintenance, or due to certain circumstances, may force shutdown of the platform or kill any agent processes. This agent is the only one that connects user agents to the ontology agent; therefore, if it is killed or terminated, the system will not be functional at all. In the safeguard agent creation process, the agent is created when the platform is initialised and running. Then the services that are provided by the safeguard agent are initialised, set up and posted to the yellow page, which is the directory of the platform. From this directory, all agents will search and find the service they are looking for. The safeguard agent will idle until another agent establishes a link or makes contact with it. Once a connection has been established, the service type is checked and if it matches the service type the that safeguard agent is providing, further processing will be carried out; otherwise the message is discarded and the request is rejected. There will be two possible scenarios. The first is where the user requests to view, add, and update ontology. In this case, the safeguard agent needs to verify the user account first. If the user has an authorisation, the safeguard will retrieve the user access level and then pass the request to the ontology agent. The second case is where the safeguard is requested by any agents to record user activities. Because the safeguard agent is the only agent that connects to the user database, if there is any action to log into the database, it will be done by the safeguard agent. The safeguard agent will always be alive unless it is killed by human intervention.

The purpose of having an ontology agent is to manage connections with the ontology. The ontology agent provides navigating, querying, and manipulating ontology. The design philosophy of the ontology agent is to use the in-memory or persistent storage model and serialise it into a physical document stored in the

ontology repository. There are three different services: navigating, querying and manipulating services. Users can access knowledge held in the ontology model. A user can navigate in the ontology for clarification or classification certain concepts. The information provided is in hierarchical form so upper level concepts or lower level concepts or adjacent concepts can easily be navigated. It also serves as a searching tool to help narrow down the vast amount of concepts in the ontology. Through the use of the ontology search function, the user can re-classify concepts to match his/her needs. This leads to the specific ontology. Note that the information provided by this function can be arranged in XML format, which means that it can be easily managed to display only a certain part of the information retrieved or can provide a different display interface with the same set of retrieved information.

Technically for this function, the ontology agent focuses on the ontology model, the set of statements that comprises the abstraction and instantiations. To navigate the ontology, the ontology agent reads ontology into a model and then accesses the individual elements.

The ontology agent acts as a responder by publishing its services to the directory and allowing other agents who are looking for the service it is publishing to communicate with it. As the ontology agent is a service provider, it does not terminate itself after finishing a communication with another agent. It will always be executed again whenever an agent communicates with it. This type of behaviour is classified as cyclic and the agent will terminate only if the platform is terminated or forced to be terminated. In the ontology agent creation process, the ontology agent is created when the platform is initialised and running. Then the services that are provided by the ontology agent are initialised, set up and posted to the directory. From this directory, all agents will search and find the service they are looking for. The ontology agent will be idle until another agent establishes a link or makes contact with it. Once a connection has been made, the service type is checked and if it matches the service type that the ontology agent is providing, further processing will be carried out; if not, the message is discarded and the request is rejected. There will be two different services, i.e. viewing and updating services. For the viewing services, the ontology agent simply carries out querying. For the updating service, the ontology agent considers whether it is a minor change or major change. For a minor change, the ontology agent updates the ontology immediately and also records the changes. For a major change, the ontology agent checks whether the update request has been authorised. Basically, for major changes, the ontology agent will pass the request for changes to proceed further processes, for example, gathering information and consulting the ontologies in the ontology repository. On passing through, the updating can be done by the ontology agent. Every activity will be recorded by the logger process. The results of the process are sent to the user agent that made the enquiry. The ontology agent will automatically wait for another connection and will be terminated only if it is forced to be terminated, either by a kill process or a platform collapse.

10.4 Conclusion

In this chapter, we have explored architecture of the integrated ontology and multi-agent system. The ability to make use of the knowledge described in the ontology enables applications or software agents in the integrated ontology and multi-agent system to have capabilities in managing or reasoning knowledge stored in the ontology repository located in ontology server. We have also focused on ontology evolution architecture and ontology management. Before concluding, we discussed how JADE can be used for the building of agents and an agent platform.

References

1. Bellifemine, F.: JADE Java Agent DEvelopment Framework. Telecom Italia Lab Torino, Italy (2001)
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA 2000 compliant agent development environment. In: The fifth International Conference on Autonomous Agents, pp. 216–217. ACM Press, New York (2001)
3. Lim, E.P., Vuong, B.Q., Lauw, H.W., Sun, A.: IEEE/WIC/ACM Conference on Web Intelligence, Measuring Qualities of Articles Contributed by Online Communities (2006)

Chapter 11
Case Study I: Ontology-Based Multi-Agent System for Human Disease Studies

11.1 Introduction

In this chapter, we shall put the Onto-Agents Methodology into practice and use it step by step to develop a Generic Human Disease Ontology (GHDO) and to design a multi-agent system that can use the designed GHDO for intelligent information retrieval.

In Section 2, we discuss the domain of human diseases, the purpose of the ontology, the community of users and agents for which this ontology is being developed, and applications based on the designed ontology. The aligning and merging of existing medical ontologies against the defined ontology structure is discussed in Section 3. Section 4 describes how to design the ontology base, while Section 5 explains the design of the ontology commitment layer. In Section 6, we discuss the evaluation of the GHDO.

In Section 7, we identify and describe different groups of agents according to their functions and responsibilities within the system. In Section 8, we describe a mechanism by which GHDO ontology is used in this system during the processes of problem solving, task and result sharing, and assembling of results. In Section 9, we focus on the structural organization of the agents within the system and define agents' collaborations. Individual agent structure and agents' components are described in Section 10. In Section 11, we discuss security issues within the multi-agent system. Examples of the use of the designed systems are given in Section 12.

11.2 Generalization and Conceptualization of the Medical Domain

In this section, we discuss the following points:

- Community associated with the ontology
- Purpose of the ontology
- Domain that needs to be represented by the ontology
- Application based on the designed ontology

We will also give a top-level hierarchy of the proposed Generic Human Disease Ontology (GHDO).

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 179–216.
springerlink.com

11.2.1 Community Associated with the Ontology

Firstly, we will discuss the community of users of the system.

The first group of users is faced with a situation of a disease and is interested, for example, in disease symptoms for the purpose of correct identification and diagnosis of this disease, and/or they may be interested in possible treatments for a particular disease. This group of users consists mainly of patients and physicians.

The second group of users is interested in disease causes, disease prevention and/or how to improve available disease treatments. The second group of users is comprised mainly of medical researchers whose goal is to make a disease situation easier for patients and physicians. Medical researchers may look for disease causes and on the basis of that, develop or discover more effective treatments.

Note that there are no strict boundaries here. A physician may also be interested in the genetic causes of a disease. She/he can screen the patient's DNA for the purpose of detecting any (mutated) gene to confirm the presence of a disease. Also, a medical researcher may be interested in information about disease treatments, especially if she/he is working on the drug design.

Secondly, we will mention the community of agents that are committed to, and are using, the designed ontology during the process of intelligent information retrieval. We identified four different groups of agents. Interface agents assist the user in forming queries as well as returning the retrieved and assembled information to the user. Interface agents communicate a user's request to the Manager agents. Those agents then assign specific tasks to the Information agents in order to gather the requested information in the most efficient way. The Information agents retrieve the requested information from a wide range of biomedical databases. Each Information agent may have a set of assigned databases it needs to visit in order to gather the requested information. The Information agents hand over the retrieved information to Smart agents. These agents select appropriate information, assemble it correctly, and hand it over to the other agents which will direct it back to the user as an answer to his/her query.

Due to the complexity and size of the body of knowledge represented by the human diseases ontology, we decided to partition the ontology into its subontologies, and for each of the subontologies, design agents with the four abovementioned functionalities. Such a system has a hierarchical structure and will be discussed in more detail in the second part of this chapter.

11.2.2 Purpose of the Ontology

The medical knowledge contained in various databases needs be intelligently extracted in order to improve medical research, disease control and patient care. Also, new medical information is being constantly added to the existing pool of knowledge. Researchers and medical practitioners need mechanisms to capture and diffuse domain-specific knowledge. This requires structured and standardized organization of data.

We believe that the use of ontologies would facilitate the sharing and reusing of the medical information, cooperation, querying of the information in an intelligent way, embracing all the knowledge shared by the medical community into a unifying framework and offering the possibility of one common ontology being used by various applications.

11.2.3 Ontology Domain

We need to understand the general structure and organization of the medical domain before we start building software models. The complexity of the medical domain is due to different factors:

- structure and content of information resources. The information resources are autonomous and a huge number of them contain raw and heterogeneous data. The information is distributed across various information resources and a thorough search is impossible. The medical domain is dynamic because new information is constantly being added and the number of users increases. Consequently, network traffic also increases. There is the need for an intelligent and dynamic information system to enable efficient information retrieval.
- complexity of medical research experiments. Medical research experiments are complex due to the long DNA sequence, the heterogeneity of study groups, and the multiple factors that together may cause a disease such as in the case of psychiatric illnesses.
- social issues such as security and privacy issues, social and professional acceptance, safety critical issues and legal issues need to be effectively addressed within the medical domain, especially because of the sensitive nature of patient information.

In this chapter, we focus on the medical domain regarding human diseases. We aim to embrace all the information that may be helpful in the study and control of human diseases. This may include information regarding laboratory experiments, drugs used to treat human diseases, disease symptoms, case studies etc. We believe that by combining all this information into a common framework, it becomes easier for the users to identify and/or recognise conceptual relationships in the data and identify possible links between originally specially distributed and heterogeneous information.

For example, medical research experiments may provide the results that may be useful during the drug design process. We may need to combine the information of drug treatments with disease symptoms in order to effectively examine, for example, the effect that a new drug has on disease symptoms. We believe that all subdomains (such as examining genetic causes of diseases, drug design technology, monitoring symptoms of diseases etc.) of the domain of human diseases need to be unified as they are targeting the same goal but at different knowledge levels.

11.2.4 Application Based on the Designed Ontology

We aim to design an ontology-based, multi-agent system that can make use of the Human Disease Ontology for the purpose of intelligent and dynamic information retrieval. Ontology is used by the agents at different levels to decompose the overall problem into smaller problems, to locate and retrieve the significant information, to communicate with each other, to cooperate and coordinate their actions, to analyze and manipulate the retrieved information, and to present this information to the user in a meaningful way.

11.2.5 Top-Level Hierarchy of GHDO

We identified two main user categories of the system:

- medical researchers who are mainly interested in the causes of a disease; and
- physicians and patients who are faced with the situation of a disease and are mainly interested in symptoms and treatment of a disease.

We believe that the ontological model of the Generic Human Disease Ontology (GHDO) needs to have four main branches:

1) disease types, describing different types of a disease;
2) phenotype, describing symptoms of a disease;
3) causes of a particular disease which can be environmental or genetic. Microorganisms, such as bacteria and viruses, can also cause a large number of diseases.
4) treatments, giving an overview of all treatments possible for that particular disease.

The GHDO top-level hierarchy is illustrated in Figure 11.1. The information presented in this figure indicates that a disease may have different types that may be further divided into subtypes and sub-subtypes. For each disease, there is a corresponding phenotype or observable characteristics of an ill individual, namely the symptoms of a disease. Each disease is caused by factors(s) which can be genetic (genotype), environmental, or caused by a microorganism. Information about the genetic causes of a disease can be about a mutated gene, a complex of genes or a DNA region of interest. A DNA region of interest is a region in the DNA sequence that potentially contains a gene responsible for the disease. This region needs to be further examined in order to correctly locate the mutated gene. Environmental causes of a disease can be stress, climate, drugs or family conditions. Microorganisms that may cause a disease may be viruses or bacteria. Possible treatments for a disease can be drug therapy, chemotherapy, surgery, psychotherapy or physiotherapy.

The four different branches (subontologies) of the GHDO ontology can serve as a reference point against which the concepts from the existing medical ontologies can be reorganized, aligned and merged.

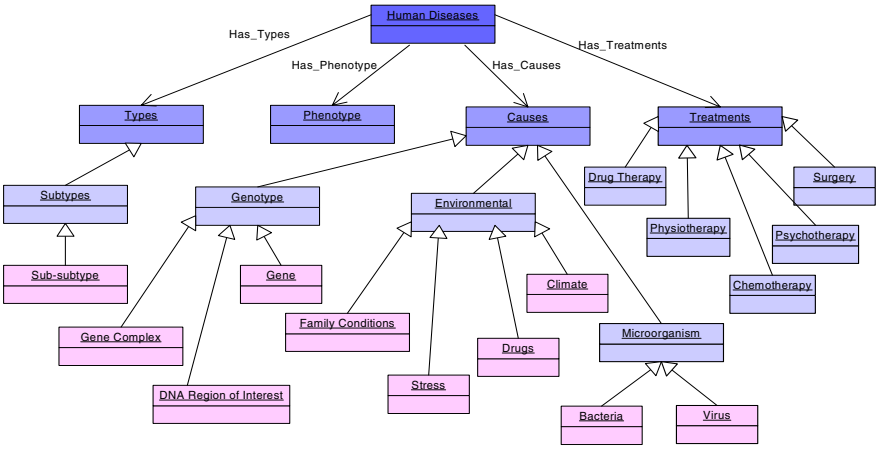


Fig. 11.1 Top-level hierarchy of GHDO

The four subontologies (disease types, symptoms, causes and treatments) are different from each other in the knowledge they represent and in the way their concepts are organized. We say that these different subontologies are orthogonal to each other.

The Types subontology is more a classifying ontology and is strongly hierarchically supported. It provides information about different types and subtypes of a specific disease. For example, psychiatric diseases have different types such as depression, schizophrenia, bipolar disorder etc. Those different types can be further divided into subtypes. For example, for bipolar disorder type, we have Bipolar I and Bipolar II subtypes. This subontology is based on classification and it does not provide a user with much scientific information.

The Phenotype subontology is more descriptive than the others. It is based on observation and diagnosis of the characteristics of an ill individual. Usually, physicians are in direct contact with patients and are able to provide this information.

The Cause subontology provides a user with scientifically proven facts and is strongly based on scientific research. Most of this information results from laboratory experiments and is provided by medical researchers. For example, bacterial and viral characteristics and their role in disease development are examined in medical laboratories.

The Treatment subontology is a combination of classification and research ontology. Comparing different characteristics of different drugs targeting the same disease and working on a drug design is medical research work but, for example, all the discovered drugs can be classified according to a hierarchical structure.

All four dimensions are different from each other and each dimension is unique. But jointly they provide an overall picture and a good overview of knowledge about a human disorder.

11.3 Aligning and Merging Existing Medical Ontologies

Researchers in the medical ontology-design field have developed different terminologies and ontologies in many different areas of the medical domain. A large number of medical ontologies cover overlapping domains. The need for sharing and reusing this body of knowledge becomes increasingly critical. It would be of great value to construct a common terminology for all different applications within the same knowledge domain and so enable the integration and exchange of information between different applications.

This stage of ontology design focuses on the process of building a new ontology by reusing and adapting ontological terms from other ontologies. In this section, we will focus on the following points:

- identifying suitable ontologies that can be reused
- defining merge and alignment tool
- importing source ontologies
- identifying correspondences between different ontologies
- aligning and merging ontologies

11.3.1 Identifying Ontologies Suitable to Be Reused

Rather than creating a new terminology, we suggest reusing existing medical ontologies. When designing GHDO, other medical ontologies, such as LinkBase (Ceusters et al. 2001), can be reused. The way that these concepts are organized within the existing medical ontologies is not suitable for use by our system. Moreover, LinkBase contains more than one million concepts and only those concepts suitable for, and needed by, our information system need to be selected. TAMBIS ontology (Stevens et al. 2002) represents general knowledge in regard to proteins and associated genes. If a gene is mutated, it may result in a non-functional protein. The presence of a non-functional protein in the human body can cause a disease. This is the reason that some biomolecular ontologies, such as TAMBIS ontology, would be suitable for that part of our ontology that relates to genetically caused diseases. UMLS (Bodenreider 2004) and SNOMED (Ceusters et al. 2004) terminologies can be used to validate the chosen concepts.

It is thus possible to use concepts from the existing ontologies but to organize these concepts according to the four ontology dimensions (disease types, symptoms, causes and treatments) mentioned in the previous section. The resulting ontology can then be used for our purpose and by our information system.

11.3.2 Define Alignment and Merge Tool

Because we adopted the DOGMA principle (separation of the ontology base from the ontology commitment layer) in our ontology design, we will use the DOGMA-Modeler tool (Spyns 2005) to design a Generic Human Disease Ontology (GHDO).

It seems logical then, to use an ontology alignment and merge tool associated with DOGMAModeler to align and merge source ontologies with GHDO.

11.3.3 Import Source Ontologies

The identified ontologies suitable for reuse should be imported into the new ontology design environment. As different ontologies may be designed using different tools and may have different formats, it may be necessary to convert all these ontologies into one format in order to use them efficiently. This standardized format needs to be supported by the chosen ontology alignment and merge tool i.e. DOGMAModeler in our case.

11.3.4 Identify Correspondences between Different Ontologies

The ontology structure of the source ontologies, the manner and order in which the terms are organized within these ontologies may be similar but is almost always different from the structure of the ontology being designed.

The same fact may be represented differently by two different ontologies. This is the reason that correspondences among the source ontologies need to be established. The set of overlapping concepts, concepts that are similar in meaning but have different names, and concepts that are unique to each of the sources, need to be determined.

11.3.5 Align and Merge Ontologies

By aligning the source ontologies, links will be established between the aligned ontologies. This will allow the designer to have more control over the ontology design process and to get a clear overview about the knowledge represented by different ontologies. In places where different ontologies overlap, the ontology designer is able to choose the most suitable concept definitions/relationships. In other places where there is no overlap between different ontologies, the ontology designer can either accept the suggested knowledge representation, or define its own.

After making and acting upon these choices, a single ontology will be created as a merged version of the original ontologies.

11.4 Formal Specification of Human Disease Domain Conceptualization

Formal ontology representation enables ontologies to be used by computers. In this section, we will focus on the formal specification of conceptualization while in the following section we will focus on the formal specification of ontology commitments.

In this section, the aim is to define:

- Ontology concepts
- Relationships between concepts
- Lexons
- Relationships between lexons
- Groups of related lexons

11.4.1 Ontology Concepts

At this stage, we need to identify and precisely define ontology concepts needed to represent the domain of human diseases. Every concept definition is dependent on other concepts and we have to rely on the commonly accepted understanding of some basic terms. The main ontology concepts are shown in Figure 11.1.

11.4.2 Relationships between Concepts

This step involves identifying intentional and extensional ontology relationships:

1) Intentional relationships (conceptual relationships) map every concept from the domain of human disease to the Generic Human Disease Ontology (GHDO) structure. The output of this function is a set of chosen ontology concepts that is going to be used to design GHDO.
2) The GHDO structure consists of extensional relations between the chosen domain concepts representing the domain knowledge. An overview of the extensional relations is given in Figure 11.1 and is more precisely defined through the use of lexons.

11.4.3 Lexons

An ontology base is (Jarrar and Meersman 2002):

- A set of context-specific binary fact types, called lexons. Notation: $<\mu, t1, r, cr, t2>$. Here is $\mu \in \Delta$ an abstract context identifier chosen from a set Δ; t1, r, cr, t2 are term1, role, co-role and term2 respectively. The lexical terms (t1, r, cr, t2) are constructed from a given alphabet.
- For each $\mu \in \Delta$ and each term t occurring in a lexon, the pair (μ, t) specifies exactly a unique concept.
- Lexons are conceptual constructs expressing a binary conceptual relationship that is agreed to hold within a given context (among all the parties involved in the ontology).

For example, in ontology designed for representing the knowledge of human diseases, we have the following lexons expressing binary relationships:

- Lexon1: <human diseases, disease, has, is of, type> of the form <μ, t1, r, cr, t2>.

  In this expression, we have:

  a. context identifier μ is "human diseases"
  b. term t1 is "disease"
  c. role r is "has"
  d. co-role cr is "is of", and
  e. term t2 is "type"

We describe in the context of human diseases, two complementary binary relationships of the forms <t1, r, t2> and <t2, cr, t1>: <disease, has, type> and <type, is of, disease> respectively. This means that in the context of human diseases, "disease has type" and "type is of disease".

- Lexon2: <human diseases, disease, shows, characterizes, symptom> of the form <μ, t1, r, cr, t2>.

  In this expression, we have:

  a. context identifier μ is "human diseases"
  b. term t1 is "disease"
  c. role r is "shows"
  d. co-role cr is "characterizes", and
  e. term t2 is "symptom"

We describe in the context of human diseases, two complementary binary relationships of the forms <t1, r, t2> and <t2, cr, t1>: <disease, shows, symptom> and <symptom, characterizes, disease> respectively. This means that in the context of human diseases, "disease shows symptom" and "symptom characterizes disease".

- Lexon3: <human diseases, disease, is caused by, causes, cause> of the form <μ, t1, r, cr, t2>.

  Here, we have:

  a. context identifier μ is "human diseases"
  b. term t1 is "disease"
  c. role r is "is caused by"
  d. co-role cr is "causes", and
  e. term t2 is "cause"

We describe in the context of human diseases, two complementary binary relationships of the forms <t1, r, t2> and <t2, cr, t1>: <disease, is caused by, cause> and <cause, causes, disease> respectively. This means that in the context of human diseases, "disease is caused by cause" and "cause causes disease".

11.4.4 Relationships between Lexons

The binary nature of relationships serves as a conceptualization constraint and enables modelling of the most basic and atomic fact within the domain of human

disease. To allow for the introduction of higher order semantic relationships, it is possible to treat a lexon as a term in another lexon.

Consider the following example:

- Lexon4: <causes, Lexon3, is, is of, genotype> of the form <μ, t1, r, cr, t2>.

Here, we have:

a. context identifier μ is "causes"
b. term t1 is "Lexon3"
c. role r is "is"
d. co-role cr is "is of", and
e. term t2 is "genotype"

We describe in the context of disease causes, two complementary binary relationships of the forms <t1, r, t2> and <t2, cr, t1>: <Lexon3, is, genotype> and <genotype, is of, Lexon3> respectively. This means that in the context of causes of a disease, <"disease is caused by cause" and this "cause is genotype"> and <"genotype is of cause" and this "cause causes disease">.

Consider the following lexon:

- Lexon 5 <causes, genotype, is, is a, gene mutation> of the form <μ, t1, r, cr, t2>.

Here, we have:

a. context identifier μ is "causes"
b. term t1 is "genotype"
c. role r is "is"
d. co-role cr is "is a", and
e. term t2 is "gene mutation"

We describe in the context of causes of a disease, two complementary binary relationships of the forms <t1, r, t2> and <t2, cr, t1>: <genotype, is, gene mutation> and <gene mutation, is a, genotype> respectively. This means that in the context of causes of a disease, "genetic (cause) is gene mutation" "gene mutation is a genetic (cause)".

Medical researchers are searching the human DNA to find situations like this, while the physicians use this information as they can screen human DNA for the presence of this mutation and evidence of a disease.

11.4.5 Groups of Related Lexons

Context identifiers group related lexons in an intended conceptualization of a domain. A context can be defined as a mapping from Δ to a collection of sources, such as a corpus of documents on the same topic. In the example of Lexon4, we can cluster together all binary relationships expressing facts regarding disease causes and map this cluster to a corpus of documents on the topic of causes of human diseases. Note that sometimes a disease is also partly affected by the environmental conditions such as stress, climate, family conditions etc.

Lexons are always true and free of further interpretations. Specifications of improbable or impossible (contradictory) worlds are also possible, especially in the early stages of engineering an ontology, but in practice no applications can commit to them. For example, it is impossible for an application not to commit to Lexon3 and to commit to Lexon4 that is based on Lexon3.

All lexons in the ontology base are free of any specific interpretation and all rules and constraints implied on the lexons are moved to the commitment layer.

11.5 Formal Specification of Human Disease Ontology Commitments

The commitment layer is organized as a set of ontological commitments. Each commitment is a consistent set of rules, axioms in a given syntax that provide a specific interpretation to a subset of lexons in the ontology base. Within a commitment, we distinguish:

- Intra-commitments: rules that constrain and attribute specific interpretations to a selected subset of lexons contained within the lexon base (Jarrar et al. 2003), and
- Inter-commitments: a set of mappings that link elements of this subset of lexons to elements of specific applications (Deray and Verheyden 2003).

In this section, we aim to:

a. identify intra- and inter-commitments
b. formalize the ontology commitments
c. identify reusable knowledge components

11.5.1 Identify Intra- and Inter-Commitments

We will take examples from the previous section:

Lexon1: <human diseases, disease, has, is of, type>
Lexon2: <human diseases, disease, shows, characterizes, symptom>
Lexon3: <human diseases, disease, is caused by, causes, cause>

The intra-commitments are rules that constrain and attribute specific interpretations to this selected subset of lexons contained in the ontology base of the human diseases ontology. As the intra-commitments for those lexons of the ontology base, we have respectively:

- each disease may have a disease type
- each disease shows at least one symptom
- each disease is caused by at least one factor

Inter-commitments characterized on this subset of lexons are a set of mappings that link these lexons to elements of specific applications. For example, the lexons associated with these commitments can be linked to an application that retrieves information on the topic of human diseases.

11.5.2 Formalize the Ontology Commitments

Given a lexon <μ, t1, r, cr, t2>, its commitment can take the form of two perspectives <t1, r, t2> or <t2, cr, t1>. The first element is referred to as the theme, the second as the transition, and the third as the rheme (Jarrar and Meersman 2002).

In the examples of:

Lexon1: <human diseases, disease, has, is of, type>
Lexon2: <human diseases, disease, shows, characterizes, symptom>
Lexon3: <human diseases, disease, is caused by, causes, cause>

As the intra-commitments of the type <t1, r, t2>, we have respectively:

Commitment1: <each disease, has, zero-or-more type>
Commitment2: <each disease, shows, at least one symptom>
Commitment3: <each disease, is caused by, at least one cause>.

Note that a commitment of the form <t1, r, t2> or <t2, cr, t1> operating on a lexon of the form <μ, t1, r, cr, t2> from the ontology base does not mean that t1,t2, r and cr are the same in the lexon and in the commitment. For example, Commitment1 <each disease, has, zero-or-more type> operates on the Lexon1 <human diseases, disease, has, is of, type>. For the commitment layer, t1 is "each disease" and t2 is "zero-or more type". For the ontology base, t1 is "disease" and t2 is "type".

11.5.3 Identify Reusable Knowledge Components

The ontological commitments may be seen as a set of reusable knowledge components. In practice, similar applications reuse or inherit commitments from each other. New applications can commit to, and use, the existing ontology. This is a very useful feature that can be effectively used within multi-agent systems. Cooperatively working agents differ from each other, but require minor differences in their ontology as they are all working towards the same goal. We can use the same ontology base combined with different commitment layers to design slightly different ontologies for different agents.

Assume that Ag1 is an Interface agent and Ag2 is an expert on general knowledge regarding human diseases. The commitments of these two different agents regarding the same lexon <human diseases, disease, has, is of, cause> will be different as these agents have different responsibilities within the system. For Interface agents, it is not necessary for each user of the system to require information regarding diseases causes. For example, a user may be interested only in treatments for a disease. In this case, commitment of the form <t1, r, t2> for agent Ag1 that operates on the lexon < human diseases, disease, is caused by, causes, cause > is < each disease, has, zero-or-more cause>. On the other hand, commitment of Ag2 that operates on the same lexon is < each disease, is caused by, at least one cause >. Ag2 is expert on general knowledge of human diseases and it needs to know that a disease has at least one cause.

11.6 Human Disease Ontology Evaluation

The constructability of GHDO can be evaluated mathematically using Set Theory (Wouters et al. 2004), and practically through design of a prototype.

11.7 Classification of Agents According to their Responsibilities within the Human Disease Information Retrieval System

The outcomes of this step are analogous to the examples shown in Section 8.3.

11.7.1 Establish Intuitive Flow of Problem Solving, Task and Result Sharing

Intuitive flow of problem solving, task and result sharing can be accomplished in the following sequence of actions:

query formulation → task sharing → information retrieval → result sharing → result analysis → result assembly and result presentation

11.7.2 Identify Corresponding Agent Functions

We need agents to help query formulation, to decompose the overall problem and assign task to various agents that need to retrieve the needed information. We also need agents to analyze the retrieved information and assemble the selected information to be presented to the user.

11.7.3 Identify Corresponding Agent Types

We will use a Sequence Diagram where Composite Classes have more than one port and represent different roles of the same agent (Wongthongtham et al. 2008). This will enable us to model agents of GHDO-based multi-agent system which play more than one role concurrently. For example, Smart agent plays 2 different roles: collection of information from various Information agents and sending SHDO to the Interface agent.

In the examples shown in Figure 11.2, a number of agents play multiple roles which is represented by multiple ports. Information agent plays four roles; it retrieves information about disease types, symptoms, causes and treatments. Smart agent plays two roles; it collects the retrieved information and sends the SHDO to the Interface agent. Depending on which role the agent is acting in when it sends/receives messages, the sequence diagram shows arrows to/from a particular lifeline for the agent.

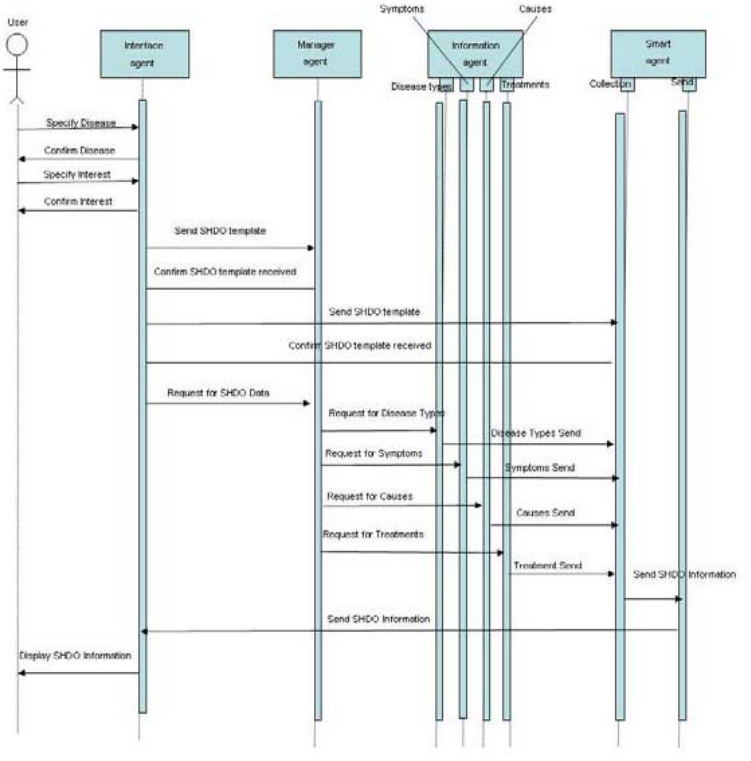


Fig. 11.2 Sequence diagram representing sequence of processes within GHDO-based multi-agent system

We illustrate a system that has four agents. The first agent is called Interface agent. The user initially specifies the disease that they want information about by sending a 'Specify Disease' message to the Interface agent. The agent confirms that this message has been received. The next step is that the user specifies their specific interest in the disease. That is, do they want information about disease types, causes, symptoms or treatments? Again, the Interface agent confirms that this message has been received. Once the Interface agent builds a SHDO template (called a Specific Human Disease Ontology Template) using Generic Human Disease Ontology, there is a structure established which can later be filled out with data. For example, the user may specify that they want information about Diabetes, and only on causes and symptoms, not treatments.

The SHDO template is immediately sent to the Manager agent, whose responsibility is to direct requests to Information agents who actually retrieve the data required to fill in the template. Thus, the Manager agent needs to have the empty SHDO template, in order to know which Information agents to activate. The SHDO template is also sent to the Smart agent. The responsibility of the Smart agent is to collect (from Information agents), analyze, select relevant information and fill out the SHDO template with data returned.

Once the SHDO template has been sent by the Interface agent, this agent makes a request to the Manager agent for Disease Data. The Manager agent then successively sends messages to an Information agent about Disease Type, Symptoms, Causes and Treatments. Each different type of information will require that the Information agent play a different role in order to gather it. Hence the messages go to one of four distinct ports, each of which models a distinct role for the Information agent.

Each time the Information agent gathers information it sends it to the Smart agent, who is responsible for collecting the information in order to fill out the SHDO template. Once the Smart agent has all the SHDO data collected and the template has been filled out it plays a different role to send the information back to the Interface agent. The Interface agent then returns the data to the user.

11.8 Identify the Need for Human Disease Ontology to Support Agents' Intelligence

11.8.1 Problem Decomposition and Task Assignments

The overall problem to be solved is constructed as a Specific Human Disease Ontology (SHDO) template by Interface agents. Retrieving and adding of relevant information to this SHDO template results in Specific Human Disease Ontology (SHDO). This is shown in Figure 11.3.

The SHDO template is decomposed into smaller subproblems by Manager agents. This kind of decomposition is hierarchical and the subproblems are further decomposed into smaller sub-subproblems, and so on. The SHDO template is first decomposed into its four subontologies (disease types, symptoms, causes and treatments). These subontologies are further decomposed into smaller sub-subontologies. The goal of this problem decomposition is to reach a stage where subproblems can be solved by individual Information agents.

A task assigned to an individual Information agent can be composed of more atomic actions. Information agents continue problem decomposition until the subproblems represent atomic actions that cannot be decomposed any further. The different levels of decomposition will often represent different levels of problem abstraction. For example, 'genetic cause of a disease' is on a higher abstraction level than 'gene'.

Agents must have appropriate expertise to decompose the overall problem and assign corresponding tasks to the agents of the system. They must have knowledge of the task structure and must know how the task is put together. For example, the Manager agent needs to know which Information agents are suitable for performing a specific task so that the Manager agent can assign this task to appropriate Information agents when the need arises. Also, the Information agent needs to know how and where to perform atomic actions of the overall task that was assigned to it. This is the reason why the ontology can be used to represent domain knowledge as well as the task structure.

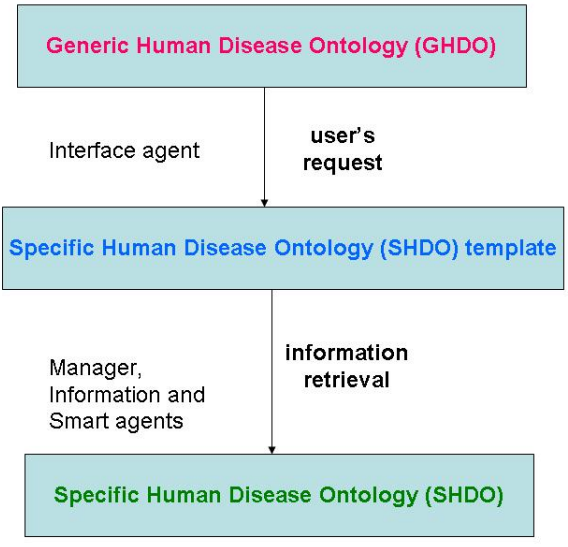


Fig. 11.3 GHDO, SHDO template and SHDO

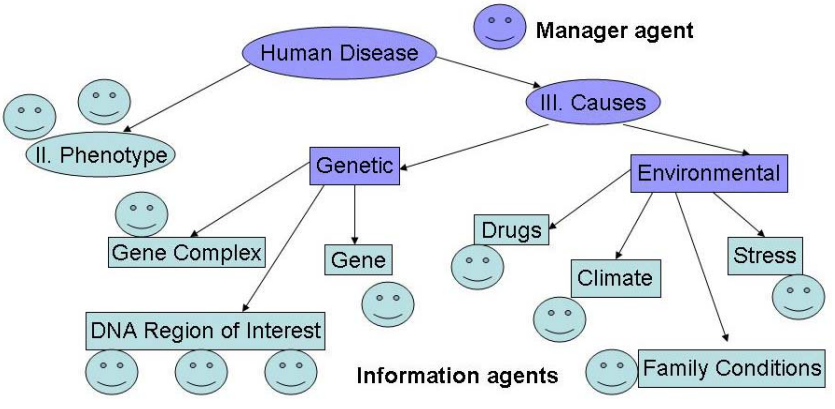


Fig. 11.4 Manager agent assigns tasks to different Information agents

As described above, a problem is decomposed into smaller subproblems by the Manager agent. Because the Manager agent knows exactly which Information agent is appropriate for the execution of a particular task, tasks are allocated to different Information agents. In Figure 11.4, we show an example where four different types of Information agents exist. Each agent is responsible for retrieving information on a specific topic, namely, on disease types, symptoms, causes or treatments. The directed contract is then established between Manager and Information agents. One Information agent is also aware of tasks assigned to other Information agents. Because different databases are assigned to different agents, it

is possible that one Information agent finds information significant to other Information agents. This is a situation where sharing of the information between different Information agents becomes important.

Generally, messages within a multi-agent system can be implemented as request and as information messages. In the case of simple requests for information, the construction of an SHDO template is not needed. The request message can then be used to encode a straightforward request for information. The information message can be used as a response to a request message. Some examples of the messaging within a multi-agent system include situations when a user has a simple request that needs activation of only one Information agent or situations between different Information agents when one Information agent finds information that is significant for another Information agent.

11.8.2 Information Retrieval

In this section, we will discuss two related features: atomic problems solutions and result sharing.

In the stage of atomic problems solution, subproblems identified during the problem decomposition phase are individually solved by Information agents. Usually, a task assigned to individual Information agents addresses a specific problem and is composed of more atomic actions. From the example given in Figure 11.4, one Information agent has the task of retrieving information about the effect of drugs on the onset of a specific disease. One aspect of this problem can be related to the medicines used to treat some other disease while the other aspect can be related to substance abuse. Each of these subproblems can be classified further into, for example, natural or synthetic drugs. The Information agents will perform the atomic actions and migrate from one database to another in order to accomplish their overall task, e.g., to retrieve information on the negative association between drugs and a specific disease.

Because different databases are assigned to different agents, sharing of the information between different Information agents within the system can be very useful. Agents share information relevant to their subproblems. The cooperative exchange of information covering different areas of the originally defined SHDO template enables the solution to be developed progressively. Solutions to small problems are gradually refined into larger, more abstract solutions resulting in the final result which is presented to the user.

11.8.3 Agent Communication

The agents are able to interact, freely share, and combine their results in the most accurate and efficient way only when communicating by use of a common language. All agents within this multi-agent system need to agree and commit to this common ontology language.

The ontology of an individual agent has two main components. One component is internal and more stable. This ontology component includes domain, procedural, task, cooperative, environment knowledge and similar. We note this component as

$\mathcal{A}$Oi. Conversely, the other component is dependent on the user's query and is easily changed. This is the $\mathcal{A}$Oe component. The agent ontology is then written as $\mathcal{A}$O, where

$\mathcal{A}O = \mathcal{A}Oi \cup \mathcal{A}Oe$.

The difference between the ontologies of different types of agents (such as Interface, Manager, Information, Smart agents) is larger than the difference between individual ontologies of the different agents of the same type.

The component $\mathcal{A}$Oi is different for different types of agents since they have different tasks assigned to them. The same type of agents may also have minor differences in the $\mathcal{A}$Oi component since different information resources are assigned to different Information agents.

We need to mention that the difference in $\mathcal{A}$O for the same task of the same agent is possible. This is the result of the dynamic multi-agent system where it is possible for the same action performed twice in apparently identical circumstances to have different effects.

Ontologies that represent medical knowledge as well as users' queries about this knowledge can be derived from a single generic ontology (Generic Human Disease Ontology, GHDO). We defined Template Constructor Function τ to map:

- generic ontology base relations $\mathcal{R}$ between terms t12 and t2 to true or false values, and
- generic ontology commitments $\mathcal{C}$ to true or false values.

We say that the specific ontology templates Specific Human Disease Ontology (SHDO) templates) are formed from the Generic Human Disease Ontology (GHDO) by Template Constructor Function τ,

$\tau : [(\mathcal{R} \rightarrow \{\text{true, false}\}) \wedge (\mathcal{C} \rightarrow \{\text{true, false}\})]$.

We represent this as **<GHDO, τ > → SHDO template**.

The Interface agent gives the resulting SHDO template over to the Manager agent. The Manager agent assigns different tasks to different Information agents. This would mean that the SHDO template needs to be partitioned into smaller tasks, $\mathcal{A}$Oe. $\mathcal{A}$Oe represents the task assigned to an individual Information agent. $\mathcal{A}$Oe is constructed from the SHDO template using the Subtemplate Constructor Function τ_a which maps relationships and commitments of the SHDO template to true and false values,

$\tau_a : [(\mathcal{R}_{\text{SHDO template}} \rightarrow \{\text{true, false}\}) \wedge (\mathcal{C}_{\text{SHDO template}} \rightarrow \{\text{true, false}\})]$.

If there are n Information agents, then n of such $\mathcal{A}$Oe are formed so that ideally the whole information within the SHDO template is covered, or

$\mathcal{A}Oe1 \cup \mathcal{A}Oe2 \cup \ldots \mathcal{A}Oen\text{-}1 \cup \mathcal{A}Oen$ = SHDO template.

All the resulting $\mathcal{A}$Oe are subsets of SHDO template.

We represent this as <SHDO template, τ1, τ2…τn-1, τn>, where τ1, τ2…τn-1, and τn are n different formation functions for agents Ag1, Ag2…Agn-1, Agn respectively, or **<SHDO template, τ1, τ2… τn> → AOe1, AOe2...AOen**.

Now that their task has been specified and formatted as AOe, the Information agents are able to search for and retrieve the requested information. The information retrieved by Information agents is given to the Smart agent. This agent analyses this information and selects relevant information to be added to the SHDO template. Addition of the selected information to the SHDO template results in an SHDO which is then presented back to the user.

There are three different parties involved in communication within this information system: user, agent and environment. The Interface agents communicate to the user during the process of query initiation and also during the presentation of results. The Information agents communicate with the environment and must move between different environments in order to access data. Communication between agents, such as communication between Smart and Information agents, allows agents to share information and coordinate activities, thereby enabling them to perform collaborative work.

11.8.4 Information Analysis and Manipulation

In this phase, the Smart agent analyses information provided by different Information agents. In the example from Figure 11.4, information regarding "DNA region of interest" is coming from three different Information agents. This information may be different or the same. If different information covers the same topic, information with the highest value needs to be selected by the Smart agent and incorporated into the SHDO template. In our case, "DNA region of interest" contains information about regions of human DNA which may contain a gene responsible for the onset of a particular disease if mutation (abnormal change of gene structure) of this gene occurs.

Table 11.1 Information retrieved by different Information agents regarding DNA region of interest

Agent1	2, p13-19	**10, q21-24**	17, q11-14	**17, q11-12**	X, q24-27
Agent2	10, q21-26	10, q21-25	**12, q23-24**	17, q11-13	**X, q24-25**
Agent3	2, p13-17	**2, p13-16**	**12, q23-24**	12, q23-26	17, q11-13

The Information agents may provide, for example, the following information for the case of manic-depression (Craddock and Jones 2001; Liu et al. 2003). A part of this information is presented in Table 11.1. The numbers represent chromosomes in human DNA that may contain the gene of interest (2, 10, 12, 17 and X chromosome) followed by the precise region of this chromosome where this gene is positioned (p13-16, q21- 24, q23-24, q11-12, q24-25 etc.).

The Smart agent compares this information on two levels:

1) On the first level, the Smart agent clusters the information according to the chromosome. For example from Table 11.1, we would have 5 clusters for chromosomes 2, 10, 12, 17 and X.

2) On the second level, the Smart agent compares information regarding chromosome regions for each of the chromosomes. In Table 11.1, for chromosome 17 we have regions: q11-13 (information provided by Agent 2) and q11-12 and q11-14 (information provided by Agent 1). In this context, a smaller DNA region of chromosome means being closer to the gene of interest. For this reason, smaller regions of chromosomes are selected by the Smart agent to be incorporated into the SHDO template. In the example of chromosome 17, region q11-12 would be selected.

The selected information is assembled into the SHDO template resulting in SHDO. In the example from Table 11.1, the following information would be selected and incorporated into the SHDO template: chromosome 2, region p13-16; chromosome 10, region q21- 24; chromosome 12, region q23-24; chromosome 17, region q11-12 and chromosome X, region q24-25.

11.8.5 Meaningful Information Presentation

Solutions to atomic problems are integrated into the overall solution by Smart agents. As in problem decomposition, this stage is hierarchical with partial solutions assembled at different levels of abstraction.

The use of ontology for meaningful representation of a knowledge domain is equally important in this stage. Information retrieved by Information agents is compared, analyzed and assembled according to the SHDO template that was constructed at the beginning by the Interface agent. This step results in SHDO which is presented to the user as the answer to his/her query.

11.9 Define Agent's Collaboration within the Intelligent Human Disease Information Retrieval System

11.9.1 Establish Efficient Organization of Agents

We need to focus on responsibilities and functions of different agents, and on the nature of communications within a multi-agent system in order to establish the most efficient way of organizing agents within a multi-agent system.

As GHDO has a hierarchical structure, we believe that a multi-agent system based on this ontology functions best when forming itself a hierarchy of agents corresponding to the four different ontology branches. We propose a GHDO-based Holonic Multi-agent Structure (GHMS) (Hadzic et al. 2006) as a nested hierarchy of four holarchies in which each of the four GHDO dimensions (Disease types, Symptoms, Causes and Treatments) is associated with one holarchy (see Figure 11.5). Highest in the agent hierarchy is the Disease Mediator Agent. For each of the four holarchies, we have corresponding Mediator, Specialist and Representative Agents. The information is interpreted and analyzed at the higher levels of the hierarchy while collection of the data happens at the lower level holarchy.

11.9.2 Establish Correspondence between Agent Types and Their Organization

In this section, we will discuss different agent types within the GHDO-based Holonic Multi-agent Structure (GHMS).

Disease Mediator Agent (DMA). The GHMS has Disease Mediator Agent as its main entry point.

On the basis of the SHDO template provided by the Interface agent, DMA decides which of the four holarchies needs to be engaged in order to generate the SHDO. For example, sometimes a user may be interested only in the causes of a disease so that there is no need to engage the Disease types, Symptoms or Treatments holarchy.

DMA has the function of a Manager agent. As there are also other agents from lower levels of the holonic multi-agent system that have the function of the Manager agents, we say that the DMA corresponds to the first level Manager agent.

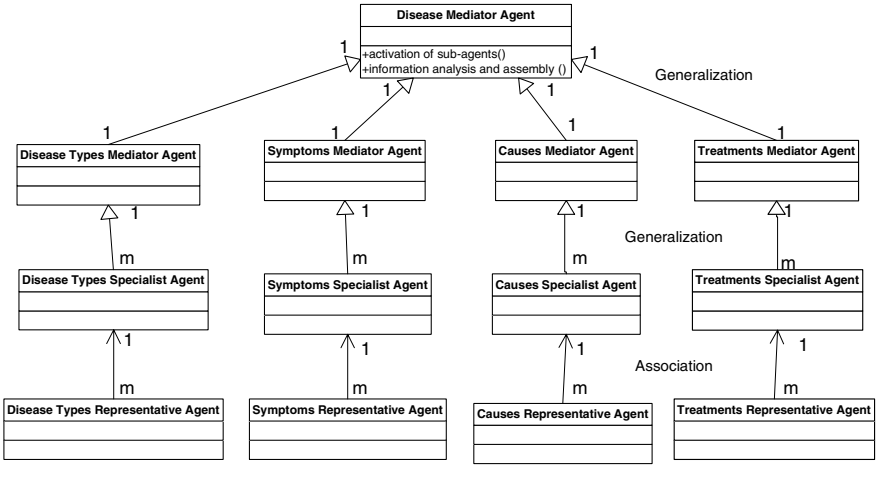


Fig. 11.5 GHMS structure

Mediator Agents (MAs). Each branch of the main entry point of GHMS (DMA) has its own mediator agents, respectively Disease Types, Symptoms, Causes and Treatments Mediator Agents (D-MA, S-MA, C-MA and T-MA).

Their task is to decide which other subordinate Specialist Agents (SAs) or Representative Agents (RAs) need to be activated in order to retrieve the information requested by the user. MAs correspond to the second level Manager agents.

Specialists Agents (SAs). Holarchy inner nodes represent Specialist Agents (SAs). We differentiate Disease types, Symptoms, Causes and Treatments Specialists Agents (D-SA, S-SA, C-SA and T-SA).

They represent decision makers and are specialists in a specific topic of the corresponding subontology. For example, one C-SA may be specialized in the genetic causes of a disease while another C-SA may be a specialist in the environmental causes of a disease.

SAs assign different tasks to different RAs. For example, C-SA that is specialized in the genetic causes of a disease activates one RA to look for "DNA regions of interest" and another RA to look for a specific "gene" that, when mutated, causes the disease in question. SAs correspond to the third level Manager agents.

After subordinate agents (RAs) have returned their data, SAs interpret, compare, and evaluate these data. In this way, all the delivered data are properly ranked and prepared to be assembled into the SHDO template. SAs also correspond to the third level Smart agents. Note that DMA and MAs correspond to the first level Smart agent and the second level Smart agents respectively.

Not only do they define a proper ranking among all the delivered data, but also an important function of SAs is to interpret the incoming data and conclude whether there is sufficient information retrieved in regard to the SHDO template. If not, the SA has to decide - on the basis of the delivered information - whether it makes sense to consult other RAs.

Representative Agents (RAs). The leaves are so-called Representative Agents (RAs). We differentiate Disease types, Symptoms, Causes and Treatments Diseases Representative Agents (D-RA, S-RA, C-RA and T-RA).

Each RA is an expert on the lowest level concept within the ontology. Note that RAs differ from SAs in that they need to recognize the significant information inside the appropriate database and retrieve that information. RAs correspond to the Information agents. The retrieved information is then passed over to the SA which will do the analysis, comparison and assembly of the retrieved information which is then passed over to the respective mediator agents.

For example, article_1 claims that a gene located somewhere on chromosome 6 is responsible for a disease in question, while article_2 gives more precise information regarding the gene of interest such as location 6p11-p17. C-RA retrieves both articles while C-SA gives to the C-MA only information from article_2. C-MA will do the matching and assign the value '6p11-p17' to the concept "DNA region of interest" of the SHDO template, telling the user that the DNA sequence positioned on chromosome 6 in the region between p11 and p17 potentially contains a gene which may be causing the specific disease. In this way, selection of information is performed and only significant information is presented to the user. This is especially important when a great deal of information regarding a specific topic is available.

11.9.3 Query Processing and Information Integration within GHMS

A user's query in the form of an SHDO template is firstly constructed by the Interface agent. This query is sent to the Disease Mediator Agent (DMA).

DMAs partition the SHDO template into four subontologies templates: $\mathcal{A}$Oe (disease types), $\mathcal{A}$Oe (symptoms), $\mathcal{A}$Oe (causes) and $\mathcal{A}$Oe (treatments). The resulting subtemplates of the form $\mathcal{A}$Oe (subontology) are sent to corresponding Mediator Agents (MAs).

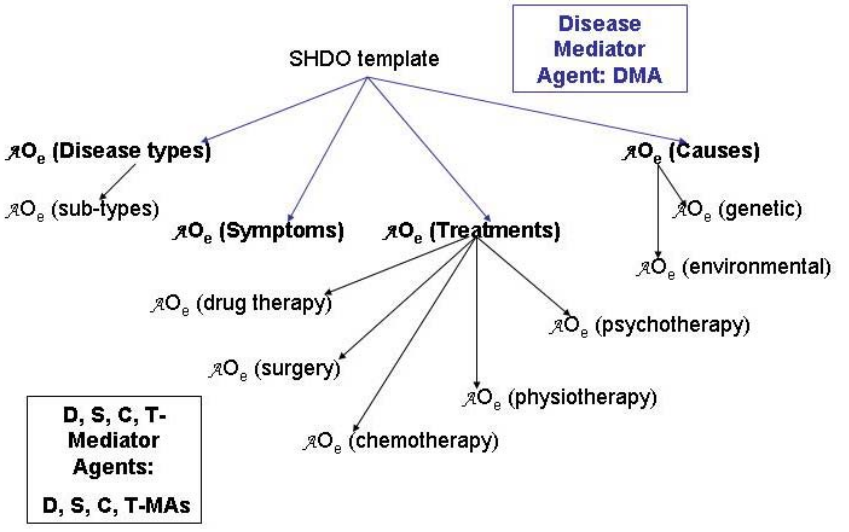


Fig. 11.6 Assigning tasks to different holarchy agents

MAs also receive information regarding other subontologies templates, but it needs to process only its own. For example, Causes-Mediator Agent (C-MA) receives all four subontology templates but it further processes only the $\mathcal{A}$Oe (causes) template. The $\mathcal{A}$Oe (subontology) is further partitioned by MAs into sub-subontologies and parts of them. For example, $\mathcal{A}$Oe (drug therapy), $\mathcal{A}$Oe (surgery), $\mathcal{A}$Oe (chemotherapy), $\mathcal{A}$Oe (physiotherapy) and $\mathcal{A}$Oe (psychotherapy) in the case of Treatments-Mediator Agent (T-MA), and $\mathcal{A}$Oe (genetic) and $\mathcal{A}$Oe (environmental) in the case of Causes-Mediator Agent (C-MA). The resulting sub-subtemplates are in the form of $\mathcal{A}$Oe (sub-subontology).

The process is shown in Figure 11.6. According to the sub-subtemplates, tasks are assigned to Specialist Agents and/or Representative Agents. SA and RA also received information regarding other parts of the SHDO template and are aware of tasks assigned to other agents.

During the information assembly process, the DMA on the higher level and MAs on the lower level of the hierarchy function as Smart agent. We mentioned that they correspond to the first level Smart agent and the second level Smart agent respectively.

MAs combine information coming from RAs and SAs in the form of $\mathcal{A}$Oe (sub-subontology), and present it to DMA as a single unit in the form of $\mathcal{A}$Oe (subontology). Four different $\mathcal{A}$Oe of the four different subontologies are combined by DMA: $\mathcal{A}$Oe disease types) $\cup$ $\mathcal{A}$Oe symptoms) $\cup$ $\mathcal{A}$Oe (causes) $\cup$ $\mathcal{A}$Oe (treatments) = $\mathcal{S}$O. This step results in Specific Human Disease Ontology (SHDO) that after final information reorganization is presented to the user.

11.10 Construction of Individual Agents of the Human Disease Information Retrieval System

11.10.1 Identify Required Agents' Components

An agent interacts with users of the system via the agent interface and constructs the SHDO template (from GHDO) according to the user's requests. As all agents of the system interact and communicate with each other, they all have an agent interface.

Procedural knowledge of an agent is also constructed as ontology. It contains information regarding problem solving and the goal prioritization method. For example, the SA needs to know how to compare all information delivered by RAs and select significant information that needs to be incorporated into the SHDO template. In the example from Table 11.1, SA needed to choose the smallest DNA regions to be passed over to the MA and further incorporated into the SHDO template.

Communication component processes messages. Messages are written in ontology specification language. Agents need to speak the same language as they are exchanging information and cooperatively working towards the same goal. All agents within the system agree and commit to this common language.

Cooperative knowledge component, also constructed as ontology, enables agents to cooperate with each other. An agent needs to know how to negotiate with other agents, coordinate its own actions with the actions of other agents, and make various decisions. For example, an agent needs to decide for itself whether it is appropriate, in a specific situation, to contact other agents. These decisions are made on the basis of information regarding the organization of agents within the system and on the basis of information regarding the functions of these agents within the system. Agents that need to be contacted can be from the same, higher or lower levels within the holarchy. For instance, the Mediator Agent (MA) needs to know how and when to contact a specific Specialist Agent (SA), and whether it is appropriate to contact it in regard to the overall task that needs to be accomplished.

Task knowledge of the agents, written as ontology, contains knowledge regarding tasks assigned to that particular agent. This knowledge enables an agent to decompose its task into smaller tasks when needed and perform atomic actions. Note that task knowledge regarding tasks assigned to other agents of the system is in cooperative knowledge module. The activation of the cooperative knowledge and task knowledge module is connected as all agents within the system are cooperatively working towards the same goal. For example, Disease Mediator Agent (DMA) needs to decompose overall task (SHDO template) into four parts and assign those subtasks to the corresponding MAs. Through the decomposition process, part of this task is assigned to each agent. An agent needs to be aware of tasks assigned to other agents and to contact them if needed.

An agent also relies on past experiences. These files are stored in the history files component that is also written as ontology. Files are stored according to

disease name. The reason for this is that one SHDO requested by one user may contain information partly covering four different GHDO 'dimensions'. Another SHDO queried for the same disease by another user, may complement this information by covering other 'dimensions' or extending existing 'dimensions'. Covering as much as possible of the information requested by different users, the latest version of the SHDO regarding a particular disease is saved in the history files module. Note that in most cases, the SHDO last requested covers only a part of the SHDO from the history files. This is illustrated in Figure 11.7. If a difference is found between the SHDO last requested and SHDO from history files, the new SHDO should be checked for consistency. If the difference is consistent, the new SHDO should be used to update the corresponding part of SHDO file. In Figure 11.7, 'f' needed to be replaced by 'l'. This latest version of the SHDO is stored in the history file module and used next time for matching. We call this kind of SHDO update an 'update own files' update. We also have an 'update suggestion' update. This is specifically required in the case where the environment of agents is dynamic and changes within this environment occur constantly. Here, an agent can be designed to detect these kinds of updates and suggests an update of its environment knowledge module to system designers.

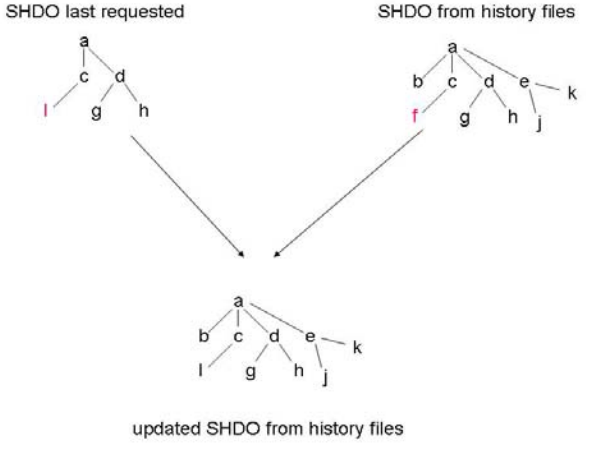


Fig. 11.7 Updating SHDO from history files

Domain knowledge of an agent, written as ontology, contains general information regarding human diseases. In our model, this is Generic Human Disease Ontology (GHDO). A user specifies which part of this knowledge is of interest to him/her. The Interface agents then construct the SHDO template from GHDO on the basis of user's query.

Environment knowledge is also constructed as ontology. It contains information regarding information content and structure within each of the different databases assigned to different Information agents. Most of the databases within the biomedical community are autonomous, heterogeneous and different from

each other. It would be very beneficial if information content of these information resources would be described using ontology (annotation of information resources). This would enable agents to 'understand' the content of these information resources and retrieve relevant information. Some of the information resources have already begun annotation. This is important to the biomedical community as it enables other services to efficiently access and share information contained within these information resources.

We can model the goal-driven aspect of the agent by a Composite Structure Diagram with Parts, and Ports. Each part represents a distinct area of processing within the agent. Each port represents a different role played by the agent (Wongthongtham et al. 2008).

The <<Agent>> stereotype based on the Composite Structure Diagram can be used to model the Smart agent. The <<Agent>> stereotype must have a name, at least a Controller part which controls the efforts of the Agent to achieve a goal, and at least one port, which relates to its playing a role.

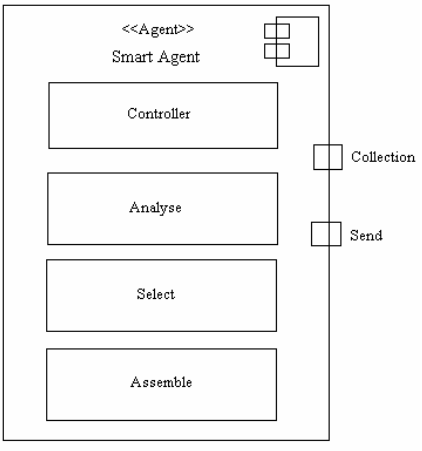


Fig. 11.8 Composite Structure Diagram representing Goal-driven characteristic of the Smart agent

We use a Composite Structure Diagram to represent the goal-driven nature of an agent. In the case of the Smart agent shown in Figure 11.8, we have two ports which correspond to two different roles of this agent, and four parts which show distinct areas of information processing within the agent. Note that the same two ports (Collection and Send) that were present in the sequence diagram (see Figure 11.2) are also present here. Each of the ports is a construct which enables the agent to interact with other agents, namely Information agent and Interface agent. Next to the Controller object, the Smart agent also has Analyze, Select and Assembly objects. The Smart agent collect the retrieved information through the Collection port. This information is analyzed, the relevant information is selected and assembled into the SHDO template resulting in the SHDO. The SHDO is sent to the Interface agent via Send port.

11.10.2 Construct Various Agents

In Chapter 8, we discussed that a variety of agents within a multi-agent system can be achieved in three different ways: (1) different components that are used to construct different agents can be the same, but the content of the components may be different for different agents; (2) content of components used to construct different agents may be the same, but different agents are constructed through a different combination of used components; and (3) different agents differ in the combination of components used to construct them and in the content of these components.

In order to design different agents of a GHDO-based multi-agent system, we will adopt the third principle.

11.11 Protect the Human Disease Information Retrieval System by Implementing Security Requirements

11.11.1 Identify Security Requirements

We need to consider the environment in which the GHMS will be situated in order to identify security requirements. Various agents located over different biomedical databases search for specific information when requested by a user. Their environment is the database environment of the biomedical community. The characteristics of this environment are that it is inaccessible, non-deterministic, dynamic and continuous (Wooldridge 2002).

In such an environment, all the security properties of authentication, availability, confidentiality, non repudiation and integrity (Mouratidis et al. 2003) as well as compliance, service and dedication should be taken into consideration.

11.11.2 Implement the Security Requirements

Because the security of a multi-agent system is closely linked to functions that agents have within the system, we will discuss system security according to the functions of different agents. We discuss Interface, Manager, Information and Smart agents in the singular to represent all agents having the same function.

Authentication: Proving the identity of an agent

Each agent needs to prove its identity to another agent when communicating and exchanging information. The whole system operates as a sequence of different actions of different agents: Interface agent (Ag1) → Manager agents (Ag2) → Information agents (Ag3) → Smart agents (Ag4) → Interface agent (Ag1). The Interface and Information agents operate inside as well as outside the system and are more critical than other agents with respect to security. The Manager agent and Smart agent operate only with the agents of the multi-agent system.

The "communication" module of the agents processes incoming identification messages. On the basis of validity of identification, the agent will either continue performing its action or its action will be cancelled.

The "procedural knowledge" component of an agent contains information regarding the problem solving and goal prioritization method. This component takes two inputs: the task that needs to be performed and information regarding the security of performing this task. On the basis of these two inputs, the agent makes the decision whether to proceed further or to cancel its action (see Figure 8.7).

This decision is communicated to security agents. These agents are specialized in the security of the system. Security agents validate the decision made by the individual agent with regard to continuing with or cancelling its action.

Two original inputs, the final decision and the consequence of this decision are stored in the agent's "history files" for its own reference. This becomes the third input on the basis of which the decision will be made next time this agent is found in a similar situation.

If the decision made by an agent had negative consequences, results are sent further to the "update suggestion" component. Changes in values given to the security of performing the action will be suggested.

Availability: Guaranteeing the accessibility and usability of information and resources to authorized agents

Only Information agents of the multi-agent system have access to information resources. These agents need to prove their identity to external agents of the information resources. After their identification validation, they are able to access all, or part of, an information resource.

Confidentiality: Information is accessible only to authorized agents and inaccessible to others

Only those agents that can prove their authority over information have access to information. This is true for inside as well as for outside the system. Information agents need to prove their identity to external agents of the information resources. Also, the Smart agent needs to prove its identity to the Information agents in order to be able to receive information from Information agents. The Interface agent needs to prove its identity to the Smart agent in order to be able to receive the SHDO after data selection and assembly.

Non-repudiation: Confirming the involvement of an agent in certain communication

Not only do Smart agents need to show their identity to Information agents, but Information agents also need to prove to Smart agents that the information they provide is from the correct source. A similar process applies to other agents of the system.

The Smart agent needs to provide evidence to the Interface agent that all the information contained within the SHDO is trustworthy and is provided by Information agents.

Integrity: Assuring that the information remains unmodified from source entity to destination entity

The query formed by the Interface agent may not be modified by the Manager agent and subqueries assigned by the Manager agent to Information agents may not be modified by Information agents.

The integrity property needs to be guaranteed to Information agents by agents of information resources. Retrieved information should not lose its meaning once found outside the information resource.

The Interface agent presents the assembled SHDO to the user and may not modify or change the meaning of the information originally assembled by Smart agents.

Compliance: acting in accordance with the given set of regulations and standards.

Manager, Information and Smart agents must be given clear instructions on how they should perform and carry out their tasks within the system. These agents must comply with the given set of laws, regulations and standards. Following instructions set up by their designer(s) will enable the agents to be successful in performing their actions for their own benefit and for the benefit of the whole multi-agent system. For example, Information agents must make information available to authorized Smart agents when requested.

Service: agents need to serve one another for mutually beneficial purposes.

Manager, Information and Smart agents need to serve one another. For example, if some Information agents are under malicious attack, other Information, Manager, and Smart agents need to make their best efforts to protect the attacked Information agents and the whole multi-agent system.

Dedication: complete commitment of the agents to the multi-agent system goal and purpose.

Manager, Information and Smart agents are working on different aspects of the overall goal; in this case, efficient retrieval of the information about human disease. All agents must commit to this goal and purpose, and function in unity for the maximum success.

11.12 Examples of Use of the Intelligent Human Disease Information Retrieval System

The importance of ontology and agent-based technologies has been recognized within the biomedical community (Marchetti and Lanzola 2001). The Onto-Agent methodology can be used to design an ontological model as well as a multi-agent system (based on the designed ontological model) for the purpose of intelligent information retrieval of knowledge regarding human diseases.

The system can be used to support scientists in gathering information on highly specific research topics and to allow users on a world-wide basis to intelligently access new scientific information much more quickly. The knowledge domain can be easier understood and new knowledge discovered. Shared knowledge improves research efficiency and effectiveness. By being aware of research done by other teams, scientists will be able to progress much faster, coming closer to answers for their common problems. In particular, being able to have shared understanding of concepts facilitates information and knowledge exchange. Having an overview of available information, it is much easier to identify research areas where more information and more examination are needed.

Some advantages of such system are that it:

- creates a unifying framework for human disease knowledge which is of great importance for medical researchers, physicians and patients. Embracing all information about human diseases makes it possible to fully understand the knowledge domain.
- enables dynamic knowledge discovery, especially in cases where there is uncertainty about the type of disease or it is not easily diagnosable. This system may also help in early identification of new diseases such as SARS.
- improves research efficiency and effectiveness, as it helps to avoid unnecessary redundancy in performing the same experiments, such as examination of the same region of a DNA sequence and/or determination of part of DNA sequence that needs to be further examined in order to find a gene mutation causing a disease.
- constructs data patterns which combine, for instance, different genetic and environmental causes and different disease types. This helps to sort out the exact combinations of genetic and environmental factors involved, as well as their individual influences on a specific complex disease type. Once factor(s) responsible for a human disease are known, it would be much easier for doctors to diagnose, treat and possibly prevent that disease.

We consider below a number of possible scenarios from the biomedical area in which such a system would be helpful. Researchers are constantly searching for, and adding more information to, the existing pool of knowledge. Physicians are directly in contact with patients and are using all available information to help and treat their patients. Especially when a new disease starts spreading to epidemic proportions, researchers and physicians are strongly connected because they are working towards the same goal, but on different knowledge levels.

11.12.1 Example 1: Help Physician to Identify Disease

If a physician queries a system, she/he will be mainly interested in symptoms and possible treatments of a particular disease.

There are some exceptions to this rule such as in the case of a new disease being encountered by a physician. Namely, a physician may have a patient showing some symptoms of a disease but he may not be able to identify the disease. By entering symptoms of a disease into the system, a doctor may be able to retrieve some information regarding that disease.

It is also possible that two different diseases are manifesting the same or similar symptoms so that different outputs are possible. This is shown in Figure 11.9. In such cases, it may be useful to look for some significance in the causes of the disease as we explain in the sequel.

- Use case 1a: disease causes are not known

On the basis of key symptoms, the doctor will choose one (set of) disease(s). This disease becomes the doctor's working hypothesis, her/his most likely choice. The doctor then starts to gather evidence in support of the working hypothesis, always keeping in mind the set of alternative hypotheses.

Such a process relies on all kinds of information such as information that is gained by interrogating the patient or by conducting necessary (physical or instrument- or implement-based) examinations and tests.

It will be assumed that all this data and information will be stored in medical records for patients and that all necessary/available medical information about a patient is kept in exactly one comprehensive ontology-based computer-readable patient record. This enables the patient record to be further processed by the system.

- Use case 1b: disease cause is known, e.g. a gene mutation.

From Figure 11.9 it follows that in case of disease X, gene X is mutated and this causes disease X. And disease Y is caused by the mutation of gene Y. Even though the two different diseases are caused by the mutation of a different gene, they can still show similarities in their symptoms. The physician can screen the patient's DNA to check whether gene X or gene Y is mutated. If mutation is found in gene X, the patient has disease X; and if gene Y is mutated, the patient suffers from disease Y.

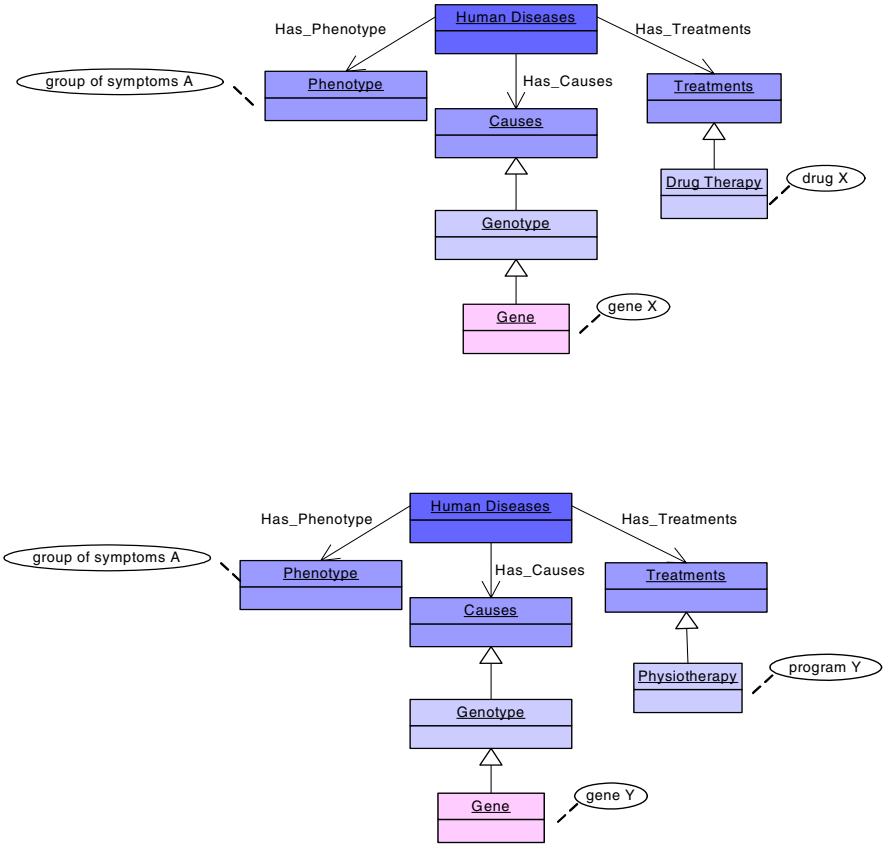


Fig. 11.9 Two different diseases caused by mutations of different genes and treated by different methods showing same symptoms

Only when the patient has been correctly diagnosed, can the physician consider possible treatments. In Figure 11.9, we see that disease X is treated by drug X. Disease Y can be simply treated by physiotherapy. Therefore, our information system also reduces risks of misdiagnosis.

11.12.2 Example 2: Support Physician to Choose Disease Treatments

It is common to have more than one treatment possible for a particular disease. A medical professional might consult the information system in order to do a one-component search (treatments).

Figure 11.10 provides an example where three different drugs are being used to target the same disease. Different drugs have different active ingredients and show differences in their characteristics. Some drugs are herbal; others are synthetic. Usually, herbal drugs are not as effective as synthetic drugs and do not give results as quickly. However, in most cases, healing with herbal drugs is more effective as long-term irreversible healing. Another advantage is that most herbal drugs show minor or no side effects.

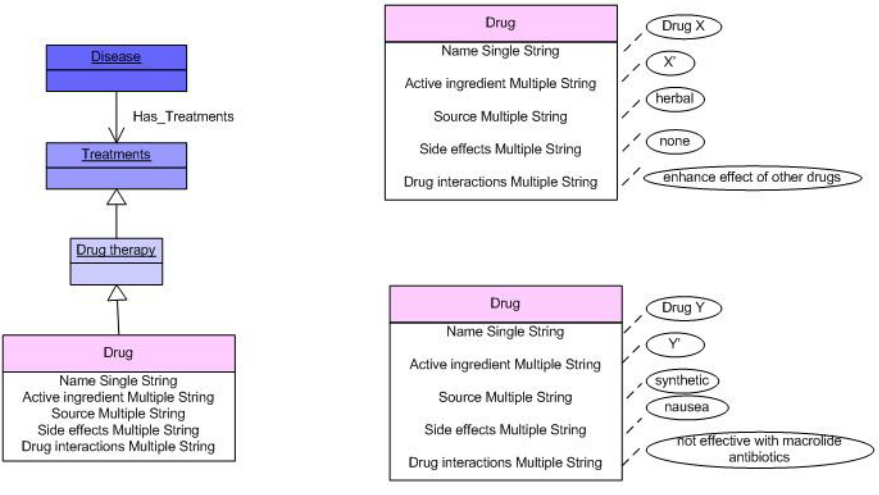


Fig. 11.10 Different drugs target same disease

A physician may wish to look at all available options before choosing one. Choosing medication is also an individual thing because not all people respond in the same way to the same medications.

11.12.3 Example 3: Help Patients and General Public to Prevent a Disease

Some complex diseases, such as manic-depression, are caused by different factors (genetic and environmental). All the different factors that cause this illness have

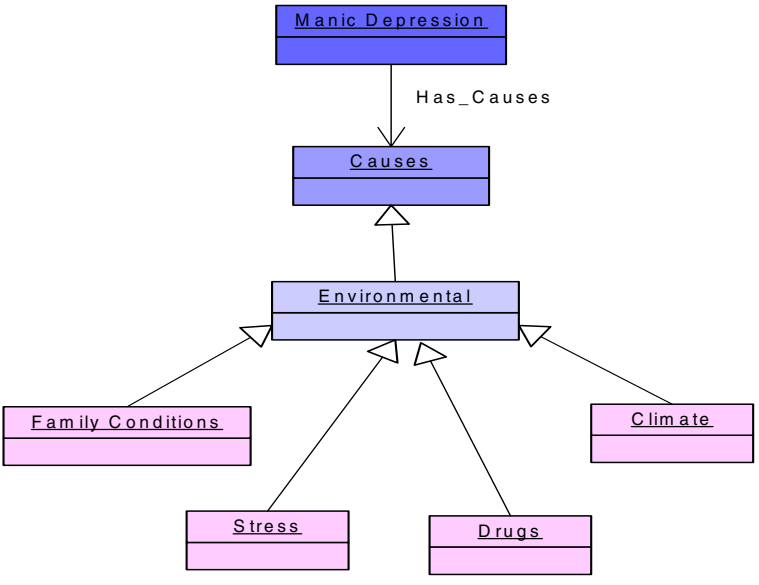


Fig. 11.11 Environmental causes of manic depression

not yet been identified. Also, the individual role of each causal factor in the onset of manic-depression is not yet known.

A person may know that manic-depression occurs frequently in his/her family line. It may be possible for him/her to inherit genetic cause(s). A patient cannot change his/her DNA, but he/she can influence the effects of environment on him/herself. In order to prevent manic-depression, the patient may wish to use the information system to retrieve information regarding environmental factors causing, or contributing to, manic-depression. In this case, the patient may alter his/her environment and reduce the risk of developing manic-depression. In Figure 11.11, we show that environmental factors such as climate, drug intake, stress and adverse family conditions may have an effect on the onset of this disease.

11.12.4 Example 4: Help Medical Researchers to Identify Disease Causes

Medical researchers may use the system to retrieve information regarding disease causes, such as in the case of manic-depression (Figure 11.12).

By querying the system and obtaining relevant information systematically represented, a researcher may be able to identify some regions of interest in the DNA sequence such as regions 2p13-16, 10q21-24, 12q23-24, 17q11-12 and Xq24-26 on chromosomes 2, 10, 12, 17 and X respectively (Craddock and Jones 2001; Liu et al. 2003).

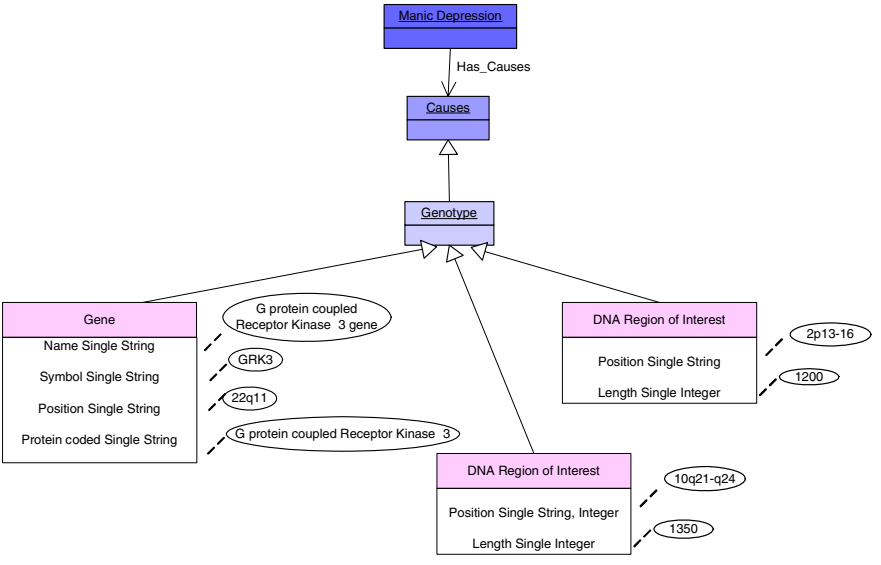


Fig. 11.12 Genetic causes of manic-depression (current research)

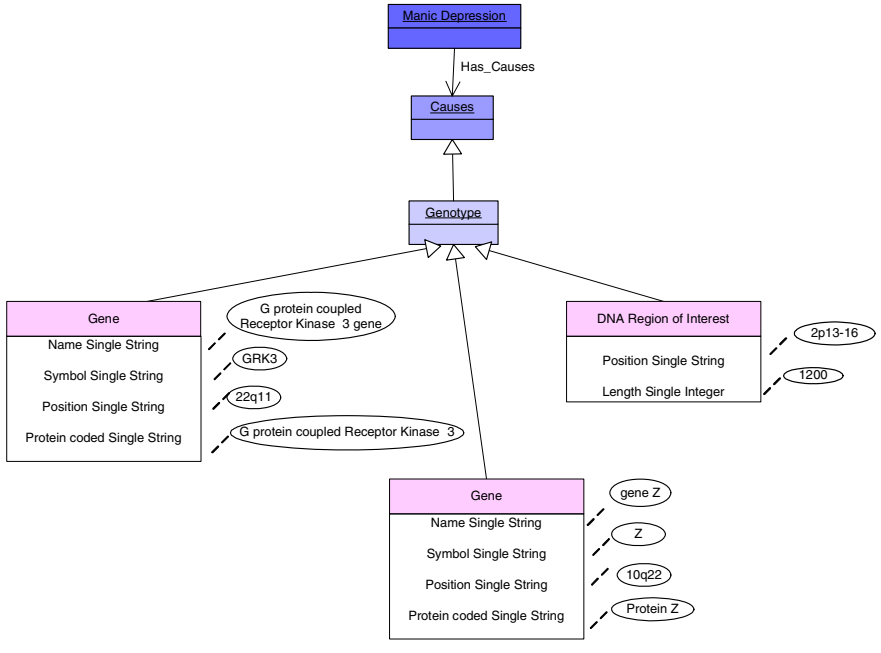


Fig. 11.13 Genetic causes of manic-depression, future research if gene of interest found on chromosome X

These DNA regions need to be further examined in order to find a gene and a mutation inside that gene responsible for the onset of this disease. For example, mutation of gene GRK3, which is positioned on 22q11, is responsible for development of manic-depression in 10% of disease cases (Barrett et al. 2003).

Because of the agreed semantics in shared ontology, it will be easier for the next person to continue research in the same direction and possibly locate the gene of interest. For example, further research on 2p13-16 DNA region of interest, may allow scientist to narrow down the region to, for example, 2p14-15 and finally precisely locate the gene of interest on 2p15. This aspect of cooperation between different teams also increases productivity, and saves time and resources.

Given the length of the DNA sequence, it is obviously much easier for a researcher to target a specific area of a chromosome such as 10q21-q24 than the whole of chromosome 10. If a new Gene Z is found on 10q21-q24, for example on position 10q22, our model will have one less instance of the term 'DNA region of interest' and one more instance of the term 'Gene' (see Figure 11.13). Note that this example of Gene Z is given just for illustrative purposes.

11.12.5 Example 5: Help Medical Researcher Study Complex Diseases

As we previously mentioned, complex diseases are characterized by different types, such as depression, manic-depression and schizophrenia types of psychiatric diseases. Also, different environmental factors such as stress, adverse family conditions, climate etc. are, together with different genetic factors (e.g. gene mutation), responsible for the onset of such diseases (Verheyen et al. 1997). All different disease causing factors have not yet been identified. Also, the influence of each identified factor on the development of disease is not yet known.

Let e1, e2, e3 and e4 be the environmental causes, g1, g2, g3, g4 and g5 be the genetic causes and t1, and t2, and t3 be different types of a complex disease. We worked out 4 different possible hypotheses in this case.

Hypothesis 1:

The same causes influence the phenotype (observable characteristics of an organism) differently.

Different influences can be expressed by different percentages and different influences explain different types of these diseases.

For example, the same causes e1, e2 and g1 are responsible for the types t1, t2 and t3 but have different influences on the phenotype. This is shown in Table 11.2.

Table 11.2 Same factors, different influences

Types/Factors	e1	e2	g1
t1	K%	L%	100 – (K + L)%
t2	M%	N%	100 – (M + N)%
t3	X%	Y%	100-(X + Y)%

In this case, different diseases types result from the different individual influences of the same factors.

Hypothesis 2:
Different factors influence the phenotype equally.

For example, type t1 is influenced by factors e1, e2 and g1; type t2 by factors e3, g2 and g3; type t3 by factors e4, g4 and g5. This is presented in Table 11.3.

Table 11.3 Different factors, same influences

Factors / Types	e1	e2	e3	e4	g1	g2	g3	g4	g5
t1	X%	Y%			100-(X+Y)%				
t2			X%			Y%	100-(X+Y)%		
t3				X%				Y%	100-(X+Y)%

In this case, different types result from different factors associated with each type.

Hypothesis 3:
Different factors influence the phenotype differently.

This is shown in Table 11.4.

In this case, different types are a result of both different causes, and different individual influences of these causes, on the development of that specific disease type.

Table 11.4 Different factors, different influences

Factors / Types	e1	e2	e3	e4	g1	g2	g3	g4	g5
t1	X%	Y%			100-(X+Y)%				
t2			K%			L%	100-(K+L)%		
t3				M%				M%	100-(M+N)%

Hypothesis 4:
All different factors have a cumulative effect and there is a threshold value that needs to be reached before the disease develops.

Note that it is possible for individual influences of different factors to have the same value. For instance, it is possible that X = Y.

In Table 11.5, we give an example where the development of diseases type t1 requires the presence of factors e1, e2, g3 and g5; of disease type t2, presence of e1, e2, g1 ad g4 factors; and of disease type 3, presence of e3, e4, g2 and g5 factors. If one of the required factors is missing, the corresponding disease type will not develop.

Table 11.5 Cumulative effect of different factors

Factors	e1	e2	e3	e4	g1	g2	g3	g4	g5
Influences	M%	N%	P%	R%	S%	X%	Y%	Y%	Z%
t1	x			x			x		x
t2	x	x			x			x	
t3			x	x		x			x

We illustrated here the importance of an intelligent information system in the information retrieval process for the purpose of testing hypothesis(es). It may be possible for medical researchers to retrieve all information regarding genetic and environmental factors as well as different types of a complex disease. Having all available information structurally organized, makes it much easier to identify relationships between the data, identify possible dependencies between different factors and prove one (or more) hypothesis(es).

11.13 Conclusion

In this chapter, we illustrated how to use the Onto-Agents Methodology to develop a Generic Human Disease Ontology (GHDO) and a multi-agent system that can use the designed GHDO for intelligent information retrieval. We also discussed numerous advantages of this system, and illustrated how physicians, medical researchers and patients can all make use of this system.

References

1. Barrett, T.B., Hauger, R.L., Kennedy, J.L., Sadovnick, A.D., Remick, R.A., Keck, P.E., McElroy, S.L., Alexander, M., Shaw, S.H., Kelsoe, J.R.: Evidence that a single nucleotide polymorphism in the promoter of the G protein receptor kinase 3 gene is associated with bipolar disorder. Molecular Psychiatry 8, 546–557 (2003)
2. Bodenreider, O.: The Unified Medical language System (UMLS): integrating biomedical terminology. Nucleic Acids Res. 32, 267–270 (2004)
3. Ceusters, W., Martens, P., Dhaen, C., Terzic, B.: LinkFactory: an Advanced Formal Ontology Management System. Interactive tools for Knowledge Capture, KCAP-2001 (2001), `http://www.isi.edu/~blythe/kcap-interaction/papers/LinKFactoryXWhiteXPaperXfinal.doc` (retrieved: May 26, 2004)
4. Ceusters, W., Smith, B., Kumar, A., Dhaen, C.: Ontology-Based Error Detection in SNOMED-CT. In: Medinfo 2004, pp. 482–486 (2004)
5. Craddock, N., Jones, I.: Molecular genetics of bipolar disorder. The British Journal of Psychiatry 176, 128–133 (2001)

6. Deray, T., Verheyden, P.: Towards a semantic integration of medical relational databases by using ontologies: a case study. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 137–150. Springer, Heidelberg (2003)
7. Mouratidis, H., Giorgini, P., Manson, G.A.: Modelling secure multiagent systems. In: The second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), pp. 859–866 (2003)
8. Hadzic, M., Ulieru, M., Chang, E.: Soft Computing agents for e-Health Applied to the Research and Control of Unknown Diseases. Information Sciences 176, 1190–1214 (2006)
9. Jarrar, M., Demey, J., Meersman, R.: On Reusing Conceptual Data Modeling for Ontology Engineering. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 185–207. Springer, Heidelberg (2003)
10. Jarrar, M., Meersman, R.: Formal Ontology Engineering in the DOGMA Approach. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 1238–1254. Springer, Heidelberg (2002)
11. Liu, J., Juo, S.H., Dewan, A., Grunn, A., Tong, X., Brito, M., Park, N., Loth, J.E., Kanyas, K., Lerer, B., Endicott, J., Penchaszadeh, G., Knowles, J.A., Ott, J., Gilliam, T.C., Baron, M.: Evidence for a putative bipolar disorder locus on 2p13-16 and other potential loci on 4q31, 7q34, 8q13, 9q31, 10q21-24, 13q32, 14q21 and 17q11-12. Mol. Psychiatry 8, 333–342 (2003)
12. Marchetti, D., Lanzola, G., Stefanelli, M.: An AI-Based Approach to Support Communication in Health Care Organizations. In: Quaglini, S., Barahona, P., Andreassen, S. (eds.) AIME 2001. LNCS (LNAI), vol. 2101, pp. 384–394. Springer, Heidelberg (2001)
13. Spyns, P.: Object Role Modelling for Ontology Engineering in the DOGMA Framework. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 710–719. Springer, Heidelberg (2005)
14. Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A., Brass, A.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. Bioinformatics, 184–186 (2002)
15. Verheyen, G., Raeymaekers, P., Van Broeckhoven, C.: De genetica van stemmingsstoornissen. Cns Review 2, 7–12 (1997)
16. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley and Sons, Chichester (2002)
17. Wongthongtham, P., Dillon, D., Dillon, T.S., Chang, E.J.: Use of UML 2.1 to model multi-agent systems based on a goal-driven software engineering ontology. In: Proceedings of the 4th international conference on semantics, knowledge and grid (SKG 2008), China (2008)
18. Wouters, C., Dillon, T.S., Rahayu, J.W., Chang, E., Meersman, R.: Ontologies on the MOVE. In: Lee, Y., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 812–823. Springer, Heidelberg (2004)

Chapter 12
CASE STUDY II: Ontology-Based Multi-Agent System for Software Engineering Studies

12.1 Introduction

In this chapter, we explore a case study in the domain of software engineering. We will go through the development of Software Engineering Ontology (SE Ontology) and a multi-agent system in which agents are interacting and mediating with the SE Ontology. The purpose of such development is for multi-site software development as a communication framework.

In Section 2 we discuss on software engineering domain including purpose of the SE Ontology, the context of software engineering domain knowledge, communities of users and agents for which the SE Ontology is being used, and applications on the SE Ontology. In Section 3, we discuss how we design the formal SE Ontology. We then evaluate the SE Ontology in Section 4.

In Section 5, we identify and describe different type of agents according to functionalities in the domain. We discuss the need for the SE Ontology to support agents' intelligence in Section 6 and describe agents' collaboration in Section 7. The construction of each individual agent is explained in Section 8. We present examples of the practical uses in Section 9. We conclude the chapter in Section 10.

12.2 Generalization and Conceptualisation of the Software Engineering Domain

In this section, we discuss respectively the following points:

- Purpose of the SE Ontology
- Context of software engineering domain knowledge
- Community of users and agents for which the SE Ontology is used
- Applications of the SE Ontology

12.2.1 Purpose of the Software Engineering Ontology

With the advent of the Internet, software development has increasingly focused on the Internet which enables a multi-site environment that allows multiple teams residing across cities, regions, or countries to work together in a networked distributed fashion to develop the software. A realisation of the advantages of

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 217–270.
springerlink.com

multi-site software development has led to major corporations moving their software development to countries where employees are on comparatively lower wages. It is this imperative of financial gain that drives people and businesses to multi-site development and the Internet which facilitates it.

However, the globalization of software development means that the problems of multi-site development are increasing. Team members who carry out the tasks and activities, team leaders who control the tasks and activities, and managers who manage the project and leaders, may or may not be at the same site in a multi-site environment. These people often have never met face-to-face, have different cultural and educational backgrounds, interpret methods in different ways, etc.

Additionally, software engineering training and practice are quite different between cities and countries. It can be difficult to communicate between teams and among team members, if strict software engineering principles and discipline are not understood and followed. The inconsistency in presentation, documentation, design and diagrams could prevent access by other teams or members. Sometimes, these issues (such as a diagram using non-standard notations) are ignored because they are not understood and no-one asks for clarification.

Despite this, software engineering has a commonly understood body of knowledge and is an easily learnt subject that includes some of the latest technology and methodology which is easily adopted. However, different teams could be referring to different texts on software engineering. Teams or team members use a particular text as their own individual guide, and when they communicate, their own knowledge base and terminology is different from that of others. Often, the issues raised or debated are related to inconsistency in understanding software engineering theories and practice.

Consequently, several practical problems arise and underlying issues need to be explored. Communication is the real challenge and there is the need to develop different ways of tackling this through better communication and conferencing systems and through systems that help resolve differences between the teams. Ontology is an important part of developing a shared understanding across a project. As Davenport and Prusak (Davenport and Prusak 1998) mentioned, people cannot share knowledge if they do not speak a common language. Representing software engineering knowledge in the form of ontology is helping to clear up ambiguities in the terms used in the context of software engineering.

12.2.2 The Context of Software Engineering Domain Knowledge

Software engineering is an engineering discipline focusing on the development of high quality software applications. It includes not only technologies and practices, but also new techniques and methods needed to control the complexity inherent in large software systems. The development of software engineering has markedly improved software applications so that they are now formal, under budget, reliable, maintainable, and deliverable. There are many kinds of systems and organisations which require a broad diversity of approaches to software development. Nevertheless, fundamental notions of process and system organisation basically underlie the essence of software engineering. Software

engineering concerns all aspects of software production from the early stages of specification through to management of software systems.

By default, all computer programmes have a fundamental ontology consisting of a standard library in a programming language, or files in accessible file systems or some other list of 'what exists'. However, the representations are sometimes poor for certain problem domains, so more specialised schema must be created to make the information useful and for this ontology is utilised.

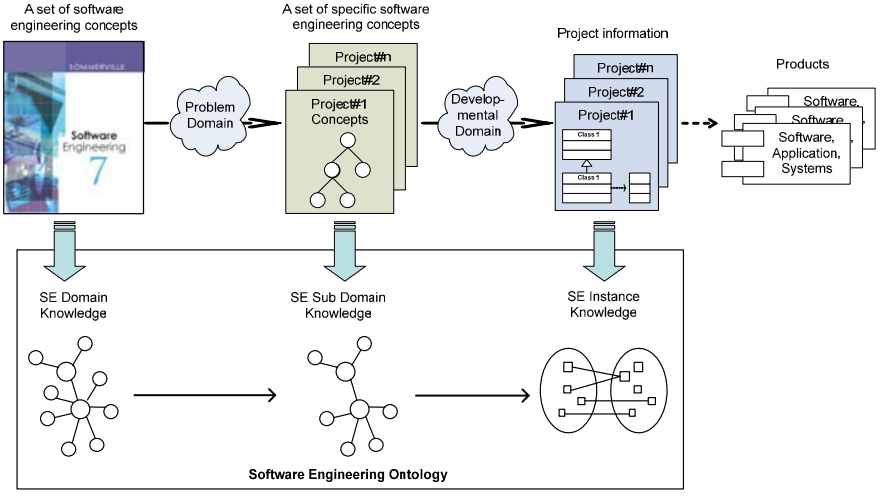


Fig. 12.1 Schematic overview of software engineering knowledge representation

An abstract view of representing the software engineering knowledge is shown in Figure 12.1. The whole set of software engineering concepts representing software engineering domain knowledge is captured in ontology. Based on a particular problem domain, a project or a particular software development probably uses only part of the whole set of software engineering concepts. The specific software engineering concepts used for the particular software development project representing software engineering sub-domain knowledge are captured in ontology. The generic software engineering knowledge represents all software engineering concepts, while specific software engineering knowledge represents some concepts of software engineering for the particular problem domain. For example, if a project uses purely object-oriented methodology, then the concept of a data flow diagram may not be necessarily included in specific concepts. Instead, it includes concepts like class diagram, activity diagram and so on. For each project in the developmental domain, there exists project information or actual data including project agreements and project understanding. The project information in particular meets a specific project need and is needed together with the software engineering knowledge to define instance knowledge in ontology. Note that the domain knowledge is separate from instance knowledge. The

instance knowledge varies depending on its use for a particular project. The domain knowledge is quite definite, while the instance knowledge is particular to a problem domain and developmental domain in a project. Once all domain knowledge, sub-domain knowledge and instance knowledge are captured in ontology, it is available for sharing among software engineers through the internet. All team members, regardless of where they are, can query the semantically linked project information and use it as the common communication and knowledge basis for raising discussion matters, questions, analysing problems, proposing revisions or designing solutions and the like.

Software engineering domain knowledge constructs should be sought in ontology, a well-founded model of reality. Ontology is used to analyse the meaning of common conceptual modelling constructs which accurately reflect the world. The notion of a concrete thing applies to what software engineers perceive based on software engineering domain knowledge. In this light, the notion of ontology is a solution for software engineering knowledge representation.

When the knowledge of the software engineering domain is represented in a declarative formalism, the set of software engineering concepts, their relations and their constraints are reflected in the representation which represents knowledge. Thus, the SE Ontology can be defined by using a set of software engineering representational terms. Then a conclusion can be determined from the knowledge of what is.

In order for the software engineering domain knowledge to be shared amongst software engineers or applications, agreement must exist on the topics about which information is being communicated. The issue of ontological commitment is described as the agreement about concepts and relationships between those concepts within ontology. When the SE Ontology is committed, it means agreement exists with respect to the semantics of the concepts and relationships represented. Therefore, in order to know what the software engineers are talking about, agreement is arrived at. The software engineers agree to share knowledge in a coherent and consistent manner.

The SE Ontology is organised by concepts, not words. This is in order to recognise and avoid potential logical ambiguities. The SE Ontology has been developed for communication purposes, and as such, it could differ greatly from other ontologies developed for different purposes. The main purpose of the SE Ontology is to enable communication between computer systems or software engineers in order to understand common software engineering knowledge and to perform certain types of computations. The key ingredients that make up the SE Ontology are a vocabulary of basic software engineering terms and a precise specification of what those terms mean. For software engineers or computer systems, different interpretations in different contexts can make the meaning of terms confusing and ambiguous, but a coherent terminology adds clarity and facilitates a better understanding. SE Ontology has specific instances for the corresponding software engineering concepts. These instances contain the actual data being queried in the knowledge-based applications. The SE Ontology includes the set of actual data or instances of the concepts and assertions that the instances are related to, each according to the specific relations in the concepts.

The main purpose of the SE Ontology is to enable knowledge sharing and reuse. In this sense, the SE Ontology is a specification used for making ontological commitments. In practice, an ontological commitment is an agreement that is consistent and coherent with respect to theory specified by the SE Ontology.

The software engineering knowledge described in the SE Ontology comes from two sources: the software engineering body of knowledge (Bourque et al. 2004) and the software engineering text of Ian Sommerville (Sommerville 2004). Ontology design for software engineering knowledge is the design of the SE Ontology. The objectives of this chapter are not to introduce software engineering, nor to provide a framework for understanding software engineering. Instead, it aims to present an ontology model of the software engineering to represent its knowledge. In order to present the SE Ontology, we utilise the notation description introduced previously. The fundamental knowledge relating to the software engineering is well described in the textbook of software engineering 7th edition by Ian Sommerville (Sommerville 2004) and white paper of the SoftWare Engineering Body Of Knowledge (SWEBOK) by the IEEE (Bourque et al. 2004) upon which the SE Ontology is based. In a multi-site distributed software development environment, SE Ontology is used as a communication framework. Its end-users are software engineers scattered around the globe sharing domain knowledge of software engineering as well as instance knowledge of multi-site project data formed in the SE Ontology. The software engineering domain knowledge presented in SE Ontology covers knowledge areas intended for use in multi-site communications.

12.2.3 Community

In this section, we discuss the community of users. There are different roles for different tasks and purposes during software development forming different groups of users as follows.

Project Manager
The project manager is the administrator who can view, add, delete, and update project information formed in the software engineering ontology and makes the decisions. An agent makes decisions based on a rule base. However, the manager can interrupt and make decisions on any outstanding issues.

Team Leader
The team leader is the person responsible for the whole process of the team. A sound understanding of multi-site software development is important in this role, enabling the team leader to trace team progress or work done by team members. He also can evaluate whether the project will be delivered within time constraints, or whether any changes are needed in the process. The team leader can view, add, delete, and update project information of the team formed in the software engineering ontology. The team leader can only view, but cannot make changes to, the team's project information. However, he can raise issues relating to changes as appropriate. An agent manages issues as they arise.

Analyst
The chief analyst is responsible for the overall software requirements of the project. The chief analyst shares the results of project analysis. The chief analyst has the highest degree of expertise on all software requirement issues when it comes to making decisions about the issues. The chief analyst can view, add, delete, and update project information formed in the software engineering ontology on software requirements. The chief analyst can only view, but cannot make changes to, other team project information such as project information of the design team, implementation team, etc. However, he can raise issues about changes as appropriate. An agent assists the chief analyst to communicate and coordinate with other analysts and team members.

Designer
The chief designer is responsible for the overall software design of the project. The chief designer shares results of the project design. The chief designer has the highest degree of expertise in all software design issues when it comes to making decisions about the issues. The chief designer can view, add, delete, and update project information formed in the software engineering ontology on software design. The chief analyst can only view, but cannot make changes to other team project information such as the project information of the requirement team, implementation team, etc. However, he can raise issues relating to changes as appropriate. An agent assists the chief designer to communicate and coordinate with other designers and team members.

Programmer
The chief programmer is responsible for the overall software construction of the project. The chief programmer shares the results of project implementation. The chief programmer has the highest degree of expertise in all software implementation issues when it comes to making decisions about the issue. The chief programmer can view, add, delete, and update project information formed in the software engineering ontology on software construction. The chief programmer can only view, but cannot make changes to other team project information such as project information of the requirement team, design team, etc. However, he can raise issues relating to changes as appropriate. An agent assists the chief programmer to communicate and coordinate with other programmers and team members.

Tester
Testers verify that the software being developed will meet the requirements of the customer and they record the testing report. Testers may be an independent team or a part of the project team. Testers are then responsible for the overall software testing of the project. The tester shares results of project testing. The tester has the highest expertise point on all software testing issues when it comes to making decisions about the issues. The tester can view, add, delete, and update project information formed in the software engineering ontology on software testing. The tester can only view, but cannot make changes to, other team project information

such as the project information of the requirement team, design team, etc. However, he can raise issues about changes as appropriate. An agent assists the tester to communicate and coordinate with other testers and team members.

12.2.4 Application Based on the Software Engineering Ontology

This section discusses application based on the SE Ontology for multi-site distributed software development. We illustrate how software engineering ontology facilitates the communication framework and allows knowledge sharing through platforms. In particular, this is how the man-machine system interfaces work. The man-machine interactions were designed and developed into four platforms: Knowledge Navigation Platform, Question and Issue Platform, Suggestion and Proposal Platform and Solution Decision Platform.

The platforms pose as a communication framework for multi-site distributed software development. This overcomes the multi-site issue of communications. The disadvantages associated with multi-site communications rather than face-to-face communications in a software development project whose tasks are distributed across remote sites, is a key issue. Platforms are a means of exchanging information with explicit details for the collaboration between multi-site teams. Basically, the platforms which constitute a communication framework, are the place to find answers in a multi-site distributed environment. The Navigation Platform is available at any time for remote team members to seek further more information, to clarify a mutual agreement, and to share information. The Question Platform is where discussions are conducted on multi-site project issues. The Suggestion Platform is used for brainstorming purposes and to discuss issues that have been raised. The Solution Platform assists the project to keep moving forward and here also the project data gets up-to-dated.

The underlying architecture of the communication framework is the software engineering ontology which identifies the domain knowledge of software development along with instance knowledge of project data. Knowledge sharing through the software engineering ontology eliminates misunderstandings, miscommunications, and misinterpretations. Software engineering ontology presents explicit assumptions concerning the objects pertaining to the domain knowledge of software development. A set of objects and interrelations and their constraints renders their agreed meanings and properties. For example, the confusing terms of ‘classes, ‘objects’, and ‘components’ in object-oriented software development can be simplified, and when perceiving them, software engineers agree to recognise their constitutes, their interrelations, and their constraints.

Platforms make things easy to manage, even remotely. Multi-site issues of coordination, co-operation, awareness, and interoperability have been overcome.

Remote team members’ coordination is important, especially in multi-site software development. The Navigation Platform assists team members to be aware of tasks to be, or being, carried out. Sharing their project information through the Navigation Platform makes project tasks explicit; hence, this makes it easy for

others to coordinate the work and be aware of it. For example, if there is an ignorance of tasks, then there can be no sharing of information. Wrong tasks that have been carried out are identified. In this case, the Suggestion Platform is where members can coordinate meetings to discuss task misinterpretation. Therefore, the potential problems, such as two groups overlapping in some work, or other work not being performed due to misinterpretation of the task, are no longer the case.

Instance knowledge is drawn based on a consensus of domain knowledge of software engineering formed in the software engineering ontology underlying the platforms. Hence, commonalities relating to the knowledge are assumed. This specially overcomes different knowledge of team members as coming from diverse backgrounds. Additionally, failing to describe their local context when team members raise project issues or share information and knowledge will not be the case no more. This is because team members are referring to the consensus software engineering domain knowledge formed in the software engineering ontology.

The process of going through navigation, Question, Suggestion, and Solution Platforms is undertaken when there are issues that need authorising and brainstorming from team members in multi-site projects. This produces awareness of issues that have been currently raised (from the Question Platform), the issues that have been clarified (from the Solution Platforms), any task that has been misunderstood (from the Solution Platforms), and an understanding of the reason(s) for a team member not following the project plan (from the Suggestion Platform). Throughout, instance knowledge explicitly displayed on the platforms specifies the current status of the project, particularly from the Navigation Platform. Team members are made aware of work that has been done, work being done as planned, work being done to incorporate other tasks, and the team members who are performing the tasks.

Software engineering domain knowledge is captured in the form of software engineering ontology. Software engineering standards, rules, and formalisms facilitate the production of high-quality task performance. Hence, the development of components from different sources and their ability to integrate these components to build a system, can be relied on. If a component integration has not been successful at some point during integration, the issue is raised through the Question Platform in the line before the project fails. Additionally, mutual adjustment, feedback and discussion during integration, in order to align team members' actions, can be carried out through the platforms.

Platforms define how tools are utilised. This enables one to overcome the issues of track and trace and just-in-time in the multi-site situation.

The platforms are where members keep track of project data being exchanged, discussed or shared, as well as member relationships. Local context and constraints issues do not remain local, but rather are shared to enable brainstorming across multi-sites. In particular, the Solution Platform is where members are advised of any updated project data trace. The platforms are available online for remote teams to send and respond to the information at any point in time.

The software engineering ontology underlying the platforms provides a source of precise and explicit software development terms or concepts which can be communicated across remote team members, multi-site projects, and applications including intelligent agents. This man-machine system, through the platforms, makes it possible for members to perceive their counterparts as they are sharing the same knowledge formed in the software engineering ontology. Additionally, automatically and immediately, members send and receive responses to sharing project data formed as instance knowledge.

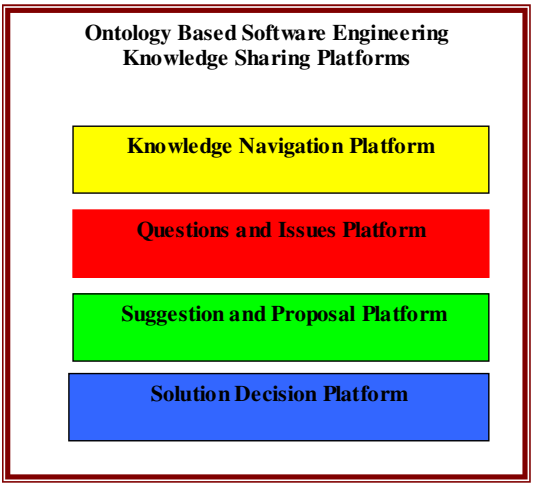


Fig. 12.2 Four levels of software engineering knowledge sharing platforms

Figure 12.2 shows the level view of the four levels of Knowledge Sharing Platforms and Figure 12.3 shows an overview of the four platforms running for the methodology of multi-site distributed software development.

The Knowledge Navigation Platform, or Navigation Platform for short, is basically for all team members to navigate or query shared domain knowledge and instance knowledge. Anyone can view domain knowledge for classification of certain concepts and can view instance knowledge for clarification of project agreements or understandings. However, permission is needed in the Navigation Platform if team members want to make changes to instance knowledge. Authorised members can make changes from the Navigation Platform. Unauthorised members can propose changes as an issue. The issue will then be processed through the Question Platform, Suggestion Platform, and Solution Platform depending on how substantial the changes are (minor or major changes).

The Question and Issue Platform, or Question Platform for short, is for raising difficulties encountered in the software development or issues which occurred within the team. A member at any one site logs a project matter through the Question Platform. Basically issues raised would be like comments or debates over project data and project agreement. After all, a question posed needs to be elucidated by clarifying the specific project data he/she wants to discuss. Technically, the project data here are known as instance knowledge. As a member progresses

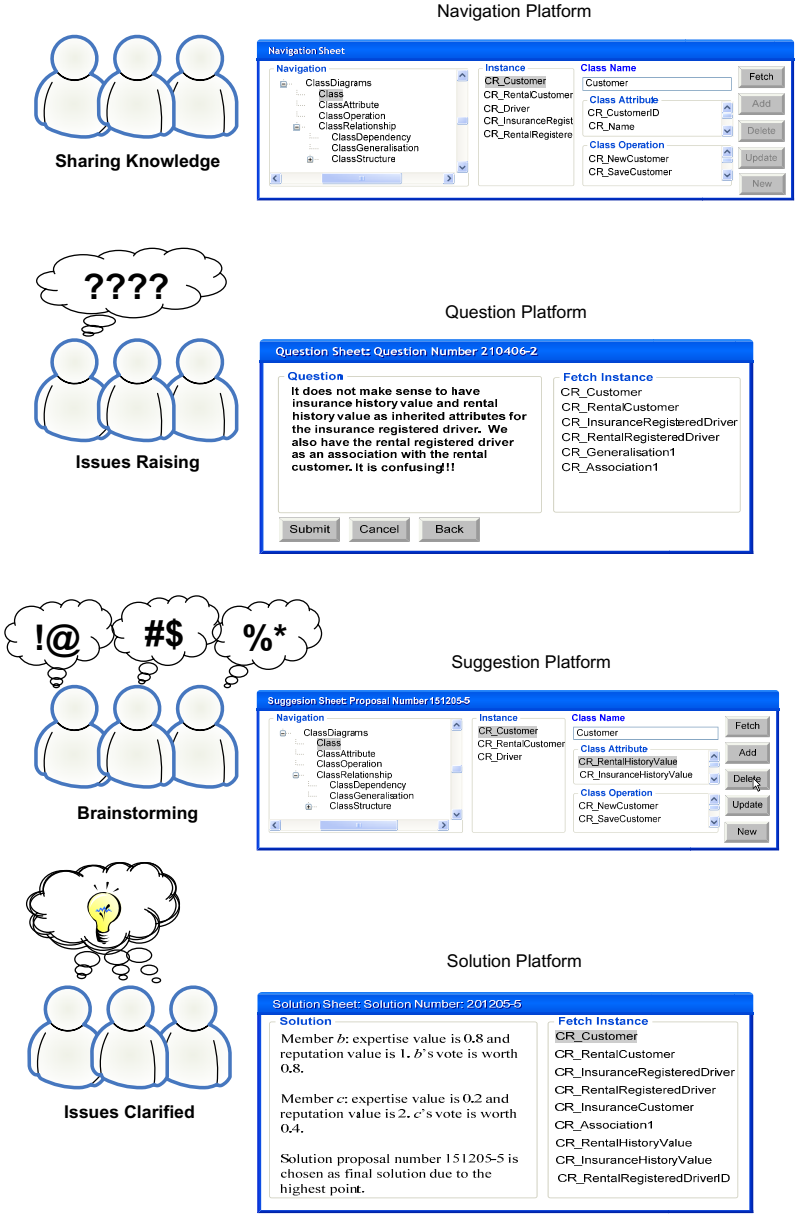


Fig. 12.3 Overview of platforms framework

towards clarifying the issue, the retrieval of relevant instance knowledge by the platform assists other members to have a clear understanding of the issue as well as assisting them to recognise the instance knowledge they have been discussing. This is particularly useful when team members work on many projects simultaneously.

The Suggestion and Proposal Platform, or Suggestion Platform for short, is as its name suggests, for all team members to propose the potential solutions of the discussion or issue proposed in the Question Platform. Thus, this platform mainly involves potential modification of instance knowledge. If any potential modification has to be made, members need to follow the software engineering knowledge defined in the software engineering ontology in order to have a consistent view and the same understanding. In the Suggestion Platform, all proposed solutions are initially pending.

After a certain time of gathering some feedback from team members, a decision maker agent in the Solution Decision Platform, or Solution Platform for short, later determines what action is needed or which solution will be updated. The actual updating process is also carried out through the Solution Decision platform. A final solution after the discussion is revealed by the Solution Platform. The software engineering ontology gets updated in the ontology repository at this stage.

The issue raised plays an important role in multi-site distributed software development. Issues are distinguished as being either minor or major problems. Access control in the Navigation Platform determines whether an issue is major or minor. Being a major issue, it needs brainstorming through the Suggestion Platform. A minor issue may mean (daily) basic update of instance knowledge by the member in the team who has got permission to do so. Figure 12.4 shows a flow chart of the processes when an issue arises.

Basically, there are two cases. The first case is when an issue is raised by a member who has no access. In this case, the member is not part of the working team where the issue has arisen. The issue from the member is then considered as

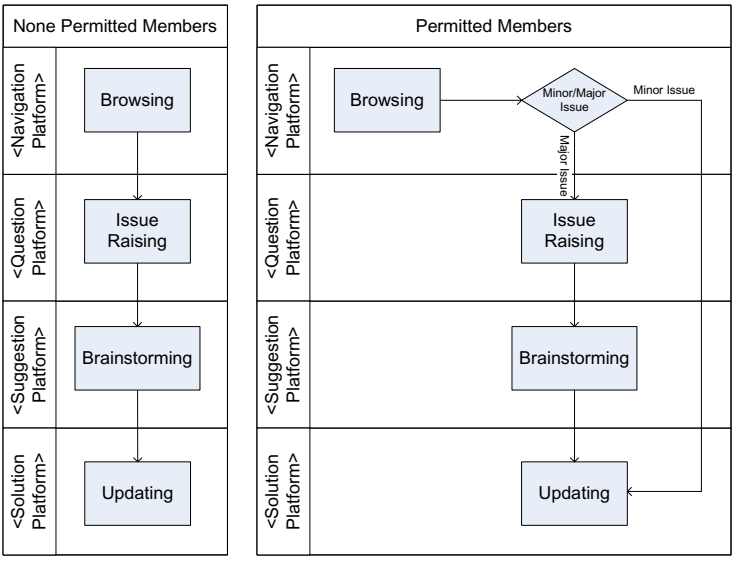


Fig. 12.4 A flow of the processes when issue arises

a major issue. For example, members in the design team may raise an issue relating to the requirements part which is the responsibility of the analysis team. The issue raised is categorised as a major issue that needs authorisation. An issue raised by a member who has access is another case. This is a case of the member being in the working team where the issue has arisen. In this case, the issue can be considered as either minor or major and this is determined by the member alone. In deciding that the issue is a minor one, it could simply be a matter of updating project data that occurs within the team. The issue, however, can be considered as a major issue if the member decides that the issue is significant enough to need brainstorming from other members or even from other teams.

In particular, platforms improve the process of sharing knowledge so that it is understood in a consistent manner. Also, they facilitate effective and efficient remote communication in the multi-site setting.

Navigation Platform

Software engineering knowledge, formed into software engineering ontology, helps communications among team members and provides consistent understanding of the domain knowledge. Software engineering ontology, together with its instance knowledge, is used as a communication framework within a project, thereby providing rational and shared understanding of project matters. In the Navigation Platform, software engineering instance knowledge, in accordance with domain knowledge that is described in software engineering ontology, is extracted. By consulting the software engineering ontology, the platform enables references of software engineering domain knowledge and enables extraction of instance knowledge. For example, class diagrams referred to in the software engineering ontology assert how a set of classes is formed in the diagram.

The specification imposing a structure on the domain of class diagrams i.e. elicitation of each class consists of class name, class attributes, class operations and relationships, hold with other classes. Using software engineering domain knowledge, together with instance knowledge, the Navigation Platform dynamically and automatically acts for a certain class instance that the member navigates to retrieve accordingly attribute instances, operation instances, and relationship instances together with the related class instance details.

Figure 12.5 shows examples of instances of class diagrams ontology that are navigated in the Navigation Platform. In Figure 12.5 (a), Class instance CR_Customer is navigated to consequently retrieve ClassAttribute instances and ClassOperation instances. ClassRelationship instances can also be navigated to consequently retrieve Class instances that hold in the relationship and applicable properties of the relationship. For example, in accessing ClassAssociation instance, Class instances held in the relationship and properties like role name and cardinalities are automatically retrieved as shown in Figure 12.5 (b). Similarly, if those Class instances are accessed, then a list of ClassAttribute instances and a list of ClassOperation instances are retrieved to show its attributes and its operations respectively. In accessing each ClassAttribute instance, details of attribute's name, attribute's data type, and attribute's visibility are shown as referred to ClassAttribute ontology in the software engineering ontology. In Figure 12.5 (c), navigating ClassAttribute instance CR_CustomerID, its name of 'Customer ID', its data type of

'integer', and its visibility of 'public' can be revealed. The same as ClassOperation ontology referred in the software engineering ontology, in accessing each ClassOperation instance, details of operation's name, operation's visibility, and operation's parameters and parameters' data type can be retrieved.

Fig. 12.5 Examples of instances being navigated in the Navigation Platform

Moreover, indicating how concepts are inter-related constrains the possible interpretations of terms. Structures of class, object and component impose differences in the software engineering domain. Therefore, through class diagrams, classes, class attributes, class operations, and/or relationships amidst classes are expected, whereas by declaring object diagrams, objects, object attributes, belonging classes, and/or relationships between objects are expected. By indicating component diagrams, which are apparently different from class diagrams and object diagrams, components, interfaces and/or relationships among components are expected. The different constituents of the domain of class diagrams, object diagrams, and component diagrams in the navigation of class,

object, and component respectively reveals divergence in their usage. If there is some kind of consensus among the software engineering community or project teams, then this eliminates ambiguous concepts or terms. With team members having the same understanding of concepts, remote communications proceed smoothly and effectively in the multi-site setting.

Question Platform

Both the Question Platform and the Suggestion Platform involve proposals for dealing with multi-site project issues. However, each is devised for different purposes. In the Question Platform, team members raise an issue, whereas, in the Suggestion Platform, they propose the possible solutions for the issue. A member interacts with the platform in order to raise an issue. To do so, the member firstly needs to clarify the problem, specifying the particular instance knowledge involved which can be fetched from the Navigation Platform. Such issues raised are distinguished as either minor or major issues. A minor issue is simply one where the necessary change does not have to be authorised. This is in the case of instances update within a team such as daily update by authorised members. An authorised approval is needed for a major issue. This is in the case of an instance change by a member in another team. For example, a member in an implementation team has raised the issue of change request on the design part which is the responsibility of the design team. The instance minor changes get updated in the ontology repository directly through the Navigation Platform and are recorded and acknowledged through the Solution Platform. The major instance issues are pending through the Question Platform and make issues outstanding until feedback and suggestions from members are made through the Suggestion Platform.

The cases of raising major and minor issues are given as follows. The first case, shown in Figure 12.6, is considered as a major issue due to the member's intention of making the issue outstanding through the Question Platform and obtaining suggestions from other members through the Suggestion Platform. By selecting instances involved and by clicking the Fetch button, issues are raised through the Question Platform.

The third case, shown in Figure 12.7, is where the changes made simply involve updating; hence, there is no need to go through the Question Platform. Instead, the member needs only to go through the solution to record changes and be acknowledged by other members.

In general, all the issues were resolved by members utilising the discipline of the software engineering represented and expressed in the software engineering ontology to gain a common understanding among the teams. This is carried out by the platforms referring to the software engineering ontology. The software engineering knowledge formed in the software engineering ontology is captured widely and thoroughly.

Suggestion Platform

The Suggestion Platform interactions involve modifying instance knowledge. The modifications include adding, adding new, deleting, and editing instance knowledge. However, modifications made in the Suggestion Platform are not instantly updated to the knowledge base in the repository. Rather, the modifications made are pending and will be updated in the Solution Platform if they are applicable.

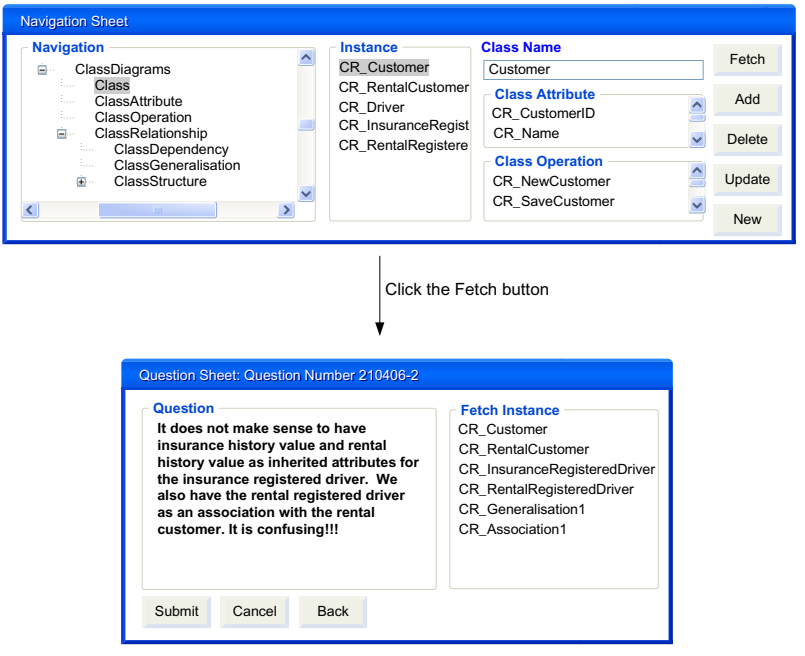


Fig. 12.6 Member raises issue in Question Platform

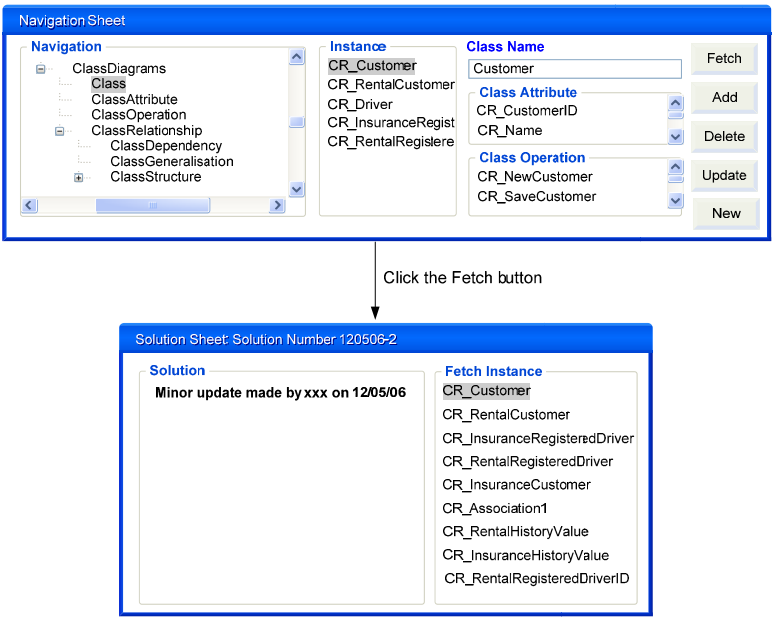


Fig. 12.7 Member makes minor changes

Mainly, the Suggestion Platform is concerned with a number of proposed possible solutions regarding the problem raised from the Question Platform. This allows all team members the freedom to advocate. In order to not be too restrictive, the software engineering ontology is left open to worthwhile ideas. However, any changes that have been proposed need to be accepted by the software engineering domain knowledge asserted in the software engineering ontology. For example, in the activity diagrams ontology, the constraint in the fork transition states that exactly one flow of control splits into at least two flows of control. If it is not in line with this notion, it is then not a fork transition. Similarly, join transition restricts the joining of at least two incoming transitions and exactly one outgoing transition.

After a certain time of accumulating various ideas from team members, determination of the final solution will be processed. The final solution will then be passed on to the Solution Platform in order to update the ontology repository as well as to inform participating members.

Solution Platform

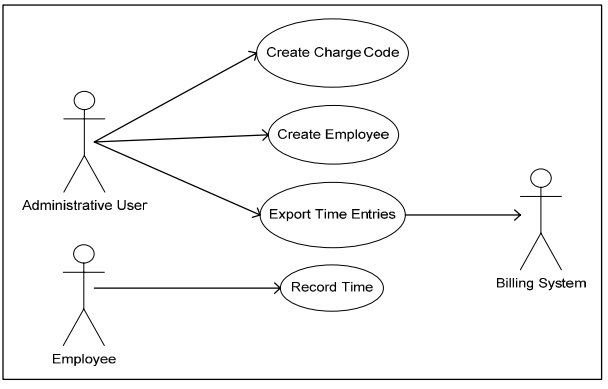
The Solution Platform is where any decision made is updated. Also the platform informs participated members of decision made. For this study, the basic techniques of voting together with individual reputation value, play roles in determining the final solution. However, it can be open to company or organisation policy to draw and develop the making decision agent for this platform.

The strategy for updating instance knowledge is given here. For example, Figure 12.8 (a) shows a use case diagram, Figure 12.8 (b) shows a revised use case diagram needing to be updated, and Figure 12.8 (c) shows an ontology model of the use case diagram referring along updating. The use case diagrams used as an example here are derived from the book of Enterprise Java with UML (Arrington 2001).

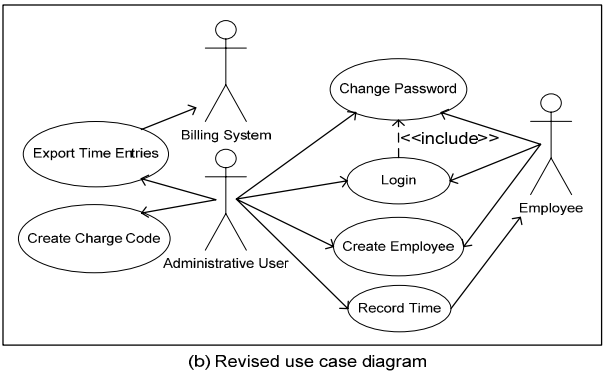
A list of updating actions is as follows:

- Adding new instances ChangePassword and Login for concept UseCase.
- Adding new instance for concept IncludeRelationship relating relations Related_Use_Case with concept UseCase instance ChangePassword and Relating_Use_Case with concept UseCase instance Login.
- Adding new instance for concept AssociationRelationship relating relations Related_Use_Case with concept UseCase instance ChangePassword and Relating_Use_Case with concept Actor instance AdministrativeUser.
- Adding new instance for concept AssociationRelationship relating relations Related_Use_Case with concept UseCase instance Login and Relating_Use_Case with concept Actor instance AdministrativeUser.
- Adding new instance for concept AssociationRelationship relating relations Related_Use_Case with concept UseCase instance ChangePassword and Relating_Use_Case with concept Actor instance Employee.
- Adding new instance for concept AssociationRelationship relating relations Related_Use_Case with concept UseCase instance Login and Relating_Use_Case with concept Actor instance Employee.

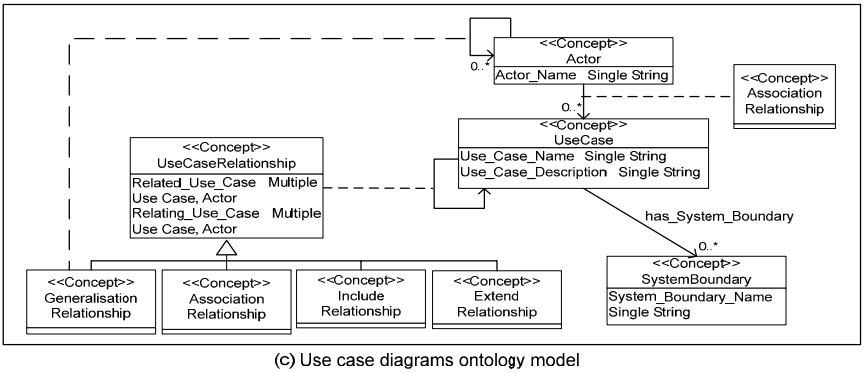
- Adding new instance for concept AssociationRelationship relating relations Related_Use_Case with concept Actor instance Employee and Relating_Use_Case with concept UseCase instance CreateEmployee



(a) A use case diagram



(b) Revised use case diagram



(c) Use case diagrams ontology model

Fig. 12.8 An example of revising use case diagram ontology instances

Updating instance knowledge can be as simple as just adding new information to it, or it can be complicated with a mix of deleting and then adding information.

12.3 Formal Specifications of Software Engineering Domain Conceptualisation

A process of design in the SE Ontology refers to the process of design concepts which includes concepts hierarchy design, relations and constraints, and instances in the software engineering domain.

12.3.1 SE Ontology Concepts

The SE Ontology contains over 362 concepts. Figure 12.9 gives an overview of a part of the SE Ontology illustrating the construction of software engineering concepts.

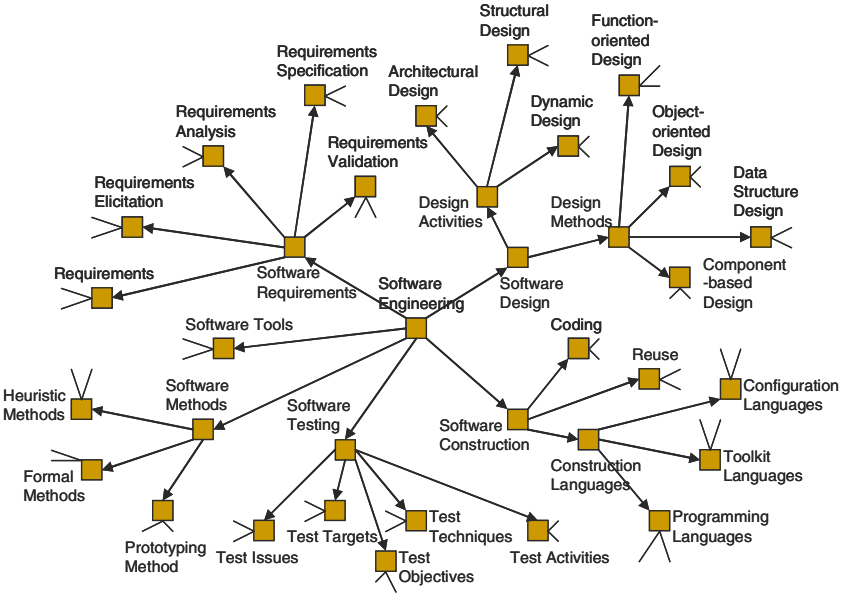


Fig. 12.9 Overview of a part of SE Ontology

As can be seen from Figure 12.9, the top level of the software engineering concept hierarchy is subdivided into five software engineering areas of Software Requirements, Software Design, Software Construction, Software Testing, and Software Tools. Hierarchy of Software Requirements ontology then consists of ontology concepts Requirements, Requirements Elicitation, Requirements Analysis, Requirements Specification, and Requirements Validation. Concept hierarchy of

Requirements ontology includes Product Requirements, Process Requirements, Functional Requirements, Non-Functional Requirements, Emergent Properties, Quantifiable Requirements, System Requirements, and User Requirements. Non-functional Requirements can be further classified into Delivery Requirements, Ethical Requirements, Implementation Requirements, Interoperability Requirements, Performance Requirements, Portability Requirements, Privacy Requirements, Reliability Requirements, Safety Requirements, Space Requirements, Standards Requirements, and Usability Requirements. Requirements Elicitation ontology builds a two-concept hierarchy i.e. Requirements Sources and Elicitation Techniques. Concept hierarchy of ontology concept Requirements Sources includes Domain Knowledge, Operational Environment, Organisational Environment, Stake Holders, and Term Goals. Concept hierarchy of ontology concept Elicitation Techniques includes Ethnography, Facilitated Meeting, Interviews, Prototypes, Scenarios, Use Cases, and Viewpoints. Interview can be further classified into Close Interview and Open Interview.

12.3.2 SE Ontology Relations and Constraints

The SE Ontology contains over 303 relations. In this section, we illustrate the design by choosing some specific examples of common widely used concepts in the software engineering domain. We utilise the notation description introduced in Chapter 9.

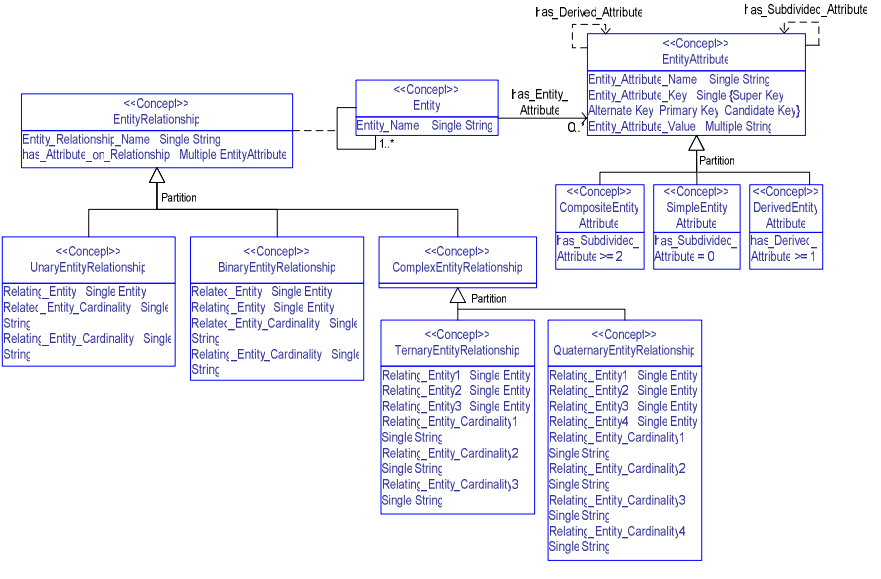


Fig. 12.10 Entity-relationship diagrams ontology relations

An entity-relationship diagram represents conceptual models of data stored in information systems. In an ontology model of entity-relationship diagrams as shown in Figure 12.3, there are three main basic components in the entity-relationship diagrams i.e. entity, attributes, and relationships which form three ontology classes i.e. Entities class, Entity_Attributes class and Entitiy_Relationships class respectively. Entity_Attributes class can be classified as being simple (i.e. Simple_Entity_Attribute class), composite (i.e. Composite_Entity_Attribute class) or derived (i.e. Derived_Entity_Attribute class). A simple attribute is composed of a single component and a composite attribute is composed of multiple components. In the ontology model, cardinality restriction in relation between Entity_Attribute classes defines attributes as being either simple or composite. A derived attribute is based on another attribute(s) and refers to relation has_Derived_Attribute restricting at least one relation. Key can be defined as attributes of super key, alternate key, primary key, or candidate key. This refers to relation Entity_Attribute_Key in the ontology model and restricts to one of super key, alternate key, primary key, or candidate key. An attribute can have a single or greater-than-one value. In the ontology model, cardinality restriction from relation Entity_Attribute_Value defines having a single or greater-than-one value. There are three main degrees of relationships which are unary (i.e. Unary_Entity_Relationship class), binary (i.e. Binary_Entity_Relationship class), and complex (i.e. Complex_ Entity_Relationship class). The complex relationship can be further divided into quaternary (i.e. Quaternary_Entity_Relationship class) and ternary (Ternary_Entity_Relationship class. In the ontology model, cardinality restriction constrains the number of entities that participate in a relationship. For example, a unary relationship represents a relationship of one entity or, more precisely, that entity is self-linked. This means that from the perspective of ontology, there is only one Entity in the relation Relating_Entity and no Entity in the relation Related_Entity. In an entity relationship, cardinality can be specified as string which can be a string of 1 (one and only one), * (zero or more), 1..* (one or more), 0..1 (zero or one) and so forth as shown in the ontology model. Attributes can also be assigned to relationships referring to relation has_Attribute_on_Relationships in the ontology model.

An activity diagram shows the control flow from activity to activity. Mainly, activity diagrams contain activities, transitions, swimlane, and objects forming ontology classes of Activity, Transition, Swimlane, and Object respectively as shown in Figure 12.4. A locus of activities is specified by a swimlane. This refers to relation in_Swimlane in the ontology model. Every activity belongs to exactly one swimlane; however, transition may make it cross lanes. This means maximum cardinality restriction in relation in_Swimlane. Objects may be involved in the flow of control associated with an activity diagram. This refers to relations set_Object_Flow and its inverse, get_Object_Flow. Transitions of activities are classified into four main transitions. Firstly, normal transition (i.e. Normal_Transition class) shows the path from one activity to the next activity. This means that, ontology class Normal_Transition that has a cardinality restriction, restricts only the one activity in the relations Related_Activity and Relating_Activity. Secondly, special transition (i.e. Special_Transition class) is further divided into an initial transityion (i.e. Start_Transition class) and a stop transition (Stop_Transition class). The initial transition is where the activity

diagrams start. This means that, class Start_Transition has a cardinality restriction and restricts at least one activity in relation Related_Special_Activity but no activity in relation Relating_Special_Activity. The stop transition is where the activity diagrams stop. This means that class Stop_Transition which has a cardinality restriction, restricts at least one activity in relation Relating_Special_Activity but no activity in relation Related_Special_Activity. Thirdly, branch transition which specifies alternate paths taken based on some guard expression refers to ontology Branch_Transition class. Lastly, concurrent transition (i.e. Concurrent_Transition class) is further divided into a fork transition (i.e. Fork_Transition class) and a join transition (Join_Transition class). The fork transition represents the splitting of a single flow of control into two or more flows of control. This means that ontology class Fork_Transition, that has a cardinality restriction, restricts at least two activities in relation Related_Concurrent_Activity and only one activity in relation Relating_Concurrent_Activity. The join transition represents the joining of two or more incoming transitions and one outgoing transition. This means that ontology class Join_Transition, which has cardinality restriction, restricts at least two activities in relation Relating_Concurrent_Activity and only one activity in relation Related_Concurrent_Activity.

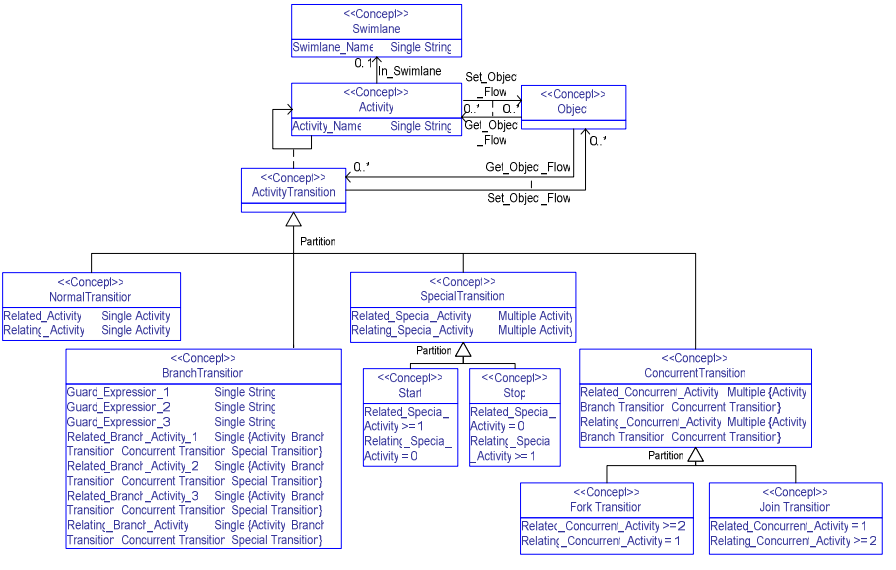


Fig. 12.11 Activity diagrams ontology relations

A class diagram is a diagram that represents a set of classes and their interrelationships (Bourque et al. 2004). Commonly, the structural details of classes expand their attributes and their operations. The five main classes' relationships are: dependencies, generalisations, aggregations, compositions, and associations. The relationships of aggregations, composition, and association are used to structure a class, so in the ontology model of class diagrams, they are grouped together. Figure 12.12 shows an ontology model of class diagrams.

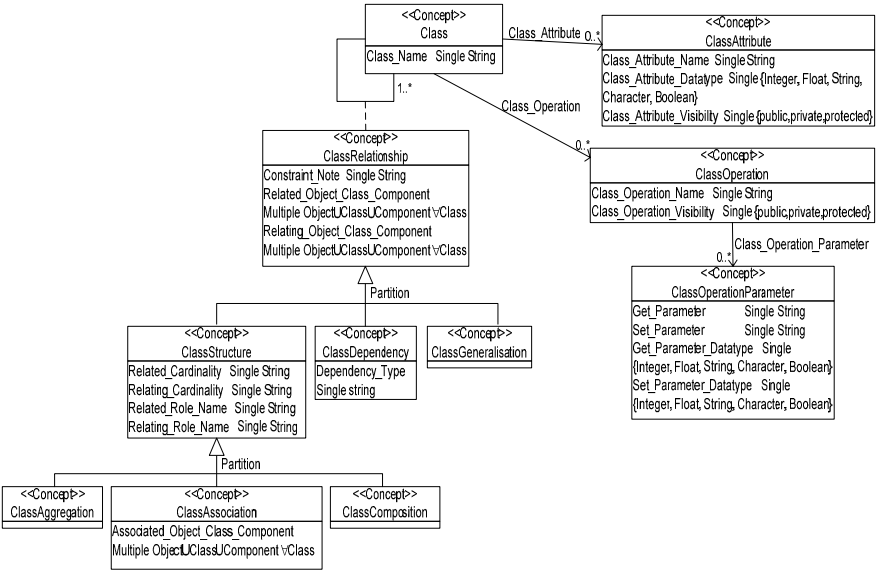


Fig. 12.12 Class diagrams ontology relations

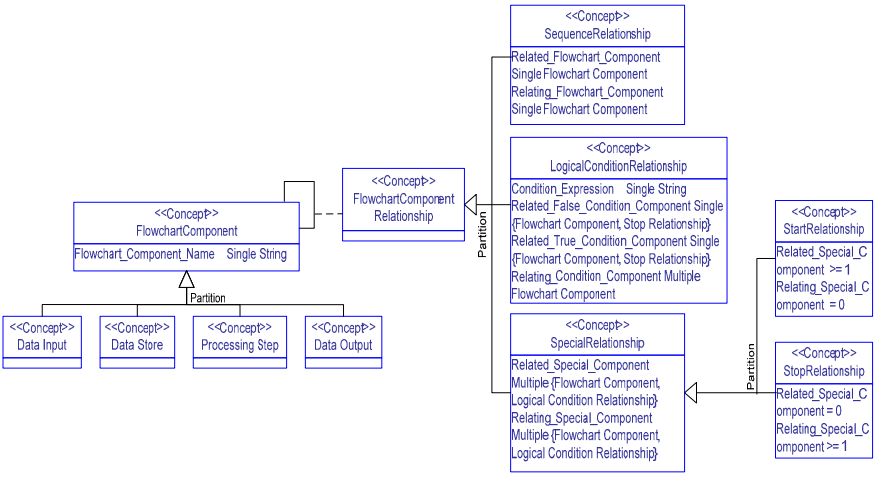


Fig. 12.13 Flowchart ontology relations

A flowchart represents the flow of control and the associated actions to be performed (Bourque et al. 2004). Basically, flowcharts consist of components of data input, data store, processing step, or data output and their relationships. Figure 12.13 shows flowcharts in an ontology model. There are three main types of component relationships. Firstly, sequence relationships representing two components connected refer to ontology class Sequence Relationship. Secondly,

logical condition relationships representing which component to go to depend on evaluation condition expression refer to ontology class LogicalConditionRelationship. Thirdly, special relationships are further divided into two types of initial relationships and end relationship. Initial relationship is where the flowchart starts referring to ontology class StartRelationship. End relationship is where the flowchart stops referring to ontology class StopRelationship.

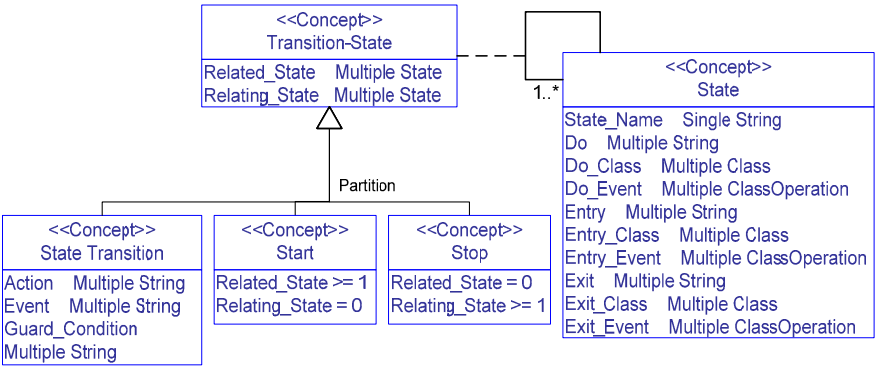


Fig. 12.14 Statechart diagram ontology relations

A statechart diagram shows the control flow from state to state in a state machine (Bourque et al. 2004). Commonly, statechart diagrams contain states and transitions. Figure 12.14 shows an ontology model of statechart diagrams. A state referring to ontology class State can have details like an entry action, exit action, and simple action referring respectively to relations Entry or Entry_Class or Entry_Event, Exit or Exit_Class or Exit_Event, and Do or Do_Class or Do_Event. A transition referring to ontology class Transition categorises into three types. Firstly, state transition refers to ontology subclass State Transition. Secondly, initial transition refers to ontology subclass Start Transition. Lastly, final transition refers to ontology subclass StopTransition. The event that triggers state transition refers to relation Event. The action that results from state transition refers to relation Action. The guard condition that allows a state transition only if the condition is true, refers to relation Guard_Condition.

A use case diagram shows a set of use cases and actors and their relationships. Commonly, use case diagrams contain actor, use cases, and relationships. Figure 12.15 shows an ontology model of use case diagrams. Actors and use cases refer respectively to ontology classes Actor and UseCase. System boundary referring to ontology class SystemBoundary defines use cases limits. Relationships between use cases, referring to ontology class UseCaseRelationship, are categorised into four types of firstly, generalisation relationship – referring to ontology class GeneralisationRelationship, secondly association relationship – referring to ontology class AssociationRelationship, thirdly include relationship – referring to ontology class IncludeRelationship, and lastly extend relationship – referring to ontology class ExtendRelationship. Only the association relationship defines the relationship between actors and use cases, and only the generalisation relationship defines the relationship between actors.

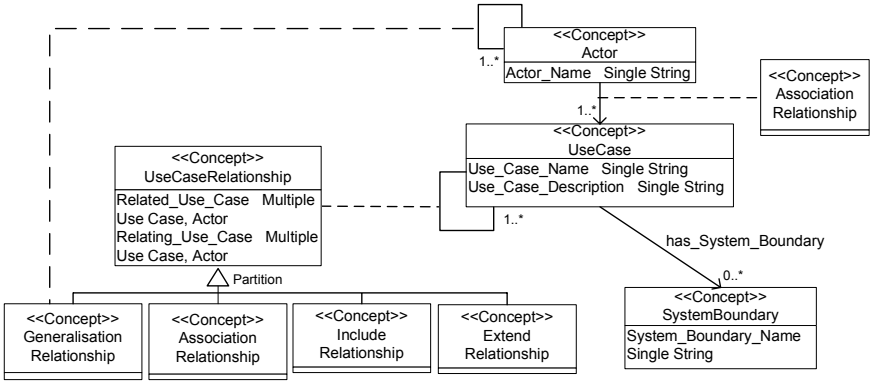


Fig. 12.15 Use case diagram ontology relations

12.3.3 SE Ontology Instances

SE Ontology instantiations are derived as a result of populating software engineering project information and are referred to as ontology instances of ontology classes. Instantiations are also known as instance knowledge of the SE Ontology. In other words, once the SE Ontology has been designed and developed, it needs to be populated with data relating to the project. This process is usually accomplished by mapping various project data and project agreement to the concepts defined in the SE Ontology. Once mappings have been created, project information, including project data, project agreement, and project understanding, is in a semantically rich form.

In this section, we describe in particular how software engineering project data are transformed or mapped into concepts formed in the SE Ontology as instance knowledge. Conversely, the instance knowledge can be transformed back to more presentable and semantic project data e.g. diagram-like project data. Once transformed, instance knowledge is available for sharing. Manipulation of semantics such as instance knowledge can be carried out by users or members.

An example of transformation is given in Figure 12.16 which shows an example UML class diagram that will be transformed into a class diagram ontology model as instance knowledge.

As from Figure 12.9, UML classes Customer, RentalCustomer, InsuranceRegisteredDriver, and RentalRegisterredDriver apply as instances of the ontology concept Class in class diagrams ontology. Explicit domain knowledge from concept Class elicit that class consists of its properties, its operations and its relationships. This is by referring respectively, in the class diagrams ontology model, to relations Class_Attribute, Class_Operation, and association ontology class ClassRelationship. The concept Class instance Customer has relation has_Attribute with concept ClassAttribute instances CustomerID, FirstName, LastName, DriverLicenceNo, etc. For example, the concept instance DriverLicenceNo has relations Class_Attribute_Datatype with xsd:string of 'Integer' and has relations Class_Attribute_Visibility with xsd:string of 'Private'. These are shown below in Figure 12.17.

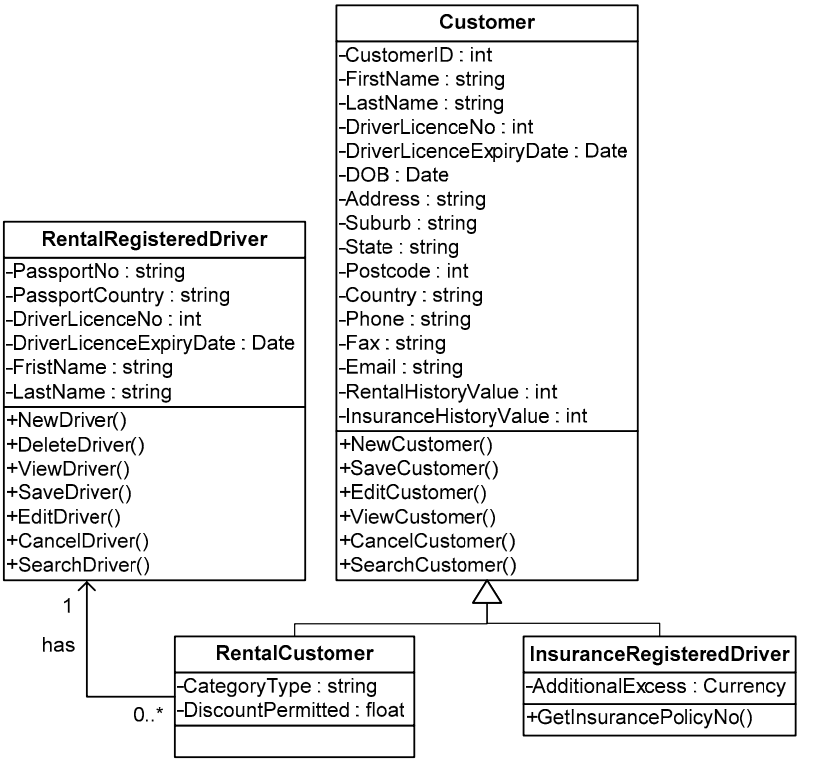


Fig. 12.16 An example of an UML class diagram

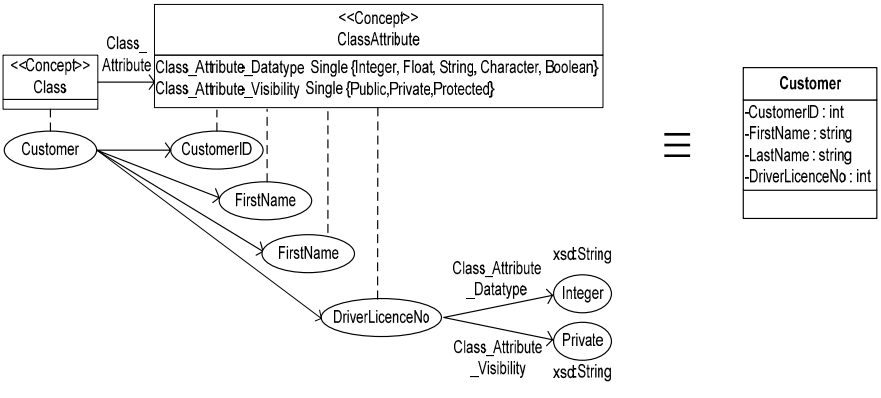


Fig. 12.17 Transformation of UML class customer and its attributes to class diagrams ontology

For a particular UML class Customer, operation NewCustomer applies as an instance of concept ClassOperation in the class diagrams ontology model. The concept Class instance Customer has relation Class_Operation with concept ClassOperation instance NewCustomer. The concept instance NewCustomer has relations Class_Operation_Visibility with xsd:string of 'Public.' These are shown below in Figure 12.18.

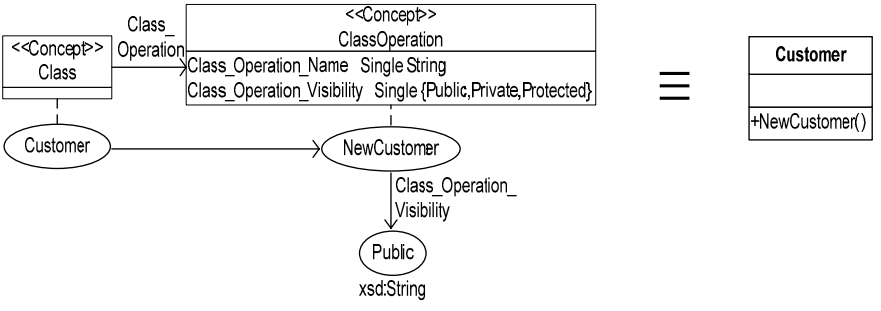


Fig. 12.18 Transformation of UML class customer and its operation to class diagrams ontology

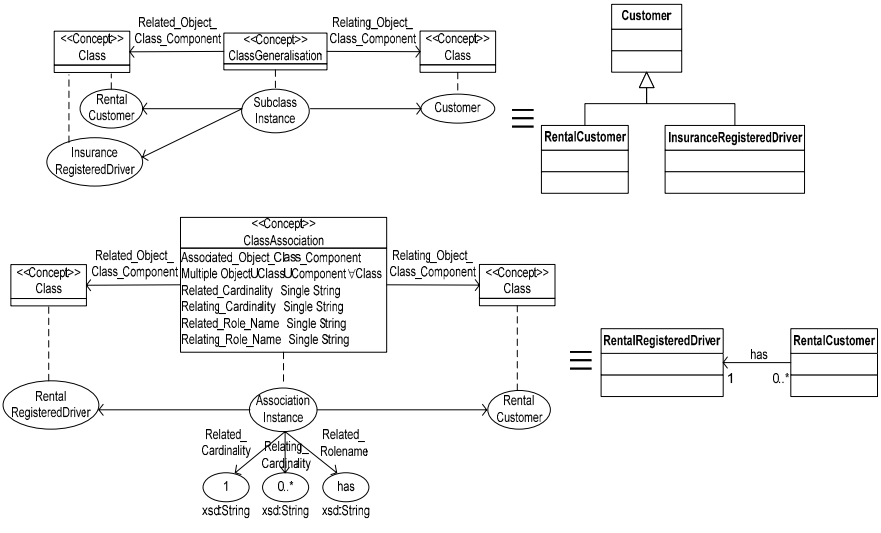


Fig. 12.19 Transformation of generalisation and association relationships to class diagrams ontology

For the particular class diagram shown in Figure 12.19, the concepts of generalisation relationship and association relationship are applied. An instance of concept ClassGeneralisation has relations Related_Object_Class_Component with concept Class instances RentalCustomer and InsuranceRegisteredDriver and has

relations Relating_ Object_Class_Component with concept Class instance Customer. Instance of concept ClassAssociation has relations Related_Object_Class_Component with concept Class instance RentalRegisteredDriver, has relations Relating_ Object_Class_Component with concept Class instance RentalCustomer, has relations Related_Cardinality with xsd:String of '1', has relations Relating_Cardinality with xsd:String of '0..*', and has relations Related_Role_Name with xsd:String of 'has'. These are shown below in Figure 12.19.

12.4 SE Ontology Validation

We use a developed prototype to test the feasibility of the software engineering ontology. The software engineering ontology can be validated though the prototype system as a proof of concept of the software engineering ontology. We validate into three different aspects i.e. validation of communications, validation of knowledge sharing, validation of knowledge management.

12.4.1 Validation of Communications

In this section, we demonstrate V&V of multi-site communications using the following example. In a multi-site environment where teams are located across different sites, member a from site A wants to communicate with member b from site B on a course offering a statechart diagram of project design. Both members are in the same team, that is the design team, but are located at different sites at the time. Consequently, they have access to project design data through platforms. Figure 12.20 shows the original course offering statechart diagram. Figure 12.21 shows the statechart diagram that member a would like it to change to and to communicate to member b. Note that the statechart diagram used as an example here is derived from the book of Visual Modelling with Rational Rose and UML (Quatrani 1998).

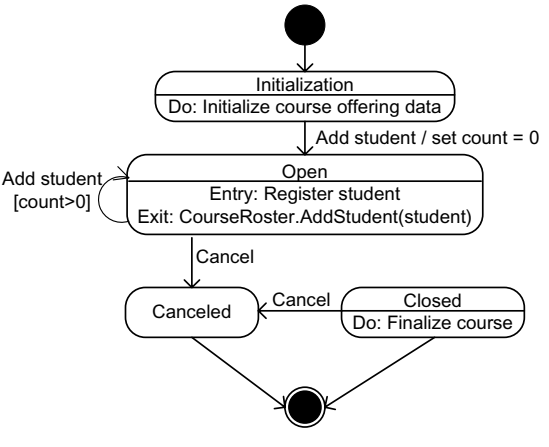


Fig. 12.20 The original course offering statechart diagram

As seen from Figure 12.20 and Figure 12.21, member a wants to communicate with member b that the course offering system should place a limit of only ten students for the course offered. Member a suggests the addition of a guard condition of 'count = 10' to the transition from state Open to state Closed.

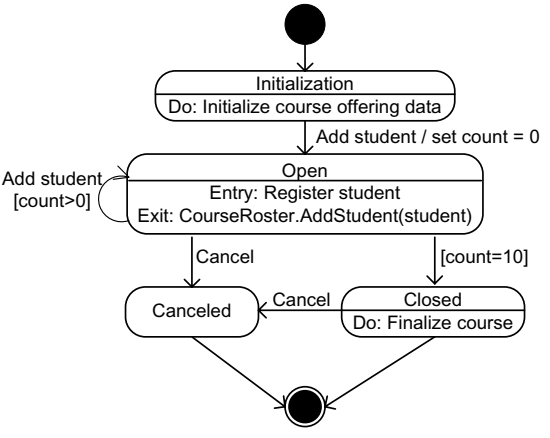


Fig. 12.21 The revised course offering statechart diagram

Member a then logs into the Navigation Platform and navigates or searches for the statechart diagrams concept from the navigation panel. Once the concept has been found, its instances are automatically extracted, shown, and can be selected for editing as shown in Figure 12.22.

In this case, member a will need to add new normal transition. By selecting concept 'NormalStateTransition' in the navigation panel and pressing the button 'New', information of related state, relating state, and guard condition of the transition can be added in as shown in Figure 12.23.

Because member a wants this updating to be an outstanding issue, the changes will not be updated in the knowledge base; rather the changes are pending. By pressing the 'Fetch' button, the changes become a major issue and information about changes will pass through the Question Platform as shown in Figure 12.24.

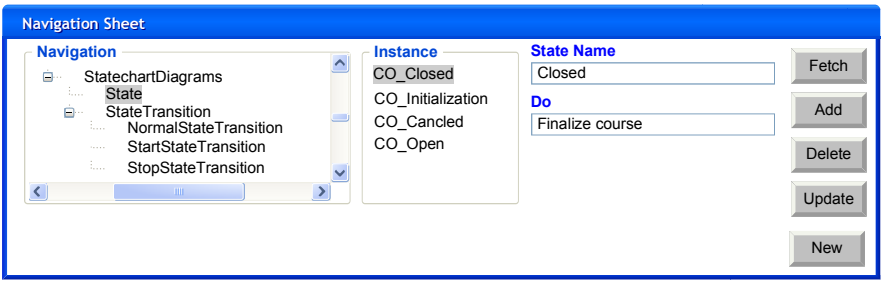


Fig. 12.22 Statechart diagrams ontology instances extracted

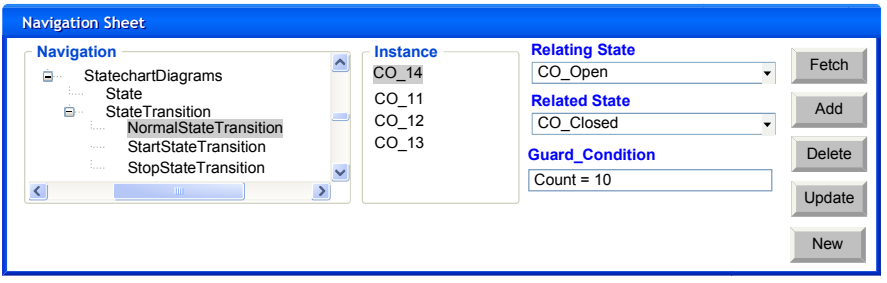


Fig. 12.23 Statechart diagrams ontology instances extracted for member to make changes

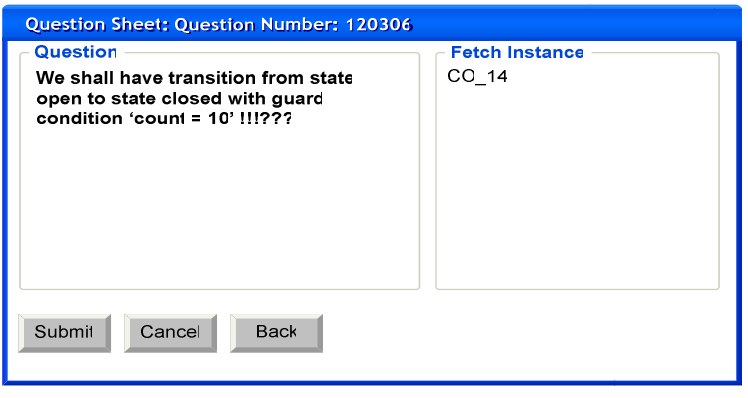


Fig. 12.24 Changes made become a major issue through the Question Platform

In the Question Platform, question number (120306) will be automatically assigned by the platforms systems. Member a can then email the number to member b and then member b can log into systems to see details of changes. Detailed changes can be found in the panel 'Fetch Instance'.

This example validates and verifies that a member can communicate about any project information that is captured as ontology instances. The course offering statechart diagram is captured, and adheres to the concept of the statechart diagram in the software engineering domain knowledge captured as software engineering ontology. This enables a meaningful communication about the course offering statechart diagram. It eliminates misunderstandings, miscommunications, or misinterpretations that may arise. Activity diagrams, statechart diagrams and state transition diagrams are related, thereby sometimes causing confusion. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in the process. The activity diagram shows how these activities depend on one another. Conclusively, in determining the concept of project information that is captured (statechart diagrams or activity diagrams), or where that project

information resides (statechart diagrams or activity diagrams), it is assumed that this is determined by the member who specifies what the project information really means in the context.

Once they are committed to the domain knowledge of statechart diagrams and recognise that it is constituted of state and transitions, and constraints attached to the class or an operation, etc., the commitment enables people to discuss the same topic (the topic of course offering the statechart diagram). Consequently, people can coordinate their activities. The example results demonstrate the potential and advantages of multi-site communications using the methodology.

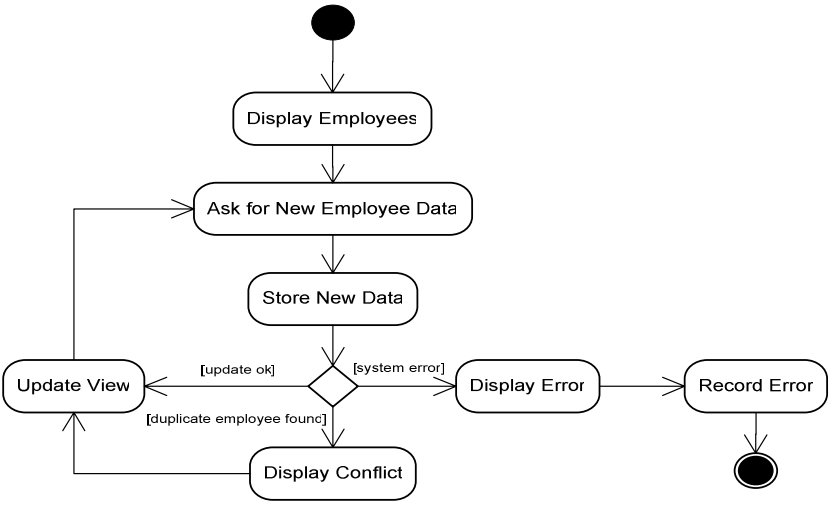


Fig. 12.25 The create employee activity diagram

12.4.2 Validation of Knowledge Sharing

In this section, we demonstrate the V&V of knowledge sharing using an example. In the methodology for multi-site software development, team members work on their individual task and once they finish this, they will need to share it through the platform. This can be used to track whether or not members have finished their assigned task. Lack of information sharing implies that work has not been done. In this section, the prototype shows sharing on activity diagram for the create employee, done for project design. Figure 12.25 shows the activity diagram for the create employee that will be shared. Note that the activity diagram used for our example here is derived from the book of Enterprise Java with UML (Arrington 2001).

In the Navigation Platform, a member navigates or searches for the activity diagrams concept from the navigation panel. Once the concept has been found, concept properties are automatically extracted and information can be added for

sharing among teams across sites. Following is a list of actions to share knowledge on the activity diagram:

- Select the concept 'Activity' using navigation panel. Press the button 'New' to add these activities 'Display Employees', 'Ask for New Employee Data', 'Store New Data', 'Display Error', 'Record Error', 'Update View', and 'Display Conflict'. Figure 12.26 shows the addition of the activity 'Display Employees'.

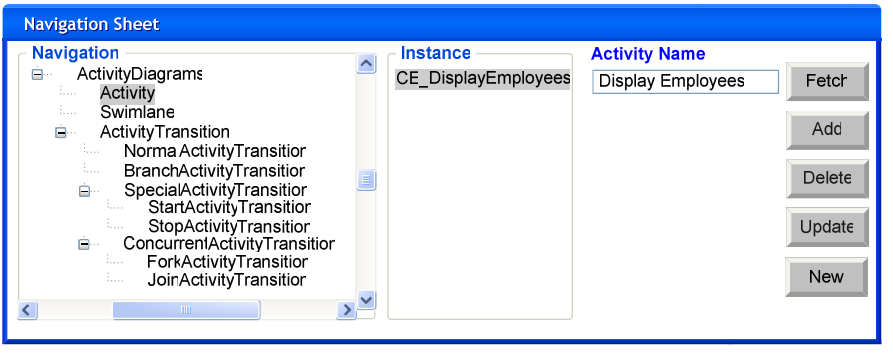


Fig. 12.26 Sharing activity 'Display Employees'

- Under the concept 'ActivityTransition' and under the concept 'SpecialActivityTransition' using navigation panel, select the concept 'StartActivityTransition'. Press the button 'New' to add the start transition. Select property 'Related_Special_Activity'. Press the button 'Add' select activity 'Display Employees' using instance panel. Figure 12.27 shows the addition of the start transition.

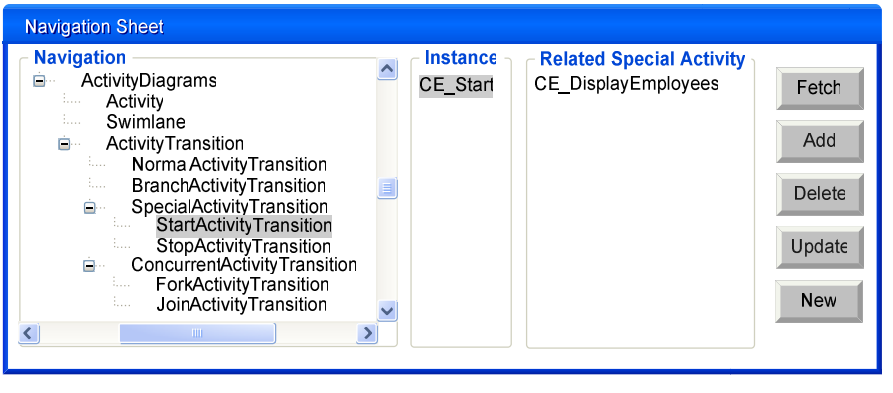


Fig. 12.27 Sharing the start transition

- Under the concept 'ActivityTransition' using navigation panel, select the concept 'NormalActivityTransition'. Press the button 'New' to add four transitions. For the first transition, select activity 'Display Employees' at property 'Relating_Activity' and select activity 'Ask for New Employee Data' at property 'Related_Activity'. For the second transition, select activity 'Ask for New Employee Data' at property 'Relating_Activity' and select activity 'Store New Data' at property 'Related_Activity'. For the third transition, select activity 'Display Error' at property 'Relating_Activity' and select activity 'Record Error' at property 'Related_Activity'. For the fourth transition, select activity 'Update View' at property 'Relating_Activity' and select activity 'Ask for New Employee Data' at property 'Related_Activity'. For the last transition, select activity 'Display Conflict' at property 'Relating_Activity' and select activity 'Update View' at property 'Related_Activity'. Figure 12.28 shows the addition of the first transition.
- Under the concept 'ActivityTransition', select the concept 'BranchActivityTransition'. Press the button 'New' to add the branch transition. Select activity 'Store New Data' at property 'Relating_Branch_Activity'. Select activity 'Display Error' at property 'Related_Branch_Activity_1' and put 'system error' at property 'Guard_Expression_1'. Select activity 'Display Conflict' at property 'Related_Branch_Activity_2' and put 'duplicate employee found' at property 'Guard_Expression_2'. Select activity 'Update View' at property 'Related_Branch_Activity_3' and put 'update ok' at property 'Guard_Expression_3'. Figure 12.29 shows the addition of the branch transition.
- Under the concept 'ActivityTransition' and under the concept 'SpecialActivityTransition' using navigation panel, select the concept 'StopActivityTransition'. Press the button 'New' to add the stop transition. Select property 'Relating_Special_Activity'. Press the button 'Add' and select activity 'Record Error' using instance panel. Figure 12.30 shows the addition of the stop transition.

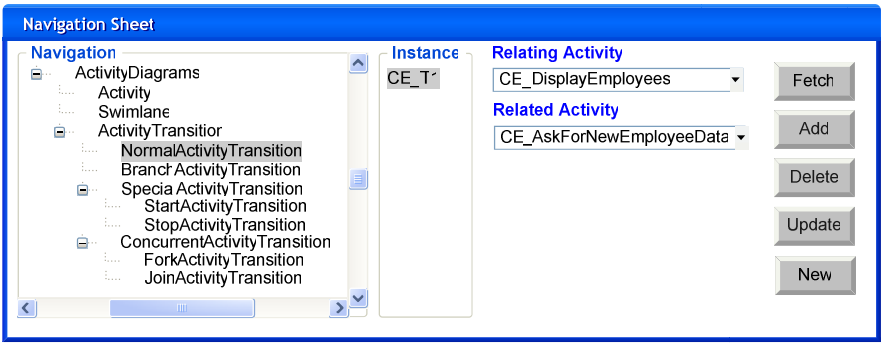


Fig. 12.28 Sharing the activity transition where activity 'Display Employees' flows to activity 'Ask for New Employee Data'

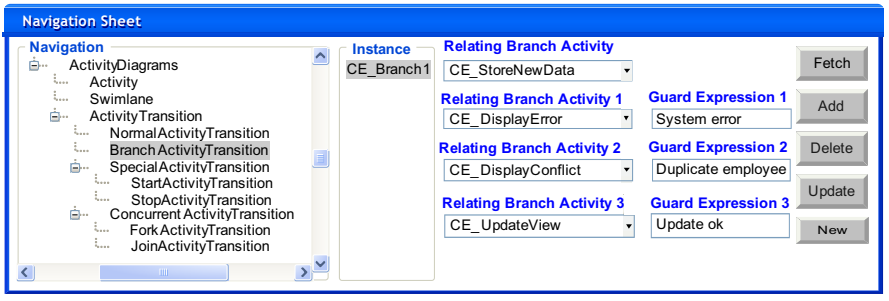


Fig. 12.29 Sharing the branch transition

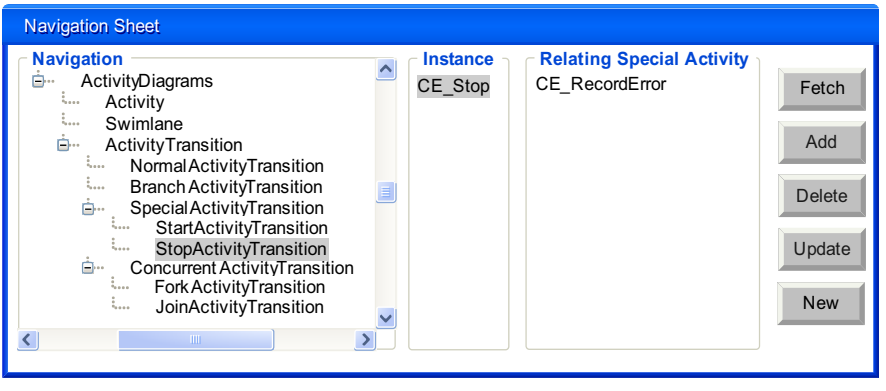


Fig. 12.30 Sharing the stop transition

The purpose of this section is to validate that a member can share knowledge on project information including project data, project understanding, project agreement, etc. used in multi-site software development. Sharing project information drawn based on a consensus of domain knowledge of software engineering formed in the software engineering ontology, makes information explicit. Having attached domain knowledge, it makes project information more understandable, more linear, predictable, and controllable as a member learns about some missing pieces that make sense of the attentive interaction among team members. In order to share the create employee activity diagrams among team members in the example, the domain knowledge of the activity diagrams specifies that members will need to be specific about:

- Activities
- Swimlane in which activities may be
- Transitions of activities which are divided into four types:
    a. Normal transition where an activity flows to another activity
    b. Branch transition where an activity flows to more than one activity depending on condition

c. Special transition which is further divided into two types:
    i. Start transition where an activity is started in an activity diagram
    ii. Stop transition where an activity is stopped in an activity diagram
d. Concurrent transition which is further divided into two types
e. Fork transition where an activity flows mutually to more than one activity
f. Join transition where a number of activities flow jointly to an activity

Some missing pieces are not accepted by the system for sharing. Alarms are then raised. For example, in a fork transition, if a member directs one activity flow to only one activity, not two activities as it should be, then this is alerted as a missing piece.

12.4.3 Validation of Knowledge Management

In this section, we demonstrate V&V of knowledge management and platforms using an example as follows. In a multi-site software development environment, member a, who is a programmer in the team of project implementation, raises an issue about the class diagram in the project design. Figure 12.31 shows the original class diagram that is shared through platforms and that the member would like to raise the issue about.

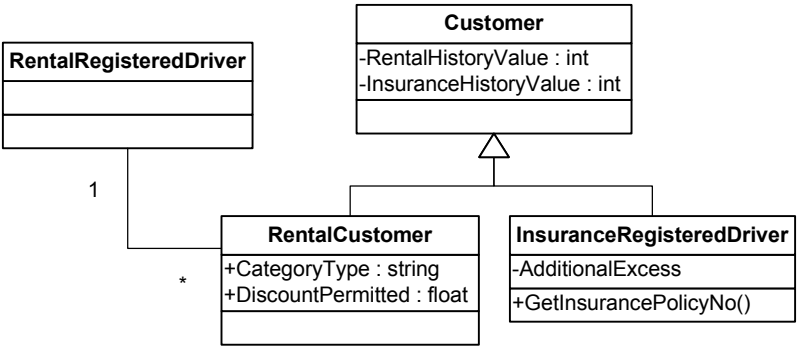


Fig. 12.31 The original class diagram that member wants to raise the issue about

Due to a member not being in the project design team, this member can view only the project design information, but will not be able to make any changes. As seen in Figure 12.32, the buttons 'Add', 'Delete', 'Update' and 'New' are disabled.

However, he can raise this issue through the Question Platform as seen in Figure 12.33. The Navigation Platform is where he can view and provide relevant information to the Question Platform. Now that the issue has been raised, all members are aware of the issue and can respond through the Suggestion Platform.

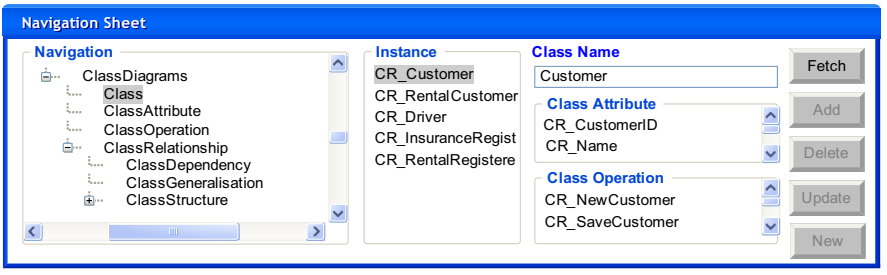


Fig. 12.32 Class diagrams ontology instances extracted

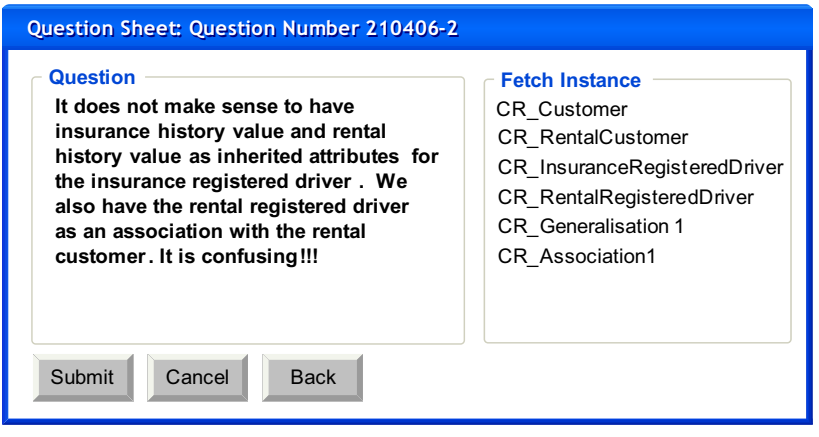


Fig. 12.33 Issue raised through the Question Platform

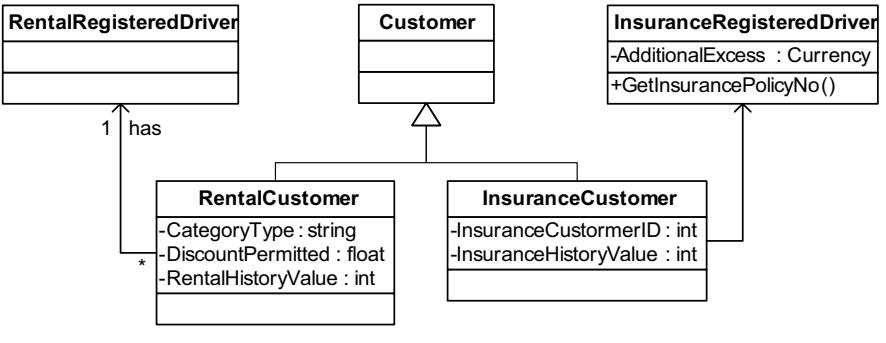


Fig. 12.34 The revised class diagram devised as solution #1

Member b, who is in the project design team, then responds to the issue through the Suggestion Platform. Figure 12.34 shows the class diagram that member b proposes as a solution to the issue that has been raised.

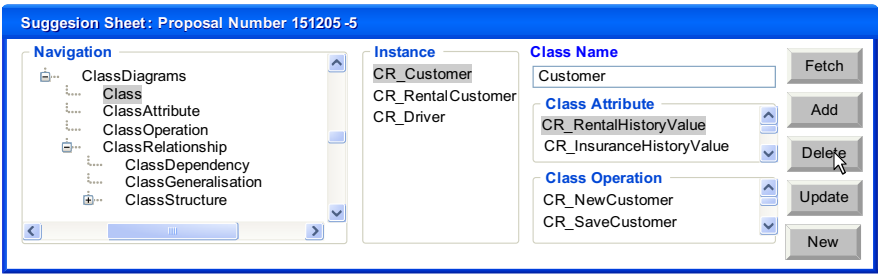


Fig. 12.35 Attribute ''Rental History Value'' is deleted from class 'Customer'

Following is a list of actions on b's proposal solution:

1. Delete attribute 'Rental History Value' and 'Insurance History Value' out of class 'Customer'. Select the concept 'Class' using the navigation panel. Select class 'Customer' using the instance panel. Select attribute 'Rental History Value' from the property 'Class Attribute' and press the button 'Delete'. Select attribute 'Insurance History Value' from the property 'Class Attribute' and press the button 'Delete'. Figure 12.35 shows that attribute ''Rental History Value'' has been selected to be deleted.
2. Link attribute 'Rental History Value' to class 'Rental Customer'. Select the concept 'Class' using the navigation panel. Select class 'Rental Customer' using the instance panel. Select property 'Class Attribute'. Press 'Add' and select attribute 'Rental History Value' using the instance panel. Press the button 'Update'. Figure 12.36 shows that attribute 'Rental History Value' has been updated to class 'Rental Customer'.
3. Add new property 'Insurance Customer ID'. Select the concept 'ClassAttribute' using the navigation panel. Press 'New', put 'Insurance Customer ID' to property 'Class Attribute Name', select 'integer' from the property 'Class Attribute Datatype', and select 'private' from the property 'Class Attribute Visibility'. Figure 12.37 shows that attribute 'Insurance Customer ID' has been added.

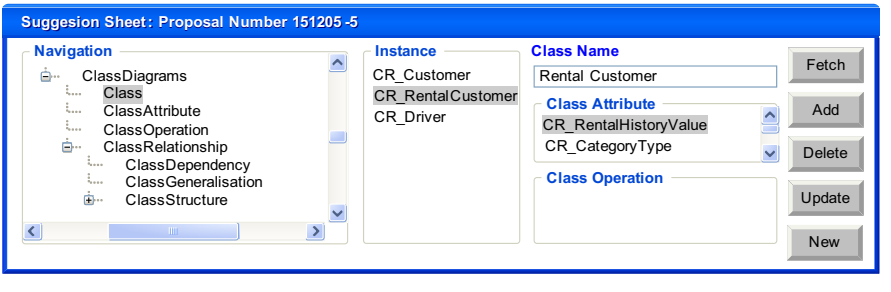


Fig. 12.36 Attributed 'Rental History Value' is linked to class 'Rental Customer'

4. Add new class ‘Insurance Customer’ with attribute ‘Insurance Customer ID’. Select concept ‘Class’, using the navigation panel. Press ‘New’ and put ‘Insurance Customer’ to property ‘Class Name’. Select property ‘Class Attribute’ and press ‘Add’. Select ‘Insurance Customer ID’ using the instance panel and press ‘Update’. Figure 12.38 shows that class ‘Insurance Customer’ has been created with attribute ‘Insurance Customer ID’.
5. Link attribute ‘Insurance History Value’ to class ‘Insurance Customer’. Select the concept ‘Class’ by using the navigation panel. Select class ‘Insurance Customer’ using the instance panel. Select property ‘Class Attribute’. Press ‘Add’ and select attribute ‘Insurance History Value’ using the instance panel. Press ‘Update’ Figure 12.39 shows attribute ‘Insurance History Value’ linking to class ‘Insurance Customer’.
6. Create association relationship between classes ‘Insurance Customer’ and ‘Insurance Registered Driver’. Under the concept ‘Class Relationship’ and under the concept ‘Class Structure’, select the concept ‘Class Association’ using the navigation panel. Press ‘New’. At property ‘Related Class’, select ‘Insurance Registered Driver’ and at property ‘Relating Class’, select ‘Insurance Customer’. Figure 12.40 shows that an association relationship has been created between associated classes ‘Insurance Customer’ and ‘Insurance Registered Driver’.

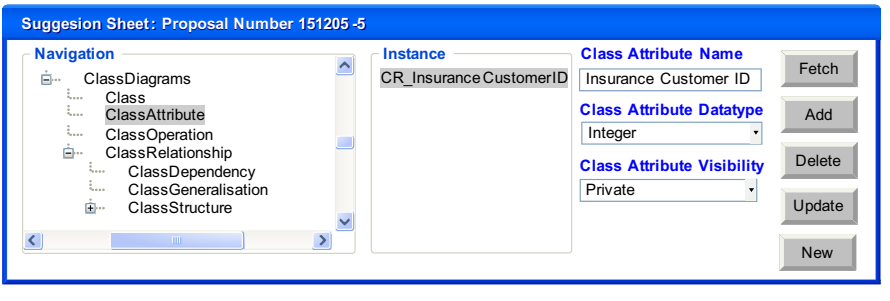


Fig. 12.37 Attribute ‘Insurance Customer ID’ has been added

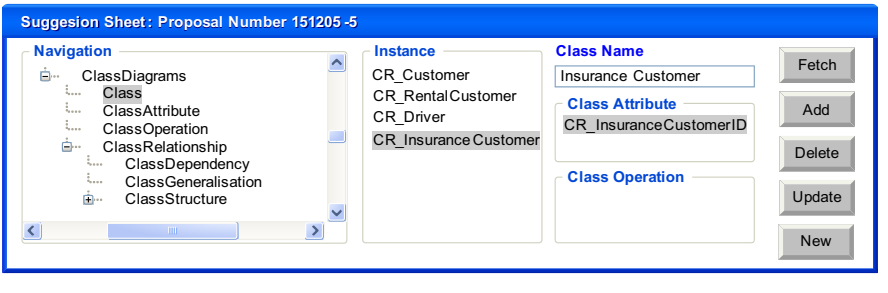


Fig. 12.38 Attribute ‘Insurance Customer ID’ is linked to class ‘Insurance Customer’

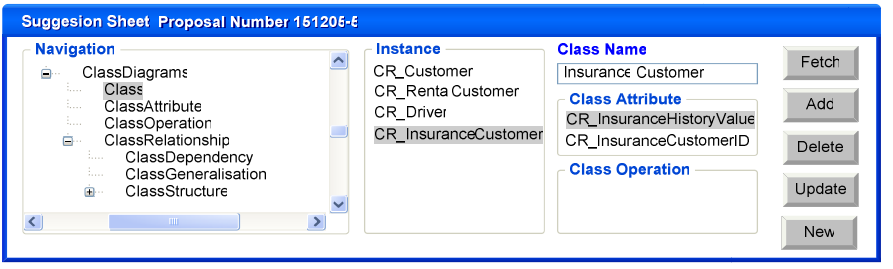


Fig. 12.39 Attributed 'Insurance History Value' is linked to class 'Insurance Customer'

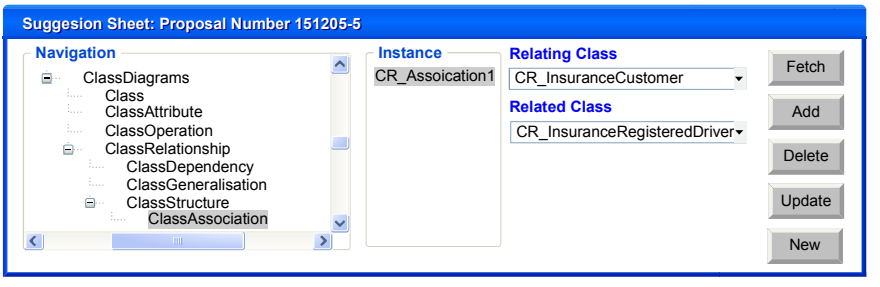
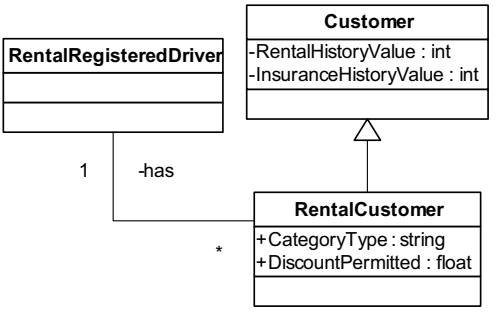


Fig. 12.40 Creation of association relationship, associated classes 'Insurance Customer' and 'Insurance Registered Driver'

Fig. 12.41 The revised class diagram developed as solution #2



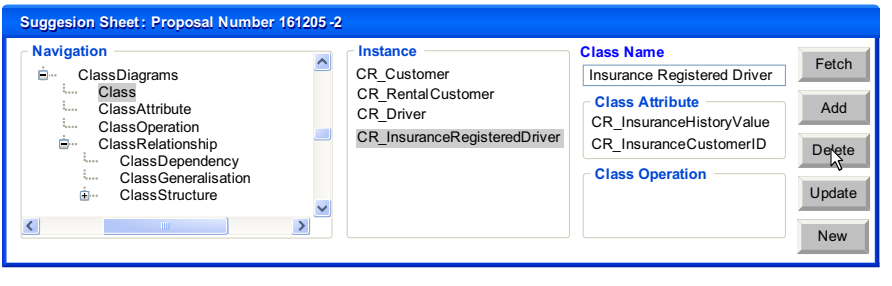


Fig. 12.42 Deletion of class 'Insurance Registered Driver'

Table 12.2 Calculation of voting point

Solution Proposal Number	Proposed by	Expertise Value	Reputation Value	Design Issue Voting Point
151205-5	Member *b*, a designer of the project	0.8	1	0.8 (0.8×1)
161205-2	Member *c*, a programmer of the project	0.2	2	0.4 (0.2×2)

Member c, who is in the project implementation team, also responds to the issue through the Suggestion Platform. Figure 12.41 shows the class diagram that member b proposes as a solution to the issue raised.

What he suggests is the deletion of class 'Insurance Registered Driver'. By selecting 'Class' using the navigation panel, selecting 'Insurance Registered Driver' using the instance panel, and pressing the button 'Delete', class 'Insurance Registered Driver' is deleted as shown in Figure 12.42.

After a certain time of accumulating a number of solutions to the issue raised, determination of a final solution will be processed. This comes through the Solution Platform. In this case, there are two solution proposals: proposal number 151205-5 has been proposed by member b who is the designer of the project, and proposal number 161205-2 has been proposed by member c who is the programmer for the project. In this study, we asserted that 'members who actually work at a task have the best understanding of that task' and we also considered the reputation of the team members involved in the software engineering project. This is so-called 'reputation-based' voting for making decisions whose details can be found in (Wongthongtham 2006). Member b, who is a designer of the project, would have the best understanding of the issue (the design issue); hence, his expertise value is 0.8. Member b has got a reputation value of 1 for a particular area of design. Therefore, his vote is worth 0.8 (0.8x1). Member c, who is a programmer for the project has an expertise value of 0.2. Member c has a reputation value of 2 on a particular area of design. Therefore, his vote is worth 0.4 (0.2x2). Eventually, solution proposal number 151205-5 is chosen as the final solution due to the highest point. Table 12.2 summarises the calculation of the voting point.

Figure 12.43 shows the Solution Platform revealing the final solution with voting point details.

In this section, we validate that members can use platforms to manage knowledge about project information used in multi-site software development. As from the example, knowledge on the class diagram is managed before it becomes wrong through platforms. In the example, discussion of the project design is raised during project implementation. It becomes a case of mutual adjustment being needed between the design team and the implementation team. The issue is raised at the right time before wrong implementation occurs.

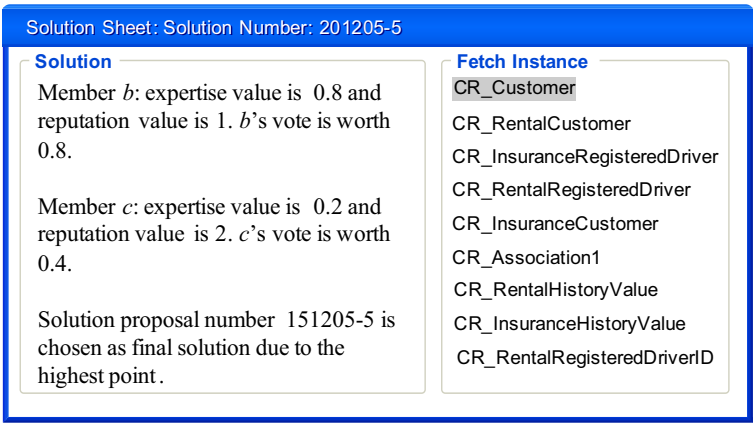


Fig. 12.43 The Solution Platform revealing final solution with voting point details

12.5 Classificaiton of Agents

We classify different groups of agents according to their functionalities and responsibilities within the systems as follows:

- user agents which represent each team member being provided with services,
- safeguard agent which represents system authentication for user authorisation and access level;
- ontology agent which represents manipulation and maintenance of the SE Ontology; and
- decision making agent which represents decision making on the matter of updating the SE Ontology.

A software engineer having and working with his/her own repository of software components, documents and codes, etc. interacts with his/her user agent in the system when he/she wants to enquire, to discuss a problem, to raise an issue, to make a decision, or to find answers in a multi-site distributed environment. If he/she requests to change or update project data, and it is beyond his/her user agent to decide, then the user agent will communicate with the decision making agent. The decision making agent then gathers information from the other team members in addition to consulting the ontologies from the ontology repository. Making the decision is based on the information obtained from consulting ontologies. The final solution(s) will then be raised up and sent back to the involved software engineers. In a case of the decision making agent has difficulties coming up with any solutions; the agent will put it through to the authorised person(s) or team leader to make a decision. Once the agent receives the solution(s) from the person or the team leader, it will automatically reconfigure or update the ontology, the knowledge base in the resources as well as sending back the solution(s) to the involved software engineers. Ontology-based contents and agent capability

descriptions are machine-processable and thus the ontology can be correctly reconfigured by the agents. In the case of changing domain knowledge requests, the user agent is to send the requests to the decision making agent which will then require the domain expert to be involved.

12.6 Need of the Software Engineering Ontology to Support Agent's Intelligence

Ontologies coupled with a multi-agents system allow greater ease of communication by aggregating the agreed knowledge about the project and the domain knowledge of software engineering into a shared information resource platform, and allow them to be shared among the distributed teams across the sites. This enables the intelligent agents to use the ontology to carry out initial communication with developers when the problem is raised in the first instance.

The system utilises software agents-based computing in the sense that the agent has knowledge through consultation with ontologies in the ontology repository. Due to agent capacities in reading and reasoning published knowledge with the guidance of the ontology, the shared ontology enables agents to have meaningful communications. We design a set of agents that cooperate with each other and interact with users or team members.

12.7 Agent's Collaborations

We illustrate an SE Ontology-based, multi-agent system that has four agents. Each user is assigned to a user agent when a login is made. Each user agent is an initiator based on the user actions; the agent will carry out the specific operations accordingly. All the operations have different logic involved, but the structure of creating a user agent is the same. In the agent creation process, an agent object is created when a user login is carried out. The user agent will kill itself if the team member decides to log off the system or the team member is idle for too long. This is typical behaviour of goal-specific agents, which exhibit one-shot behaviour. In other words, the agent is created for a purpose, and once the purpose has been achieved, the agent will be terminated. The safeguard agent will be conducting user authentication and authorization, access levels allocation, proposal management, and monitoring users' activities. The user identification will be verified with the user database as well as access level allocation. There will be five possible cases in access levels allocation. The first case is where the user navigates knowledge in the form of SE Ontology. In the second case, the user queries on the knowledge. These two cases require the ontology agent to perform navigation and querying on the SE Ontology. Knowledge manipulation can produce another three cases. The manipulation of minor instance knowledge requires the ontology agent to do minor manipulation of instance knowledge. The manipulation of major instance knowledge requires both the ontology agent and decision making agent to complete the task. Basically, the ontology agent passes the request to the decision making agent to proceed with reputation-based decision processes including gathering information, consulting the SE Ontology, etc. The manipulation of domain

knowledge also requires both the ontology agent and the decision making agent to complete the task. Similarly, the ontology agent passes the request to the decision making agent to proceed with domain expert-based decision processes. On passing through the decision making agent, the proposals are recorded through logging processes. The results of the processes are sent to the user agent that made the enquiry as well as relevant user agents that will have the affect of processes. As the name itself suggests, the task of the decision making agent is to make decisions on matters of major updates instance knowledge requests and matters of updates domain knowledge requests. The role of reputation-based decision making provides a mean for making the changes to the reflected data in the SE Ontology based on the reputation of users involved in the software engineering project. Reputation based decision making detailed processes can be found in literatures. Detailed domain expert-based decision making processes which involve human domain experts can be found in the literature.

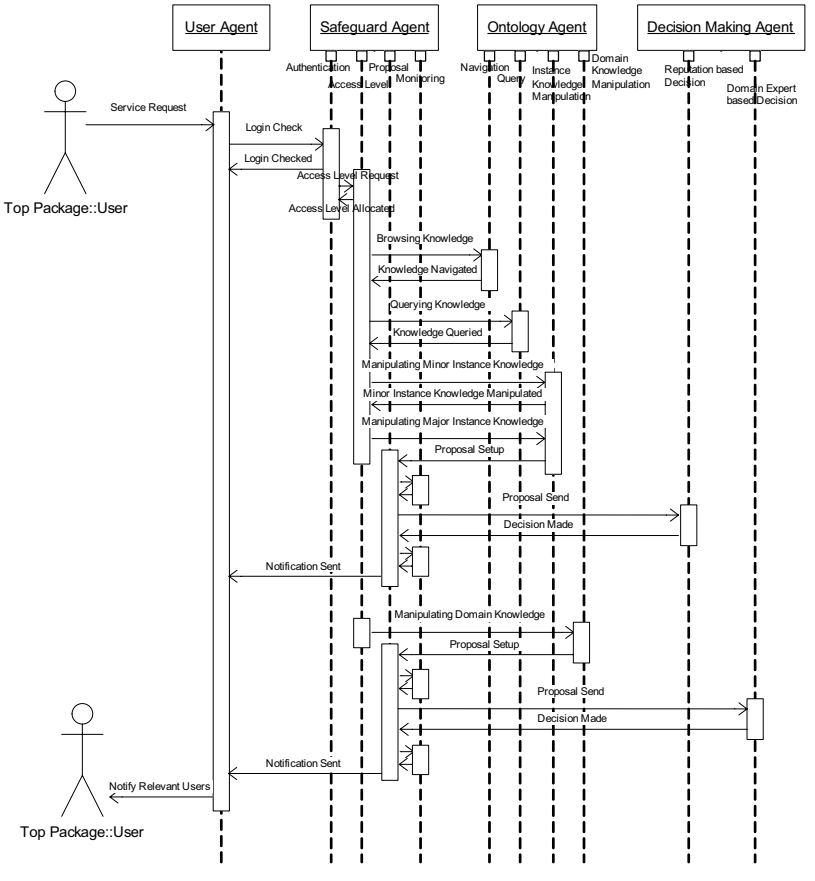


Fig. 12.44 Sequence diagram representing sequence of processes within SE Ontology-based multi-agent systems

Figure 12.44 shows sequence diagram representing sequence of processes within SE Ontology-based, multi-agent systems. A number of agents play multiple roles which are represented by multiple ports. The Safeguard agent plays four roles: systems authentication, access level allocation, proposal management, and monitoring users' activities. The Ontology agent also plays four roles: navigation, querying, instance knowledge manipulation, and domain knowledge manipulation. The Decision making agent plays two roles: reputation based decision and domain expert based decision. Depending on which role the agent is acting in when it sends/receives messages, the sequence diagram shows arrows to/from a particular lifeline for the agent.

We use a Composite Structure Diagram to represent the goal-driven nature of an agent in the system. In the case of the safeguard agent shown in Figure 12.45, we have four ports which correspond to four different roles of this agent, and four parts which show distinct areas of process within the agent. Note that the same two ports (authentication, access levels allocation, proposal management, and monitoring users' activities) that were present in the sequence diagram are also present here. Each of the ports is a construct which enables the Agent to interact with other Agents, namely the Ontology agent and the decision making agent. Composite structure diagrams representing goal-driven characteristic of the ontology agent and decision making agents are shown in Figure 12.46 and Figure 12.47 respectively. We have four ports corresponding to the four roles of the ontology agent and four parts illustrating distinct processes within the ontology agent, while we have two ports (two roles) for the decision making agent and two ports within the decision making agent.

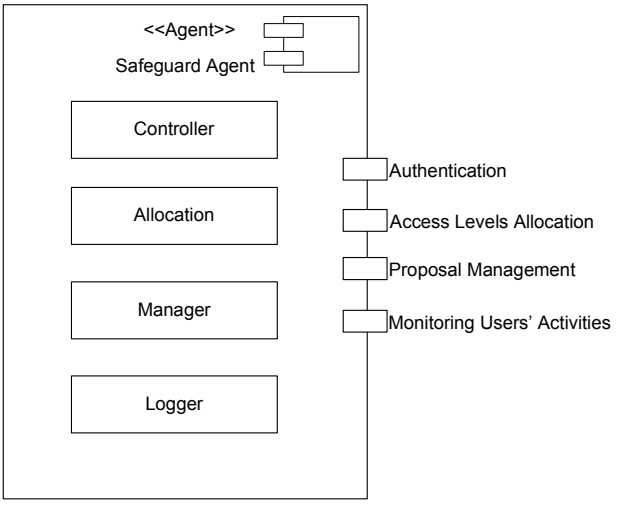


Fig. 12.45 Composite Structure Diagram for the safeguard agent

Figure 12.48 illustrates method lifting to define a composite class of the ontology agent. The ontology agent is defined by roles of navigation, query, instance knowledge manipulation, and domain knowledge manipulation, each of which is associated with its distinct interface. We specify these interfaces by the method lifting technique as shown in Figure 12.48. For example, the interface of component class navigation relates to the interface of a composite class ontology agent. Component classes of instance knowledge manipulation and domain knowledge manipulation are inherited from component class manipulation. The instance knowledge manipulation and domain knowledge manipulation interfaces of composite class ontology agent relate to the interfaces from component class manipulation.

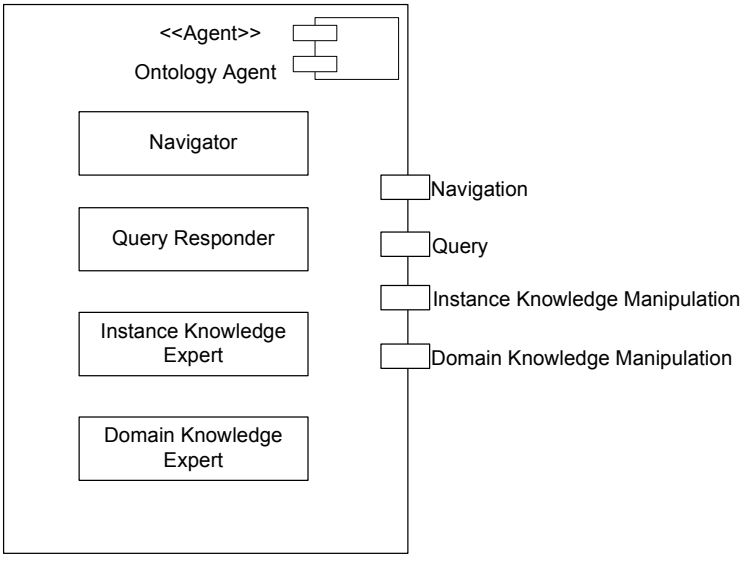


Fig. 12.46 Composite Structure Diagram for the ontology agent

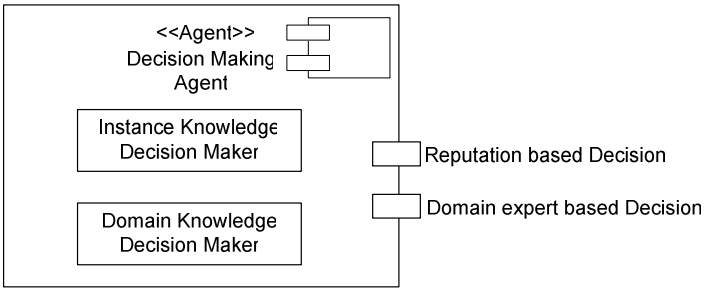


Fig. 12.47 Composite Structure Diagram for the decision making agent

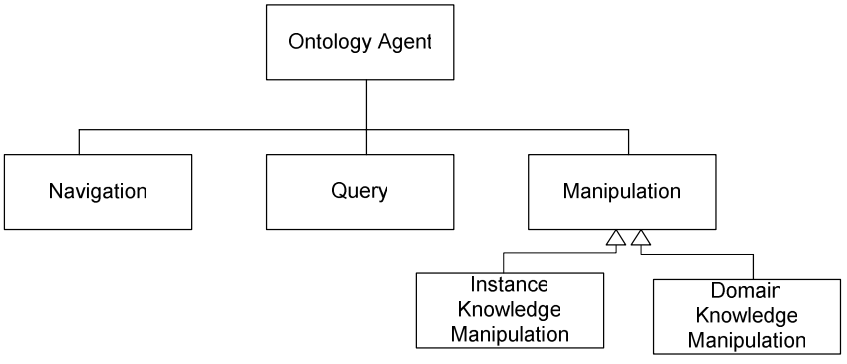


Fig. 12. 48 Method lifting for composite class Ontology Agent

12.8 Construction of Individual Agents

12.8.1 User Agents

A user agent is assigned to every team member that is logged into the system after authentication and provides different services to different access levels for each team member. All team members are provided with the service to query the SE Ontology. These access levels are given according to the different status of the team members. We also use the hierarchy of the software engineering sub-ontology categories to determine the access to the status of team members.

All agents' lifecycles reside on the server, and each team member is assigned to a user agent when a login is made. Each user agent is an initiator based on the user actions; the agent will carry out the specific operations accordingly. All the operations have different logic involved, but the structure of creating a user agent is the same. In the agent creation process, an agent object is created when a user login is carried out. In the agent listener process, the agent listens to any events based on the user actions. After a fixed waiting time expires, with no (other) actions from the team member, the agent will terminate itself. The agent will set up the required communication in the service creation process setting based on user operation e.g., setting the service type. Then the agent needs to find the identical service type in a directory which contains all the services published by agents that provide their services. The user agent will look up the directory for any services that match the one it is looking for. If the user agent cannot find the service, it deems that the service is currently unavailable and the operation will be terminated. In the process of message setup, the user agent creates a message object and sets up the content to be passed to the other agent. Normally it is a request for service. If the particular operation needs to communicate multiple times, or additional tasks need to be carried out, the message will then be setup again in a loop action. Once the goal has been achieved, the user agent will call logger to log all the activities. The

user agent will then kill itself if the team member decides to log off the system, or if the team member is idle for too long; otherwise, the agent will go back to the listener process.

This is the typical behaviour of goal specific agents, which exhibit one-shot behaviour. In other words, the agent is created for a purpose, and once the purpose has been achieved, the agent will be terminated. The life-cycle of the user agent listed above applies to all the operations that the user carries out. And if the user decides to use another operation, a new user agent will be created for that specific operation.

12.8.2 Safeguard Agents

To implement security features into the system, it has been decided to appoint another agent - in this case, the safeguard agent. The safeguard agent will be doing user authentication, authorisation and allocating access levels. The user identification will be verified with the user database as well as access level allocation.

Once the safeguard agent has been created, it cannot be terminated by normal means, and only the team leader/project manager has access to carry out platform maintenance and also, in certain circumstances, force the shutdown of the platform or kill any agent processes. This agent is the only one that connects user agents to the ontology agent; therefore, if it is killed or terminated, the system will not be functional at all. In the safeguard agent creation process, the agent is created when the platform is initialised and running. Then the services that are provided by the safeguard agent are initialised, set up and posted to the page service which provides a directory service for agents to publish their services. From this directory, all agents will search and find the service they are looking for. The safeguard agent will idle until another agent establishes a link or makes contact with it. Once a connection has been established, the service type is checked and, if it matches the service type the safeguard agent is providing, further processing will be carried out; otherwise the message is discarded and the request is rejected. There will be two possible cases. The first case is where a team member requests to view, add, and update the SE Ontology. In this case, the safeguard agent first needs to verify the team member account. If the team member has an authorisation, the safeguard will retrieve the user access level and then pass the request to the ontology agent. The second case is where the safeguard is requested by any agents to record team member activities. Because the safeguard agent is the only agent that connects to the user database, if there is any action to log into the database, it will be done by the safeguard agent. The safeguard agent will always be alive unless it is killed by human intervention.

12.8.3 Ontology Agents

The purpose of having an ontology agent is to manage connections with the SE Ontology. The ontology agent acts as a responder by publishing its services to the page service and allowing other agents who are looking for the service it is

publishing to communicate with it. As the ontology agent is a service provider, it does not terminate itself after finishing a communication with another agent. It will always be executed again whenever an agent communicates with it. This type of behaviour is classified as cyclic and the agent will terminate only if the platform is terminated or forced to be terminated through the GUI. In the ontology agent creation process, the ontology agent is created when the platform is initialised and running. Then the services that are provided by the ontology agent are initialised, set up and posted to the page service, which is the directory of the platform. From this directory, all agents will search and find the service they are looking for. The ontology agent will be idle until another agent establishes a link or make contact with it. Once a connection has been established, the service type is checked and, if it matches the service type that the ontology agent is providing, further processing will be carried out; if not, the message is discarded and the request is rejected. There will be two different services i.e., viewing and updating services. For the viewing services, the ontology agent simply carries out querying. With another service, the updating service, the ontology agent considers whether it is a minor change or major change. For a minor change, the ontology agent updates the ontology straightaway and also records the changes. For a major change, the ontology agent checks whether the update request had been authorised. Basically, for major changes, the ontology agent will pass the request of changes to the decision making agent to precede further processes, for example, gathering information and consulting the ontologies in the ontology repository. On passing through the decision making agent, the updating can be done by ontology agent. Every activity will be recorded by the call logger process. The results of the process are sent to the user agent that made the enquiry. The ontology agent will automatically wait for another connection and will be terminated only if it is forced to be terminated, either by a kill process or a platform collapse.

12.8.4 Decision Making Agents

As the name itself states, the job of the decision making agent is to make decisions on matters pertaining to requested major updates.

In the decision making agent creation process, the decision making agent is created when the platform is initialised and running. Then the services that are provided by the decision making agent are initialised, setup and posted to the page service, which is the directory of the platform. From this directory, all agents will search and find the service they are looking for. The decision making agent will idle until another agent establishes a link with it. Once a connection has been established, the service type is checked and if it matches the service type that the decision making agent is providing, further processing will be done; if not, the message is discarded and the request is rejected. Once the service has been activated, the decision making agent broadcasts messages to all team members. After that, the agent gathers responses and makes a decision on which possible solution will be finalised and be updated to the ontology. Updating the SE Ontology is done by the ontology agent after the decision making agent forwards the request to the ontology agent. Then the decision making agent sends the final decision

message to all team members and records this event. The process whereby the decision making agent makes a decision has been given in an earlier example. Also, the agent can decide to abort the update if no-one agrees to the requested update. The decision making agent will automatically wait for another connection and will terminate only if it is forced to be terminated, either by a kill process or a platform collapse.

12.9 Practical Uses

The practical uses of the systems are given in this section through examples. The text transcription is difficult because work is carried out in an environment where development teams are geographically distributed and team members are involved in many projects simultaneously. It is a typical means of global communication which is, however, neither efficient nor sufficient for a multi-site environment. Figure 12.49 shows such an example of the text transcription that, in a multi-site environment, makes less of an impression.

I am struggling to understand why we need it. I think the system will be simpler for people to understand if we deleted the insurance registered driver.
My reasons for this are that the insurance registered driver is a sub type of the customer. This means that for every insurance registered driver object there must be a corresponding customer object. However, in the customer object we store values like customer type, insurance history value and rental history value. It does not make sense to have these values for the insurance registered driver. I also think people will be confused because we have the rental registered driver as an association with the rental customer (which is a sub type of the customer) but the insurance registered driver is a sub type of the customer.

Fig. 12.49 An example of text transcription which is not efficient or sufficient for multi-site communication

With ontology-based software engineering, the software engineering terms can be parsed with software engineering ontology concepts and can recall the necessary details and relevant information. We see that it involves the terms of class (class insurance registered driver, class customer, and class rental customer), subclass (sub type), property (property customer type, property insurance history value, and property rental history value), and object (object insurance registered driver and object customer). Terms class, subclass, property, and object apply respectively to the concepts of class, generalisation relationship, class property, and class object in the software engineering ontology. By specifying ontology class instances, relevant information of those instances can be discovered dynamically and automatically. This is carried out by referring to software engineering ontology which asserts that concept class has its semantic of containing attributes, operations, and relationships holding among other classes. Automatically drawing out details facilitates others' greater understanding of the content, thereby reducing misunderstanding, and eliminating ambiguity.

For example, an issue may relate to a project design raised by a programmer in the implementation team. This is considered to be a major issue. The Navigation Platform is where the software engineer first identifies the involved data. The user raises the issue by fetching relevant instance knowledge from the Navigation Platform to the Question Platform. Figure 12.50 shows the class diagram on which the member raised the issue.

The Question Platform is where members raise problems they consider to be issues. Everyone can see the issue being raised. This allows brainstorming over the issue and possible solution proposals can then be made in the Suggestion Platform.

From here, there are two possible solutions proposed in the Suggestion Platform. Figure 12.51 shows the first solution proposal and Figure 12.52 shows the second solution proposal. As stated earlier, neither of these two potential solutions is yet to be a solution. Throughout the platform, worthwhile suggestions can be carried out by: deleting, adding, editing, or by a mix of, for example, deleting and then adding the instances.

As from Figure 12.51, the suggestion is to delete concept Class instance InsuranceRegisteredDriver. Another suggestion from Figure 12.52 is to first delete concept ClassAttribute instances RentalHistoryValue and InsuranceHistoryValue

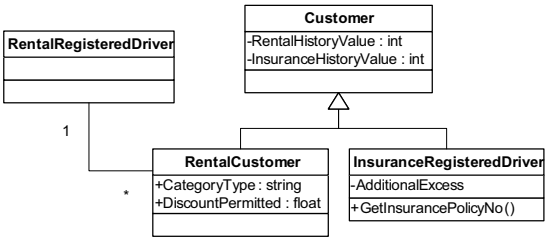


Fig. 12.50 Issue on an UML class diagram

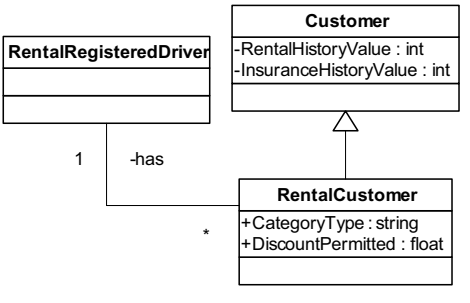


Fig. 12.51 The first solution proposal

relating relation has_Attribute with concept Class instance Customer. Secondly, new instance RentalHistoryValue is added for concept ClassAttribute relating relation has_Attribute with concept Class instance RentalCustomer. Thirdly, a new instance InsuranceCustomer is added for concept Class. Fourthly, new instances InsuranceCustomerID and InsuranceHistoryValue are added for concept ClassAttribute relating relation has_Attribute with concept Class instance InsuranceCustomer. Fifthly, a new instance is added for concept ClassGeneralisation relating relation Related_Object_Class_Component with concept Class instance InsuranceCustomer and relating relation Relating_Object_Class_Component with concept Class instance Customer. Lastly, a new instance is added for concept ClassAssociaton relating relation Related_Object_Class_Component with concept Class instance InsuranceRegisteredDriver and relating relation Relating_Object_Class_Component with concept Class instance InsuranceCustomer.

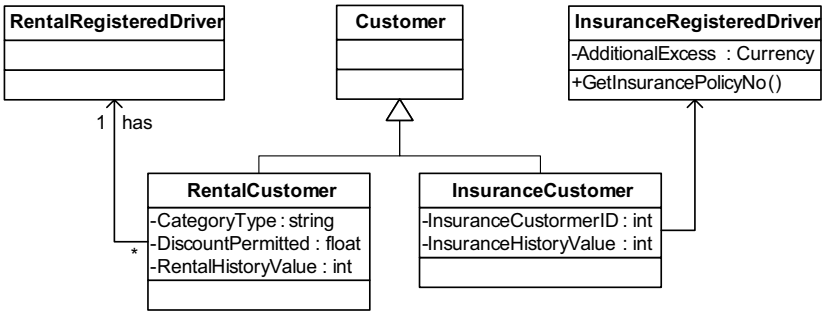


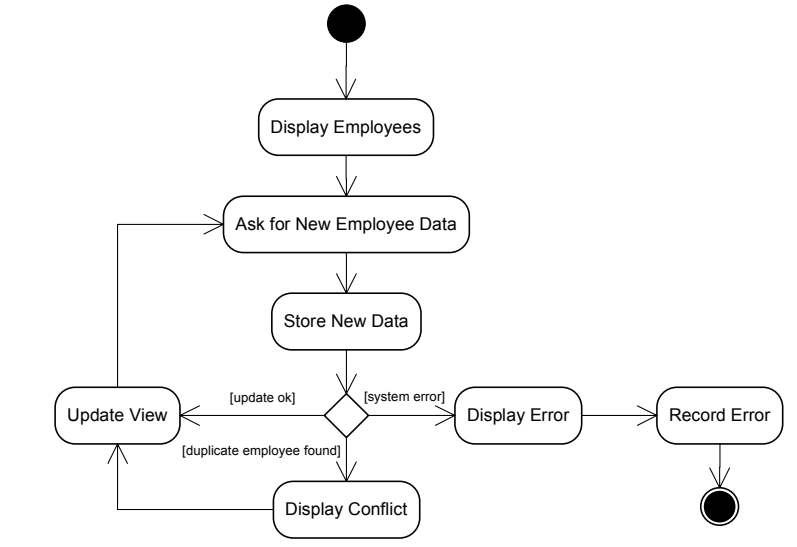
Fig. 12.52 The second solution proposal

The proposed changes become solution changes in the Solution Platform. Until such decision making systems in the Solution Platform determine the final solution, instance knowledge gets updated along the lines of the chosen solution.

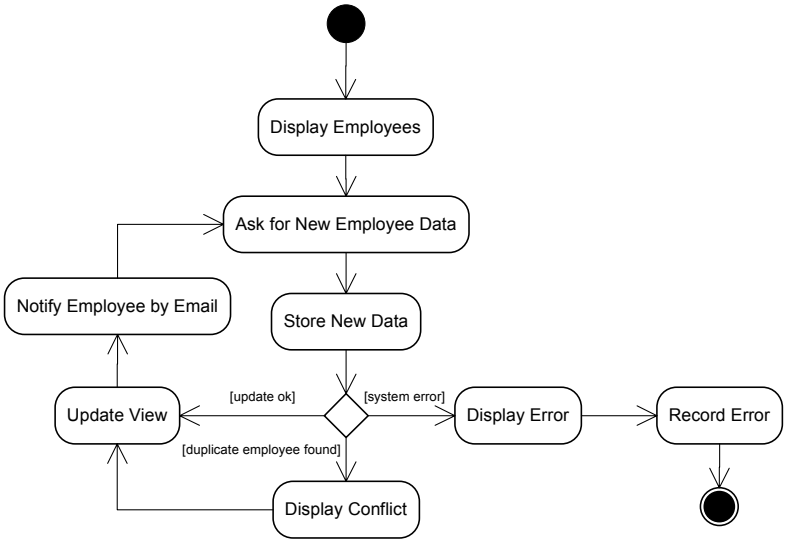
The following examples show the case of an issue raised by a member who has access. In this case, it is the member alone who determines whether the issue is major or minor. In the case of a minor issue identified by the member, it is simply a matter of updating project data through the Navigation Platform, and the Solution Platform retains and displays the record. This is no need to go through the Question and Suggestion Platforms because its purpose is to update in order to share instance knowledge.

Figure 12.53 (a) shows an original activity diagram, while Figure 12.53 (b) shows an updated activity diagram. The activity diagram used as the example here is derived from the book of Enterprise Java with UML (Arrington 2001).

As can be noted when comparing Figure 12.53 (a) and Figure 12.53 (b), the software engineer has revised the transition of activity 'Update View'. Originally, activity 'Update View' transited to activity 'Ask for New Employee Data'. Revision has been made by activity 'Update View' transited to activity 'Notify Employee by Email' and activity 'Notify Employee by Email' transited to activity 'Ask for New Employee Data'. Functioning is as following:



(a) An activity diagram



(b) Minor revised activity diagram

Fig. 12.53 Activity diagram revision in response to minor issue

- Delete concept NormalTransition instance that has relation Related_Activity with concept Activity instance named 'Ask for New Employee Data' and has relation Relating_Activity with concept Activity instance named 'Update View'.

- Add new concept Activity instance named ‘Notify Employee by Email’.
- Add concept NormalTransition instance that links relation Related_Activity with concept Activity instance named ‘Notify Employee by Email’ and links relation Relating_Activity with concept Activity instance named ‘Update View’.
- Add concept NormalTransition instance that links relation Related_Activity with concept Activity instance named ‘Ask for New Employee Data’ and links relation Relating_Activity with concept Activity instance named ‘Notify Employee by Email’.

In the Navigation Platform, an authorised member can directly update project data using add, delete, update, add new functions or else despatch this as a major issue using the fetch function to post to the Question Platform. In the case where a member considers an issue to be a major one, the process involves going through Navigation, Question, Suggestion, and Solution Platforms. For example, Figure 12.54 (a) shows an activity diagram which is the point at issue and Figure 12.54 (b) shows an activity diagram that is ultimately suggested to resolve the issue. The activity diagram used as an example here is derived from the book of Enterprise Java with UML (Arrington 2001).

A member starts progressing towards a resolution of the issue by fetching involved instances from Navigation Platform to Question Platform. Consequently, in Suggestion Platform a dummy solution is suggested. After brainstorming, the solution is given in the Solution Platform and the ontology gets updated. Along the line of the Solution Platform in this example, the list of functions is as follows:

- Update BranchTransition instance linked to Related_Branch_Activity_1 instance named ‘Display Error’ which the Guard_Expression_1 instance is changed to ‘error found’.
- Add new Activity instance named ‘Ask if Ready to Submit Timecard’.
- Update BranchTransition instance linked to Related_Branch_Activity_2 instance which is Stop instance. Delete the Stop instance and add the Activity instance named ‘Ask if Ready to Submit Timecard’.
- Add new Activity instance named ‘Build a New Current Timecard’.
- Add new BranchTransition instance which links Relating_Branch_Activity_1 instance to Activity instance named ‘Ask if Ready to Submit Timecard’, links Related_Branch_Activity_1 instance to Activity instance named ‘Build a New Current Timecard’, and links Related_Branch_Activity_2 instance to Stop instance. Its Guard_Expression_1 is a string of ‘yes’ and its Guard_Expression_2 is a string of ‘no’.
- Add Stop instance that links Relating_Special_Activity named ‘Build a New Current Timecard’.
- Add another Stop instance that links Relating_Special_Activity named ‘Display Error’.

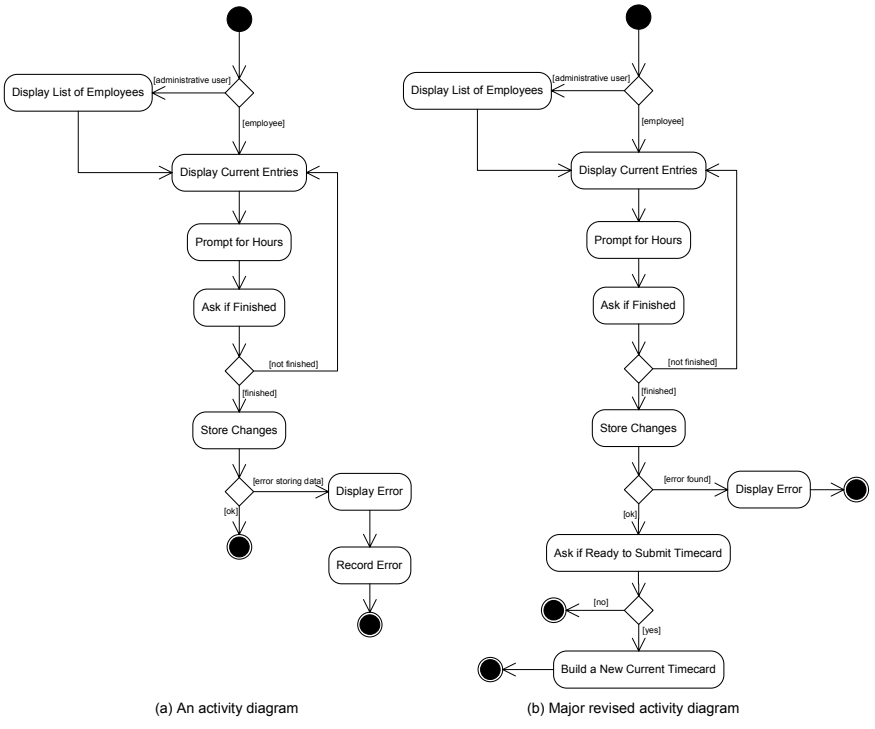


Fig. 12.54 Activity diagram revision in response to major issue

12.10 Conclusion

In this chapter, we explored a case study on the SE Ontology and a multi-agent system in which agents interact and mediate with the SE Ontology. The purpose of this is to develop multi-site software as a communication framework. We also illustrated some examples of the practical applications.

References

1. Arrington, C.: Enterprise Java with UML. John Wiley & Sons, Inc., New York (2001)
2. Bourque, P.: SWEBOK Guide Call for Reviewers (2003)
3. Bourque, P., Dupuis, R., Abran, A., Moore, J.W., Tripp, L.: Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Press, Los Alamitos (2004)
4. Davenport, T.H., Prusak, L.: Working Knowledge: How Organisations Manage What They Know. Harvard Business School Press, Boston (1998)
5. Quatrani, T.: Visual Modeling With Rational Rose and UML. Addison-Wesley, Reading (1997)
6. Quatrani, T.: Visual Modelling with Rational Rose and UML. Addison-Wesley, Reading (1998)
7. Sommerville, I.: Software Engineering. Pearson Education Limited, London (2004)

8. Wongthongtham, P.: A methodology for multi-site distributed software development. School of Information Systems. Curtin University of Technology Perth (2006)
9. Wongthongtham, P., Chang, E., Dillon, T.: Enhancing Software Engineering Project Information through Software Engineering Ontology Instantiations. In: The 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006). IEEE Computer Society Press, Hong Kong (2006)
10. Wongthongtham, P., Chang, E., Dillon, T.: Software Design Process Ontology Development. In: Second IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS 2006). LNCS. Springer, Montpellier (2006)

Chapter 13
Potential Applications of Ontology-Based Multi-Agent Systems

13.1 Overview

In this book, we introduced multi-agents systems in Chapter 2 and ontologies in Chapter 3. Many advantages of ontology-based, agent-based and the integrated ontology/agent-based systems are described in Chapter 6. We described a number of existing design methodologies for agent-based systems in Chapter 4 and ontology-based systems in Chapter 5. We have developed our own design methodology by examining the strengths and weaknesses of the existing design methodologies. This new design methodology consists of two parts: the ontology design part which is described in Chapter 7, and the multi-agent design part, described in Chapter 8. The notation for ontology-based multi-agent systems is defined in Chapter 9, while the system architecture and system implementation are described in Chapters 10 and 11. The implementation of ontology-based multi-agent system in the domain of human diseases and the software engineering domain is described in Chapters 12 and 13 respectively.

In this chapter, we will highlight the three main application areas (Collaborative Environments, Information Access and Retrieval; and Data Mining). We will also discuss some open issues related to the implementation of the ontology-based, multi-agent systems.

13.2 Collaborative Environments

We take an example from the mental health domain. Identification of the precise patterns of causal factors responsible for a specific type of mental illness still remains unsolved and is therefore a very active research focus today. As the different research teams work on different aspects of mental illness, research into mental illness requires collaboration and sharing of information between the different research teams. Usually, each research team focuses on only one factor and perhaps one aspect of a mental illness. For example, in the paper "Bipolar disorder susceptibility region on Xq24-q27.1 in Finnish families" (PubMed ID: 12082562), the research team Ekholm et al. examined one genetic factor (Xq24-q27.1) for one type of mental illness (bipolar disorder). As mental illnesses do not follow Mendelian patterns, but are caused by a number of genes usually interacting with various environmental factors, all factors for all aspects of the illness need to be

M. Hadzic et al.: Ontology-Based Multi-Agent Systems, SCI 219, pp. 271–273.
springerlink.com

considered. This requires the different research teams to share complementary information and collaborate with each other, jointly building a solution to their shared problem.

A Mental Health Ontology can be designed to capture and represent mental health knowledge. This ontology is to be shared between the different research teams, and various agents can be designed to enable management, access, sharing, analysis and exchange of the information shared between the different research teams. The integrated ontology/agent-based systems will create a cooperative environment, allowing coherence between different research teams in big research tasks. It will also be easier to ascertain and analyse any gaps that still exist in the research pertaining to the mental health domain. Research projects (past and present) would be mapped onto the mental health domain as specified by the ontology. During the planning stage all research projects could use these maps to see where overlaps, redundancies, complementary research and (more importantly) no research has occurred or is occurring.

13.3 Information Access and Retrieval

Let us again consider example from the mental health domain. Yes, we can design an integrated ontology/multi-agent system that will help the different research teams to collaborate with each other, but what can we do with the existing pool of mental health information dispersed over the Internet and over various databases? The size of the existing corpus of knowledge is so large and the possibility of searching it effectively is very low. In most cases, some important information has been neglected. We need to take a systematic approach to making use of an enormous amount of available information that has no value unless analysed and linked with other available information from the same mental health domain. We need to design an information retrieval system to help us maximize the value of the existing mental health information. In Chapter 12, we described the design of a similar system.

The Mental Health Ontology that provides a model of mental health concepts and relationships can be used to form a semantic framework to support data storage and retrieval tasks. Such a semantic framework can be used by annotation agents to systematically annotate mental health information available through various information resources. This will enable the information agents to understand the content of the information resources and retrieve relevant information, searching for the meaning of information beyond its appearance in the text.

13.4 Data Mining

Data mining involves sophisticated data analysis techniques and can be used to perform automated searches for actionable knowledge buried within a huge body of data. Data mining extracts information and finds patterns and behaviours embedded in the data. Usually, the data mining results increase understanding and help to provide predictive models for decision making and new discoveries.

In our example of mental health, data mining algorithms have great potential to expose the patterns in mental health data. This will facilitate the search for the combinations of genetic and environmental factors involved and provide an indication of influence. This knowledge has the potential to improve the infrastructure for evidence-based interventions, assist in the prevention, diagnosis, treatment and control of mental illness, and provide innovations for improvement in mental health care.

13.5 Open Issues

Ontology-based multi-agent systems are complex systems. This complexity has to stay behind the screen, and the users should simply see answers targeted in response to their questions. In general, users are not interested in the system itself or how it works, but in the answers it can provide. If the complexity of the system is mirrored on the screen, the users will avoid using such systems.

Depending on the scope of the system, it is required to comply with the local, national and/or international regulations. In the case where a distributed system has been designed with individual components located in different countries, an appropriate set of regulations needs to be carefully studied and addressed for each country.

No computer system has been designed yet to completely replace human thinking and actions. Humans need to take control when the computer fails to deal with a situation. The users need to be aware of this fact and consider the reliability and trustworthiness of the information provided to them, especially in the cases where small errors or misunderstanding can lead to catastrophic consequences.