A Requirements Elicitation Framework for Agent-Oriented Software Engineering

Richard Hill

A Thesis Submitted in Partial Fulfilment of the Requirements of

Sheffield Hallam University

for the Degree of Doctor of Philosophy

14th December, 2006

Abstract

The hypothesis of this research is as follows: "Conceptual modelling is a useful activity for the early part of gathering requirements for agent-based systems."

This thesis examines the difficulties of gathering and expressing requirements for agent based systems, and describes the development of a requirements elicitation framework. Conceptual modelling in the form of Conceptual Graphs is offered as a means of representing the constituent parts of an agent-based system. In particular, use of a specific graph, the Transaction Model, illustrates how complex agent concepts can be modelled and tested prior to detailed design specification, by utilising a design metaphor for an organisational activity.

Using an exemplar in the healthcare domain, a preliminary design framework is developed showing how the Transaction Agent Modelling (TrAM) approach assisted the design of complex community healthcare payment models. Insight gained during the design process is used to enrich and refine the framework in order that detailed ontological specifications can be constructed, before validating with a mobile learning scenario. The ensuing discussion evaluates how useful the approach is, and demonstrates the following contributions:

- Use of the Transaction Model to impose a rigour upon the requirements elicitation process for agent-based systems;
- Use of Conceptual Graph type hierarchies for ontology construction;
- A means to check the transaction models using graphical inferencing with Peirce Logic;
- Provision of a method for the elicitation and decomposition of soft goals;
- The TrAM process for agent system requirements elicitation.

Acknowledgments

Of course a piece of work this size could not have been completed without the help of others. In particular my supervisory team, Allan Norcliffe (DoS) and Paul Crowther (Supervisor) provided the day-to-day support and encouragement, as well as Simon Polovina, whom I publish with. Gary Swann provided invaluable assistance with proof-reading. I must also recognise my long-suffering work colleagues in the 'snake pit', Bob Drake, Hugh Lafferty and Peter Lake; their collective acidic wit and blunt speaking kept me sane during the tough times. Mo Rezai and Dharmendra Shadija provided undying support and constant entertainment. Paul Henderson allowed me to moan without interruption. Other smaller (but no less significant) contributions were made by Ian Morrey and Pete Collingwood. This research would not have continued without the countless discussions with 'agent-people' such as Simon Thompson, Antonio Moreno, Sue Greenwood and co. at Oxford Brookes, Steve Willmott and Julian Padgett. Although intensely irritating, some thanks is due to the vast number of friends and family that kept asking "*have you finished your PhD yet?*". Finally, I must thank 'the Wife' who relentlessly reminded me of the need to complete the thesis and return our lives to some degree of normality.

Contents

1 Introduction and Motivation for Research 1

1.1 Introduction 1

1.2 Motivation 1

1.2.1 Hypothesis 6

1.3 Research Approach 7

1.4 Contributions 7

1.5 Overview of Thesis 8

1.6 Prior Work 9

2 Agents and Agent-Oriented Software Engineering 13

2.1 Introduction 13

2.2 Intelligent Agents Explained 13

2.2.1 Agents and Objects 16

2.2.2 What is a Multi-Agent System? 20

2.3 Communication 21

2.3.1 Speech Acts 21

2.3.2 Agent Communication Languages (ACL) 23

2.4 Ontologies 25

2.4.1 Syntactic Interoperability 27

2.4.2 Semantic Interoperability 28
2.4.3 Communicating Intentions 30
2.5 Agent-Oriented Software Engineering 33
2.6 Design Methodologies 34
2.6.1 Gaia 35
2.6.2 Prometheus 41
2.6.3 Tropos 45
2.7 Discussion 51
2.8 Criteria for Framework 54
2.9 Conclusions 59

3 Modelling Concepts 61
3.1 Introduction 61
3.2 Capturing Domain Knowledge 61
3.3 Conceptual Graphs 63
3.3.1 Notation 63
3.3.2 Concepts 67
3.3.3 Relationships 69
3.4 Ontology 73
3.4.1 Types 73
3.4.2 Defining Types 74
3.4.3 Type Hierarchies 75
3.4.4 Lambda Expressions 76
3.4.5 Coreference Links 77
3.4.6 Projection 77
3.4.7 Predicate Calculus 81
3.4.8 Inferencing 81

3.5 Discussion 86

3.6 Conclusions 87

4 A Unifying Framework 88

4.1 Introduction 88

4.2 The Need for a Metaphor 89

4.2.1 Complex Systems 89

4.3 Event Accounting 90

4.3.1 Modelling an Enterprise 93

4.4 Transaction Agent Modelling (TrAM) 95

4.4.1 Capture Scenarios 97

4.4.2 Identify Agent Roles 97

4.4.3 Allocate Tasks to Agents 97

4.4.4 Identify Collaborations 97

4.4.5 Apply Transaction Model 98

4.4.6 Design Artefacts 98

4.5 Discussion 100

4.6 Using TrAM 104

4.7 Background 104

4.8 Rationale for Choice of Domain 104

4.9 A Community Healthcare Case Study 105

4.9.1 Developing an Agent-based Approach 107

4.9.2 Designing Community Care Systems 109

4.9.3 Building the Model with TrAM 112

4.9.4 Limitations of the Approach 130

4.10 Conclusions 132

5 Refining the Framework **136**

5.1 Introduction 136

5.2 The TrAM Process 136

5.3 Issues with TrAM 142

5.4 A Recap of Agent BDI Concepts 144

5.4.1 Norms and Policies 146

5.5 Towards a Refined Framework 147

5.5.1 A Transaction Ontology 148

5.6 An Improved Process 151

5.7 Discussion 155

5.7.1 Desired Characteristics 155

5.8 Conclusions 161

6 Applying TrAM to MOBIlearn **162**

6.1 Introduction 162

6.2 Rationale for Case Study 163

6.3 MBA Scenario 164

6.3.1 Capture Concepts 164

6.3.2 Transform with TM 172

6.3.3 Gather Use Cases 179

6.3.4 Verify TM Graphs 181

6.3.5 Allocate Agents 187

6.4 Conclusions 188

7 Conclusions and Further Work **190**

7.1 Hypothesis 190

7.2 Research Approach 191

7.3 Contributions 193

7.4 Further Work 196

References **199**

A OWL Listings **224**

A.1 Belief-Desire-Intention Ontology 224

A.2 Transaction Model Ontology 225

B MOBIlearn Case Study **234**

B.1 MOBIlearn Case Study Background 234

B.1.1 Objectives 237

B.1.2 On Pedagogical Issues 238

B.1.3 On Human Interaction and Technical Issues 238

B.1.4 On Business Issues 238

B.2 Case Study: Description of MBA Use-Case Scenario 241

List of Figures

2.1 The Gaia Methodology Models (redrawn from Wooldridge et al., 2000). 37

2.2 Concepts within the Analysis stage of Gaia. 38

2.3 Role Schema for Coffee-filler (Wooldridge et al., 2000). 39

2.4 The Prometheus Methodology (Padgham and Winikoff, 2002). . 42

2.5 Goal-plan tree showing goal decomposition 48

3.1 'A relation of Concept_1 is Concept_2' graph in display form. . . 64

3.2 A part of a bicycle is a wheel. 64

3.3 A bicycle is on the ground. 64

3.4 Lattice diagram of an example ontology. 68

3.5 Example of triadic relation. 72

3.6 DF Graph illustrating coreferent links between graphs. 78

3.7 First graph to be joined. 79

3.8 Second graph to be joined. 80

3.9 Possible projections. 80

3.10 Common generalisation. 81

3.11 Common specialisation. 81

3.12 An extended common specialisation resulting in a *maximal join.* 82

3.13 'If Graph A then Graph B' 82

3.14 Deiterated graph 83
3.15 Denegation 83
3.16 Original graphs 84
3.17 Specialised graphs 84
3.18 Deiterated graphs 85
3.19 Denegation 85

4.1 The Transaction Model (TM) Graph. 94
4.2 The TrAM Framework. 96
4.3 TrAM Process in Detail. 99
4.4 The Transaction Model (TM). 112
4.5 Use case model for maintaining the ICP. 114
4.6 Use case model for positive care. 115
4.7 Use case model for daily care. 117
4.8 Emergency scenario use case model. 118
4.9 Quality assurance use case model. 119
4.10 Overview of care model. 121
4.11 Initial actor to agent mappings. 121
4.12 Agent collaboration model. 123
4.13 Iterated agent collaboration model. 125
4.14 CG Model of Community Care Scenario. 125
4.15 CG Model of Generic TM. 126
4.16 Overall Transaction Model of care scenario. 126
4.17 Initial type hierarchy of care scenario. 127
4.18 Local Authority pays for care in full. 129
4.19 TM showing care recipient receiving care package at zero cost. . 130
4.20 Care recipient pays for care in full. 131

4.21 Emergent CG model. 132
4.22 Updated TM showing care recipient receiving care package at full cost. 133
4.23 Incomplete TM. 133
4.24 Part payment scenario with shared liabilities for care cost. . . . 134
4.25 New rule for ontology. 134
4.26 Revised type hierarchy. 134
4.27 Refined payment model. 135

5.1 The TrAM Framework. 141
5.2 CG Representation of a BDI Agent. 146
5.3 Type Hierarchy of BDI concepts (Absurdity Type omitted). . . 147
5.4 Visualisation of OWL file, translated from the type hierarchy. . 148
5.5 Amended type hierarchy to include `Policy` concept. 149
5.6 Type hierarchy from the generic Transaction Model. 149
5.7 Refined TrAM Type Hierarchy. 150
5.8 Visualisation of TM ontology. 151
5.9 Example of a display form graph to be parsed into natural language. 152
5.10 Specialised healthcare TM graph. 153

6.1 Modelling a MOBIlearn objective with CGs. 166
6.2 The M-Learning platform's duty of care towards students. . . . 166
6.3 CG representing an MBA student. 167
6.4 CG model of the MOBIlearn service characteristics. 168
6.5 Initial overall model Part 1. 169
6.6 Overall model part 2 (note coreferent links to Part 1 model). . . 169

6.7 Some high level desires of the stakeholders. 170
6.8 Iterated CG model of Student desires. 171
6.9 Iterated CG model of Tutor desires. 172
6.10 Exploring the *enjoy course* goal. 174
6.11 The generic Transaction Model. 175
6.12 MOBIlearn scenario Transaction Model. 176
6.13 Amended Transaction Model. 176
6.14 A trade-off between studying and spending time with the family. 177
6.15 Capturing a student transaction. 177
6.16 Type hierarchy from TM. 178
6.17 MBA top-level scenario use case model. 180
6.18 Updated TM graph from MBA use cases. 181
6.19 Refined model of Student desires. 182
6.20 The MOBIlearn system employs a pedagogical approach to facilitate mobile learning. 184
6.21 Students may participate as individuals, as a member of a group or both. 184
6.22 All learning content must be administered and managed remotely. 185
6.23 The Local Authority pays for the education in full where it is deemed that the student is eligible. 185
6.24 The Student pays for the education in full where it is deemed that the Student is ineligible for financial assistance. 186
6.25 A Student may be eligible for financial assistance if they are female and between the ages of 18 and 65 years old. 186
6.26 The eligibility is determined by reference to current educational policy. 187

List of Tables

2.1 Evaluation rankings (Sturm and Shehory, 2003). 60

2.2 Evaluation of agent design methodologies against desired characteristics. 60

3.1 Reading conceptual graphs from left to right. 65

3.2 Reading conceptual graphs from right to left. 66

4.1 Agent types and allocated tasks 122

4.2 Iterated agent types and allocated tasks. 124

5.1 Evaluation of TrAM against desired characteristics identified in Chapter 2. 160

6.1 Some of the high-level stakeholder goals from the CG models. . 173

6.2 MOBIlearn agent roles. 188

6.3 Task allocation for the Student Agent. 189

Chapter 1

Introduction and Motivation for Research

1.1 Introduction

This chapter introduces the motivation for the research and identifies the research hypothesis. Existing work is briefly introduced, highlighting the limitations of current approaches to requirements capture. The research approach is described, followed by an overview of the remainder of the thesis.

1.2 Motivation

Multi-agent System (MAS) architectures are used to build complex systems, which often comprise many autonomous entities that communicate across multiple organisational tiers. Gathering requirements for such systems is a challenge. The MAS paradigm appears however, to make this simpler since the more comprehensive abilities of agents are easier to map to real-world actors. Similarly it is possible to map the aspirations, intentions and beliefs of individual actors, thus creating constraints that become part of the design specification for each agent. This simplifies the process of gathering requirements by moving the model nearer to reality, reducing the need for functional decomposition from the outset.

In practice the process of requirements gathering for agent based systems is not simple and it is common for agent systems to be modified post-model creation in order to achieve the requirements of the relevant stakeholders. This gap between understanding of the system (the model representation) and implementation (program code) is not uncommon, and is a continuing challenge for software engineering in general.

The collection of data pertaining to processes and specific terminology is normally conducted with the assistance of domain expertise. MAS architectures must be able to communicate freely, employing communicative acts as a fundamental part of their collaboration mechanism. Agent Communication Languages (ACL) typically comprise a performative and some message content that must be represented in a way that can be understood by potential agent collaborators. The key to a common, shared understanding of knowledge in a particular domain is by the use of a description of the concepts within a particular domain, or an *ontology*. Consequently, any ACL must make use of an ontology in order to enable communication between different parties, ensuring what was said is what was meant.

One aspect that proves particularly difficult is the generation of the ontology. It would seem that ontology creation requires a significant input from domain experts and the design models need to be iterated in order to develop the ontology to a more comprehensive state. It should be noted that an ontology comprises not only domain specific concepts (and their associated terminology), but also the relationships between those concepts plus any domain constraint rules.

Methodologies and tools tend to require an ontology as an input into their respective methods. Tools in particular can then use the ontology to check the models that are developed against a conceptual representation. Unfortunately the amount of effort and expertise required to generate the ontology in the first instance is considerable and therefore it would be helpful if a method existed to assist this first step. If it was possible to generate even a rudimentary ontology from the outset then existing tool-based methods for MAS modelling would be better supported.

Of course ontology generation is not straightforward, and whilst MAS architectures seem easier to map to real systems, the complexity lies in the ontological representation of that knowledge.

Once an ontological representation has been produced, it is prudent to verify the domain concepts and relations, typically utilising the services of a domain expert. This activity is also fraught with difficulties as it is likely that the representation of the ontology will not be familiar to the domain expert and thus some transformation is needed in order that the domain expert can concentrate on verifying the model. Since the resources of a domain expert are generally regarded as scarce, it would be advantageous if the demands upon such a role were minimised.

When considering the domain specific terms, there also exists the complexity of qualitative concepts. Unlike quantitative concepts, which can be measured, qualitative concepts have not yet evolved into measurable entities. For instance consider the goal 'maintain quality of life'. How can this be considered by a MAS? In this case the ontology requires some work before such a qualitative issue can be expressed and understood quantitatively.

It also follows that the MAS might be designed differently if an ontology existed prior to modelling; indeed the fact that systems are modified after initial modelling suggests that the current methods are flawed. If the ontology could be generated earlier, then it would seem reasonable to assume that fewer modifications to the system would be required post design specification.

Therefore, in order to generate the ontology earlier, there needs to be a framework that can:

1. Capture fundamental domain concepts whilst minimising the use of a domain expert;

2. Expose qualitative issues much earlier in the process, in order that they might be quantified later;

3. Produce representations that can be tested prior to design specification and;

4. Represent complex qualitative issues in a repeatable way.

A key challenge for an improved agent design framework is the ability to capture domain knowledge in a way that faithfully represents the needs of the intended system, whilst permitting the expression of that knowledge in the widest sense possible. Since ontologies can assist the design of new applications, be it through the process of capturing domain knowledge or the sharing and re-use of existing domain ontologies, it seems prudent to consider the development of such a framework.

Furthermore, 'early' requirements capture is important as it contains the high level goals (hard and soft) of the stakeholders. Conventional approaches to modelling, with the subsequent modelling iterations, can dilute these goals (desires) to the point where they lose importance. The capture and expression of high-level concepts is therefore fundamental to the requirement for a more faithful representation.

Whilst it is feasible that much of this work can be performed manually by the agent system designer, the potential complexity of these systems is such that it is inevitable that inconsistencies will present themselves. Therefore it is necessary to consider processes that support either the automation of tasks, or the individual steps are able to implicitly build a rigorous model. This would assist the agent system designer considerably, and reduce the reliance upon domain experts.

It follows that there is a need for a modelling environment which:

1. Utilises a notation that is rich, expressive and can tolerate both quantitative and qualitative high-level domain concepts;
2. Provides a mechanism whereby models can be queried, reasoned against and verified;
3. Supports the implicit capture and explicit expression of ontological data;
4. Imposes a rigour upon the modelling process.

This supports a tool-based approach to MAS modelling as it would assist the initial (and currently 'pre') requirements gathering stages by creating an ontology that could subsequently be used for automated model-checking. It would also enable higher-level issues to be discussed and debated much earlier. It is feasible that high-level goals are not captured and represented correctly and therefore compromised by a system implementation. Thus the motivation for this research is described.

1.2.1 Hypothesis

The hypothesis of this research is: "Conceptual modelling is a useful activity for the early part of gathering requirements for agent-based systems." For the purposes of this thesis, 'usefulness' is characterised by the following:

1. An opportunity to reduce the need for input from domain experts;
2. A means by which system models are tested earlier in the requirements capture process;
3. An ability to capture abstract domain terms as concepts;
4. The elicitation of an ontology that reflects the domain more faithfully;
5. An approach that complements other MAS design methodologies and;
6. An approach that is sufficiently abstract to be generally applicable in the wider context.

The use of the TrAM framework illustrates how high-level concepts can be captured in the community healthcare and m-learning domains, and demonstrates the process by which qualitative concepts are quantified and used to populate a hierarchy of types prior to ontology generation. From the earliest stage, concept types, relations and domain terms can be qualified with domain experts. TrAM offers the significant advantage of being able to focus in on areas that require concentrated analysis, thus guiding the agent system analyst, whilst also concentrating the efforts of the domain expert. The capture, representation and subsequent analysis of early requirements is also supported by TrAM, and since the framework explicitly supports BDI concepts the resulting design artefacts can be used as a precursor to detailed implementation with existing agent design methodologies. Finally, the TrAM approach conveniently uses a transaction metaphor that is sufficiently abstract to be domain independent. As such, it is established that conceptual modelling is a useful activity and therefore the hypothesis is believed to be true.

1.3 Research Approach

The research combines the characteristics of the case study approach with those of action research. Initially an in-depth study of a complex scenario in the community healthcare domain is used to develop a draft requirements elicitation framework. A second case study in a disparate domain (m-learning) is then used in order to:

- provide new insight and refine the proposed framework;
- communicate the process undertaken whilst applying the framework;
- demonstrate the characteristics of the framework that are generally applicable, and identify domain specific aspects of the framework.

Whilst the application of the framework to a domain is in itself a contribution, it is also recognised that there is significant benefit in terms of rigour to be gained from documenting and monitoring the process of applying the framework to a second domain.

1.4 Contributions

The primary contributions of this research are as follows:

- Use of the Transaction Model to impose a rigour upon the requirements elicitation process for agent-based systems;
- Use of Conceptual Graphs type hierarchies for ontology construction;
- A means to check the transaction models using graphical inferencing with Peirce Logic;
- Providing a method for the elicitation and decomposition of soft goals;
- The TrAM process for agent system requirements elicitation.

1.5 Overview of Thesis

Chapter 2 establishes some basic agent concepts before examining the current literature in relation to existing Agent-Oriented Software Engineering (AOSE) approaches. Requirements capture for AOSE is introduced, and considered in relation to three Agent-Oriented design methodologies. The limitations of each of the approaches are briefly described and the basic criteria for a design framework is introduced, upon which the rest of the research is based.

Chapter 3 explores the use of conceptual modelling for AOSE and introduces Conceptual Graphs (CG) as a notation for gathering agent system requirements. The formal underpinnings of CGs are explained and type hierarchies are used to describe the concepts and relations in a domain in order to generate an ontology. Finally, inferencing using Peirce logic is utilised to test conceptual models prior to detailed design specification.

Chapter 4 looks at some theoretical foundations upon which an improved requirements elicitation design framework might be based. Event accounting is explored and offered, through the Transaction Model (TM), as a means by which conceptual models can be queried and tested during the requirements gathering process. Additionally the TM is used to illustrate how domain ontologies can be derived from CG Type Hierarchies and how a unified, robust approach to model creation and checking assists AOSE. Chapter 4 introduces the Transaction Agent Modelling (TrAM) Framework and describes its use by way of an exemplar case study in the community healthcare domain. In particular the complexities of healthcare payments are examined and the framework demonstrates the ease with which this complex problem was modelled and tested prior to design specification. Additionally the case study illustrates limitations of the framework and provides an opportunity to refine the process accordingly in Chapter 5.

After the framework has been developed further in Chapter 5 it is then applied to MobiLearn, an EU Funded project in the m-learning domain in Chapter 6. The results illustrate the extent to which each of the key criteria identified in Chapter 2 are addressed. Areas of generic applicability are identified, as are domain specific aspects of the modelling process. Chapter 7 explores the results and establishes commonality between the two disparate domains, identifying the elements of the framework that are generally applicable, prior to a discussion of the areas for future development.

1.6 Prior Work

Elements of this thesis have been published in the following:

- Hill, R., (2007). "Capturing and Specifying Multi-agent System Requirements for Community Healthcare" in In H. Yoshida, A. Jain, A.

Ichalkaranje, L. Jain, and N. Ichalkaranje, eds., "Advanced Computational Intelligence Paradigms in Healthcare", Volume 48 of *Studies in Computational Intelligence*, Chapter 6, Springer-Verlag, Berlin, pp. 121-158.

- Hill, R., Polovina, S., & Shadija, D. (2006). "Transaction Agent Modelling: From Experts to Concepts to Multi-agent Systems", In Proceedings of the Fourteenth International Conference on Conceptual Structures (ICCS '06): Conceptual Structures: Inspiration and Application, July 16-21, Aalborg, Denmark, *Lecture Notes in Artificial Intelligence* (LNAI) 4068, Springer-Verlag, Berlin, pp. 247-259.

- Hill, R., Polovina, S., & Beer, M. D., (2006). "Improving AOSE with an Enriched Modelling Framework", In Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering (AOSE-2005), Utrecht University, The Netherlands, *Lecture Notes in Computer Science* (LNCS) 3859, Springer-Verlag, Berlin.

- Hill, R., Polovina, S., & Beer, M. D. (2005b). "Managing Community Healthcare Information in a Multi-agent System Environment". In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05) - BIOMED Workshop, Utrecht University, The Netherlands.

- Hill, R., Polovina, S., & Beer, M. D. (2005a). "From Concepts to Agents: Towards a Framework for Multi-agent System Modelling". In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05). ACM Press, Utrecht University, The Netherlands, pp. 1155-1156.

- Hill, R., Polovina, S., & Beer, M. D. (2005). "Managing Healthcare Workflows in a Multi-agent System Environment", In Proceedings of the Agents Applied in Healthcare Workshop, Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), University of Edinburgh, Scotland, UK, 9 pages.

- Polovina, S., Hill, R., & Beer, M. D. (2005). "Enhancing the Initial Requirements Capture of Multi-Agent Systems through Conceptual Graphs". In Pfeiffer, H., Wolff, K. E., & Delugach, H. S., eds., Conceptual Structures at Work: Contributions to ICCS 2005 (Thirteenth International Conference on Conceptual Structures), *Lecture Notes in Artificial Intelligence* (LNAI), Springer-Verlag, Berlin.

- Hill, R., Polovina, S., & Beer, M. D. (2004). "Towards a Deployment Framework for Agent-Managed Community Healthcare Transactions". In The Second Workshop on Agents Applied in Health Care, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 13-21. ECCAI, IOS Press, Valencia, Spain.

- Polovina, S., Hill, R., Crowther, P., & Beer, M. D. (2004). "Multi-agent Community Design in the Real, Transactional World: A Community Care Exemplar", In Conceptual Structures at Work: Contributions to ICCS 2004 (Twelfth International Conference on Conceptual Structures), Pfeiffer, H., Wolff, K. E., Delugach, H. S., eds., Shaker Verlag (ISBN 3-8322-2950-7, ISSN 0945-0807), pp. 69-82.

- Huang, W., Beer, M. D., & Hill, R. (2003). "Community Care System Design and Development with AUML", 9th International Conference on

Information Systems Analysis and Synthesis (ISAS '03), July 31-August 2 2003, Orlando, Florida, USA.

Additionally this work has been partly funded by an AgentCities Deployment Grant from the European Union AgentCities.rtd Project (IST-2000-28385), AgentCitiesUK (GR/S57617/01) and the MOBIlearn project, IST-2001-37440.

Chapter 2

Agents and Agent-Oriented Software Engineering

2.1 Introduction

This chapter introduces agents, multi-agent systems and Agent-Oriented Software Engineering (AOSE). Three popular methodologies for specifying and designing agent based systems are examined and some criteria for an improved design framework are proposed.

2.2 Intelligent Agents Explained

As computing moves from isolated, standalone systems into vast, powerful distributed networks of processing, the need for systems to be able to operate with more autonomy is greater than ever. It is clear that there are tangible business benefits associated with integrating disparate, heterogeneous information systems and repositories, and presently much effort is being expended within the computer science community to make this happen.

This evolution of computing is creating a demand for systems with traits that would normally be considered 'human'. For instance, businesses are composed of many functions that interoperate with other functions, within constrained and open environments, making decisions and forming strategies to attain pre-determined goals. Such businesses may be physically distributed worldwide, and there might be functions that are mobile, moving from place to place. Information systems have developed considerably to support these functions, to the extent where it is difficult to imagine an organisation operating without one.

Indeed, many organisations are reliant upon their systems, as critical business processes harness the capabilities of the information system, especially as new developments in technology offer new opportunities to conduct business in new, more effective ways. These systems however, may help coordinate, organise and distribute information, but the real power of the organisation is within the employees who can communicate, interact, react and plan; both themselves and those that they manage, to achieve tangible business goals.

The Object Oriented (OO) paradigm has much to offer businesses in terms of system design. Business 'objects' can be created that allow the organisation to be modelled, designed, deployed and maintained at an abstract level. Such abstraction hides detail which might prove distracting, enabling much more tangible representations to be developed. Additionally the abstraction assists those who wish to coordinate the activities of 'islands' of information and functionality, as they can concentrate on the flow of messages that are passed between discrete objects. Considering an object as a 'black box', which accepts inputs and produces predictable outputs, improves maintainability and future extensibility. As a result there has been a proliferation of OO systems developed using technologies such as Sun Microsystems Java 2 Enterprise Edition

(J2EE) (Sun, 2004) and Microsoft .NET (Microsoft, 2004).

To develop an organisation further, it is necessary to provide a means by which existing systems can be integrated. For some time now organisations have been integrating internal systems, creating 'enterprise level' applications such as SAP (2004). With the trend towards eliminating geographical boundaries and using the Internet as an infrastructure, forward-thinking businesses are developing open systems that can be interconnected between organisations; extending the possibilities for rationalising the effort expended and becoming more profitable.

Additionally, it is vital that organisational systems can themselves be delegated tasks to complete autonomously in pursuit of the business goals. So far the integration of disparate systems is creating even more quantities of information, that still needs processing. If we are to make use of this information then we have to find a way of automatically processing it, whilst ensuring that our pre-determined goals are met, in a dynamic and rapidly changing environment.

Finally we need to find a mechanism that permits these actions to be coordinated and communicated, in a way that the pertinent information is available when required. More importantly, the relevant knowledge is shared and understood across different parties.

Thus we require systems that have an ability to:

- React to a dynamic, open business environment;
- Coordinate future activities in a way that takes control of a process in order to meet a business objective;
- Communicate across business functions to facilitate the devolvement of

information and knowledge, whilst also providing an interaction mechanism to support the above.

In essence we require business functions that can operate with some degree of autonomy, and act in our best interests. To do this we need a computer system that is (Wooldridge and Jennings, 1995):

> "...situated in some environment, and that is capable of autonomous action in this environment to meet its design objectives."

Such a system is described as an *agent.*

Wooldridge (2002) has further refined the description of an agent by providing some distinguishing characteristics of an *intelligent* agent, that being reactive, proactive and social behaviours.

The definition that follows is probably the most accepted description of an agent, though discussion still exists within the research community (Wooldridge and Jennings 2005):

> "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."

2.2.1 Agents and Objects

There has been significant debate as to whether agent-oriented applications are merely OO applications, and many discussions as to the distinction between 'agents' and 'objects'. Some would argue that an object can be made to be proactive and possess some degree of autonomy (two facets that seem to provide most of the necessary differentiation). Wooldridge (2002) argues that such assertions are "*missing the point*". If an object exhibits the characteristics of an agent, then it *is* by definition an agent.

Put another way, if an object is told to do something, it will do it in a predictable way. An agent will decide for itself whether it wishes to complete the task. The notions of choice, planning and autonomy are immediately attractive concepts for system designers and business modellers. These facets lift the potential capabilities of software systems to new levels, promising adaptive, flexible and self-maintaining business processes.

Of course such promise also demands the clarity of thought and practical skills to be able to successfully design and deploy these capabilities. It is at the point of implementation that the system development can become confusing; software agents are typically deployed with OO tools and application programming interfaces (API). The primary distinction between agents and objects is that agents are autonomous entities that choose what they are going to do, and who they are going to interact with. It follows that an agent's *behaviour* (method) cannot be directly invoked, unlike an object. All interaction with an agent is performed by communication only, as one would expect when interacting with a human agent. These traits however, do not preclude the construction of agents from OO tools.

There are many solutions to computational problems that are dealt with reactively, continually adapting to their environment in response to changing conditions. Similarly, proactive solutions demonstrate that it is possible (and eminently practicable) to model systems that take pre and post conditions, and compute an action in pursuit of a goal. It is the combination of these two characteristics however, that best describes what we want from an agent. Human agents regularly exhibit the ability to plan whilst also reacting to changes in the environment. This is important if software agents are to offer capabilities beyond those of objects.

An agent in the context of this thesis will generally be assumed to be a software system (or software agent) rather than the human equivalent, though when modelling a business organisation it is inevitable that human agents will exist. When such cases arise however, the distinction will be reiterated.

The first characteristic described in the definition of an agent is of *situatedness.* This becomes important when we consider the environments in which we expect an agent to exist. One of the attractions of the agent paradigm is that they are expected to operate in dynamic, open environments and as such they should exhibit reactive characteristics in order to cope with a rapidly changing set of conditions and achieve a particular goal.

In order for an agent to influence its own environment in response to unpredictable external influences, it must possess autonomy. Autonomous behaviour is a key distinction between agents and objects, and agents are often referred to as *autonomous agents.* To avoid confusion from the proliferation of terms this thesis will persist with the use of *agent.*

Whilst it is feasible that an agent should react to its environment to pursue a goal, this in itself provides nothing more than the capabilities of an object. Proactive behaviour serves to further differentiate agents from objects by enabling the pursuit of several goals. Similar to a human being, an agent can react to pursue one goal in isolation or plan to achieve multiple goals. Similarly, an agent must also strike a balance between reactive and proactive behaviours if it is to be successful. Achieving such a balance is a particular challenge for agent designers. One way of addressing this challenge is to adopt the *Belief*, *Desire* and *Intention* (BDI) architecture (Georgeff et al., 1999).

Finally agents need to be able to communicate if they are to interact with other agents. Amongst other things they need to describe their intentions, make requests and deal with responses. Social interactions are then converted into the most appropriate method to invoke by each agent. From a programmer's perspective an object has no control over its publicly available methods; any other object can directly invoke the receiving object's methods at will. An agent remains in control of its own methods however, and will exercise its autonomy to decide whether or not it will respond to a request from another agent. The messages exchanged by agents are referred to as communicative acts, and are usually expressed in an Agent Communication Language (ACL).

The characteristics described so far are used to classify agents as those with *weak agency.* The embodiment of additional characteristics such as mental attitudes tends to lead to an agent being described as having *strong agency,* though this has even been extended further to include mobility, veracity and benevolence.

Irrespective of strong or weak agency, the combination of reactive and social abilities enables agents to react to changing circumstances by influencing other agents in the same environment, making them extremely flexible. The ability to make decisions by exercising autonomy, together with goal-directed, proactive behaviour goes some way towards providing robust systems, which are essential if agents are to operate in truly open environments.

Thus, the definition of an agent for the purposes of this thesis is a software system that:

1. Is situated in an environment;

2. Reacts to changes in its environment;

3. Exhibits autonomy and controls itself;
4. Can demonstrate proactivity in pursuit of several goals;
5. Communicates socially with other agents in order to interact.

Agents therefore have much to offer compared with objects and they appear to offer many advantages when designing complex software systems.

2.2.2 What is a Multi-Agent System?

If an agent can embody reactive, proactive and social characteristics, then it also possesses the necessary traits to permit interaction with other agents. Immediately this notion opens up a new set of possibilities whereby agents can interact to exchange knowledge, whilst also using their planning abilities to coordinate and control activities, thus influencing their own environment. Similarly their ability to work towards goals by combining proactive with reactive behaviour, using a degree of autonomy, means that agents can initiate other agents into action. The autonomous trait also imbues an agent with the ability to say 'no', or select another means of completing a task.

In the same way that collections of autonomous business functions formulate an organisation, a number of interacting agents within an environment is referred to as a *Multi-Agent System (MAS).*

Jennings (2000) offers a description of a MAS whereby the control exerted by an individual agent over its environment is referred to as a *sphere of influence.* The human-like traits of an agent suggest that a MAS is a society that needs managing. The convenience of replacing abstract representations of business actors with agents makes the MAS approach an immediately attractive proposition for modelling complex systems. Each of the autonomous roles can now be suitably described, without decomposing roles into behaviours and entity objects. Jennings et al. (2001) remark that agents provide a *design metaphor* for system designers that supports the development of the autonomous systems required to solve complex computational problems. Methodologies for MAS development are still immature and with the rapid expansion of Web Services and the Semantic Web (Berners-Lee, 1999), tools and architectures are now more in demand.

2.3 Communication

So far we have considered agents as an abstraction of a particular computer system that exhibits enhanced behaviours, and MASs as collections of interacting agents in a particular domain. For the collaboration to take place, agents must be able to communicate their intentions in an unambiguous way.

As described earlier, the OO approach is to allow 'message passing' by directly invoking a publicly available method of an object. As an agent has control over its own state and behaviour, there is no concept of a 'public' method. An initiating agent would communicate its intention to invoke a behaviour (method). The receiving agent would then consider its own agenda before responding, or not, as the case may be.

Such characteristics place demands on the format and structure of agent communication, as there are fundamental conversational protocols that need to be represented.

2.3.1 Speech Acts

Austin (1962) writes of *speech acts* as a collection of utterances that appear to have some influence over a physical world. He identified a number of speech acts that can be represented by *performative* verbs, and defined three aspects of the acts:

- *Locution* - 'Please wash the dishes.' The act of making an utterance.
- *Illocution* - 'She asked me to wash the dishes.' The action that was performed in response to the utterance.
- *Perlocution* - 'She got me to wash the dishes.' The resulting effect of the act.

The performative verbs are a means by which the action of the speech act is described. Examples include *request*, *inform*, and *promise*. Successful completion of the performative was classified as three *felicity conditions* (Austin, 1962):

- There must be an accepted conventional procedure for the performative, and the circumstances must be as specified in the procedure.
- The procedure must be executed correctly and completely.
- The act must be sincere, and any uptake required must be completed, insofar as is possible.

Searle (1969) extended this work to include a much more rigorous specification of the domain in which the coversation takes place. For instance we must consider whether the 'hearer' can hear the 'speaker', or if the domain has specific characteristics that might affect the comprehension of a speech act. We would expect that an agent that receives a speech act instructing the murder of someone would be able to differentiate between the domain context of a theatre play and the real world.

2.3.2 Agent Communication Languages (ACL)

The means by which agents communicate is via an Agent Communication Language (ACL). Such languages have been informed by the work of Austin (1962) and Searle (1969), and most notably with regard to agents has led to the Knowledge Query Manipulation Language (KQML) described by Finin et al. (1993) and Labrou et al. (1999). The intention of this work was to generate (Finin et al., 1993):

> "...protocols for the exchange of represented knowledge between autonomous information systems."

KQML

KQML itself is akin to a *wrapper* that is used to transport the message between agents whilst also encoding the illocution part of the speech act. Using an example from Wooldridge (2002) each message is composed as follows:

```
(ask-one
:content (PRICE IBM ?price)
:receiver stock-server
:language LPROLOG
:ontology NYSE-TICKS
)
```

The performative is represented by the `ask-one` parameter which is interpreted as a question that requires a single answer. Next is the `:content` parameter, which contains the actual message content. Note that KQML ignores the message content (Patil et al., 1992), (Mayfield et al., 1996), and as such this field could contain natural language, Structured Query Language (SQL) or some other message format. `:receiver` specifies the identity of the intended recipient, and `:language` identifies the language of the `:content` field. In this case the message content is expressed in LPROLOG, and the `:language` parameter confirms this. The last parameter of the message, `:ontology` describes the terminology that the message uses. To satisfy the requirement for *knowledge exchange*, the Knowledge Interchange Format (KIF) (Genesereth and Fikes, 1992) was developed from first-order logic (Enderton, 1972), and the intention was not for KIF to represent messages, but to represent the message *content*. Thus KQML communications can contain message content encoded in KIF, for automated knowledge transfer between agents.

FIPA ACL

As an alternative to KQML, the Foundation for Intelligent and Physical Agents (FIPA) attempted to develop a standards-based ACL that had a much wider collection of suitable semantics (FIPA, 1999) than KQML. FIPA used the work of Cohen and Levesque (1990) and Bretier and Sadek (1997) when developing the ACL, creating a Semantic Language (SL) that permits not only actions to be communicated but also *beliefs*, *desires* and *uncertain beliefs* as well.

The inclusion of a representation for semantics means that it is possible to specify constraints that the sending agent must adhere to if it is to be FIPA ACL compliant. FIPA describes this as the *feasibility* condition. Additionally, the SL is used to describe the purpose of the message (perlocution) by encoding the *rational effect* of the communication. Since an agent can choose whether or not to respond to a communicative act, conformance to the FIPA standard cannot be enforced when receiving messages.

However, ACLs such as FIPA and KQML only provide protocols for communication between agents. Using the term agent as an abstraction for a variety of autonomous entities (such as human agents, or other MASs) it is conceivable that it will be necessary to consider communication protocols that can cross more disparate domains. In fact it is vital that issues such as human/agent interaction interfaces, knowledge transfer from user to agent and back again, and the level of abstraction required when delegating tasks to agents are resolved before significant progress can be made.

2.4 Ontologies

As we gradually explore the behaviours and abilities of agents, it is important to consider the practicalities of what, in some cases, are quite abstract concepts. If agents are to communicate successfully then there has to be some shared understanding of the message structure and content. ACLs offer an architecture that facilitates the exchange of speech acts, permitting the construction of messages that describe beliefs and intentions; although if knowledge is to be exchanged then there has to be some consensus as to to what the message content means. Without this we cannot delegate tasks to agents and MASs and thus derive the benefits of devolved decision-making.

Philosophy describes *ontology* as the study of *being*. Within the context of a system that shares knowledge, an ontology is an explicit, formal specification of how to represent the objects, concepts and other entities that exist in a domain of interest, together with the relationships between them (Gruber, 1993).

There are three principal objectives of an ontology:

1. It must represent a conceptualisation that can be shared and re-used;

2. The ontology must represent all of the applications within a domain and not be specific to one type;

3. It must contain all of the required information to permit knowledge to be explicitly stated, together with rules and constraints to facilitate the inference of new knowledge.

The advent of the eXtensible Markup Language (XML), (W3.org, 2004a), which is a subset of the Standard Generalized Markup Language (SGML) (W3.org, 1986), has made the generation of ontology documents much more accessible. Darpa Agent Markup Language (DAML, 2001) and latterly the Web Ontology Language (OWL) W3.org (2004b) are examples of XML-based languages that enable the inclusion of semantic information within documents, permitting automated analysis and processing.

One such example of a practical use for an ontology is a *knowledge dictionary*, in which the domain concepts, their relationships and constraints are defined in order to improve consistency of communication and facilitate system integration. Ontologies are also beneficial when constructing domain-specific applications, as they guide the system designer who has to interpret the demands of the end-users, the result being a set of more realistic application requirements and better long-term reliability (Uschold and Grüninger, 1996). Additionally the recording of ontological artefacts can be used as part of the requirements specification, assisting the design, build and test of domain applications.

Tools that facilitate ontology generation are crucial to improvements in business performance as the increased amount of information available still needs filtering, sorting and correlating, even though predominantly it is performed manually. Automating such tasks presents new opportunities as well as the challenges of dynamically generating ontologies as information changes. The Knowledge Reuse and Fusion/Transformation (KRAFT) project (Preece et al., 2000) demonstrated that agents can locate information from distributed, heterogeneous data sources and 'fuse' the knowledge to create new, pertinent knowledge for problem-solving.

2.4.1 Syntactic Interoperability

As discussed above, an agent communicates a representation of its mental state to a receiving agent, rather than directly manipulating the receiver's methods. This means that the responsibility for the outcome of an action is transferred to the receiving agent, rather than lying with the sender in an object environment.

This enables an agent to delegate responsibilities (and by implication its *goals*) to other agents, in the same way that responsibilities are delegated by human managers in a typical hierarchical management structure (Castelfranchi, 1998). Since agent architectures facilitate delegation, the following issues are addressed:

- It is easier to capture and specify the requirements of hitherto complex systems, thus embodying autonomy, delegation and proactivity;
- The resulting software models represent the domain more faithfully.

The imperative message passing approach of objects results only in receiving objects being forced to perform actions in a particular way, whereas agents can make requests without specifying how that request might be achieved. The fact that objects need to specify how something is performed, has led to the development of messaging protocols that rely on syntactic arguments; an object message specifically orders the execution of a method.

Such an approach delivers systems that need developers to program to the appropriate interface, using the correct syntax. It also means that the receiver has no information as to the intended outcome of the request, other than the specific method invocation, thus the receiver cannot reason about how an outcome might be achieved.

Whilst interoperability can be achieved between systems using specific syntactics, this is somewhat restrictive if the systems to be integrated are disparate, and certainly prevents the potential capabilities of agent architectures for open systems, since every agent needs to understand every other agents' communication syntax.

2.4.2 Semantic Interoperability

If the semantics of a request are considered, a different scenario is presented:

- Communicating agents would not have to rely on restrictive syntactic messages, and would be able to interpret those messages within the context of the agent's own belief-base;
- Agents could delegate responsibilities to achieve goals without specifying how those goals should be achieved;
- Environments of semantically-able agents could be assembled automatically, safe in the knowledge that they could interoperate and cooperate independently of syntactical restrictions, whilst maintaining loose coupling and agent autonomy;

- Communication is simplified as agents only have to communicate the goals that they wish to achieve.

Agent Communication Languages (ACL) such as FIPA-ACL (FIPA, 2006) define syntactic and semantic standards for inter-agent communication in terms of *speech acts* (Austin, 1962).

In particular, FIPA-ACL has a rich set of performatives that formally specify meaning for communication primitives, based upon speech acts, enabling agents to interpret messages correctly and act accordingly. Agents that can understand the meaning of a communication, by interpreting its semantics, stand a much better chance of reacting properly. This, in turn, facilitates improved system flexibility within open environments.

If an agent wishes to know the time it would need to express a communicative act that represents *"What is the time?"*. Using FIPA-ACL this would look like:

```
(Query-ref
      :sender Agent_A
      :receiver Clock_Agent
      :content ''((any ?t (time ?t)))''
)
```

The following statement is also valid: *"I want to know the time"*. This would result in the `Inform` performative being used.

```
(Inform
      :sender Agent_A
      :receiver Clock_Agent
      :content ''((I Agent_A (exists ?t (B Agent_A (time ?t)))))''
)
```

In essence, both of these communicative acts should receive the same answer, even though the original language is grammatically different. An agent with semantic capability can interpret both of these messages and use the following communicative act as a reply:

```
(Inform-ref
     :sender Clock_Agent
     :receiver Agent_A
     :content ''((any ?t (time ?t)))''
)
```

This approach simplifies agent construction considerably as the agent system designer can concentrate upon developing cooperative and domain-specific features instead of attempting to capture (or predict) every variant of conversation with the associated message handling protocol.

Domains in an open environment will be rich with diversity and inconsistency, particularly since they are composed of many disparate heterogeneous systems. Such environments demand flexible communications, and applications that rely upon syntactic exchange of knowledge cannot offer the potential of a semantic agent approach.

2.4.3 Communicating Intentions

Having established that a semantic approach to agent communication is desirable for an open environment, it is necessary to consider the means by which the agent *intentions* (communicative acts) can be constructed. The process of capturing requirements, gathers together, amongst other artefacts, the business rules by which an organisation operates. If agents are to operate as a flexible MAS, then it is important that the business processes, rules and protocols are captured in order that they can be utilised by a MAS. In general:

1. Domain rules should be written by the individuals who perform the tasks, not necessarily domain experts;

2. Each role within the domain may require a bespoke interface for composing context dependent rules;

3. Domain rules should ideally be dynamically generated by interacting with the system;

4. Rules are likely to be incomplete and will be refined over a period of time.

The complexity of a MAS domain is such that a large proportion of the knowledge is held with the users of the various systems. This, in turn, leads to informal processes and protocols that have evolved over time to accommodate deficiencies in the existing command and control systems. Sowa (2000) proposes Controlled English as a formal language for description, which could be used to facilitate the generation of ACL message content whilst maximising semantic interoperability.

Organisation protocol rules can be convoluted however, and it is probable that rule generation can become an overwhelming task. Compton et al. (2006) and Compton and Jansen (1990) describe 'Ripple-Down Rules (RDR)', a method of '*rules maintenance*' whereby rules are created or edited within the context of a specific task, resulting in easier comprehension and more stable rule building. A key part of this approach is the realisation that the post-conditions of a task in a particular context need capturing and expressing if a representative rule is to be generated.

For example, a Bank Agent needs to assess the financial status ('credit check' in UK) of a potential Loan Applicant, via the Loan Applicant Agent. This rule can be expressed simply as:

```
If Loan_Applicant has salary < 20000 Then
   Loan_Applicant is rejected
```

However, this blanket rule takes no account of other circumstances, such as whether the Loan Applicant is self-employed. Since the overall rule has been created, modification is required rather than composing a new rule.

```
If Loan_Applicant has salary < 20000 Then
   If Loan_Applicant is self_employed Then
      Loan_Applicant must submit proof_of_income
   Else
      Loan_Applicant is rejected
```

Whilst the nesting of these statements will undoubtedly result in large rule trees, it is necessary only to consider each rule within the context of the particular case of use, and therefore the justification for a change is localised. Such an approach therefore enables agents to update their belief sets as they encounter new scenarios, by tailoring general rules with new specific variants.

2.5 Agent-Oriented Software Engineering

Agent-Oriented Software Engineering (AOSE) is a variation of traditional software engineering approaches that facilitate the analysis, design and implementation of software systems. In particular, the additional characteristics offered by agents are offered as a means by which more complex software applications can be constructed, in favour of more established software engineering approaches such as structured programming and object orientation (OO).

Early attempts at encapsulation, using subroutines in structured programming, has steadily matured into the OO paradigm whereby the four software engineering goals are much easier to satisfy. The ability to encapsulate code and abstract away from low-level detail is a significant advantage of the OO approach, and as a result the software industry is heavily influenced by trends in OO development. Whilst OO goes some way towards simplifying software design by providing a better fit with 'the real world' through object representations, the essential characteristics of passive objects do not support the dynamic and proactive abilities that agents possess. AOSE addresses this by applying agents to the analysis, design and construction of software, in order that more de-centralised capabilities are available to systems, as demanded by increased take-up of the Internet and emergent Semantic Web. The autonomous behaviours of agents and multi-agent systems makes AOSE particularly suited to the design and robust construction of complex applications. These systems are able to take the initiative and exhibit qualities such as self-healing, negotiation and brokering in dynamic open environments. One aspect of AOSE that is particularly interesting is the potential for the approach to be used for the analysis and design of software systems, that may eventually be constructed with established OO methods. The application of agents to a design problem allows system designers to manage the software engineering process in a more realistic way; the abstraction of discrete, autonomous entities drastically simplifies requirements gathering, and as such is a case for using agents as a *design metaphor*. Similarly, the increased capabilities of agents permits complex organisational workflows to be represented, whilst also enabling the application of existing organisational models to agent representations, in order to represent inter-dependencies and complex interactions (Luck et al., 2004).

2.6 Design Methodologies

Even though agents and AOSE appear to simplify the design of software systems, the process of eliciting requirements, system analysis, design and construction still requires guidance if a system is to be successfully completed. Methodologies provide the steps required to convert abstract, high-level requirements into a design specification, and should include the detail necessary to enable the process to be repeatable. Since many of the programming languages for agent systems are based upon OO principles, and OO design methodologies are now quite mature, it seems sensible to use an OO approach when designing an agent system. OO methodologies offer design abstraction with objects and communication via message passing, which could be used to produce a representation of an agent based system. Such a representation would allow a reactive system to be built, albeit with passive objects, which compromises any agent model somewhat. If an agent design methodology is to produce a faithful representation of an agent system then the design methodologies for agent systems need to reflect the enhanced characteristics that agents demonstrate.

For example, as described in Section 2.2, agents have proactive behaviours that require goals to be expressed. As such, a methodology should provide the steps required to elicit and model goals if the proactive behaviours are to be included as part of an agent's capabilities. OO methodologies generally model all objects as passive, and do not differentiate between the active states of agents and passive data. Similarly, architectures such as BDI require mental attitudes to be elicited, described and applied to agent software, which is not something that is included within OO design methodologies.

Consequently there is a motivation to develop an agent design methodology that embraces the enhanced abstract characteristics of agents and agency. A number of methodologies (Massonet et al., 2002) have emerged from established software engineering methodologies such as Gaia (Zambonelli et al., 2003; Garcia-Ojeda and Arenas, 2004; Juan et al., 2002), Prometheus (Padgham and Winikoff, 2002), MaSE (DeLoach, 1999), and Tropos (Bresciani et al., 2001), together with a number of toolkits that assist the generation of MASs (Bergenti and Poggi, 2001), (DeLoach and Wood, 2000). Three of these agent design methodologies, Gaia, Prometheus and Tropos, are now briefly described and discussed in relation to their relative strengths and weaknesses.

2.6.1 Gaia

The Gaia Methodology (Wooldridge et al., 2000) attempts to provide an abstract framework for the design of agent systems. It is based upon OO principles and as such makes the transition from OO to agent design much easier as it is likely that the system designer will have at least some familiarity with OO methodologies. Whilst the selection of an OO approach is 'safe', it also means that enhanced agent characteristics need to be appended to the OO concepts, and as a result they are more abstract than in some other methodologies. Also, Gaia assumes a requirements specification as an input, thus restricting the extent to which agents can simplify requirements capture through use of an agent design metaphor. The design process consists of two distinct phases:

1. *Analysis* - abstract conceptual models are built from the requirements specification, prior to;

2. *Design* - whereby the models are transformed into entities via a design specification language, in order that program code can subsequently be generated.

An overview of the models within Gaia is shown in Figure 2.1. The key thrust with Gaia is that the eventual system should be viewed as an organisation, comprising entities, roles, goals (individual and organisational) and interactions. This organisational metaphor serves to represent the system at macro and micro levels. The two stages are described briefly below.

Analysis

The analysis phase enables the system designer to explore and understand the structure and organisation of the *system* (Figure 2.2), expressed as a collection of roles that interact with each other.

Roles are a key concept within Gaia, which are used to provide a conceptual representation of the system. The role model contains a role schema for each of the roles identified. This describes the behaviours that an agent would need to possess. The role model is defined by the collection of *Role Schemata* for the entire system. Figure 2.2 illustrates the concepts and relationships within

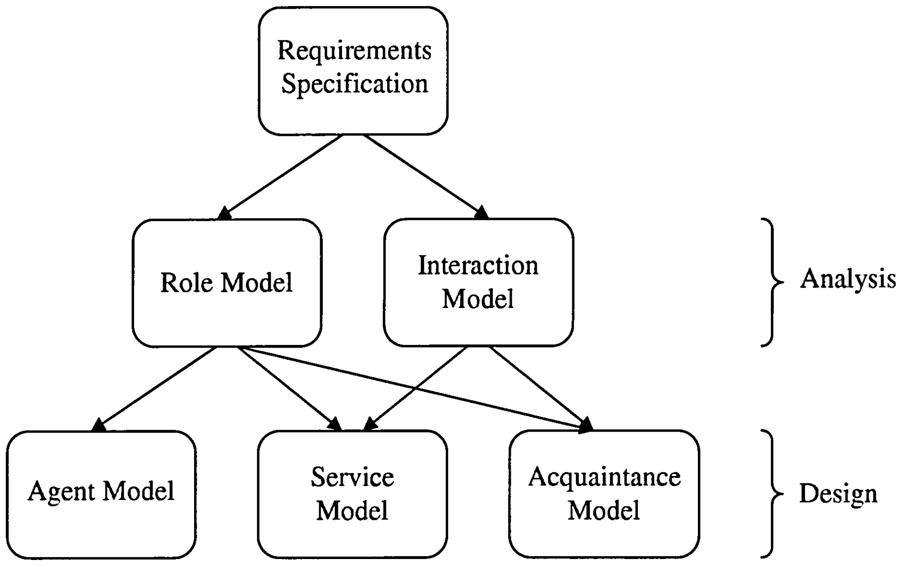


Figure 2.1: The Gaia Methodology Models (redrawn from Wooldridge et al., 2000).

the Gaia Analysis phase.

The first step upon receipt of the requirements specification is to build a *Role Model* which comprises a list of identified roles and role descriptions. These specifications give an abstract representation of the functionality of each role, by specifying the following attributes:

1. *Permissions* - This is a description of the scope of the rights of a role's behaviours, in terms of what resources the role has access to, and what it can (or cannot) modify or create.

2. *Responsibilities* - These describe the functionality of each role and are categorised by two types: (1) *Liveness Properties* state the ideal, in that they describe the solution that an agent must bring about in certain environmental conditions, whereas (2) *Safety Properties* are the properties that an agent must always protect when undertaking the role, in an attempt to maintain stability during execution.

3. *Activities* - These are specific actions that the agent might perform without involving any other agents.

4. *Protocols* - These define how a role can interact with other roles, such as the use of a Contract Net protocol (FIPA, 2002) for instance.

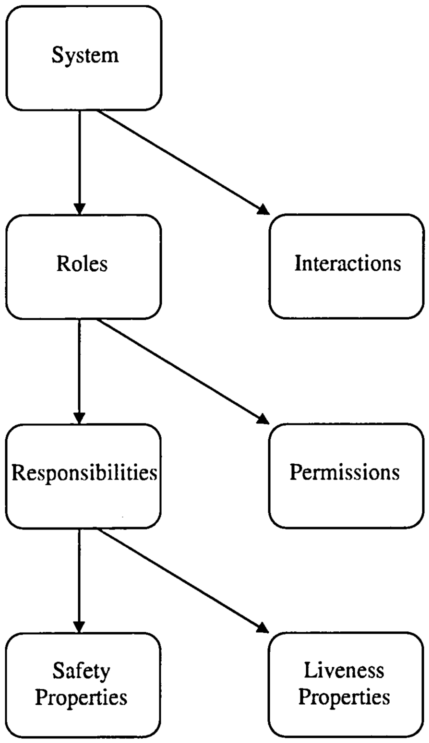
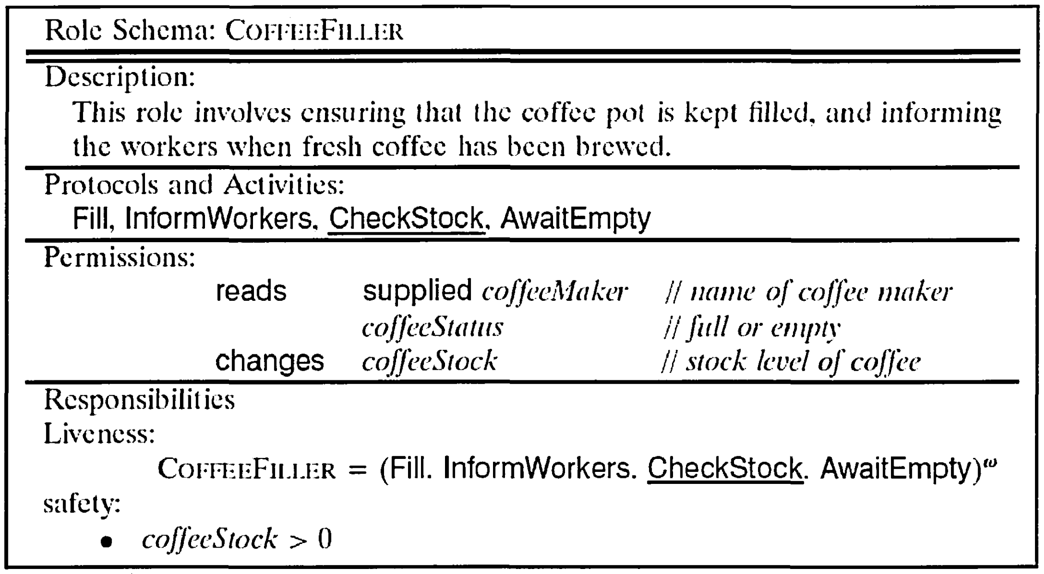


Figure 2.2: Concepts within the Analysis stage of Gaia.

The other artefact produced during the analysis phase is the *Interaction Model.* The collaboration between roles is explored in order to represent the interactions that need to take place for the system to function correctly. Gaia offers guidance by specifying the interaction characteristics for each role, otherwise known as a *Protocol Definition.* Each protocol is defined in terms of its *purpose*, the *initiator* role, the *responder* role, any *inputs* and *outputs* and finally any *processing.* Before progressing to the Design phase, more detail is added to each of the roles identified, and appended to the *Role Schema.* An example of a Role Schema is shown in Figure 2.3 (Wooldridge et al., 2000).



Role Schema: CoffeeFiller

Description:
This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.

Protocols and Activities:
Fill, InformWorkers, CheckStock, AwaitEmpty

Permissions:

reads	supplied *coffeeMaker*	*// name of coffee maker*
	coffeeStatus	*// full or empty*
changes	*coffeeStock*	*// stock level of coffee*

Responsibilities
Liveness:
CoffeeFiller = (Fill. InformWorkers. CheckStock. AwaitEmpty)$^{\omega}$
safety:
- *coffeeStock* > 0

Figure 2.3: Role Schema for Coffee-filler (Wooldridge et al., 2000).

Design

The Gaia Design phase consists of three models, that when completed form the output design artefacts from this approach. The first model (*agent model*) describes the agent types that are required in the system. This model is similar to a class model in that each agent will result in one or more instances that are realised at execution time. Each agent type may undertake one or more roles identified in the Analysis phase, and conversely a role may be attributed to one or more agent types.

The second model describes the services provided by each of the roles. Wooldridge et al. (1999) defines a service as a:

"...single block of activity in which an agent will engage."

The *protocol definition* from the Analysis phase is used to define the inputs and outputs for each service, and the pre and post-conditions are mapped from each role's safety properties. Thus the *Service Model* is a comprehensive list of the services that each agent offers.

Finally the *Acquaintance Model* is derived from the Interaction Model and Agent Model. It identifies the communicative links between each of the agent types, providing a representation of agent coupling.

Issues with Gaia

Gaia was the first MAS design methodology that addressed the need to explicitly deal with agent abstractions rather than use other, more compromised approaches. Whilst it has been developed with MASs in mind, there are some significant issues that should be considered.

Gaia assumes that the requirements gathering/specification phase has been completed, and offers no guidance as to how this might be performed. The organisational metaphor goes some way to allowing agent models to be harmonised with more traditional methods of requirements capture, but the use of agents as a metaphor for gathering system requirements is not addressed. As a result, the potential simplification of requirements models is missing and therefore the methodology could be more comprehensive.

The capture and representation of domain knowledge is a fundamental part of any MAS design process, and Gaia offers no guidance for the definition and modelling of ontologies.

Gaia assumes that all of the agents will cooperate, and therefore the environment is deemed to be closed and controlled, rather than open and dynamic.

There is no explicit means of modelling agent goals, nor a means of defining goal and task delegation. Since the content and sequences of the messages are ignored, the interaction model does not provide the necessary detail required to fully represent message content semantics.

The models produce agents that have a pre-determined organisational structure, and there is no provision for organisational relationships that might change during execution. This facet is not unique to Gaia, though it does reinforce the importance of an effective, accurate requirements capture stage.

Finally, whilst a design methodology need not specify an implementation platform, Gaia requires an experienced agent designer to convert the abstract concepts into concrete entities.

2.6.2 Prometheus

Prometheus is a comprehensive design methodology that attempts to encompass the whole system design life-cycle, from initial requirements specification through to testing and debugging. The approach is based upon a process that has been designed to be used by both experienced agent developers and newcomers to agent development. Most of the steps of the process result in an artefact, leading to a comprehensive set of design documents. Prometheus consists of three phases:

1. *System Specification* - this phase concentrates on eliciting system goals and functionality, and investigates and documents inputs and outputs of the system to be designed.

2. *Architectural Design* - this phase determines the types and quantities of agents required to deliver the outputs of the system specification phase.

3. *Detailed Design* - after determining the types of agent required, specific details of each agents' capabilities are described, in order to develop an

implementation.

The last phase of the Prometheus approach is implementation. The first three stages result in a design that is 'platform neutral' in keeping with a general purpose methodology. However, through the use of the Prometheus Design Tool, PDT (Padgham et al., 2005) and the JACK agent platform (Busetta et al., 1999), automatic code generation is provided. Figure 2.4 illustrates the first three phases of Prometheus. The following sections explore each of the

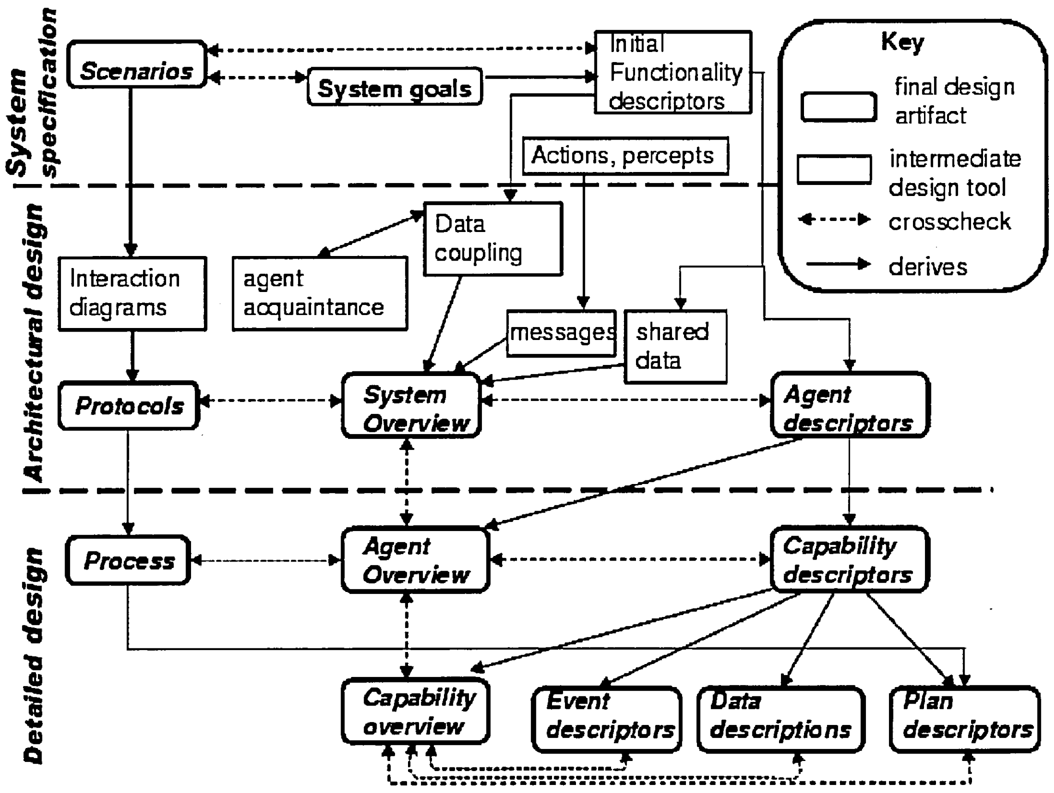


Figure 2.4: The Prometheus Methodology (Padgham and Winikoff, 2002).

phases in more detail.

System Specification

This phase concentrates upon eliciting the goals and functionality of the system, developing use case scenarios, and describing the requirements of an interface between the system and its environment. *System goals* are used as a means of capturing high-level requirements, and after some iteration *sub-goals* are discovered, together with their *relationships*, thus creating a hierarchy. Eventually this will allow similar sub-goals to be grouped together, in order to specify *functionalities.* The following information is contained within each *functionality descriptor*:

- Name and description of the functionality;
- Event triggers;
- Goals to be achieved;
- Actions performed;
- Messages sent and received;
- Data used and created.

Use case models enable the system designer to use graphical models to visualise the system in particular scenarios, whilst scrutinising the textual sequence of steps during execution. This 'process check' serves to elicit any goals that have not yet been identified, that are essential for the functionality to meet the overall system goals.

Architectural Design

The architectural design phase determines:

- The agent types and how they are described with *agent descriptors*;
- The overall structure of the system with the *system overview diagram*;
- The dynamic behaviour of the system with *interaction diagrams* and *interaction protocols.*

Agent types are determined by grouping together agent functionalities, with particular attention paid to coupling and cohesion. *Data coupling diagrams* are used to assess how successful this has been, in conjunction with an *agent acquaintance diagram* as a cross-check. The culmination of this is the *agent descriptor.*

After determining the agent types, each agent is appraised in terms of whether it reacts to a percept, and the actions it performs upon the environment are also described. Message exchange is also specified at this point, enabling the *system overview diagram* to be assembled. Whilst the message exchanges have now been identified, it is necessary to explicate the timing and sequence of messages. This is provided within the *agent interaction diagrams.* Such diagrams utilise Agent-oriented Unified Modelling Language (AUML) to represent interaction protocols.

Detailed Design

This phase develops a hierarchical capability model for each agent, in order to determine events, plans and any data structures that might be required. The *capability descriptor* is the means by which information about events is contained, including interactions with other capabilities, access to data, and details of any events that might be received. Moving nearer to implementation, descriptors for *events*, *data* and *individual plans* provide the required detail.

Agent overview diagrams illustrate the inner workings of an agent by describing the organisation of an agent's capabilities. If the system designer has chosen a platform other than JACK, then the latter part of this phase will be influenced by the elected agent platform. If JACK is to be used, then the PDT can assist with code generation during the subsequent *implementation phase.*

Issues with Prometheus

Unlike Gaia, Prometheus does support the gathering of requirements in terms of goal elicitation, and this is useful for indicating the intentions of the eventual system stakeholders. Since goals are often high-level concepts, they are much less likely to change than requirements (van Lamsweerde, 2001), which are often modified as the models are iterated. This makes goal elicitation important for agent oriented systems and Prometheus supports this in part during the system specification stage. From the architectural design phase onwards there is a shift in emphasis from goals to agent communication and data access, thus resorting back to a coding approach to agent system development.

Beyond the identification of goals however, Prometheus does not provide as much detail in the system specification phase as the approach does in latter phases. Thus the elicitation of system requirements is relatively weak and there is a reliance upon expert knowledge from the target domain.

2.6.3 Tropos

Tropos attempts to facilitate the modelling of systems at the knowledge level and highlights the difficulties encountered by agent developers, especially since notations such as UML (OMG, 2005) force the conversion of knowledge concepts into program code representations (Bresciani et al., 2001). The design of Tropos is influenced by the i* framework from Yu (1997). It seeks to capture and specify 'soft' and 'hard' goals during an 'Early Requirements' capture stage, in order that the Belief-Desire-Intention (BDI) architectural model (Georgeff et al., 1999) of agent implementation can be subsequently supported.

Once the goals have been specified, plans for actors can be constructed that enable agent desires to be pursued. Model-checking is provided through the vehicle of Formal Tropos (Fuxman et al., 2001); although this is an optional component and is not implicit within the agent realisation process. Briefly, Tropos consists of four phases:

1. *Early Requirements* concentrates on the modelling and analysis of the intentions of system stakeholders. Using the work of Yu (1995), goals are elicited that, after subsequent analysis, can be later specified as functional and non-functional requirements (Dardenne et al., 1993).

2. *Late Requirements* provides a prescriptive requirements specification that describes all of the functional and non-functional aspects of the system.

3. *Architectural Design.* This stage develops an overall architecture, by harmonising the MAS architecture into its organisational setting.

4. *Detailed Design* provides guidance for the development of agent behaviours and interactions by applying 'social patterns' (Do et al., 2003).

The notion of 'early requirements' is a key differentiator between Tropos and other agent-oriented design methodologies, and it supports agent design by providing guidance at the earliest stage of the development process.

Early Requirements

Tropos attempts to assist the process of eliciting both stakeholders and their intentions, in order that they can be specified as *actors* and *goals.* Tropos classifies the intentions as either *hard goals* or *soft goals.* Hard goals have satisfaction conditions that can be specifically defined, whereas soft goals have conditions that are difficult to define or represent. The identification of hard goals will enable functional requirements to be specified. Similarly, soft goals allow non-functional requirements to be specified.

Early requirements in Tropos are communicated using two models:

1. *Actor Diagram* - This diagram shows the actors and the relationships between actors. Tropos describes the relationships as "social dependencies", since the actors will depend on other actors for goals to be achieved, tasks to be delegated and resources to be consumed.

2. *Goal Diagram* - Each actor undergoes an analysis that will result in an individual specification of goals and plans, in order to give the actor (agent) the required capabilities in the final system.

The methodology offers three approaches to the analysis of the goals:

1. *Means-end Analysis* requires goals to be scrutinised for smaller, sub-goals, in order that the associated plans and resources required for successful attainment of the goal are specified.

2. *Contribution Analysis* represents goal interactions, and illustrates how the achievement of one goal can affect another.

3. *AND/OR Decomposition* (Nilsson, 1971). Since each goal could be achieved

in several different ways (plans), then each sub-goal that has been identified can also have a number of plans associated with it. This can be represented by the goal-plan tree in Figure 2.5, in which the OR relationship indicates the alternative plans that could be used to achieve a goal or sub-goal. Since a goal can only be achieved if all of the sub-goals have been successfully achieved, an AND relationship relates these two concepts together.

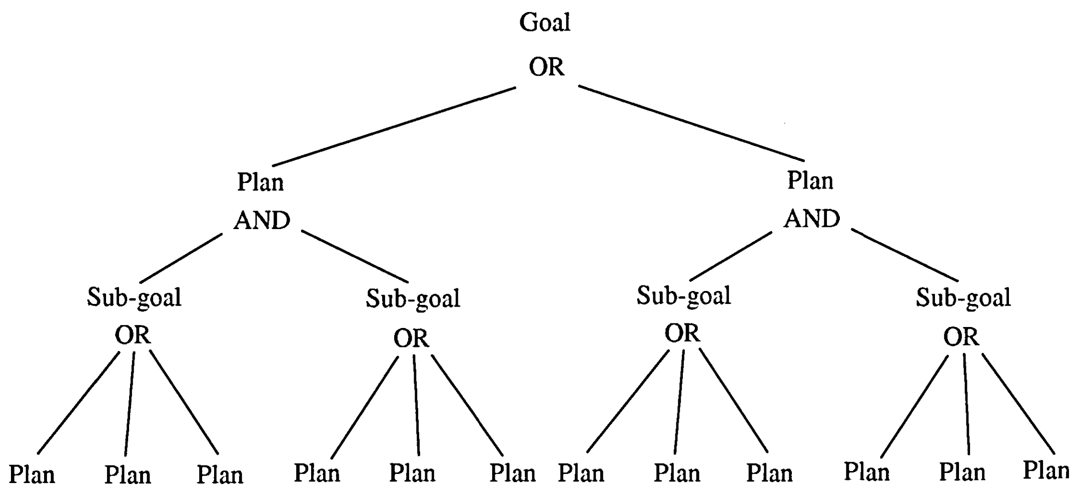


Figure 2.5: Goal-plan tree showing goal decomposition

The processes of defining social dependency relationships and the specification of goals and plans for each actor are inextricably linked, and it is necessary to iterate the early requirements stage until a suitable specification for the late requirements stage is generated.

Late Requirements

The late requirements stage is more akin to the system specification phase of Prometheus, except that the Tropos approach has not only identified *what* the system should do, but also conducted some analysis based upon the rationale for the functionality, or the *'why'*. This stage takes the individual goals of each actor and builds a strategic rationale model that represents the contributions of all of the system actors. As a consequence it is possible to represent quantitative (soft) goals with some alternative qualitative measure as is common during late requirements analysis (Dardenne et al., 1993).

Architectural Design

A series of architectural organisation patterns are provided to assist the agent system designer refine the models produced so far in order that sufficient detail for a complete design specification can be produced. Architectural analysis enables detailed actor capabilities to be explored, which may result in new actors or sub-actors being introduced into the model. Subsequently, each sub-actor will have intentions, and therefore goals and plans, which will require further iterations to develop completely.

Detailed Design

This stage uses social design patterns (Do et al., 2003) to enrich the design specification for each actor and its subsequent agent, by specifying the specific behaviours required to achieve a goal, with respect to the organisational and social architecture of a particular domain. Interactions between agents are described (typically using AUML) and a detailed design specification is produced in readiness for implementation.

Model Checking

Throughout the Tropos methodology there is an opportunity to perform model checking by using formal analysis techniques described by the Formal Tropos

(FT) specification language (Fuxman et al., 2001, 2004). FT permits the dynamic aspects of the model to be considered from a strategic viewpoint, and the model can be checked by posing queries.

The automation of many of the tasks in Tropos is yet to be realised, however, the 'T-Tool' described by Fuxman et al. (2004) is an example of tool support.

Implementation

The Tropos methodology is closely related to the JACK agent platform (similar to Prometheus), and the agent system designer must map Tropos concepts to BDI concepts, before mapping BDI concepts to the JACK language constructs.

Issues with Tropos

Tropos is an agent design approach that differentiates itself from other agent-oriented methodologies in two key areas. Firstly it makes the process of *early requirements* gathering not only explicit for agent-oriented design, but it also offers techniques and a method for analysing and modelling the system from an intentional standpoint.

Secondly, the concepts of stakeholders, actors, roles, intentions and social organisation are carried through all of the stages of the methodology. Whilst there is the potential for such 'high-level' concepts to be considered in an undisciplined fashion, there is the formal specification language FT to assist any desire for rigour.

FT, however, is an optional component and is not a pre-requisite for use of Tropos. Additionally there is a need to conduct some modelling activity before the use of FT, if unnecessary effort is not to be expended. Fortunately the T-Tool goes someway towards preventing this.

Even though Tropos provides support for the specification of systems to be implemented upon the JACK platform, the ability to represent models in terms of BDI concepts permits the model to be transferable across BDI compliant platforms.

Tropos embraces the use of 'organisational' patterns to assist the specification, analysis and design of the agent system, and also utilises more design patterns for more detailed agent description. There lacks guidance however, as to how any agent interaction protocols (other than FIPA) might be defined that support the specific semantic demands of an organisation's domain. Additionally, the task of goal decomposition can be difficult without specific domain expertise, and it would be useful to have an organisation-focused metaphor to assist in this process.

2.7 Discussion

Gaia, Prometheus and Tropos all describe different approaches to the design of agent-oriented systems. Gaia is the oldest, and as a result has been discussed at length in the research literature. The main criticisms of Gaia are its lack of a requirements analysis stage and a need for a richer set of semantics to better model the organisation in an open environment. Variations such as ROADMAP (Juan et al., 2002) and Gaia v.2 (Cernuzzi et al., 2004) have introduced extensions that enable more aspects of open environments to be catered for, but the methodology is still weak in relation to other approaches in this area.

The Prometheus approach is very comprehensive and the focus upon a process of design artefacts provides structure during the analysis and design phases. Methods for goal elicitation are provided; although only during the system specification phase.

Tropos addresses the need for more guidance during the early requirements stage and provides organisational metaphors to assist the determination of goals and actors. Formal rigour is introduced (optionally) by the use of the FT specification language. However, like Gaia, Tropos assumes that the agent system designer has control over a closed environment (Dastani et al., 2004).

None of the above methodologies offer support for the generation of an ontology. It is important to recognise that just as UML models for OO systems require a degree of expertise on the part of the designer, the creation of agent based and domain ontology models is complicated (Ehrler and Cranefield, 2004). This arises not only because an agent solution is generally more complex (Chopra and Singh, 2004) (protocols, tasks, roles, etc.), but the problem and domain is almost always more convoluted (Beer et al., 2003b).

Prior experience with AUML and the Zeus Methodology (Beer et al., 2001) illustrated that several problems remain at the requirements capture stage (Dastani et al., 2004):

1. Most agent design methodologies do not incorporate inherent model verification. It is therefore probable that some significant details are missed from the first iteration (Mellouli et al., 2002). Whilst actors are identified, they might not offer the best approach for the revised solution (Dastani et al., 2003a). Together these problems require an experienced systems analyst who can look beyond the notation and offer improved business processes so that the new system offers significant worthwhile added benefits.

2. Use case analysis captures process-level tasks without challenging qualitative issues. If the potential of an agent based system is to be realised, then the agents must be able to understand and process decisions or actions that require qualitative reasoning.

3. Role modelling is an inherent part of the MAS modelling process (Depke et al., 2001), yet there is little guidance as to how roles should be allocated for best performance (Dastani et al., 2003b).

4. Generation of terms for an ontology is largely based upon the existing processes together with the system analyst's knowledge and experience. From a systems modelling perspective, the process of describing and articulating use cases serves to elicit the majority of the eventual agent behaviours.

5. Even though actors appear to map straight to agents, the assignment of behaviours is often arbitrary, based on current practice, rather than systematically developing a coherent model (Dastani, 2004). Whilst this offers a distinct advantage for the systems modeller as the capture of system requirements is quick, simple and readily verifiable by reference to the current system users, there is no inherent check to verify the validity of each process or role, nor how the roles were delegated.

Agent design and development makes specific demands upon the developer (Jennings, 2001), especially with regard to the capture of system requirements. Except for Tropos (Bresciani et al., 2004) however, little work has been published that encompasses the whole cycle from early requirements capture through to implementation of an agent system (Giorgini, 2003).

An alternative approach utilises MAS platforms and toolkits such as the Java Agent Development Environment (JADE) (Bellifemine et al., 2001), to replicate existing systems from class model representations. This is potentially attractive to developers as relatively simple mappings from classes to agents are realised, though it does require a familiarity with low-level agent programming concepts and an understanding of how real-life interaction scenarios can be decomposed and translated into program code. A common result however is that the program code can quickly deviate from the model, as the limitations of such simplistic translations are realised (Dastani et al., 2003b).

It is important to have a much deeper level of understanding of a system from the outset, ensuring that fundamental business concepts are captured, described and understood. Whilst conceptual modelling is often a means by which rich, flexible scenarios can be captured, there is an inherent difficulty in specifying a design later in the development life cycle. This is compounded by the fact that flexibility often leads towards lack of discipline, or consistency, in modelling, thus there is a need for a concept-led, rigorous elicitation process, prior to MAS specification and design. A conceptual approach that has the capability to capture complex, real-world problems, yet with the addition of model-checking, consistency and rigour, would address these challenges.

2.8 Criteria for Framework

Whilst extensions to the UML meta model such as AUML (Bauer, 2001; Bauer et al., 2001), have simplified the design and specification of agent characteristics such as interaction protocols (Odell et al., 2001), the process of gathering and specifying initial requirements using established notations such as use case modelling (OMG, 2005) is often limited by the discipline and experience of the MAS designer.

It would therefore be useful if it was possible to provide an extended and more rigorous means of capturing requirements for agent-based systems by addressing the need to scrutinise and verify agent concepts that exist in the environment, prior to more detailed analysis and design with existing methodologies (Hill et al., 2004, 2005a,b, 2006a; Polovina et al., 2004).

Considering the review of the three methodologies considered so far, a number of key characteristics for an agent design framework become evident. Each of these characteristics is identified in relation to the relative strengths and weaknesses of each approach.

1. *A clearly defined process that describes how the framework is applied together with the details of any implicit process.* Gaia addresses this aspect to a limited extent in that an abstract process is described that lacks detail. Both Prometheus and Tropos are much better in this respect in that the process is articulated in much more depth, particularly Prometheus which is described in considerable detail. Process detail reduces ambiguity and improves repeatability and rigour. It is important however, that the imposed rigour does not limit expressivity.

2. *An ability to manage differing levels of abstraction, from the highest (societal) down to the most detailed (agent) descriptions.* Again this is addressed partially by Gaia in that the different levels of abstraction are clearly identified, and there is a reliance upon the use of roles within the models. However, the assumption that requirements capture has already taken place means that the potential benefits of agent-oriented requirements capture are not exploited. Similarly the lack of a mechanism to

model agent goals or task delegation results in an equally abstract interaction model that does not provide the detail necessary to represent rich message content semantics. Prometheus is much more prescribed in that there is a clear route from requirements gathering, through goal elicitation and role allocation through to a more comprehensive specification model. Whilst Prometheus does provide a model that is 'closer' to the code, there is a clear emphasis upon data access rather than goal description and delegation, thus resorting to a coding approach to design. Tropos offers the most comprehensive range of representations by supporting the gathering of *early* requirements, whilst providing a means of managing the hard and soft goals elicited. The use of BDI concepts enables the abstractions to be specified in an implementation language that supports BDI constructs such as JACK. The process of verifying the requirements modelling process is less comprehensive and is either ignored, supplemented with another approach, or satisfied using the Formal Tropos tools. Tropos is also supported by the use of organisational patterns that assist the specification, analysis and design of an agent system, supporting the need for added relevance that an organisational metaphor can bring to the approach.

3. *An ability to capture and model high-level qualitative concepts at an* 'embryonic' *requirements stage.* Gaia permits concepts to be modelled at a high level, providing that they have been captured already. Prometheus supports the capture of requirements, though there is the assumption that the abstract concepts have already been defined. Tropos provides explicit support for the acquisition and management of early requirements for agent based systems.

4. *A guide to the elicitation of stakeholders and their goals, and be able to manage the discovery of system goals.* Both Tropos and Prometheus explicitly treat the elicitation and decomposition of goals in some detail, though it is only Tropos that provides guidance during early requirements. Gaia does not directly address the need to elicit goals and assumes that this is performed as part of the initial requirements generation.

5. *A mechanism for eliciting and deriving pertinent agent and domain concepts, allowing the representation and open expression of agent concepts such as: belief, desire, intention, role, society, task.* Gaia is abstract enough to allow these concepts to be accommodated, but offers little guidance as to how the process can be managed. Both Prometheus and Tropos support the specification of agent models for BDI platforms, however only Tropos directly supports and manages the process of high-level goal discovery.

6. *A process that includes an implicit model check to verify the elicitation of key domain concepts at the earliest opportunity. This process must be able to enable checking of the model's consistency, ideally with tool support.* Again the abstract Gaia approach does not provide explicit instructions to support the verification of models. Prometheus has a series of opportunities to check the models as part of the process, but there is no absolute demand for model verification. Tropos can be used in conjunction with the Formal Tropos tool, though this is an optional component.

7. *A process whereby focus is directed upon inconsistencies or parts of the model that are ambiguous.* This particular aspect is influenced by the

amount of detail that is required by each of the design approaches. Clearly there is much more (unspecified) work to do when using Gaia, and it is therefore likely that required features of the eventual system may be missing. Prometheus is very prescriptive and therefore provides an indication of any gaps in the models. However this does assume that the early requirements stage has been completed successfully and the the high-level concepts have been captured. Tropos therefore provides a much better foundation upon which the subsequent stages can be based, though unfortunately it lacks the later rigour imposed by Prometheus.

8. *A means by which domain terms, constraints and rules can be captured and represented in an ontology.* This is a particular weakness of Gaia in that there is no attempt to support the generation of an ontology. Prometheus and Tropos support the creation of ontologies in part by at least attempting to specify the key domain concepts, though there is no emphasis upon reasoning against the ontology in order to verify its existence.

9. *A representation medium that permits the transfer of models across domains, and that serves to complement other agent design methodologies.* Gaia is sufficiently abstract to be completely transferable. Prometheus and Tropos offer direct support for the use of JACK, whilst also supporting any other platform that embraces BDI constructs as the model representation. The rigour of Prometheus suggests that some support for early requirements gathering (which is absent) would be useful. Conversely Tropos would benefit from more rigour during the architectural phase, to make better use of the early requirements capability of the

approach.

10. *A process that is intuitive and enables novices and experts to design agent models.* All of these approaches require some familiarity with the agent design process, therefore this aspect will consider the extent to which agent expertise is required. Gaia is the most abstract and demands implementation expertise. Tropos is gaining maturity and offers good support for the capture of high-level system goals and stakeholders. Prometheus has a very prescriptive approach that permits implementations with some degree of repeatability to be produced. As such Prometheus restricts the flexibility or creativity that can be applied to the agent design process, by defining clearly what artefacts have to be produced and in what order.

For comparison, the three agent design methodologies discussed earlier were evaluated against these criteria, using the ranking method proposed by Sturm and Shehory (2003) and described below in Table 2.1. Each of the specific criteria was ranked from 1-7 and the results are summarised in Table 2.2.

2.9 Conclusions

So far we have looked at the concepts of agents and multi-agent systems, and started to explore some of the issues for AOSE. In particular we have seen that popular agent-oriented design methodologies have a general lack of support for ontology capture and modelling, and only (at the time of writing) does Tropos assist the analysis of high-level *early* requirements. Finally some criteria for an improved agent-oriented design framework are identified. Chapter 3 explores the modelling of agent concepts in more detail.

Rank	**Evaluation criteria**
1	Indicates that the methodology does not address the property.
2	Indicates that the methodology refers to the property but no details are provided.
3	Indicates that the methodology addresses the property to a limited extent. That is, many issues that are related to the specific property are not addressed.
4	Indicates that the methodology addresses the property, yet some major issues are lacking.
5	Indicates that the methodology addresses the property, however, it lacks one or two major issues related to the specific property.
6	Indicates that the methodology addresses the property with minor deficiencies.
7	Indicates that the methodology fully addresses the property.

Table 2.1: Evaluation rankings (Sturm and Shehory, 2003).

Characteristic	**GAIA**	**Prometheus**	**Tropos**
1. Process	4	6	5
2. Abstraction	4	5	5
3. Early requirements	1	1	5
4. Goal discovery	1	5	5
5. Agent concepts	2	5	5
6. Consistency checking	2	3	4[a]
7. Analysis by exception	2	3	3
8. Ontology support	1	2	2
9. Transferability	4	4	4
10. Intuitive	3	6	5

[a]If however, Formal Tropos is used (which is optional), the rating would be 7.

Table 2.2: Evaluation of agent design methodologies against desired characteristics.

Chapter 3

Modelling Concepts

3.1 Introduction

This chapter explores and reviews the use of an established method of conceptual modelling to assist the representation of agent and ontological concepts. Conceptual Graphs (CGs) and Peirce (pronounced 'Purse') logic are introduced as a means of building agent-based conceptual models that can be verified, whilst also assisting the creation of ontologies. In particular, CG type hierarchies are used to illustrate how CGs can implicitly provide the concepts and relationships required for an ontology, together with the capability to produce inference rules.

3.2 Capturing Domain Knowledge

As discussed in Chapter 2 a key challenge for an improved agent design framework is the ability to capture domain knowledge in a way that faithfully represents the needs of the intended system, whilst permitting the expression of that knowledge in the widest sense possible. Since ontologies can assist the design of new applications, be it through the process of capturing domain knowledge or the sharing and re-use of existing domain ontologies, it seems prudent to consider the development of such a framework.

Furthermore, 'early' requirements capture is important as it contains the high level goals (hard and soft) of the stakeholders. Conventional approaches to modelling, with the subsequent modelling iterations, can dilute these goals (desires) to the point where they lose importance. The capture and expression of high-level concepts is therefore fundamental to the requirement for a more faithful representation.

Whilst it is feasible that much of this work can be performed manually by the agent system designer, the potential complexity of these systems is such that it is inevitable that inconsistencies will present themselves. Therefore it is necessary to consider processes that support either the automation of tasks, or the individual steps are able to implicitly build a rigorous model. This would assist the agent system designer considerably, and reduce the reliance upon domain experts.

It follows that there is a need for a modelling environment which:

1. Utilises a notation that is rich, expressive and can tolerate both quantitative and qualitative high-level domain concepts;

2. Provides a mechanism whereby models can be queried, reasoned against and verified;

3. Supports the implicit capture and explicit expression of ontological data;

4. Imposes a rigour upon the modelling process.

The following section reviews the background to this research by considering an approach to conceptual modelling that attempts to address the criteria above.

3.3 Conceptual Graphs

Conceptual Graphs (CGs) are a means of representing knowledge using *concepts* and the *relationships* between those concepts. They are based upon the work of John Sowa (1984), who was influenced by the work of Charles Sanders Peirce, and semantic nets (Sowa, 2000). John Sowa's prime motivation was to be able to represent the semantics of natural language, such that meaning could be described in a '*logically precise, humanly readable and computationally tractable*' way (Sowa, 2000). The formal underpinnings provided by Peirce logic has enabled CGs to be used as an intermediary between natural language and computer oriented formalisms, and as a consequence they have been implemented in a variety of projects for information retrieval, database design and expert systems (Sowa, 2000).

The remainder of this chapter introduces the use of CGs and describes the features of this approach that are particularly appropriate for the capture and organisation of knowledge for an agent.

3.3.1 Notation

A CG contains concept and relation nodes that are linked together by arcs. Each arc has a direction that describes how the linked nodes (concepts and relations) should be interpreted. For example:

```
[Bicycle]->(Part)->[Wheel].
```

This graph, expressed in linear form (LF) makes the statement that, '*Part* of a *Bicycle* is a *Wheel*'. The following convention should be used when reading a graph:

```
[Concept_1]->(Relation)->[Concept_2].
```

This equates to 'the *Relation* of *Concept_1* is *Concept_2*'. The full stop indicates the end of the graph.

So far we have seen CGs represented in linear form (LF). An alternative representation of a CG is the display form (DF)[1]. The graph above would now look like Figure 3.1. From earlier, Figure 3.2 illustrates the DF of '*Part* of a

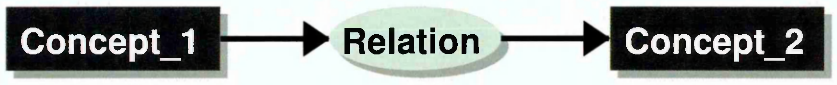


Figure 3.1: 'A relation of Concept_1 is Concept_2' graph in display form.

Bicycle is a *Wheel*'. Similarly, Figure 3.3 describes 'A bicycle is on the ground'. Note that the reading convention does not always give the 'best' grammar; the reading of graphs soon becomes intuitive and it should only be necessary to resort back to graph decomposition when a particular CG is complex. Often, rather than the graph being complex, it is the concept or relation names that are unsuitable. These should be revised accordingly until the graph becomes more readable.

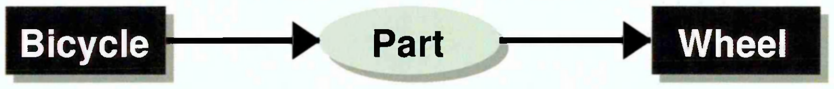


Figure 3.2: A part of a bicycle is a wheel.

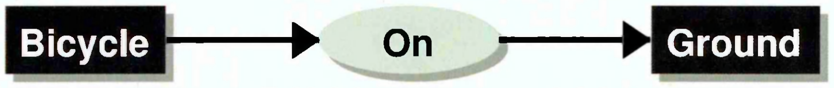


Figure 3.3: A bicycle is on the ground.

[1]All DF graphs were drawn using the CharGer tool (Delugach, 2006a).

Standard Language

Since conceptual graphs require interpretation from the inside out (starting with the relation), and then often from right to left, a significant barrier towards becoming proficient at reading graphs is presented. The period between decomposing graphs and reading graphs 'intuitively' is often too great for non-technical people, which could of course include domain experts. In such cases, a standard language exists to assist graph comprehension. Using standard language, conceptual graphs can be read either *in the direction* of the arrows, or *against* them. Tables 3.1 and 3.2 below, redrawn from the online course materials developed by Aalborg University (2006), illustrate the use of standard language when reading conceptual graphs. Applying these rules to the following graph:

`[Wheel]<-(Part)<-[Bicycle].`

Reading from *left to right*: "Wheel *is a* Part *of* Bicycle". From *right to left*: "Bicycle *has a* Part *which is* Wheel".

Exceptions

Unfortunately these rules are not all-encompassing and the exceptions are those graphs that are based upon prepositions. Figure 3.3 illustrates one such prepositional relation, 'on'. Expressed in LF the graph is:

`[Concept]<-(Relation)` "is a"	`[Concept]->(Relation)` "has a"
`(Relation)<-[Concept]` "of"	`(Relation)->[Concept]` "which is"
Example: `[Fat]<-(Attr)<-[Cat].` Fat "is an" attribute "of" Cat	Example: `[Cat]->(Attr)->[Fat].` Cat "has an" attribute "which is" Fat

Table 3.1: Reading conceptual graphs from left to right.

`[Concept]<-(Relation)` "which is"	`[Concept]->(Relation)` "of"
`(Relation)<-[Concept]` "has a"	`(Relation)->[Concept]` "is a"
Example: `[Fat]<-(Attr)<-[Cat].` Cat "has an" attribute "which is" Fat	Example: `[Cat]->(Attr)->[Fat].` Fat "is an" attribute "of" Cat

Table 3.2: Reading conceptual graphs from right to left.

```
[Bicycle]->(On)->[Ground].
```

Using the convention from earlier, the graph is read as "the *On* of *Bicycle* is *Ground*". This is clearly nonsense and does not assist comprehension. Applying the rules of standard language, the graph reads:

"*Bicycle* is an *On* of *Ground*" or "*Ground* has an *On* which is *Bicycle*".

Similarly the standard language offers no assistance, and the preposition should be stated as the graph is read in the direction of the arcs. Therefore `[Bicycle] -> (On) -> [Ground].`, becomes 'A Bicycle is On the Ground'.

Well-formed Graphs

Whilst it is useful to show multiple concepts and relations within a graph, the existence of a singular concept by itself, without any connecting arcs, is an acceptable, or 'well-formed' graph. For instance:

```
[Bicycle].
```

This is a well-formed graph, as it means 'There is a bicycle'. This type of CG is referred to as a *singleton*. Conversely a relation must have at least one arc attached to it for the CG to be deemed well-formed. Thus:

```
(Part)
```

is not a well-formed graph, whereas, `[Wheel] -> (Part) -> [Spoke].` demonstrates a well-formed graph as the relation has at least one arc associated with it.

3.3.2 Concepts

All concepts comprise a *type* and a *referent*, and are represented in the general form:

```
[Type: Referent].
```

An example would be:

```
[Bicycle: Brompton].
```

which means '*there exists a bicycle whose name is Brompton*'. Sometimes there may not be an explicit referent; in such a case the graph looks like this:

```
[Cyclist].
```

meaning '*there exists a cyclist*'. Whilst referents can be blank, types cannot.

Concept Types

Types allow the concepts to be categorised into groups of entities with similar characteristics. The ability to define and specify concept types is a fundamental part of building an ontology. However whilst it is useful to be able to specify different concept types, the resulting ontology is incomplete without some definition of the relationships that exist between the concepts. Inter-concept relationships are described by using *subtypes* and *supertypes.* An example of an ontology is as follows:

`Person, Vehicle` $<$ `Entity`

`Cyclist` $<$ `Person`

`Commuter, Professional` $<$ `Cyclist`

`Bicycle, Car` $<$ `Vehicle`

`Racer, Tourer, Folder` $<$ `Bicycle`

Referring to this ontology, `Person` is a *subtype* of `Entity`. `Vehicle` is a *supertype* of `Car`. Such relationships are written `Subtype` $\leq$ `Supertype`. Figure 3.4 illustrates this graphically. The lattice shows that every concept type is a subtype of `Entity`, and `Absurdity` is a subtype of every other concept. Since `Entity` is the supertype of every concept in the lattice (otherwise referred to as the *universal type*), everything can be referred to as being of type `Entity`. Similarly, no concept can be a subtype of `Absurdity`. Since subtypes inherit the characteristics of supertypes, an instance of the `Absurdity` type cannot be realised hence the name.

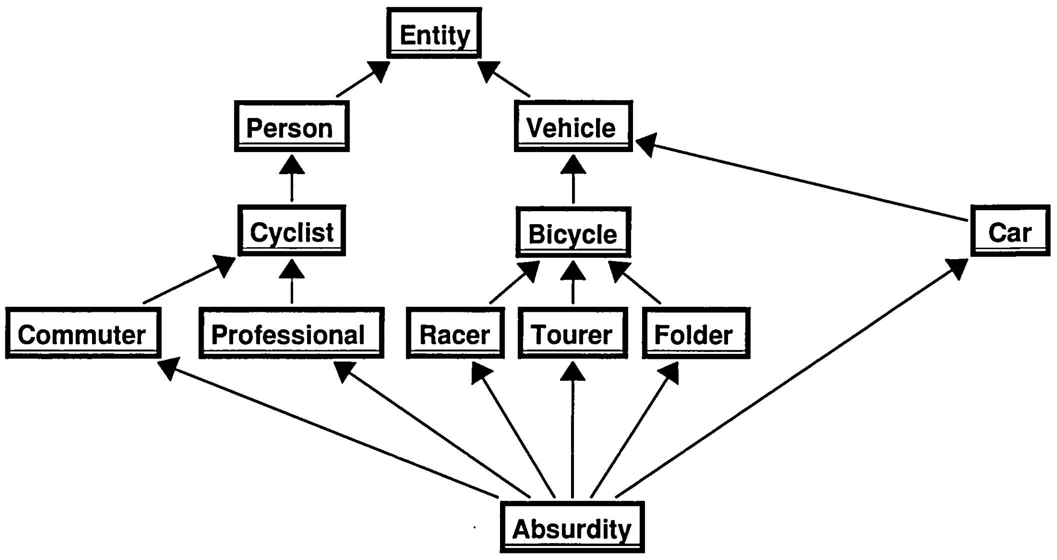


Figure 3.4: Lattice diagram of an example ontology.

The importance of the Entity and Absurdity types will be explained in Section 3.4.3.

Concept Referents

Referring to the earlier example of `[Bicycle: Brompton]`, the concept type is `Bicycle` and the referent is `Brompton`. In other words, `Brompton` is an *instance* of the `Bicycle` concept type. Some other examples of referents that conform to the ontology described above are:

```
[Person: Daniel]
[Cyclist: Lance]
[Bicycle: Brompton]
[Car: Land-Rover]
[Folder: Brompton]
```

3.3.3 Relationships

A CG is a *bipartite graph*, which means that the nodes can be partitioned into two distinct sets. In the case of CGs the concept nodes are joined to other concepts via relationships. Therefore an arc must connect a concept to a relation. Arcs connecting concepts together, and likewise relations, are invalid. Referring back to an earlier graph: `[Bicycle]->(Part)->[Wheel].`, the concepts `Bicycle` and `Wheel` are related by `Part`. Relations can also be classified by *type*, as well as *valence* and *signature*.

Relation Type

Relations are similar to concepts in that they must have a type. However unlike concepts, relation nodes do not have referents. A relation type is determined by the name given to the relation. Examples of relation types so far are:

```
(Part)
(On)
```

Each of these types has been used to relate concepts to each other, and the name (and therefore type) has been chosen to suit the situation that the CG is describing. Other relation types include:

- `(Obj)` - Object
- `(Srce)` - Source
- `(Rcpt)` - Receipt
- `(Chrc)` - Characteristic
- `(Agnt)` - Agent

Sowa illustrates examples of concepts and relations in his conceptual catalogue (Sowa, 1984) to assist in the generation of graphs, but the list is not meant to be definitive and other words can be introduced as required. Long before the advent of agent oriented computing, Sowa introduced the concept `Agnt` (agent) as a means of relating `act` concepts to `animate` concepts. For example:

```
[Cycle]->(Agnt)->[Person: Richard]->(Loc)->[City: Sheffield].
```

This can be read as: '*There is a person called Richard who is the agent of cycle. This same person is located in Sheffield*'. A less unwieldy representation might read thus: '*Richard is cycling in Sheffield*'. Whilst there are parallels with agent computing in terms of how Sowa uses 'agent' as a relation, it is important to recognise the distinction as an agent relation might not include all of the characteristics identified in Chapter 2 as befitting an intelligent agent. It follows that the presence of an agent relation in a CG also does not necessarily identify an intelligent agent, nor indeed a multi-agent system.

Valence

Valence refers to the number of arcs that belong to a relation. The number of arcs that belong to `Agnt` is always two, as the relation always connects an `act` concept to an `animate` concept.

```
[Cycle]->(Agnt)->[Person: Richard]-
->(Loc)->[City: Sheffield].
```

Similarly `Loc` (location) connects a concept of any `type` to a `place` concept. If n refers to the valence of a relation, then it is described as an *n-adic* relationship. The last graph demonstrates examples of *dyadic* or 2-adic relations with `Agnt` and `Loc`. The application of tense to a graph illustrates a *monadic* or 1-adic relation:

```
(Past)->[Situation: [Cycle]->(Agnt)-
->[Person: Richard]->(Loc)->[City: Milan]].
```

'*In the past, there was a situation where Richard cycled in Milan*'. `Betw` (Between) is an example of a triadic (3-adic) relation.

```
[Person: Daniel]<-(Betw)-
<-1-[Person: Mum]
<-2-[Person: Dad].
```

This graph reads: '*Daniel is between Mum and Dad*'. The LF graph also illustrates how the order of the concepts is mandated by the numerical designation.

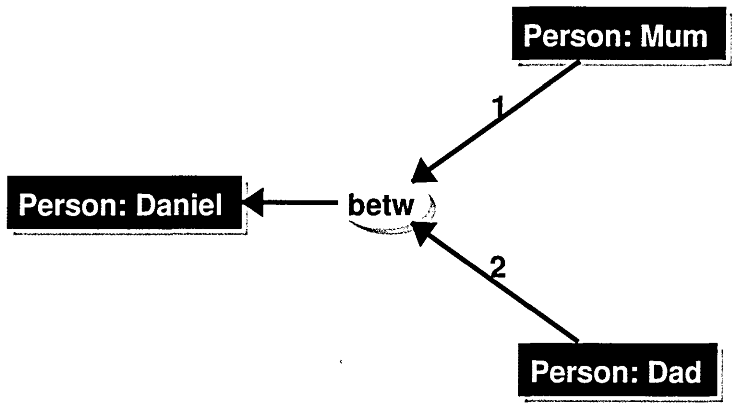


Figure 3.5: Example of triadic relation.

Figure 3.5 illustrates the DF equivalent. The third arc is left without a number, though by deduction it is the only arc that points away from the relation, hence it need not be considered in the same way as the other concepts.

Signature

Sowa (1984) explains that each n-adic relation (r) has a signature of n concept types associated with it. For the triadic relation `Betw`, three concepts are associated. For the diadic relation `Agnt`, the signature is two concepts, `Act` and `Animate`. This is written as: `<Act, Animate>`. The signature enforces the concept types that can be related, so for `Agnt`, the following is true:

```
[Cycle]->(Agnt)->[Person].
```

The two concepts that make up the signature are `Cycle` and `Person`. `Cycle` is a *subtype* of Act and `Person` is a *subtype* of `Animate`. Relation signatures also provide one other piece of information. The direction of the arcs between concepts and relations affects how the graph is read. Each signature indicates the order in which the arcs should be interpreted. For the previous example, the signature of `Agnt` is `<Act, Animate>`. Therefore the arc points *away* from the `Act` concept and *towards* the `Animate` concept.

`[Act]->(Agnt)->[Animate].`

For monadic relations, of which the signature is one, the arc points away from the relation.

3.4 Ontology

Chapter 2 introduced ontology as a means of describing domain knowledge in order that collaborating agents can share understanding. In essence an ontology is a categorisation of the entities that exist in a domain, and it is conceivable that an agent would require this information in order to process the inputs from its sensors as well as considering incoming messages from other agents. This section explores ontologies and examines how conceptual graphs can be utilised to build new ontologies.

3.4.1 Types

Conceptual graphs are composed of two types:

1. Concepts

2. Relations

With reference to Figure 3.4, concept types are arranged within a lattice structure, and it is possible to deduce information about a particular concept type from the position within the lattice. Each concept type refers to a collection of entities that have similar characteristics, whether they are concrete or abstract. Whilst a type may describe a concrete entity, the type itself is still an abstract label, as it refers not to an individual entity but to the collection of entities. Consequently, `Daniel` is a specific instance of the type `Person`.

3.4.2 Defining Types

In order to compile a robust ontology, it is necessary to be able to accurately define a type. There are four basic approaches:

1. By *extension.* Types are defined by extension when a comprehensive list of each instantiation is made. Therefore the type `TheHillFamily` contains `<Richard, Hazel, Daniel>` as these are the names of each individual in the Hill Family.

2. By *intension.* Rather than listing each individual belonging to this type (which might result in a very long list), the properties of each member are listed. This defines whether or not an individual can conform to the overall properties of the type. For example, if the intension of the type 'bird' consists of the characteristics 'lays eggs', 'has wings', 'flies', and 'builds nests above ground', every member of the category must demonstrate all of the defining properties.

3. By *axiom*, which is a statement that need not be proven or, has been accepted that a proof is not required.

4. By *referring to other types.* When concept types are created, it is possible to create new types by defining additional criteria. For instance, `FoldingBicycle` is 'a `Bicycle` that has a hinge in the main frame to permit folding to a smaller overall size for storage'. When new concept types are created with conceptual graphs (in particular conceptual type hierarchies discussed in Section 3.4.3), each new type is specified by referring to previously defined types.

Since the specification of types is central to conceptual graphs, it follows that type definition can only help the creation of ontologies.

3.4.3 Type Hierarchies

As shown earlier (Figure 3.4), related types can be presented as a *type hierarchy* or *lattice.* The subtype relation is used to relate concepts by a partial order. `A` $\leq$ `B` means that `A` is a *subtype* of `B`. Referring back to Figure 3.4, `Person` $\leq$ `Entity`, `Cyclist` $\leq$ `Person`, `Commuter` $\leq$ `Cyclist`, and so on.

When a subtype relation is declared such as `A` $\leq$ `B`, then either `A` *is* `B` or `A` is a *specialisation* of `B`. It follows that `Commuter` $\leq$ `Cyclist`, since `Folder` is a specialisation of `Bicycle`. When a type is specialised, the original properties of the supertype are inherited by the subtype, with the addition of some extra constraints.

A `Folder` therefore, has all the properties of `Bicycle` with some extra characteristics such as:

- It has a hinge in the main frame to permit folding;
- It has a hub gear;
- It has a folding left-hand pedal.

A *proper subtype* relation signifies that the subtype is only a specialisation of the supertype and they are not the same. This is represented as `A` $<$ `B`.

Type hierarchies can also be used to describe *transitivity.* For example:
If `Folder` $\leq$ `Bicycle` and, `Bicycle` $\leq$ `Vehicle`, then

`Folder` $\leq$ `Vehicle`.

Entity and Absurdity

`Entity` is the universal supertype (often referred to as '$\top$') of every type in a hierarchy. Conversely, `Absurdity` ($\perp$) is a subtype of everything in the hierarchy. Since `Absurdity` cannot be equivalent to `Entity` it is a *proper subtype* as follows; `Absurdity` $<$ `Entity`. Similarly, `Entity` $>$ `Absurdity` holds also, as `Entity` is a *proper supertype* of `Absurdity`. The universal supertype enables any entity to be represented, which can assist the generation of graphs before all of the types have been specified.

3.4.4 Lambda Expressions

Lambda expressions allow graphs to illustrate referents to other concepts. The following expression specifies a person, with an associated λ referent:

```
[Person:  λ]<-(Agnt)<-[Cycling].
```

"A person, λ, is cycling" More commonly the λ is replaced with the `?x` designation. Thus the previous graph would read as follows:

```
[Person:  ?x]<-(Agnt)<-[Cycling].
```

"A person, ?x, is cycling" Another example is:

```
type FoldingBicycle(*x) is [Bicycle: ?x]->(Chrc)-
->[Frame: Folding].
```

If we consider another graph:

```
[FoldingBicycle:  Brompton]->(Attr)->[Colour:  Black].
```

Both graphs can now be combined and expanded to give the following:

```
[[Bicycle: ?x]->(Chrc)->[Frame: Folding]: Brompton]->(Attr)
->[Colour: Black].
```

Therefore the new type `FoldingBicycle` can be added to the type hierarchy since `FoldingBicycle` is a proper subtype of `Bicycle` due to the specialisation of `Bicycle`:

`Bicycle` > `FoldingBicycle`

3.4.5 Coreference Links

Figure 3.6 illustrates how graphs can be related to other graphs via coreference links (Sowa, 2000). The `Person: Richard` has two beliefs, bounded by a `Proposition` context. Thus `Cycling` is `Fun`, yet `Decorating` is `Tedious`. The dashed line indicates the coreference link (or line of identity), meaning that the link refers to the same instance of that concept. Therefore the `Person` who believes that `Cycling` is `Fun` and `Decorating` is `Tedious` is the *same person* who is `Decorating`.

3.4.6 Projection

As a graph increases in complexity with the addition of more concepts, types and relations, it becomes more *specialised.* Similarly the substitution of specific referents for generic referents, or subtypes for types, increases the 'uniqueness' of a graph. Since a general graph can be specialised in many different ways, the general graph must exist within all of the specialised variants. A general graph is said to *project* into a specialised graph. If the projection can exist beyond one graph, then this is referred to as a *common generalisation.*

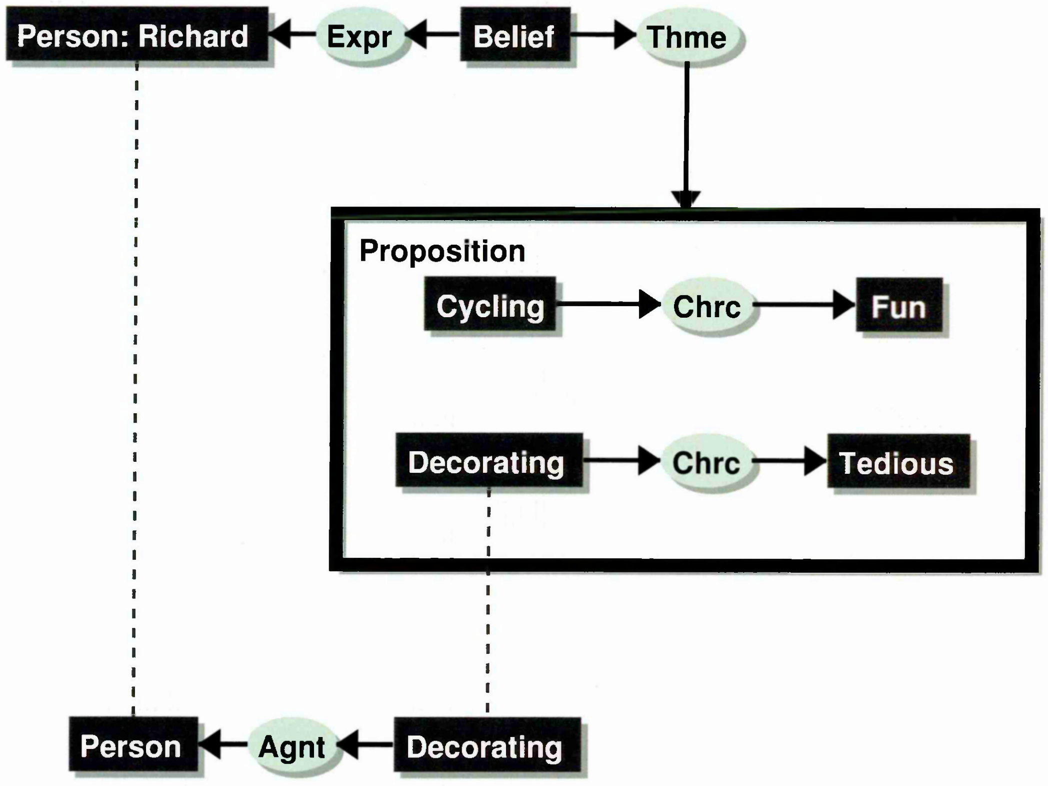


Figure 3.6: DF Graph illustrating coreferent links between graphs.

Combining Graphs

Projection is important when graphs are to be combined to produce larger, more specialised graphs. As graphs are joined, new projections become possible. The largest projection, whereby the maximum commonality between both graphs is achieved, is referred to as a *maximal join*. Once the graphs have been joined, the resulting projection is referred to as a *common specialisation*. This is now illustrated with an example.

Figure 3.7 shows a graph that reads:

"A Cyclist located in Sheffield is a Brompton enthusiast".

Figure 3.8 reads:

"Richard, a cyclist, is enthusiastic about his black bicycle".

We shall now consider how these two graphs can be joined. Using the following type hierarchy, we can attempt to identify some projections.

```
Cyclist < Person

Bicycle < Vehicle
```

`Person`, subtyped as `Cyclist`, exists in both graphs, therefore there is the possibility of a projection. However, Figure 3.9 illustrates that two other graphs exist that are potentially larger projections. Figure 3.10 shows the

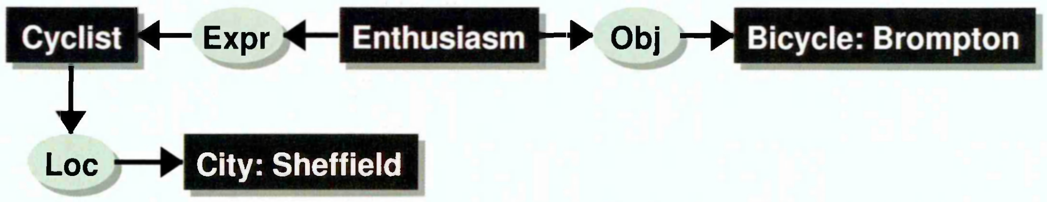


Figure 3.7: First graph to be joined.

common generalisation graph that is:

```
[Cyclist]<-(Expr)<-[Enthusiasm]->(Obj)->[Bicycle].
```

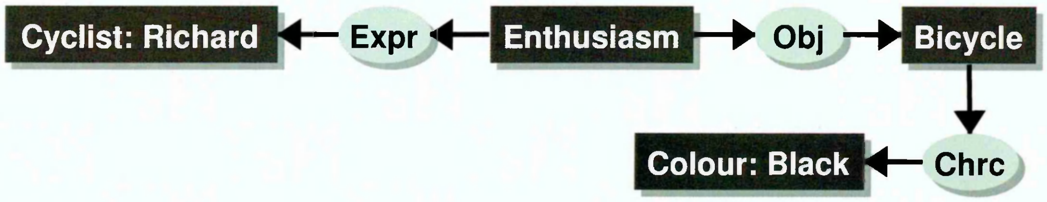


Figure 3.8: Second graph to be joined.

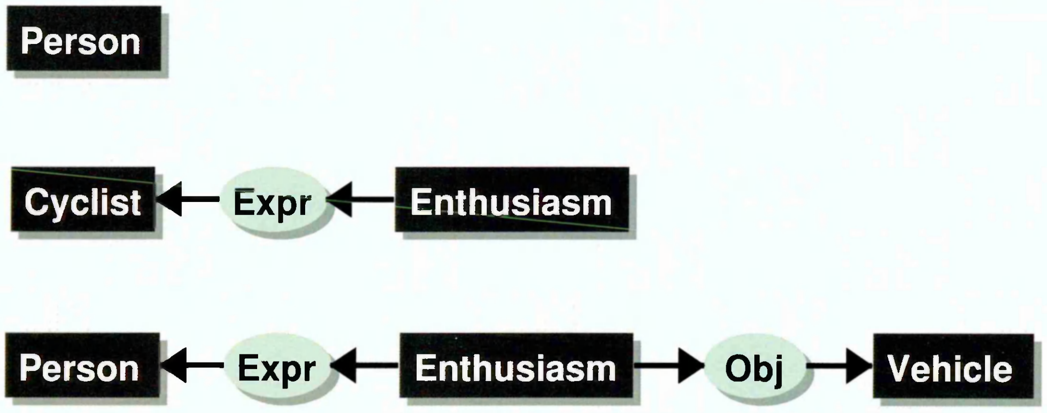


Figure 3.9: Possible projections.

The common specialisation in Figure 3.11 shows the concepts and relations where the two original graphs join. We can now see that `Cyclist` has been specialised to `Cyclist: Richard` and `Bicycle` to `Bicycle: Brompton`. This graph can be extended further by including the specialised concepts `City: Sheffield` and `Colour: Black`, together with the associated relationships `Loc` and `Chrc` - the *maximally joined* graph in Figure 3.12, is also a *common specialisation*. It is important to realise that graph joining assumes that the contexts are identical, and therefore by implication that the concepts are known to be coreferent. `Cyclist` might be any cyclist other than `Richard`, and `Bicycle` could feasibly be a `Pace RC3` or any other unspecified type. 2

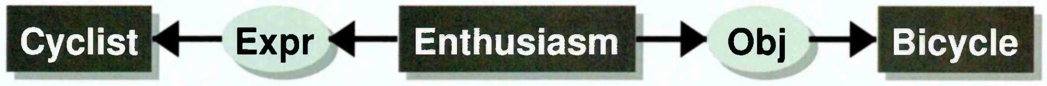


Figure 3.10: Common generalisation.

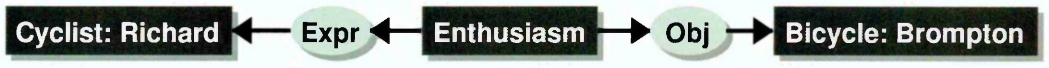


Figure 3.11: Common specialisation.

3.4.7 Predicate Calculus

Conceptual graphs map readily to predicate calculus. Using the 'Bicycle on the ground' example from Figure 3.3, in LF the graph reads:

```
[Bicycle]->(On)->[Ground].
```

This is represented in Conceptual Graph Interchange Format (CGIF) as:

```
[Bicycle:  *x] [Ground:  *y] (On?x?y)
```

To translate to predicate calculus, the following mappings are used:

- Relations become predicates;
- Arcs become arguments;
- Concepts become typed variables.

Thus the following is derived:

```
(∃x:Ground)(Bicycle) ∧ on(Bicycle, x)
```

3.4.8 Inferencing

Sowa developed Conceptual Graphs as an existential notation, permitting direct mappings between graphs and first order predicate logic (Sowa, 2000), the basis of which is the logic of Charles Sanders Peirce. This capability enables CGs to be inferenced against, allowing the representation of concepts

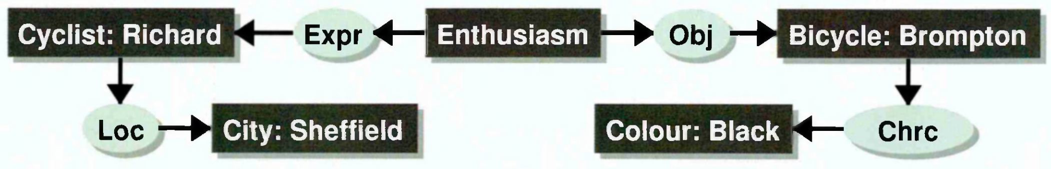


Figure 3.12: An extended common specialisation resulting in a *maximal join*.

and reasoning between those concepts. This is particularly attractive for the representation of complex systems since the graphical view (display form, DF) captures 'visual semantics', whilst also supporting logic and inferencing.

To illustrate, consider the following simple example:

`if Graph A then Graph B`

This can be interpreted as:

'If ***`Graph A`*** *projects into any graphs in the knowledge base, then* ***`Graph B`*** *can be asserted'*.

Logically this could be written as:

`not (Graph A and not (Graph B))`

Figure 3.13 illustrates this using DF. It can be seen that the 'Knowledge Base Graphs' dominate both `Graph A` and `Graph B` as they are contained within a negative context (black border). Indeed, `Graph B` is also dominated by `Graph A`, as it is in yet another negative context. The example above illustrates that parentheses replace the black borders in DF. If a graph projects

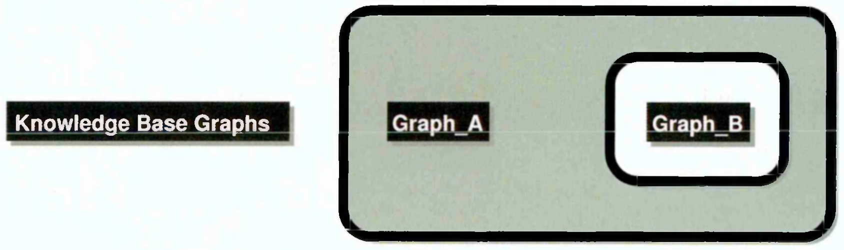


Figure 3.13: 'If Graph A then Graph B'

into a dominating graph, then it can be *deiterated*. Therefore, if Graph A is projected into the Knowledge Base Graphs the following would occur:

`((Graph B))`

Since each pair of the parentheses represents a negative context, the statement reads logically as:

`not(not Graph B)`, therefore `Graph B` is asserted as true.

The process of removing oddly-enclosed negative contexts is known as *double negation* or *denegation*.

Repeating these operations graphically, we derive Figures 3.14 and 3.15.

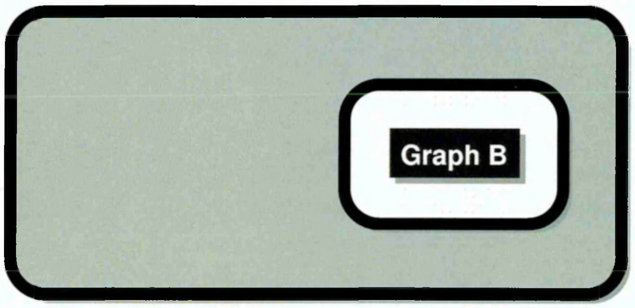


Figure 3.14: Deiterated graph

Figure 3.15: Denegation

To further illustrate the power of this approach for designing more complex models, let us consider another example:

'A resident of Sheffield who is a taxpayer can receive care from the United Kingdom Welfare system if they are ill.'

We also have a particular case, *'Betty'*, who will be used to test our model. *'Betty is a taxpayer, resident in Sheffield who is ill. Can Betty receive care from the United Kingdom Welfare system?'*

Thus these two statements are represented by the initial graphs in Figure 3.16.

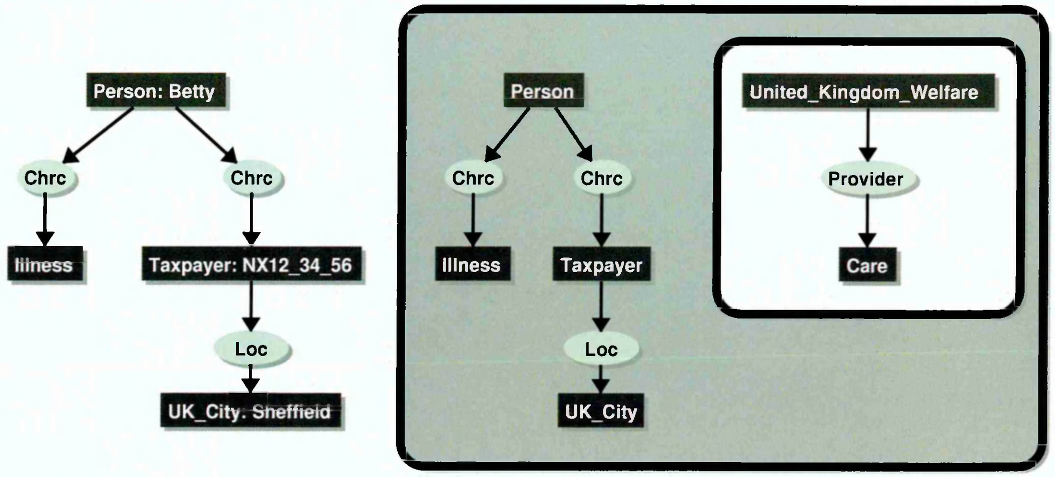


Figure 3.16: Original graphs

The first step is to *specialise* the projecting graph with those of the query graph. As shown in Figure 3.17, `Person`, `Taxpayer` and `UK_City` are all specialised in the projecting graph to become:

```
Person: Betty

Taxpayer: NX12_34_56

UK_City: Sheffield
```

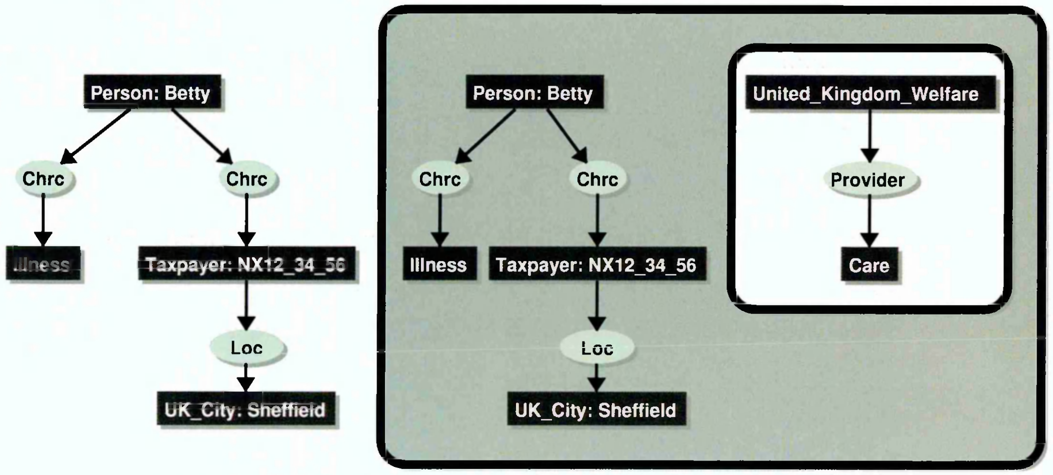


Figure 3.17: Specialised graphs

The projecting graph, once specialised with the query graph, can now be *deiterated*, giving Figure 3.18.

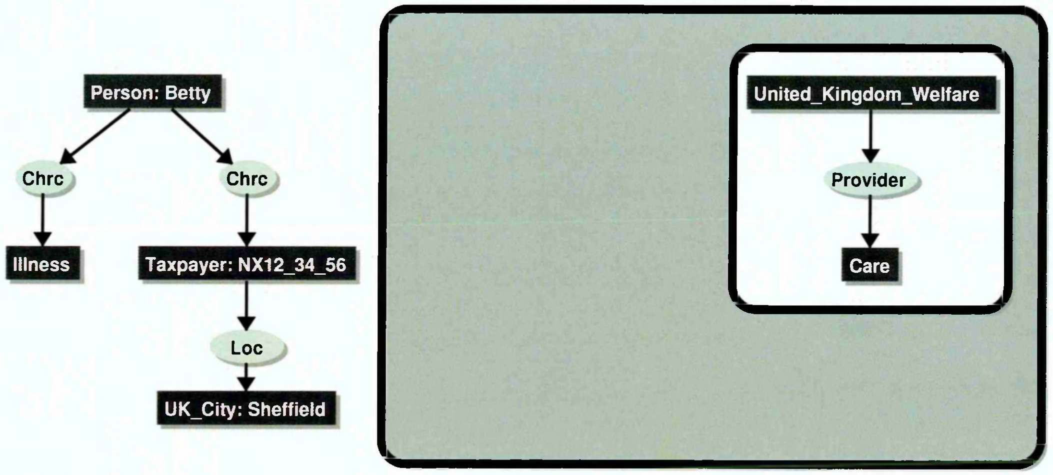


Figure 3.18: Deiterated graphs

Finally the two negative contexts can be removed by *denegation* to leave Figure 3.19. Thus Betty does receive care from the United Kingdom Welfare system, since she has the characteristics of 'illness', is a taxpayer and is resident in Sheffield.

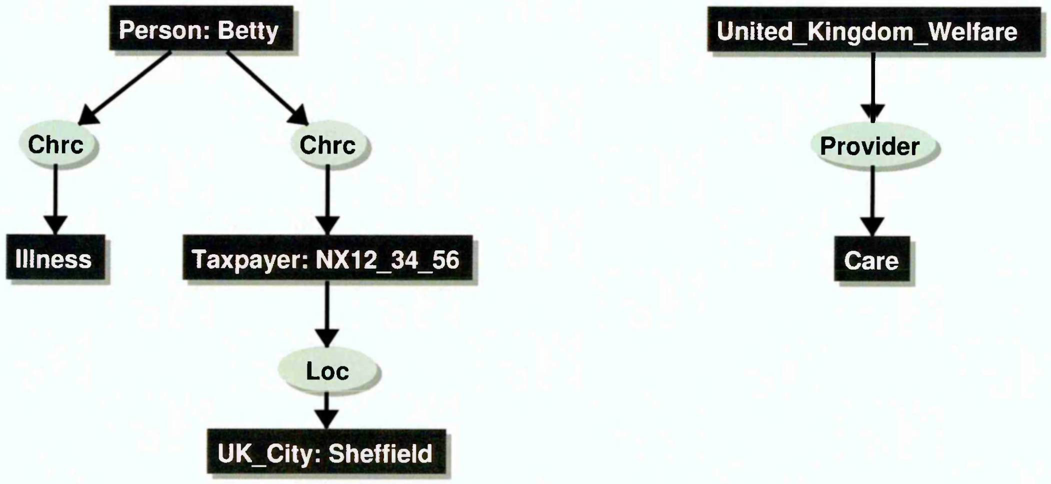


Figure 3.19: Denegation

The graphical placement of concepts into contexts allows dominating concepts to be identified and the resulting number of contexts to be reduced. This is particularly powerful when combined with agent models, as the initial graphs enable high-level concepts to be captured, yet the inferencing capability permits the models to be queried in a repeatable way.

3.5 Discussion

This chapter has introduced Conceptual Graphs as a means of representing agent and ontological concepts. The underlying formality of CGs means that not only can knowledge be captured and represented, but also the models can be reasoned against. Reasoning is an important capability for agency, which any agent design approach must be able to accommodate.

Section 3.2 presented some criteria by which a modelling environment might be evaluated. Firstly, CGs provide a notation that permits the richest concepts to be represented, and it is tolerant of qualitative, as well as quantitative concepts.

Secondly, the inferencing capabilities permit graphs to be queried, enabling model checking at a conceptual level. This is seen as a key feature that would serve to address the difficulties and ambiguity faced when gathering 'early' system requirements.

Ontological concepts are fundamental to modelling with CGs, and are a crucial part of building agents that can communicate and share knowledge across the myriad repositories that exist in an open environment. The use of type hierarchies enables ontologies to be created as graphs are assembled. Furthermore, the iteration of graphs also enables lattices to be updated in accordance with any new modelling insight.

There is a dichotomy however, between the richness of expression possible with CGs and the desired modelling rigour; whilst CGs can be mapped to First Order Logic, concepts and relationships can be added, removed and subtyped at will. It is conceivable that the breadth of expression possible, whilst attractive when gathering high-level organisational concepts, may lead to a lack of coherence and consistency when attempting to refine the models and build agent design specifications. Whilst CGs can contribute much towards the development of agent system models, there is a need for some overall rigour or indeed a design metaphor, that can be used to guide the design process.

3.6 Conclusions

This chapter has considered the needs of a modelling environment and has proposed Conceptual Graphs as a modelling notation. CGs provide a rich notation that enables high-level agent and organisational concepts to be captured and represented, whilst providing mechanisms for model checking and transfer to other representations such as FOL. For the agent design framework to be complete however, there still remains the requirement for an overall design metaphor that can permit the richness of early requirements capture, whilst also providing the discipline of process to iterate design models. Chapter 4 will explore how CG models can be rigorously developed to take account of not only the knowledge requirements of an agent-based system, but also the operational aspects such as business protocols, which govern how individual agents will behave in their intended environment.

Chapter 4

A Unifying Framework

4.1 Introduction

One of the criteria required in Chapter 2 is that of being able to check models prior to implementation, and if possible, to introduce model checking at the earliest opportunity. Chapter 3 introduced Conceptual Graphs as a means of offering the necessary formality for model-checking, whilst including a notation that enables a rich expression of high-level concepts. As such, there is now a suitable foundation for the modelling of domain knowledge, upon which the protocols by which an agent will act can be built. It is the business and organisational protocols that provide the relevant 'business rules' for a system and so it is crucial that any design framework must utilise a design metaphor to guide the agent system designer. This chapter looks at some theoretical underpinnings upon which an agent design framework might be based. Event accounting is explored and offered, through the Transaction Model, as a means by which conceptual organisational models can be queried and tested during the early requirements gathering process. A transaction ontology is produced from the model and proposed as part of the process for an improved agent design framework. Finally, use of the framework is demonstrated by way of an exemplar case study in the community healthcare domain.

4.2 The Need for a Metaphor

Whilst Conceptual Graphs offer the combination of an expressive notation and formal rigour, this is not enough in itself to provide a useful agent design framework. Conceptual Graphs, and other notations for that matter, can be used to develop abstract agent models with varying degrees of success. The ability to capture early requirements and incorporate model checking, whilst building a rudimentary ontology, are all facets that can improve the design process if a suitable framework exists to guide the designer through the maze.

What is lacking so far, is an over-arching framework that enables the powerful capabilities of Conceptual Graphs to be used in a business setting. The following sections explore an approach to unify the various capabilities discussed so far, within a framework that serves to exploit the potential of agents and also produce model artefacts that are faithful representations of the eventual system.

4.2.1 Complex Systems

Prior work (Hill et al., 2004) demonstrated that consideration of the qualitative aspects of complex agent managed community care systems gave insight into concepts which had not been clear at the outset, thereby demonstrating a greater need to more accurately map the problem domain. Lucid representations of qualitative and quantitative transactions have been demonstrated by Sowa (1984) using Conceptual Graphs. This was not only to accurately record complex interactions, but also to provide a means of eliciting domain facets that are difficult to determine with other more recognised notations.

An aspect of the Conceptual Graphs approach that is particularly relevant to agent systems is that the production of Conceptual Graphs, and the resulting predicate logic, can be easily transferred across domains using Conceptual Graph Interchange Format (CGIF) and Knowledge Interchange Format (KIF) (Harper and Delugach, 2003). This could assist the rapid generation of domain ontologies, whilst also considering qualitative issues from the initial modelling activities. This capture of the qualitative transactions allows much broader issues to be modelled, and through an iterative process, representations can 'drill-down' to reveal new aspects. Complex agent systems need to manage an enormous range of services, and this inevitably will include the resolution of unsatisfactory service, as well as the provision of satisfactory service.

The use of Conceptual Graphs for such analysis only serves to simplify the process if either the system designers have access to a domain expert, or they have the domain expertise themselves. It is therefore necessary to establish the following:

1. A suitable guiding metaphor for the representation of business operations;
2. An implicit means of providing model checking of *domain processes*;
3. A series of discrete activities for the production of design artefacts.

Firstly a guiding metaphor shall be explored.

4.3 Event Accounting

McCarthy (1979, 1982) and Geerts and McCarthy (1991) have proposed a framework for understanding accounting, based upon traditional bookkeeping that avoids the restrictions of double-entry systems. It attempts to address the difficulties experienced when accounting for qualitative entities in enterprises, and is based upon the notion of *economic scarcity.*

Double-entry bookkeeping allows all economic events to be documented in a ledger. The ledger is divided into two parts, *debits* and *credits.* As each business transaction is completed, the value of each transaction is entered into the ledger, thus allowing an economic view of the enterprise to be created.

The traditional bookkeeping approach however, assumes that all of the economic events have a prescriptive monetary value and therefore cannot take account of qualitative amounts.

For instance, an individual may wish to become a Landlord and purchase a property to rent out to tenants. Using the double-entry bookkeeping model, a debit of £150,000 would be recorded in the Cash Account, and a corresponding credit of £150,000 would be recorded in the Fixed Assets Account. Additionally, the potential Landlord recognises that there might also be some other, more qualitative 'costs' associated with this transaction. These might manifest themselves as:

- Reduction in time spent with family;
- Reduction in time available to complete PhD studies;
- Reduction in time available for research funding applications.

The benefits of not engaging with the transaction are all visible; a stable and rewarding upbringing for a young family; recognition for a sustained research effort resulting in a contribution to a body of knowledge; increased income generation for the University and enhanced reputation amongst peers. Such costs and benefits are difficult to quantify and as a result do not rest easily within the double-entry model as it stands. This is an example of the qualitative, rather than purely quantitative, exchange of resources (Piaget, 1973), indicating that there are many more aspects to explore and scrutinise in complex multi-agency domains.

The Resource-Event-Agent (REA) model (McCarthy, 1979, 1982) is an attempt to abstract away from the prescriptive nature of double-entry bookkeeping, yet still permit the recording of *exchanges of economic resources*, or *transactions*. To quote Ijiri (1967):

> "In a sense, the economic activities of an entity are a sequence of exchanges of resources - the process of giving up some resources to obtain others. Therefore, we have to not only keep track of increases and decreases in the resources that are under the control of the entity but also identify and record which resources were exchanged for which others."

Event accounting with REA enables models to be constructed that reflect business activities which *may* include monetary transactions. These models are built using the following core concepts:

- *Resource* - any resource that is the subject of an exchange or *transaction*;
- *Event* - the activities that are required for a transaction to take place;
- *Agent* - a person, system or organisation that participates in the transaction.

As such, REA captures the essence of accounting by providing abstract constructs to model organisational transactions, whilst including the bookkeeping notion of *duality*. The duality relationship permits two economic events to be represented as a mirror-image exchange of resources, thereby forming the basis of a transaction.

4.3.1 Modelling an Enterprise

REA supports the modelling of enterprises by facilitating the representation of non-accounting activities. Each economic process embodies two, opposing, economic events (associated by a duality relationship) that exchange scarce resources.

Indeed many organisational scenarios are rich with qualitative transactions. Each transaction concludes when the relevant parties have gained from the participation, and is represented as a *balance* in that very *debit* is countered by a *credit*. The inclusion of a balance within the transaction ensures an implicit validation that the transaction has occurred successfully. The agent transactions evident in community healthcare systems (Hill et al., 2004) are one such example that a desire for robust multi-agent systems must be underpinned by a solid transaction foundation. Figure 4.1 illustrates the REA model of a transaction using Conceptual Graphs (Polovina, 1993).

In a MAS trading environment, the goal-directed behaviour of an agent dictates that success occurs when both parties have gained from their participation in a transaction. In essence, the transaction describes a condition where both parties have exchanged resources, resulting in a balance.

Figure 4.1 illustrates that all transactions comprise two `Economic Events`, denoted by `*a` and `*b`. The `transaction` is complete when both `Economic Events` balance, which indicates that `*a` always opposes `*b`, representing debits and credits. Additionally there are two related `Economic Resources`, `*c` and `*d`, each having independent `source` and `destination` agents. The `Inside Agent` and `Outside Agent` refer to the parties involved with the transaction. The `Inside` and `Outside` prefix denotes the relative perspective of the transaction for each party.

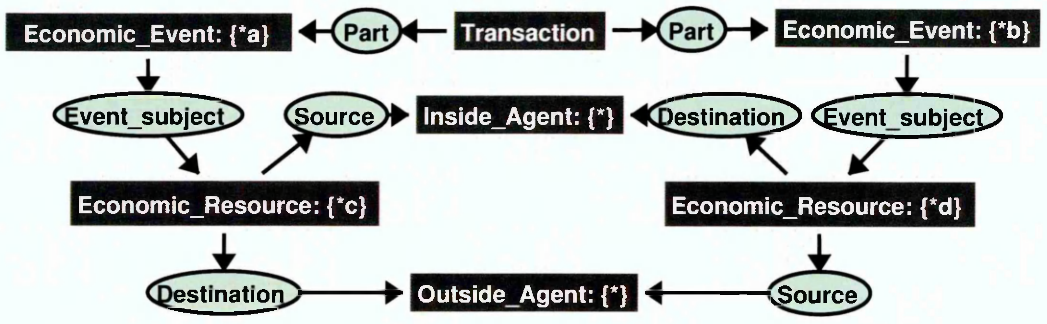


Figure 4.1: The Transaction Model (TM) Graph.

The TM graph provides guidance for modelling organisational processes in an abstract way which satisfies in part the first criterion in Section 4.2.1. It also introduces the concept of *balance*, which supplements the existing capability of the Conceptual Graph notation for model-checking, by ensuring that the TM is completely populated (criteria 2). This provides two significant benefits for the agent system designer:

1. A transaction can only be deemed complete when all of the corresponding nodes have been populated;

2. Each concept reflects a type in the type hierarchy, which in turn forms the basis of an ontology. In particular, the choice of term to represent a concept can affect how a concept is understood or processed in the future. The TM forces discussion about the most appropriate domain term (and its name in the type hierarchy, and subsequently the ontology) at a very early stage.

Since the use of Conceptual Graphs allows qualitative concepts to be represented, the TM graph also permits qualitative transactions to be represented. As such, concepts such as ‘quality of service received’ can be included within the TM. Of course for an agent to be able to compute a result for a qualitative transaction, it will require a more detailed representation of how the qualitative concept can be represented in a quantitative way. In such cases the TM serves to focus attention upon qualitative concepts, in order that a suitable representation is derived. The third criteria (Section 4.2.1) will now be addressed with the following proposed framework.

4.4 Transaction Agent Modelling (TrAM)

To briefly summarise the discussions so far, a framework is required that can:

- Address the gathering of early requirements and provide a means of representing those findings;
- Provide consistency checks for design artefacts;
- Implicitly build an ontology of domain terms;
- Provide a representation medium that permits the transfer of models across domains, and that serves to complement other agent design methodologies;

These basic criteria are addressed by the Transaction Agent Modelling (TrAM) framework. An overview of TrAM is shown in Figure 4.2.

Chapter 2 identified that support for the capture of early requirements for agent systems is generally lacking, and therefore provides much of the

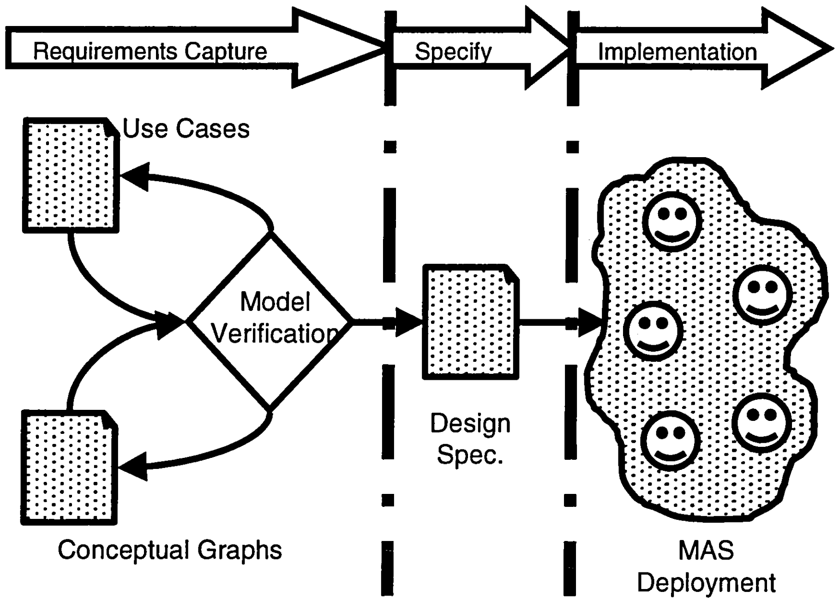


Figure 4.2: The TrAM Framework.

motivation for this research. TrAM employs Conceptual Graphs to enrich the gathering of early requirements by:

1. Providing a means of capturing and modelling high-level, qualitative concepts;
2. Exploiting the formal underpinnings of Conceptual Graphs and Peircian Logic to enable consistency checks in the notation to be made;
3. Using the Transaction Model (TM) to provide both design guidance and a mechanism for checking high-level transactions;
4. Deriving a hierarchy of types and a set of constraints upon which an ontology can be built.

The following sections and Figure 4.3 illustrate the TrAM process in more detail.

4.4.1 Capture Scenarios

The first step of the approach is to identify the key stakeholders in the system, represent them as UML actors and describe the roles that they undertake. Once the actors have been discovered, system interactions can then be described with the aid of written use cases and use case models.

4.4.2 Identify Agent Roles

After the individual use cases have been captured, the next stage is to identify the agents that will be required to complete a model of the eventual system. Prior work (Hill et al., 2004) has shown that actors can be mapped straight to actors.

4.4.3 Allocate Tasks to Agents

Once the agents have been identified, the next step is to identify and allocate tasks to each of the agents. Each task is taken from the use case descriptions and assigned to the perceived owner of that task, or the agent who is deemed to be responsible for its satisfactory completion.

4.4.4 Identify Collaborations

It is now possible to examine the collaborations between the agents. Each allocated task is considered in terms of identifying the agents that will be involved in the collaboration. As each task is mapped to an instance of collaboration between two agent types, each potential conversation is considered and new tasks are derived.

4.4.5 Apply Transaction Model

The application of the TM consists of two discrete activities. First, the concepts in the domain are captured and documented.

Model Concepts

The high level concepts are modelled as Conceptual Graphs. All of the concepts deemed relevant are recorded, irrespective of whether a quantitative representation exists or not.

Inference Model with Queries

Secondly the TM is populated and a type hierarchy is produced. Concept names are examined for suitability and modified where appropriate. Once the TM is complete, queries are created from the use case scenarios and used to test the TM using Peirce Logic. The results of these queries are used to further specialise the relevant TM and provide rules for the type hierarchy, so that domain specific constraints can be captured and included within the final solution. This process is iterative and will serve to elicit new stakeholders, goals and qualitative concepts, which are used to enrich the use case models generated earlier.

4.4.6 Design Artefacts

The TrAM Framework produces the following outputs:

- Use case descriptions and models that describe high level business processes and the relevant stakeholders;
- Task allocations for each agent;

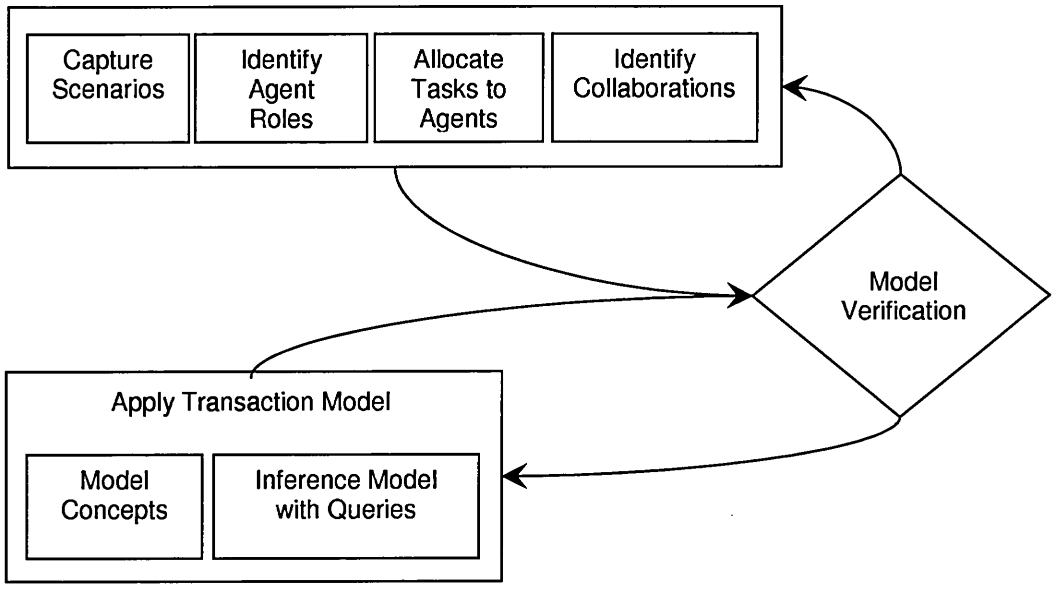


Figure 4.3: TrAM Process in Detail.

- Agent collaboration diagrams;
- Transaction Model Conceptual Graphs for each of the high-level transactions identified;
- A rudimentary ontology consisting of a type hierarchy together with domain constraint rules modelled with Peirce Logic.

These artefacts result in a design specification for the eventual system that does not impose a particular implementation architecture, and serves to complement agent design methodologies that lack a requirements gathering stage such as Gaia. Furthermore, since TrAM addresses *early* requirements, it can be used as a precursor to methodologies such as Prometheus.

4.5 Discussion

Agent based architectures provide the semantic interoperability capabilities to accommodate complex scenarios, enabling delegation, brokering, negotiation, cooperation and coordination to take place across the myriad systems. The notion of economic transactions provides a framework by which agent systems can be designed and implemented by addressing the following:

1. Gathering agent system requirements can be difficult, and the lack of model verification (even though use case models enable the various actor representations to be established) presents a significant risk that some details are missed from the first modelling iteration (Mellouli et al., 2002).

2. A successful MAS must include the ability to reason about the qualitative issues that exist in the community healthcare domain, and system designers must be able to challenge the issues from a business process perspective.

3. The capture and modelling of roles is a crucial step in the MAS modelling process (Depke et al., 2001), yet there is little guidance as to how roles should be allocated for best performance (Dastani et al., 2003b).

4. Ontological rules and terms enable semantic interoperability, however, system designers tend to rely on the process of eliciting use cases from existing processes to obtain the majority of the agents' behaviours.

5. The convenience of actor-to-agent mappings means that the assignment of agent behaviours is often arbitrary and based on current working practices. Whilst the capture of current working practices is vital to the

proper analysis of the existing system, this approach restricts the potential of an interoperable MAS solution. Additionally there is no implicit check as to the validity of a role, nor is there an audit trail of how the roles were delegated.

The TrAM approach enables the early elicitation of domain knowledge, and subsequent outputs for an ontology, whilst incorporating a robust transaction model from the beginning. This allows representations of agent-managed transactions to be assembled at a much faster rate, especially since there is greater confidence that the underlying design is based upon a solid framework. The key features of this approach are as follows:

1. CGs represent the problem in a more abstract way, and provide a foundation for modelling the knowledge exchange within a system. The abstraction is such that high-level, qualitative issues such as 'quality of health care received' are addressed, so it is feasible that the system is questioned from the point of view of concepts, rather than relying on an individual's prior experience.

2. CGs are similar to AUML in that there are some obvious mappings from concepts to agents, however there are also subtleties that CGs reveal more consistently.

3. The inherent balance check of the model ensures that ontological terms are agreed upon before the model is complete.

4. The transactions approach makes model verification implicit as any missing nodes (concepts or relations) render the model out of balance and unable to satisfy both sides of the transaction.

The TrAM approach allows agent based systems to be designed that exploit many different aspects of agent technology. In particular:

1. The development of conceptual models that support conversation semantics to characterise interaction between local and remote agents (Beer et al., 1999);
2. Matching capabilities of agents with current system needs using negotiation protocols (Beer et al., 2001);
3. Building scalable communities of agents (Beer et al., 2003b);
4. Defining semantic economic models to represent the complex relationships that exist between agents (Hill et al., 2005b) in an interoperable way;
5. Implementing agent architectural models that promote agent autonomy and privacy while ensuring that organisational commitments are realised.

The production of conceptual graph models enables higher-order issues to be captured, scrutinised and considered in an abstract way. This complements use case analysis and promotes early discussion. Use of the Transaction Model means that these concepts can be evaluated in a way akin to transactional analysis. The implications of 'duty of care', 'debt to society', and other high level concepts typically would attract little interest as they are difficult to model and even consider. The richness of conceptual graphs firstly allows these concepts to be represented lucidly.

Secondly, the application of the TM enables opposing concepts to be represented. Often one side of the transaction is clearly evident, but the opposing concept or concepts are not always clear. The application of the TM forces such hidden concepts to the fore, promoting discussion and consideration from the outset.

Thirdly, the ensuing discussion results in the generation of the most suitable term to represent each concept. This definition assists the documentation of an ontology, lessening the requirement for a domain expert. Indeed the process steers system designers so that at least the most pertinent questions can be asked of the expert, rather than requiring the system designers to be domain experts themselves.

Also, the ability to query the representation allows models to be tested and verified much earlier in the agent system design process. The use of a collaborative agent architecture for a community care system illustrates how agent cooperation can accomplish the provision of health care services and resources for both routine and emergency scenarios (Hill, 2007). This approach also indicates the possibility that agent-based technologies could be utilised in order to achieve distributed demand and supply issues within an integrated domain, whilst retaining existing actors and agencies.

Additionally, each actor's autonomy is still retained. Integrating external data sources by the use of information agents enables MAS models to be assembled rapidly and show that it is possible to integrate disparate data sources as part of an overall agent-based system, especially those associated with a variety of organisations.

4.6 Using TrAM

This section describes how the Transaction Agent Modelling framework is used by way of an exemplar case study in the community healthcare domain. In particular the complexities of healthcare payments are examined and the framework demonstrates the ease with which this complex problem was modelled and tested prior to design specification. Additionally the case study illustrates limitations of the framework and provides an opportunity to propose refinements.

4.7 Background

Previous chapters have discussed how a representation such as Conceptual Graphs might assist the design of an agent system. With reference to Event Accounting (Chapter 4) the use of the Transaction Model (TM) illustrates both how concept types and relationships could be identified earlier in the requirements gathering process. This enables an ability to query the models produced in order to develop a more comprehensive conceptual model. This chapter uses an exemplar scenario in the Community Healthcare domain to demonstrate how the draft Transaction Agent Modelling (TrAM) (Hill et al., 2006a) framework should be used, and explicates the individual steps of the process.

4.8 Rationale for Choice of Domain

In order to demonstrate the effectiveness of the TrAM approach a suitable case study is required. Community healthcare was selected as it is a domain that exhibits the following characteristics:

- Community healthcare management is inherently a multi-agency model, and there is a multitude of complex communications and interactions between many agencies; each of which demonstrate autonomous behaviours. The infrastructure is rich with policies, norms and traditions.
- The process of care delivery is distributed, in that each agent is required to deliver care either wholly or in conjunction with another agency.
- It is a domain that includes 'hard' and 'soft' goals; some of the goals are straightforward to elicit, yet there remains a large number of difficult to manage, qualitative goals, that may be excluded from the current care model.
- Collaboration is a fundamental system mechanism. Human agents regularly conduct transactions, but any process automation reaches far beyond the capabilities offered by the OO model, which explains why agents are required.

Additionally, prior work (Beer et al. (1999)) established that there were several shortcomings with regard to the modelling of complex community care management systems, which could potentially be addressed by the use of TrAM.

4.9 A Community Healthcare Case Study

The delivery of home-based community healthcare services to frail and disabled people provides a complex set of challenges for UK Local Authority Care Managers. Whilst there are arguments that support the perceived desire for people to remain in their home environment for as long as possible, it is extremely difficult to coordinate and control the wide range of separate care agencies, both in terms of effective delivery and efficient resource utilisation. Since there is a strong motivation to effectively manage the recipients' quality of life, there is a temptation to introduce redundant resources. This contributes towards high levels of cost.

Each care service is provided by an independent autonomous party, a practice that has been encouraged by UK Local Authorities in pursuit of cost savings generated by an open, economic market. Inevitably each party instigates and maintains their own management information systems, leading to a scenario that includes many disparate heterogeneous repositories. The prospect of integrating these resources seems rather onerous, and as a consequence there is a continued reliance upon more informal methods of control.

Beer et al. (1999) proposed the development of an architecture to address these issues that utilised collaborative intelligent agents (Wooldridge and Jennings, 1995) to mediate queries amongst the myriad agencies and platforms. As discussed in Chapter 2 it would seem that the reactive, proactive, social and autonomous behaviours exhibited by intelligent software agents have much to offer in terms of designing and developing more effective healthcare management systems.

The realities of attempting to accurately capture healthcare system requirements indicates that more assistance is required, particularly during the earlier stages of analysis, over and above a convenient mapping of actors to agents. In particular, collaborating agents must be able to share and re-use domain knowledge if they are to interact effectively. Therefore the means of capturing and expressing domain knowledge must be able to accommodate not only complex interactions, negotiation and brokering, but also the complicated qualitative information that exists in healthcare environments.

4.9.1 Developing an Agent-based Approach

The use of agents enables disparate systems to be integrated into a single, collaborative, cooperative system since the social abilities of agents permit conversational changes to be made between different agencies. Such an approach makes the job of monitoring the whole system much easier, with tangible benefits for UK Local Authority Care Managers who need to assemble the most effective package of care for each care recipient, without employing redundant resources.

Aside from appropriateness of care, a fundamental goal of a fully integrated community healthcare system is to provide a timely response to care requests, both from the care recipient and the care assessors such as Social Workers (SW) and Occupational Therapists (OT). Such a system should be able to negotiate at many levels if disparate, autonomous care services are to be managed and coordinated, especially since the response must be suited to the nature of the request.

For example the speed of response is more of an issue in the event of an emergency. It follows that there is also an associated cost that is a function of the response time. The system therefore must be able to assess an incident and select the most appropriate response, balancing economic costs against the quality of care delivered. Existing systems are generally limited by the lack of relevant information that is available at any particular time, and there are many instances of care scenarios whereby comprehensive informal systems have evolved to supplement the more formal system operated by the Local

Authority.

For instance, a neighbour may be able to offer assistance based upon their proximity to the care recipient, providing support until a care professional arrives at the scene of the incident. Similarly, help from extended family is often ignored by formal care systems, leading to duplication of resources. This level of cooperation is too advanced for current systems, and demands much more comprehensive exchanges of information between the relevant agencies.

Speed of response is particularly important when the system has to accommodate real-world scenarios such as service delivery failures. In such cases, the system needs to be able to recognise faults and offer an alternative course of action. Human agents would generally negotiate a new commitment, or find an alternative supplier. MAS architectures permit individual agents to act in a similar fashion, enabling not only the better provision of services, but also the generation of a history of the reliability of various services, assisting decision-making and subsequent negotiations in the future.

Modelling Systems

Bauer et al. (2001) describe Agent Oriented UML (AUML) as a notation for the description of agents and their environment. All of the models can be constructed, viewed, developed and evaluated during systems analysis and design. It is based on the meta-model that is the Unified Modeling Language (OMG, 2005), which is a notation for expressing object-oriented analysis, and presents a consistent representation for specifying, visualising, constructing and documenting the artefacts of software systems. Agent modelling requires a greater richness of description, especially since the complex interactions often need to be represented graphically to assist comprehension.

Prior work by (Beer et al., 2001) with the Intelligent Community Alarm (INCA) Demonstrator, illustrated that the design process created a requirement to formally represent various aspects of the agent-managed community healthcare system. AUML facilitated a large proportion of this work, enabling agent models and the resultant stub-code, to be generated in readiness for deployment with the ZEUS Agent-Building Toolkit (Nwana et al., 1999).

Whilst it was possible to produce models of the agents that embodied the required behaviours, and consider the nuances of the community healthcare domain concurrently, it became apparent that some real-world issues were much more difficult to capture. In particular the representation of relatively simple payment transactions proved elusive, as AUML lacked the ability to capture high-level qualitative scenarios.

4.9.2 Designing Community Care Systems

A specification for INCA was initially determined by consulting the ZEUS role-modelling guide (Nwana et al., 1999), and using this approach to derive the roles of agents, services offered and task descriptions to be described using AUML. These models were then used as an input specification for implementation activities with the ZEUS Agent-Building Toolkit. The abstract input specification was described by a collection of use case, class, interaction and deployment diagrams, which provided a consistent representation of the community healthcare complexities across a number of disparate domains (Huang et al., 2003).

Whilst this process was remarkably simple in some areas, as the agent architecture mapped directly onto significant portions of the problem domain, a number of areas were identified that proved more problematic. The tasks of selecting the most appropriate care service and brokering service requests were particularly difficult without compromising the accuracy of the model.

It is fundamentally important that agent representations are not unduly compromised if they are to gain acceptance as a resilient and life-like solution for complex management problems, and it was deemed appropriate to investigate the aspects of the community healthcare scenario that did not translate as effectively to an agent architecture.

The payment transactions required for community healthcare do not immediately appear complicated as they are conducted (albeit often quite inefficiently) by human agents, who are familiar with the concept that the agency who requests a service does not always pay for that service, or pays a proportion of the total amount, depending on a variety of circumstances (Beer et al., 2001).

It is also noted that in effect, community healthcare management systems are similar to commercial enterprise systems that manage the delivery of services by controlling and recording transactions. Human agents have of course become accustomed to interact with transactions in a commercial environment, and they often question computer-delegated transactions, particularly with regard to their robustness.

The allure of reduced resource requirements, improved service delivery, and quality assurance offered by multi-agent architectures, combined with the complexities of community healthcare systems means that we need our agents to assume control of the fundamental transaction workload. The inclusion of human agents implies that issues of 'trust' with regard to agent-managed services will arise. It is therefore fundamental that the transactions should be represented in a robust way, and paramount that any solution should include a robust transaction model as its foundation from the outset.

Using the TrAM approach, agent representations of the community care system model have been developed that address the issues of community care payment complexity and agent-managed transactions. In particular, Local Authority agents who tender the services of community care provider agents, is an example agent trading scenario that has a fundamental requirement for a model that is robust and life-like. Initially AUML representations of auction protocols (Huang et al., 2003) were included within INCA, but the combination of quantitative and qualitative aspects of transaction management, together with the 'gap' between abstract life-like representations and low-level deployment practicalities directed the research towards an alternative method of representation. This resulted in the TrAM Framework.

TrAM addresses the difficulties attributed to the production of agent-based models in the following ways:

1. The transactions approach makes model verification implicit as any missing nodes (concepts or relations) render the model out of balance and unable to satisfy both sides of the transaction.

2. The richness of CGs permits qualitative issues to be challenged and documented, before refining further by drilling-down for more detail. Qualitative reasoning is an important agent capability and the use of conceptual graphs addresses this at the earliest opportunity within the design lifecycle.

3. Roles are identified using the transaction model via the 'inside' and 'outside' agents.

4. Ontological terms are derived from the transaction model during the

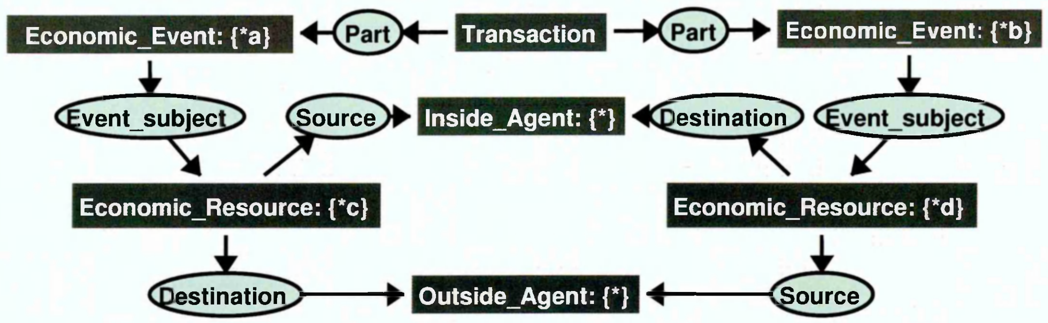


Figure 4.4: The Transaction Model (TM).

process of capturing requirements.

5. CGs are similar to AUML in that there are some obvious mappings from concepts to agents. Prior experiences with AUML illustrated that actors mapped to agents.

A combination of the requirement for a transactions-based model, and a need to represent a community care domain that is inherently complex, has led to the demand for an MAS design framework that embodies the notion of robustness. This represents the real-world scenario more faithfully, negating the need to compromise the implementation unduly.

4.9.3 Building the Model with TrAM

Having considered the various methods of representing the complexity of the community care environment, this section illustrates by way of an exemplar how Transaction Agent Modelling (TrAM) can be utilised to gather system requirements and produce a model.

Capturing Care Scenarios and Early Requirements

Beer et al. (2002) describes five scenarios that a community healthcare system would be required to manage. The following section illustrates how a MAS approach can be used to accommodate such scenarios within an integrated community care system. These scenarios are summarised as follows:

1. The creation and maintenance of an Individual Care Plan (ICP) for each care recipient, which details the package of care services that are required to address the specific needs of an individual.

2. The provision of positive care to maintain and improve the quality of life of a care recipient (Sixsmith et al., 1993).

3. Using the ICP as a reference, the delivery of regular routine care in order to support daily living.

4. The provision of emergency care in response to some unexpected event, such as an accident or medical emergency.

5. Quality assurance management, by monitoring the delivery of care, managing exceptions and interventions to the ICP when required.

The first step of the approach is to identify the actors in the system and describe the roles that they undertake. Once the actors have been discovered, system interactions can then be described with the aid of written use cases and use case models.

Maintaining the Individual Care Plan

The Individual Care Plan (ICP) is created by taking information from one or more assessments of the potential care recipient. This activity is managed

Thesis no.
24943

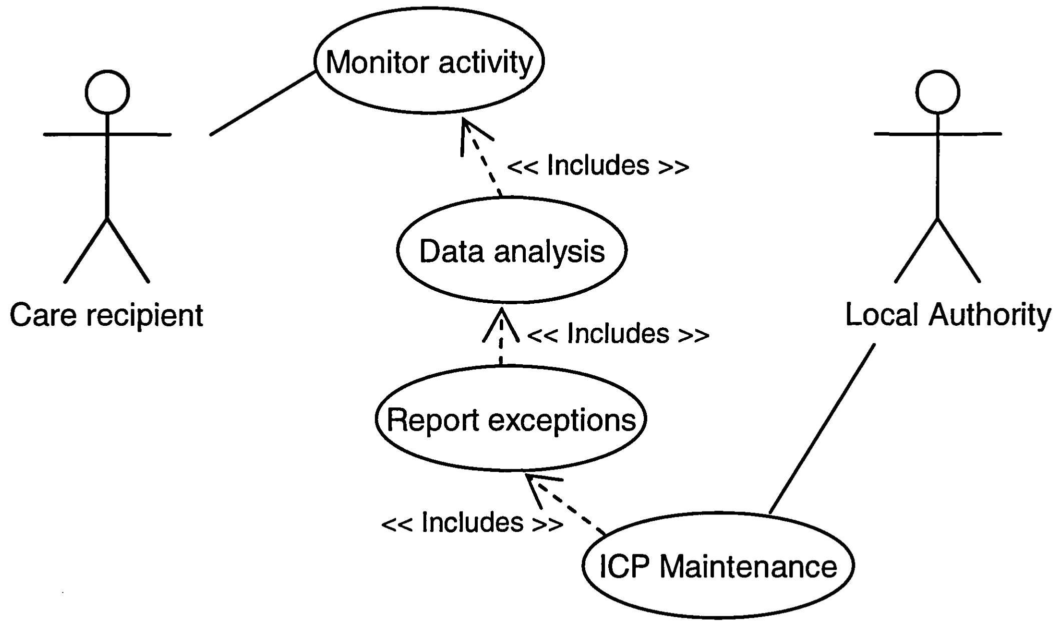


Figure 4.5: Use case model for maintaining the ICP.

by the Local Authority and typically employs the services of an Occupational Therapist (OT) for an initial assessment. Once the need has been assessed, the ICP is created to specify the package of care services that are required to meet the needs of the care recipient.

In-home assessments enable all aspects of the home environment to be taken into account, though they do require a significant amount of resource to execute. Since a community care system like INCA can monitor the activities of each individual, there is a wealth of information available for analysis. Figure 4.5 illustrates the use case model representing this scenario.

Improving Quality of Life

The argument for improving quality of life is compelling and it is often the case that when the delivery of care breaks down for some reason, the reaction is to over-allocate resource to the scenario until the situation returns to normal

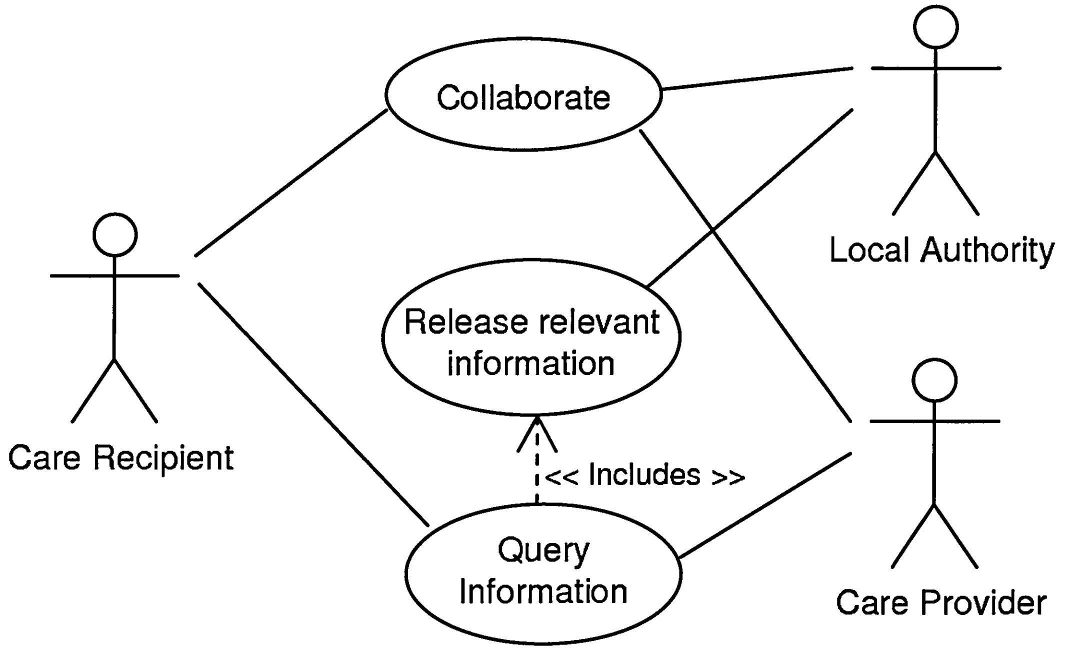


Figure 4.6: Use case model for positive care.

operating conditions. Successful delivery of the ICP not only includes the effective allocation of resources, but also the inclusion of care services that at least maintain and preferably improve the care recipient's quality of life. Such actions are referred to as 'positive care'. Positive care aims to improve the psychological and social well-being of the care recipient, by supporting and promoting:

1. Enhanced social interaction between the care recipient, Local Authority and care providers;

2. The provision of information surrounding leisure activities and opportunities for new experiences. Such information needs to be tailored to the specific needs and preferences of the care recipient.

Figure 4.6 shows the use cases required to facilitate positive care.

Providing Daily Care

The objective of daily care (Figure 4.7) is to provide each care recipient assistance with eating, washing, bodily functions, or any other care need. Maintenance of an accurate ICP is paramount and it is important to monitor the actual delivery of care services and report back any exceptions. Unfortunately, towards the end of a care recipients' life, the rate of deterioration is much greater than the responsiveness of the care management system.

This is less of an issue in a hospital or residential home environment as the care is delivered on demand. In-home care delivery is provided however, in relation to a strict schedule to minimise logistical arrangements. This results in a care service that is inflexible, and that cannot accommodate exceptions unless there are informal carers who are able to provide the assistance required.

Emergency Support

Support for emergency situations presents a challenge for community care systems. Whilst it is feasible that monitoring of the care recipient would enable a more proactive approach to care management, an emergency scenario is unpredictable and therefore the system must provide the most appropriate response in a timely manner. The use of agents to collaborate and coordinate their activities means that the results of all interventions can be monitored, and therefore used to update the dynamic ICP. These interactions are shown in Figure 4.8.

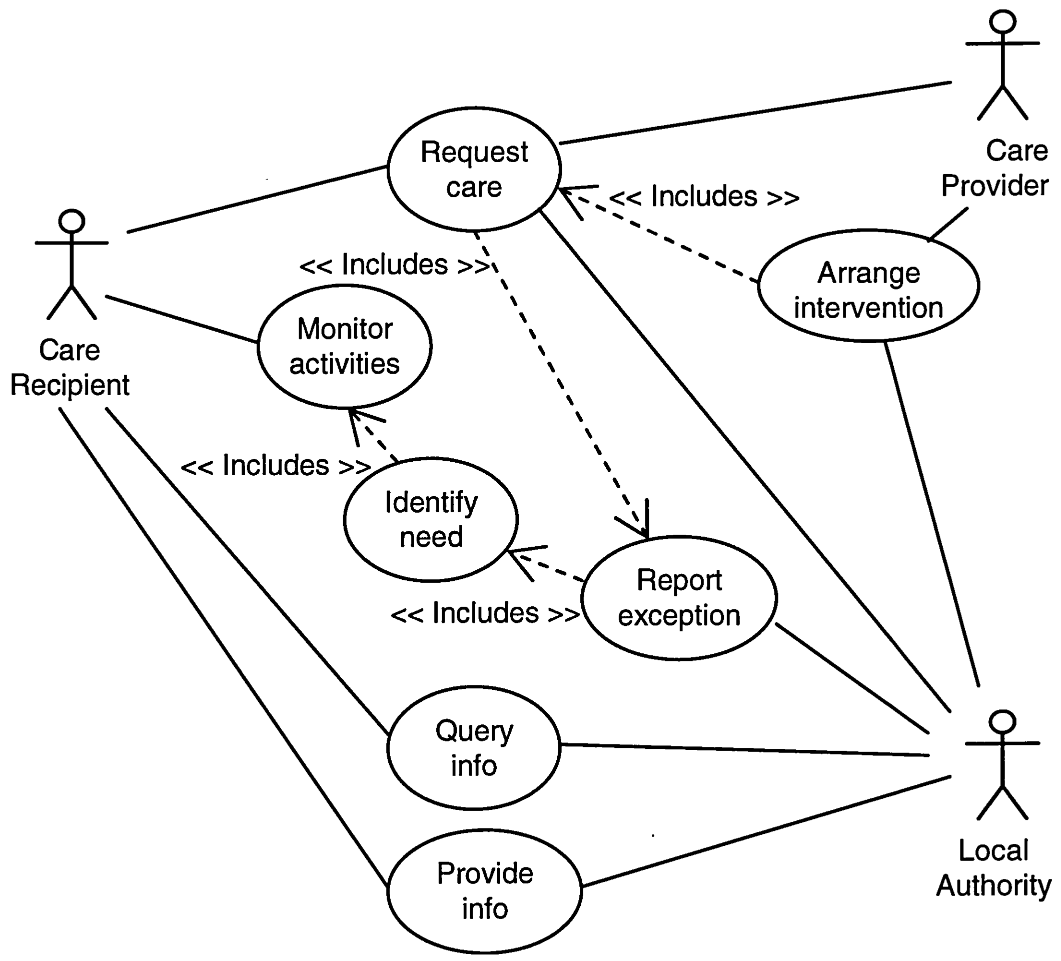


Figure 4.7: Use case model for daily care.

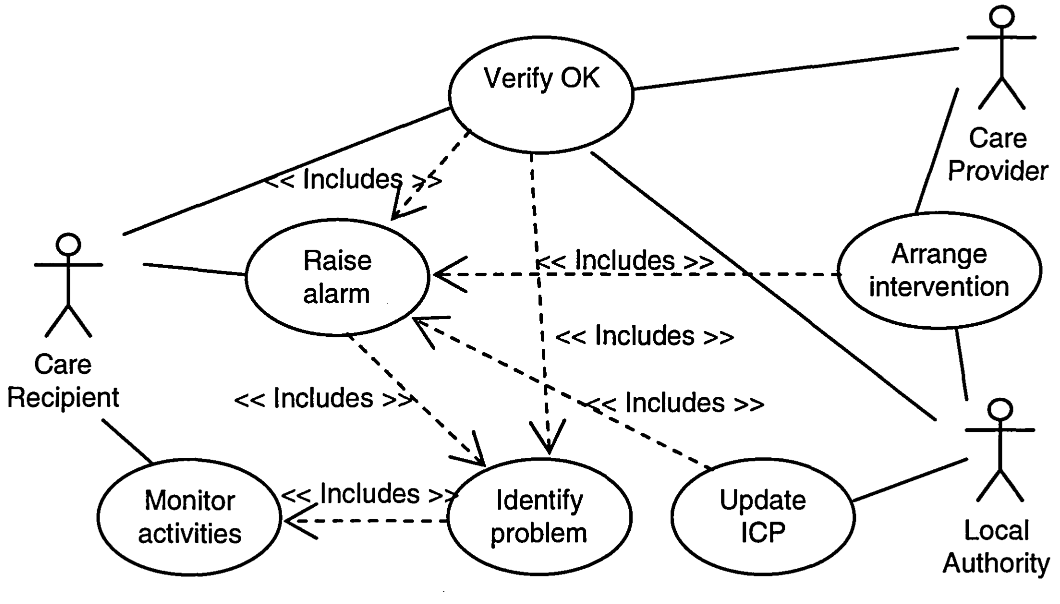


Figure 4.8: Emergency scenario use case model.

Quality Assurance

After creation of the ICP, it is necessary to monitor the requirements of the care recipient in order that the ICP can be updated to reflect any changes. Figure 4.9 shows the interaction involved in Quality Assurance procedures. Figure 4.10 shows the Local Authority as the manager of this role. The concept of a dynamic rather than static ICP is fundamental to community care management, if quality of life is to be improved whilst also minimising duplication of resources.

It is also important to ensure that all the care specified in the ICP is delivered at a satisfactory service level, at the appropriate time, standard and in the correct place. The monitoring of care staff is problematic in the community context, as direct supervision is difficult. The community care system needs to facilitate effective monitoring in two ways:

1. Care providers should log their interventions directly into the system at

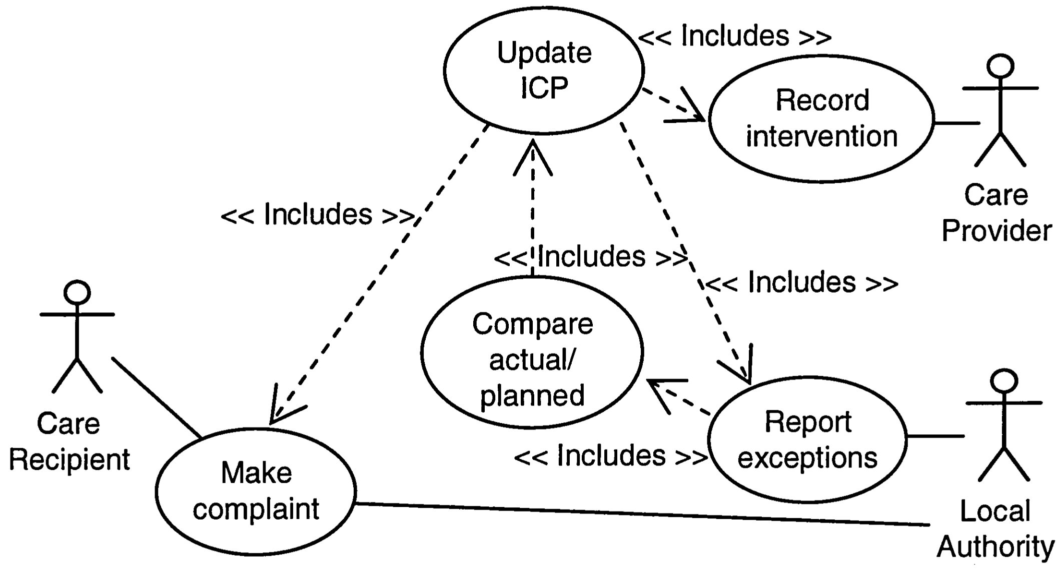


Figure 4.9: Quality assurance use case model.

each visit. These can then be compared directly with the contents of the ICP. Any deviations can then be investigated and either the ICP can be updated or other appropriate action undertaken.

2. Complaints procedures can be based upon direct communication with the Local Authority, improving monitoring and responsiveness.

Now that the early requirements have been gathered, the next stage is to produce an agent model, identify and allocate tasks to each of the agents and then scrutinise the transactional nature of inter-agent communication.

Identify Agents

After the individual use cases have been captured, the next stage is to identify the agents that will be required to complete a model of the eventual system. Prior work (Hill et al., 2004; Beer and Hill, 2006a,b) has shown that actors can be mapped straight to agents. Thus, using Figure 4.10 as an exemplar overview model, the following agents can be quickly derived:

1. Care Recipient Agent (CR Agent)
2. Occupational Therapist Agent (OT Agent)
3. Local Authority Agent (LA Agent)
4. Care Provider Agent (CP Agent)

Whilst the agent characteristics of reactivity, proactivity, autonomy, intelligence and social ability assist the representation of human agent roles, there still exist a number of entities that do not possess such characteristics, such as knowledge bases and databases.

One such example is the use case 'Query ICP' from Figure 4.10, which will need to access a repository to read the contents of a particular ICP. Similarly the use case 'Schedule care' will also require access to a database so that care delivery can be managed. In these cases, each information repository is assumed to map to an 'information agent', who manages the access to each data source. Figure 4.11 illustrates both the actor to agent mappings, plus the information agents who marshal each data source.

One of the key facets of an agent-based community care management system is the ability to harmonise all of the disparate data sources together without resorting to the drastic action of re-writing existing legacy code. Therefore in this example it is suitable to introduce information agents that reduce interference with existing systems.

Allocate Tasks to Agents

Once the agents have been identified, the next step is to identify and allocate tasks to each of the agents. Each task is taken from the use case descriptions

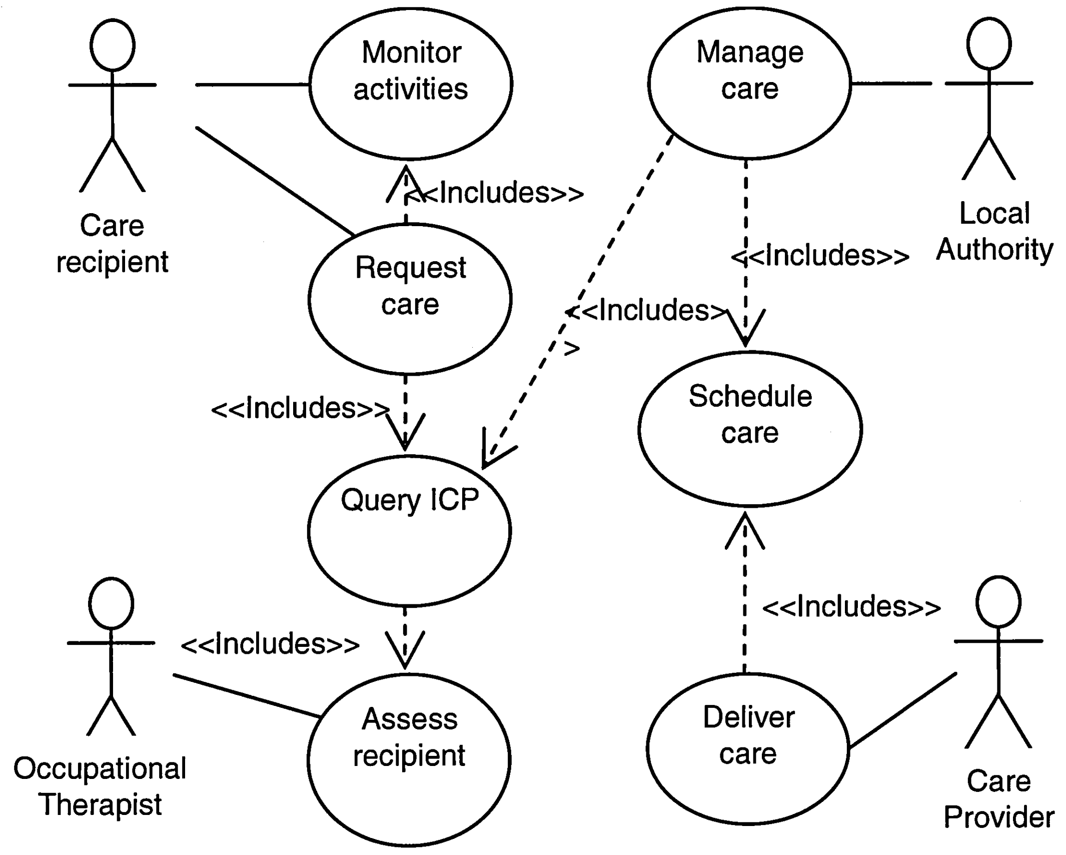


Figure 4.10: Overview of care model.

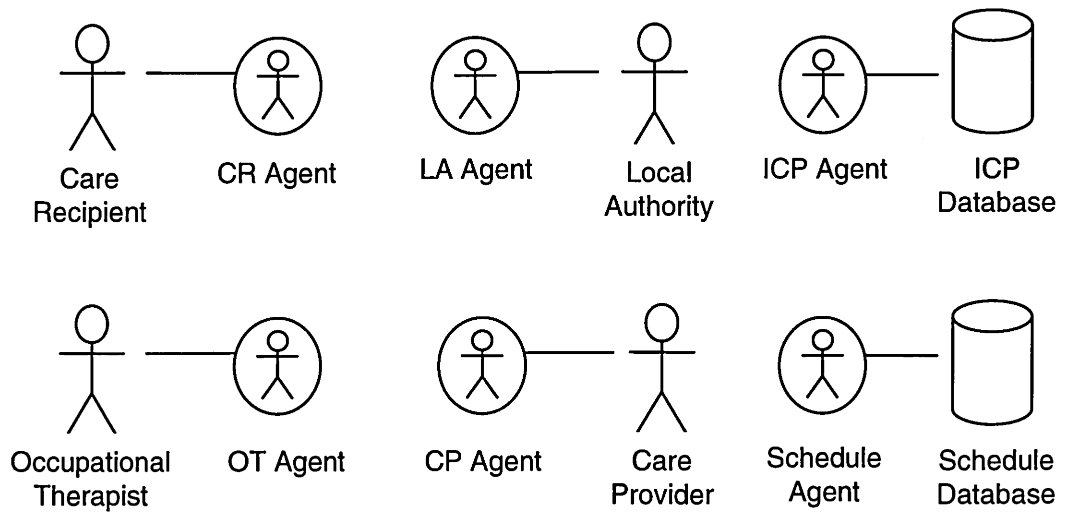


Figure 4.11: Initial actor to agent mappings.

and assigned to the perceived owner of that task, or the agent who is deemed to be responsible for its satisfactory completion. Table 4.1 shows the initial task allocation.

Agent Type	**Task**
Care Recipient Agent (CR Agent)	1. Make request for care. 2. Raise alarm. 3. Interact with home monitoring unit.
Occupational Therapist Agent (OT Agent)	4. Assess Care Recipient.
Care Provider Agent (CP Agent)	5. Deliver care to Care Recipient. 6. Query schedule information.
Local Authority Agent (LA Agent)	7. Query Individual Care Plan (ICP). 8. Schedule care services. 9. Monitor ICP.

Table 4.1: Agent types and allocated tasks

Identify Collaborations

It is now possible to examine the collaborations between the agents. Each allocated task is considered in terms of identifying the agents that will be involved in the collaboration. As each task is mapped to an instance of collaboration between two agent types, each potential conversation is considered and new tasks are derived. Figure 4.12 shows the agent collaboration model, together with the tasks allocated from Table 4.2.

This process is iterative and it is likely that several refinements are required before a comprehensive model is produced. For brevity the results of only one iteration are shown in Table 4.2, each additional task being shown in italics. Once the tasks have been discovered, they are added to the overall agent collaboration model as in Figure 4.13.

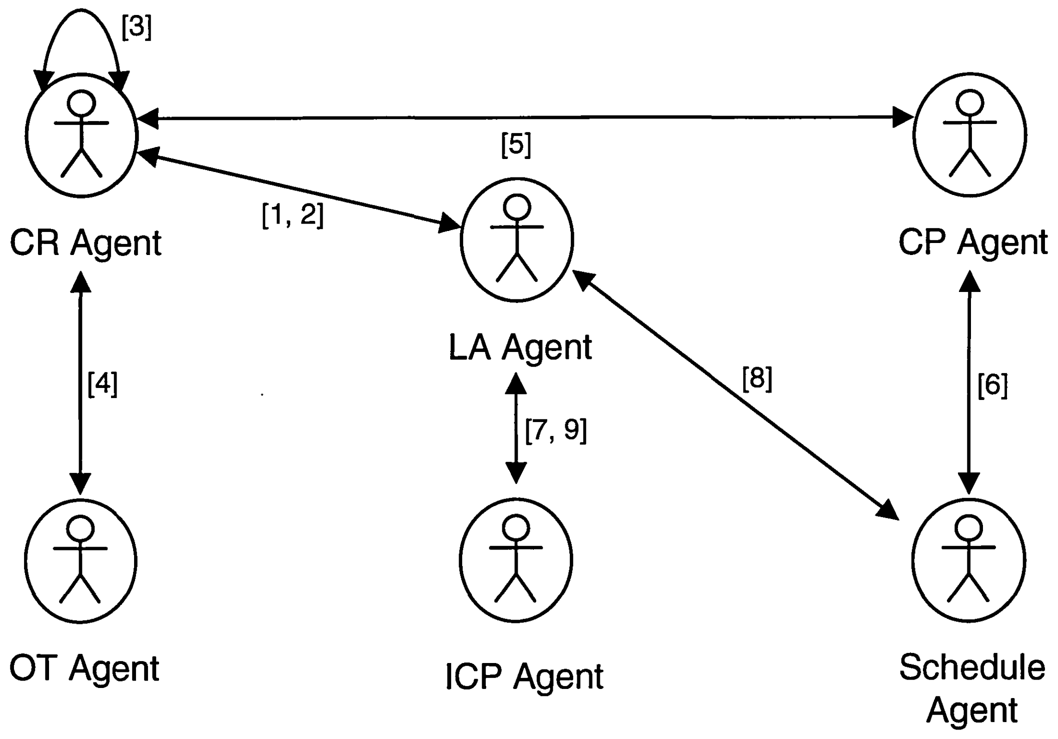


Figure 4.12: Agent collaboration model.

Apply Transaction Model

After identifying the set of overall collaborations from the use cases and subsequent task allocation stage, the model now undergoes further scrutiny in order to ensure robustness. Using the event accounting model described in the previous chapter and the Transaction Model (TM), the community care scenario is scrutinised in terms of specific transactions. In such a complex environment it is clear that many transactions exist.

For the purposes of this explanation, only one transaction will be demonstrated. As discussed earlier, prior work with INCA demonstrated that existing representations such as AUML could not successfully express the complexities of community care payment management, particularly with regard to qualitative transactions. To demonstrate the power of a transactional approach

Agent Type	**Task**
Care Recipient Agent (CR Agent)	1. Make request for care. 2. Raise alarm. 3. Interact with home monitoring unit.
Occupational Therapist Agent (OT Agent)	4. Assess Care Recipient. *10. Update ICP.*
Care Provider Agent (CP Agent)	5. Deliver care to Care Recipient. 6. Query schedule information.
Local Authority Agent (LA Agent)	7. Query Individual Care Plan (ICP). 8. Schedule care services. 9. Monitor ICP. *11. Select care provider.*

Table 4.2: Iterated agent types and allocated tasks.

to modelling an MAS, the following exemplar will describe how the payment modelling was finally resolved.

Model Concepts

Initially, the whole care scenario is represented as a Conceptual Graph (CG)(Figure 4.14). This notation is utilised as it permits the lucid representation of qualitative as well as quantitative concepts.

As described earlier, the Transaction Model (TM) provides a useful means of introducing model-checking to the requirements gathering process (Hill et al., 2006a,b). The specialisation of the generic TM of Figure 4.4 and Figure 4.15 onto the community healthcare scenario (Figure 4.14) is illustrated by the CG in Figure 4.16. This specialisation serves two fundamental objectives:

1. The concepts identified within the care scenario are 'balanced' and therefore represent a transaction;

2. Since each concept is classified in terms of type, a hierarchy of types for an ontology can be derived.

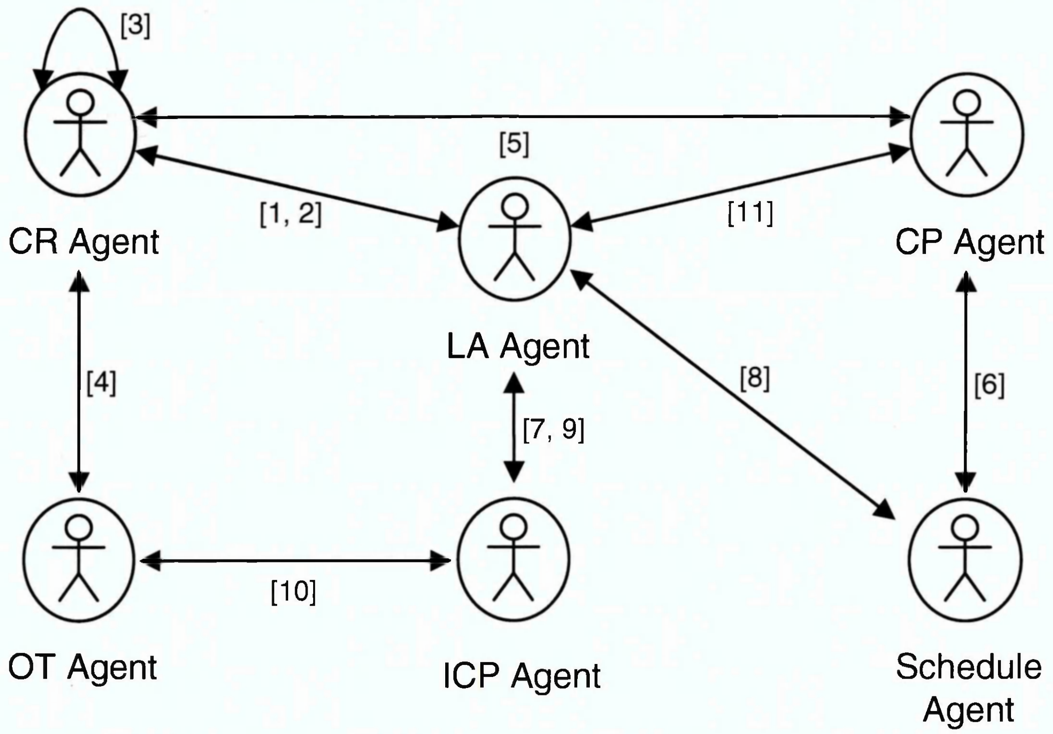


Figure 4.13: Iterated agent collaboration model.

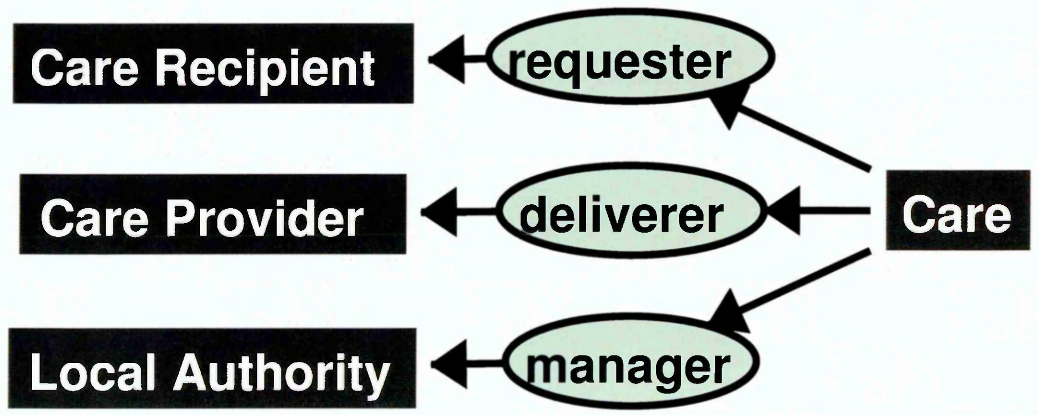


Figure 4.14: CG Model of Community Care Scenario.

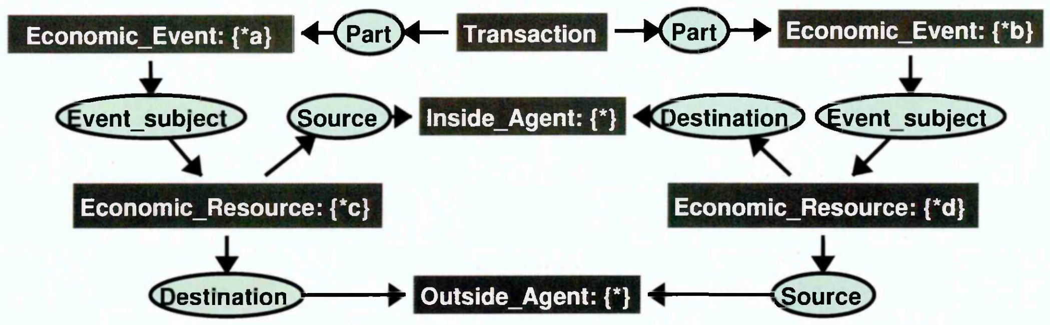


Figure 4.15: CG Model of Generic TM.

The overall model (Figure 4.16) does not explain which party pays the bill for the care, or who is the 'source' of the money. The UK Welfare System has three particular scenarios:

1. The Local Authority pays for the care in full.
2. The Care Recipient pays for the care in full.
3. The Local Authority and the Care Recipient make 'part payments' that amount to 100% of the care cost.

'Purchase Agent' is derived as the supertype of 'Local Authority' and 'Care Recipient' in order to satisfy the TM.

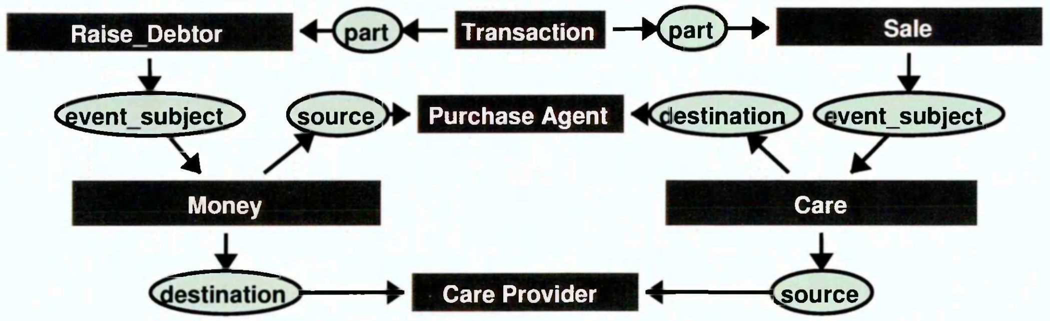


Figure 4.16: Overall Transaction Model of care scenario.

The most significant contribution of this stage is the implicit 'balance check' that immediately raises the analysts' awareness of the need for appropriate terminology. Figure 4.17 illustrates the hierarchy of types deduced from Figure 4.16.

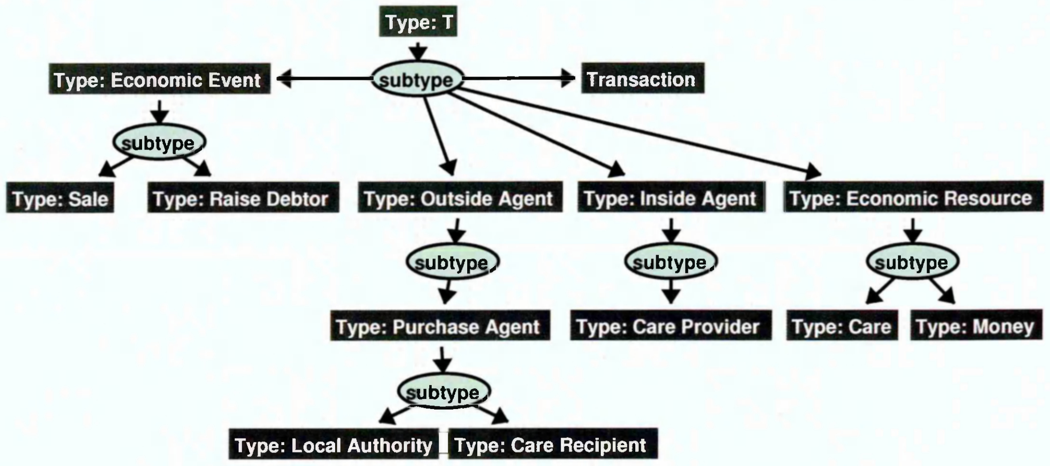


Figure 4.17: Initial type hierarchy of care scenario.

Once the generic model has been created, it is tested with some general rules. First, the specific scenario (Figure 4.18), whereby a Care Recipient has been assessed and is deemed to be eligible to receive care at zero cost is explored.

Figure 4.19 shows that the 'source' of the money to pay for the care is the Local Authority 'Sheffield City Council (SCC)', who also manages the provision of the care.

However, the care package is not delivered by the Local Authority, who buys services from designated Care Providers. For this example, the Local Authority is managing a 'Meals on Wheels' service. The party which incurs the cost of the care package is represented by the 'destination' concept.

Alternatively the Care Recipient may be deemed to have sufficient assets, and is therefore ineligible for free care (Figure 4.20).

Figure 4.19 illustrates eligibility for free care, where it can also be seen that the care package is still managed by the Local Authority. In both cases, whether the care recipient has sufficient funds to pay for the care (Figure 4.19) or not (Figure 4.22), the original relationships of Figure 4.14 are included. This ensures that the relevant aspects of the transaction are retained and can be recognised in subsequent development.

Inference Model with Queries and Validate

From the prior figures the general CG pattern in Figure 4.21 emerges. To evaluate this scenario we query the model. Firstly, we examine the case where the Care Recipient's ('Betty') 'Assets' are deemed to be less than a particular threshold set by the Local Authority. In such a case, the Local Authority (Sheffield City Council) would be the destination of the care, and would therefore be liable for the bill. Figure 4.18 shows this particular query graph, which states:

```
If requester of Care is Care Recipient whose
   characteristic is assets < threshold Then
       Local Authority is destination of Care
```

Updating the TM with this gives Figure 4.19.

Alternatively, the Care Recipient may be deemed to have sufficient assets to be able to afford the care package. Figure 4.20 illustrates the relevant query graph, showing the 'less-than-threshold' asset test being set in a negative context (false):

```
If requester of Care is Care Recipient whose
   characteristic is assets > threshold Then
       Care Recipient is destination of Care
```

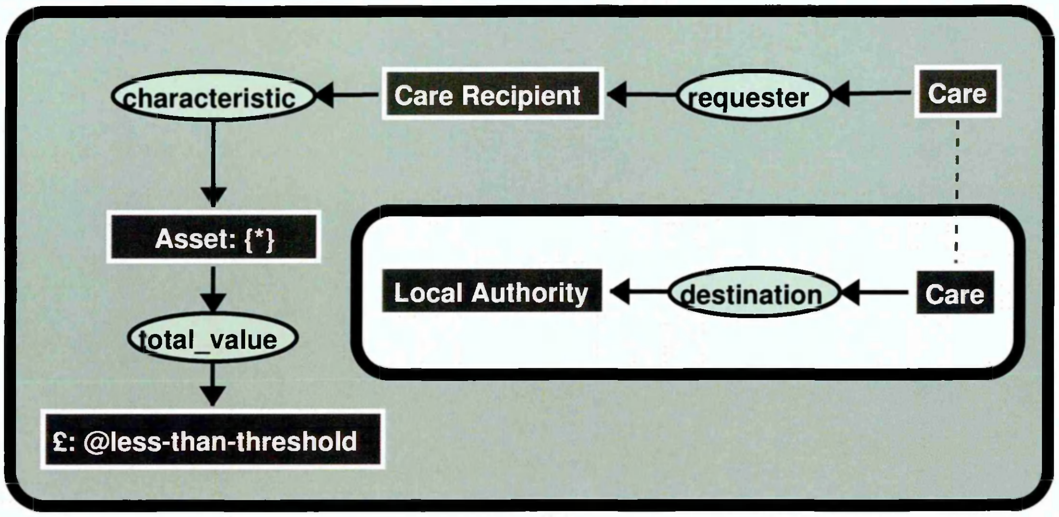


Figure 4.18: Local Authority pays for care in full.

Again the TM is specialised and is illustrated in Figure 4.22, showing that the Care Recipient is indeed the destination of the care, and therefore is liable for the full cost.

So far the opposing scenarios whereby either the Local Authority or the Care Recipient settles the bill for the care in full have been explored; for completeness the part-payment scenario, whereby each party makes a contribution towards the total cost, must also be examined. As before, the generic model of concepts is produced, before specialising with an individual scenario.

The part-payment model in Figure 4.21 comprises Local Authority and Care Recipient, plus the Purchase Agent derived earlier in Figure 4.17. After specialisation of the TM (Figure 4.23) the OR relationship between `Local Authority: SCC` and `Care Recipient: Betty` does not allow joint parties to be the Purchase Agent.

First we consider the scenario whereby the Local Authority and Care Recipient have a split liability for the care costs. The liability is apportioned in relation to the amount of assets that a Care Recipient is judged to have.

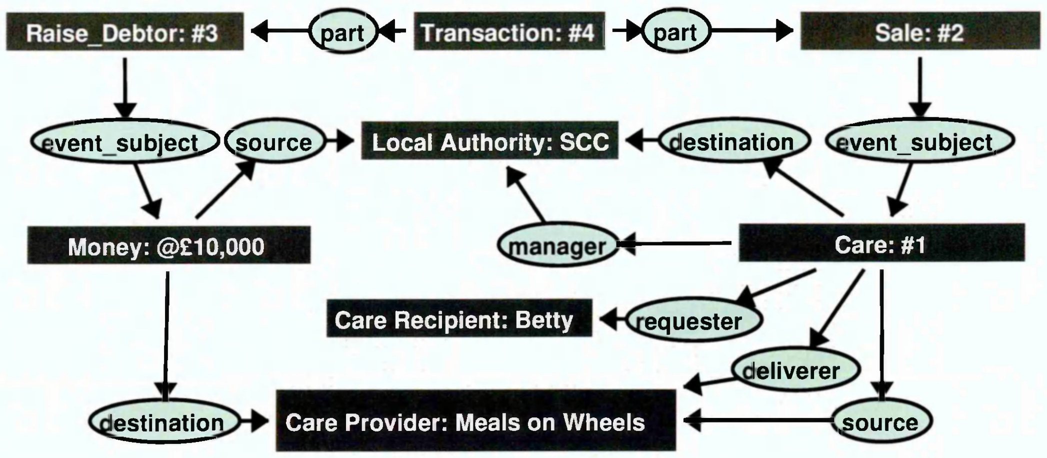


Figure 4.19: TM showing care recipient receiving care package at zero cost.

Figure 4.24 illustrates that the Care Recipient and Local Authority agents are no longer sub-types of the Purchase Agent as originally illustrated, but are instead associated via 'liability' relations.

In order to correct the original type hierarchy (Figure 4.17), the case whereby Care Recipient and Local Authority agents are sub-types of Purchase agent is false. Accordingly a rule is created, which informs the eventual ontology that such a type relation is also false. Figure 4.25 demonstrates this rule, which is negated by setting in a negative context. Having elicited this information, the type hierarchy is modified to reflect the new insight and is illustrated in Figure 4.26. Subsequently the TM is also updated with the liability relationship (Figure 4.27) in order that the model can now accommodate all three payment scenarios.

4.9.4 Limitations of the Approach

Whilst some significant advantages have been demonstrated by the TrAM approach so far, it would be prudent to consider some of the limitations that the

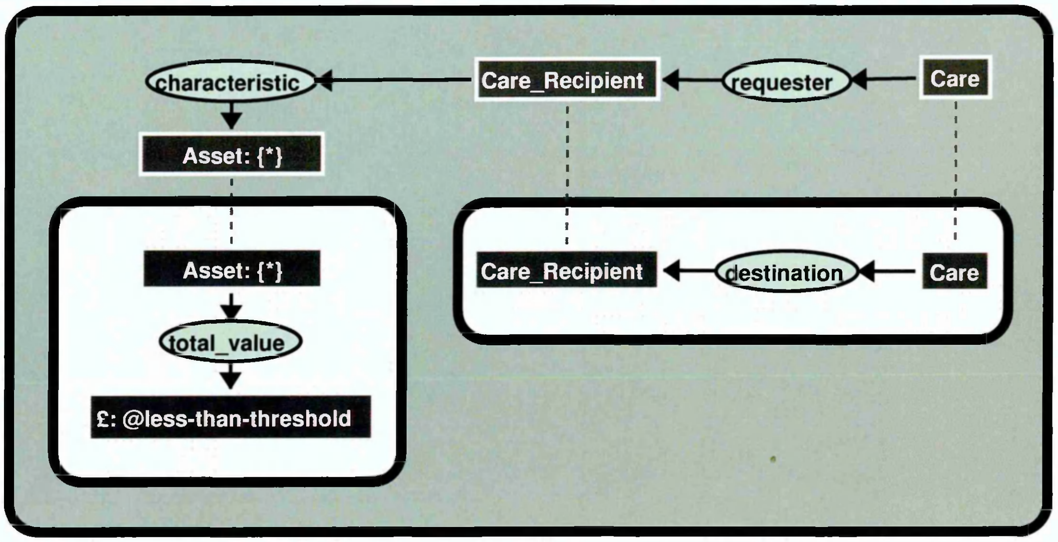


Figure 4.20: Care recipient pays for care in full.

exemplar also illustrates.

The process of using TrAM produces a set of design artefacts, including type hierarchies and constraint rules modelled with Peircian Logic. Unless the ontology is specified using a common standard (such as OWL, W3.org (2004b)) then the output requires further translation.

Also, the use of CGs as a modelling notation assumes that the resultant models will be used to communicate knowledge between those who understand CGs. Bearing in mind the issues discussed in Chapter 3, this is likely to restrict the process to a smaller audience.

Additionally, whilst the approach enables high-level concepts to be captured and analysed, the declaration of goals is not mandatory and can be ignored. This relies on the agent system designer's self-discipline, and may result in applications that cannot accommodate agent concepts such as goals, plans, beliefs and reasoning.

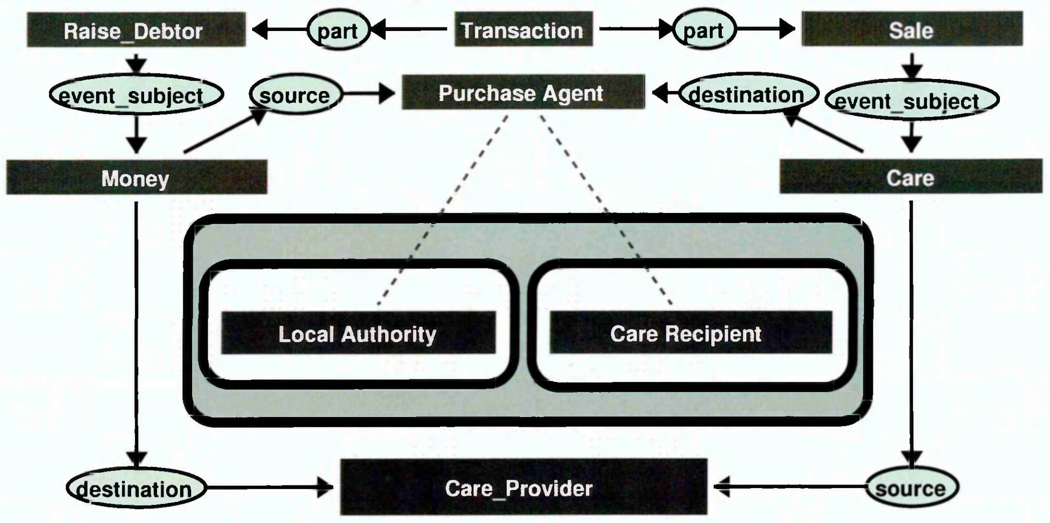


Figure 4.21: Emergent CG model.

4.10 Conclusions

This chapter has proposed Transaction Agent Modelling (TrAM) as a means of eliciting early requirements for agent based systems and demonstrated the use of TrAM in the community healthcare domain. The exemplar has illustrated some of the potential of this approach, particularly with regard to the robust elicitation and analysis of early requirements. The case study has also indicated some limitations, which will be examined in the next chapter.

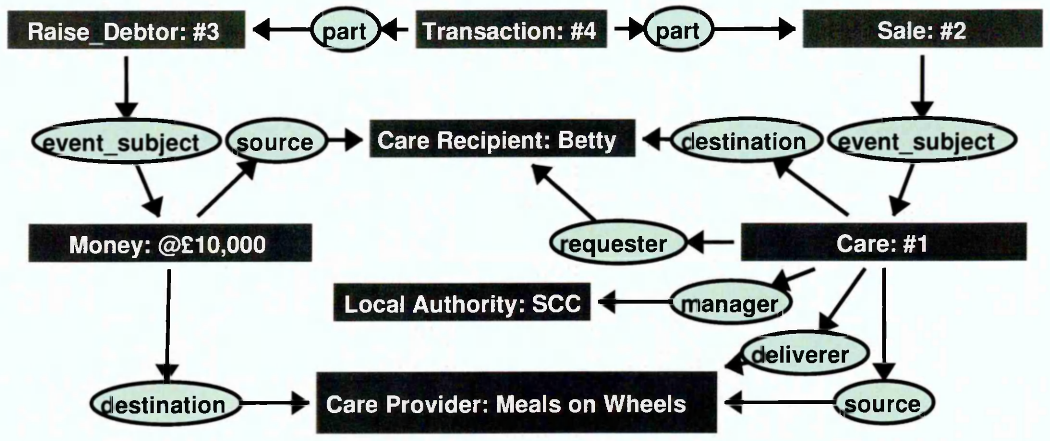


Figure 4.22: Updated TM showing care recipient receiving care package at full cost.

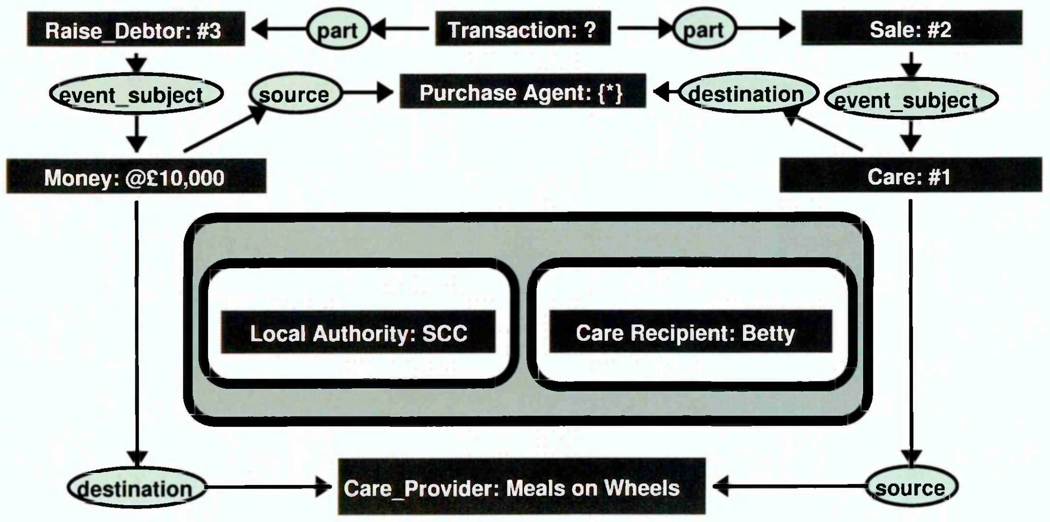


Figure 4.23: Incomplete TM.

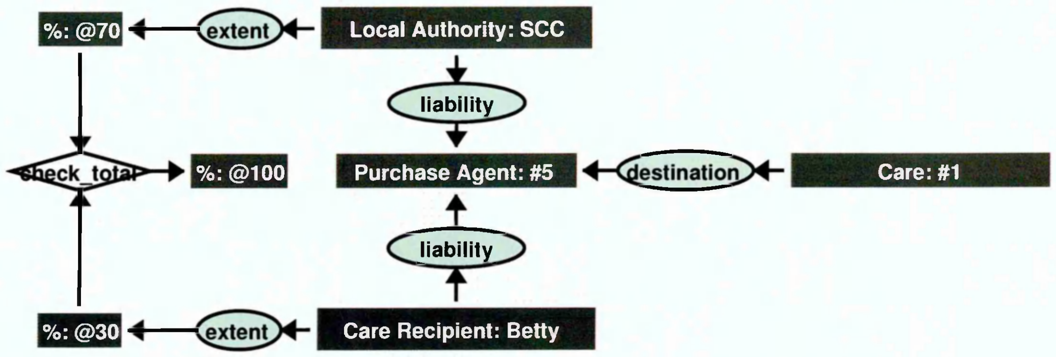


Figure 4.24: Part payment scenario with shared liabilities for care cost.

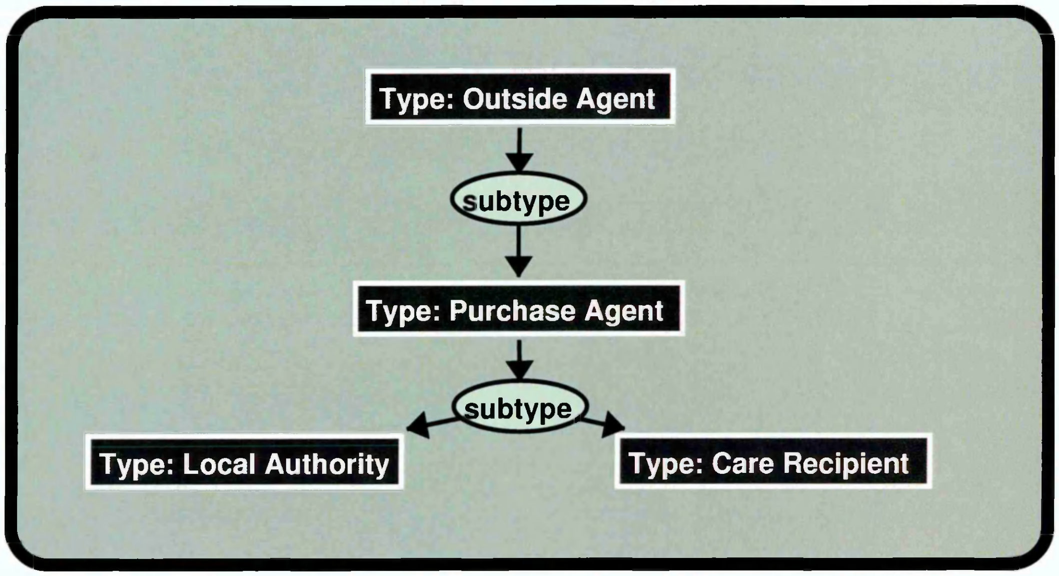


Figure 4.25: New rule for ontology.

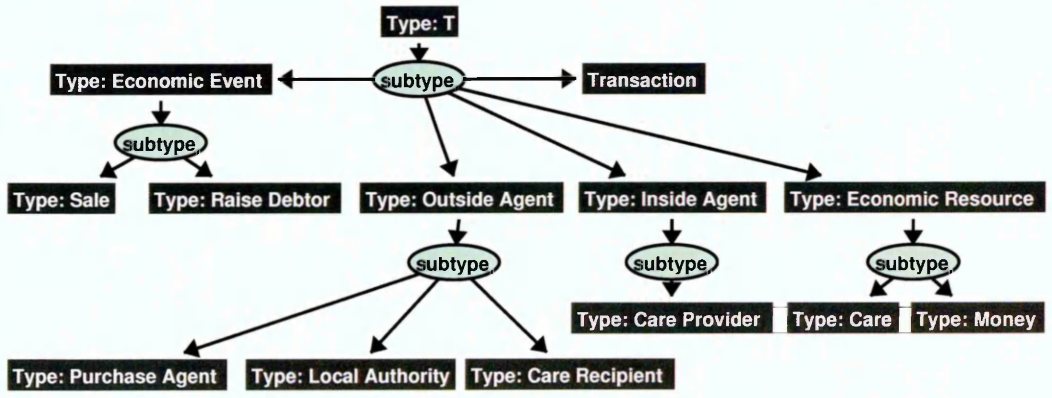


Figure 4.26: Revised type hierarchy.

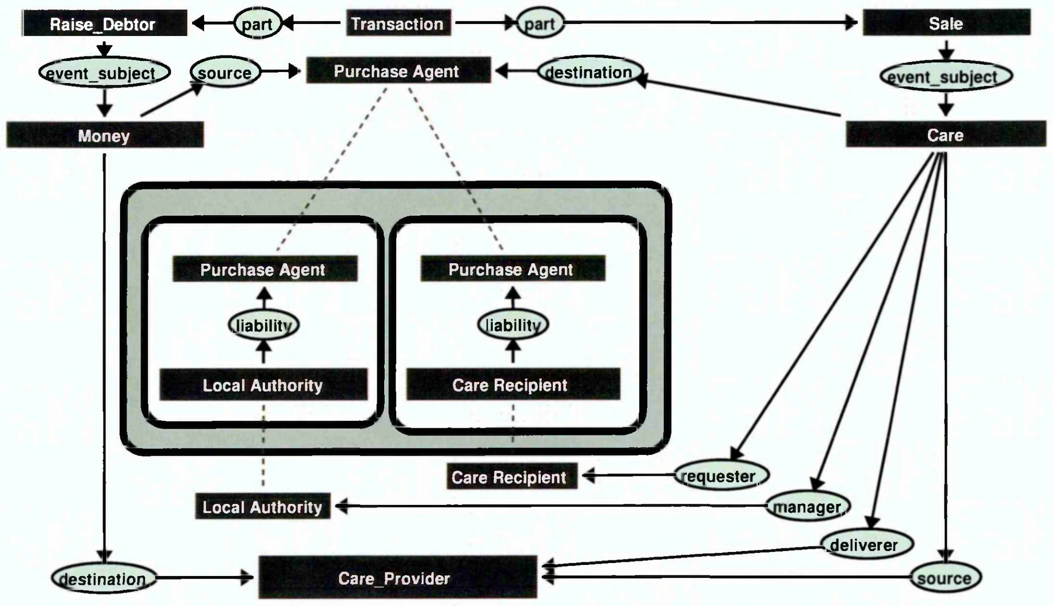


Figure 4.27: Refined payment model.

Chapter 5

Refining the Framework

5.1 Introduction

The framework introduced in Chapter 4 is now discussed in relation to the experience of applying it to the community healthcare case study, and the process for applying TrAM is described. Some limitations of the approach are examined and improvements to the framework are proposed. The inclusion of ontologies to support the framework, that recognise the explicit recognition of BDI concepts is presented, concluding with a summary of the refined framework.

5.2 The TrAM Process

A graphical representation of the framework is shown in Figure 5.1. The process steps are as follows:

- **TrAM Requirements Phase**
  - *Step 1* - Model the system with CGs. Since this is a requirements capture exercise, all concepts, relations, stakeholders and goals need

to be gathered and modelled 'as-is' by freely generating graphs of concepts and relationships. For instance some sample graphs might be:

```
[Care]->(Manager)->[Local Authority].
```

```
Person, Vehicle < Entity
Carer, Care Manager, Care Recipient < Person
Van, Car < Vehicle
Meals-on-wheels < Van
Paramedic, General Practitioner, District Nurse < Car
```

Not only are the system stakeholders being identified, the type hierarchy is being built by the process of the concepts being specified. This initial stage should be performed with the domain expert, to ensure that important key concepts and the accurate vocabulary is captured. The graphs should now be reconciled by examining for joins and common specialisations (Chapter 3). This assists the identification and specification of quantitative and qualitative goals in the following step. The intention of this stage is to produce a graph of the whole scenario, and it may be necessary to abstract some of the detail by using Lambda Expressions 3.

– *Step 2* - Using the high-level graph artefact from Step 1, transform the graph with the TM. It is now possible to identify the system goals. These should be expressed as individual graphs. For instance, the initial capture process may produce goals such as *enjoy social contact* and *provide healthcare service*. The equivalent in TrAM would be:

```
[Social contact]->(Exp)->[State: Enjoy].

[Provide]->(Obj)->[Service]->(Chrc)->[Healthcare].
```

This process enables the type hierarchy for each transaction to be populated and identify any missing concepts or relations. Should any concept nodes be missing, the relationships surrounding the missing concepts are scrutinised and reasoned against in order to determine a concept or concepts that provides a good fit. Equally it may be required to consider the fit of the new concept within the type hierarchy, amending the TM to suit if the ontology appears to be inaccurate. This process is repeated until the missing nodes are populated, and the goals are not seen to be violated. The graphs can now be parsed into controlled English and used to help query the representation of the system with a domain expert. This assists the clarification of concept terms and relationships. As a result of this new knowledge will be generated and this is appended to the TM and type hierarchy graphs. Finally the type hierarchies are considered and examined for any concept types that could be generalised.

- *Step 3* - Use cases for each scenario are gathered in order to provide the means by which the TM model can be tested. Potential queries are determined from considering the information contained within each use case.
- *Step 4* - Each of the queries raised from the use cases are represented as query graphs that depict a particular scenario. Using

graphical Peirce logic inferencing the query graphs are refined until the specific scenario is accurately depicted. Once the query graph has been defined it is appended to the overall TM graph. Steps 2-4 are iterated to refine the specification of requirements. The process is not concluded until all of the use case requirements have been accommodated within the overall TM.

- **TrAM Architectural Phase**
  - *Step 5* - Using the stakeholders identified in Step 1, along with the use case scenarios in Step 3, agents are allocated individual roles. Following on from this the tasks and goals are then allocated to agent roles. At this stage it may be prudent to introduce agent roles, particularly if some of the roles have a large number of tasks. In such cases the goals assigned to an agent role should be delegated to other agent roles, thus creating a management task for the managing agent role.
  - *Step 6* - Define agent interactions and specify interaction protocols. Each of the interactions required to support the use cases and the overall TM are defined for each interacting agent. If additional agents have been appended to the model to balance workloads, then it is necessary to identify the message semantics of the additional communicative interactions by referring to Step 5. Furthermore, depending upon the messaging protocol utilised (or demanded by the target domain), it may be necessary to add further communicative acts. Such acts may require tasks adding to the respective agent role.

- **Detailed Implementation Phase**
  - *Step 7* - Use the design artefacts created as an input for an agent implementation approach. The TrAM process has enabled the production of a set of models whereby a rigour has been imposed upon the requirements elicitation process for agent-based systems. These models can now be implemented using the agent construction toolkit of choice, though there is a particular emphasis upon the generation of a design that supports BDI constructs.

Refinements to the process from the experience of modelling exemplars have influenced the process so that far more emphasis is placed upon the modelling of conceptual requirements, exploiting the power not only of CGs, but also the agent design metaphor. Use cases are only dealt with after the process has gathered the high-level, qualitative concepts, in order that the *soft* goals can be elicited. There is also a discrete set of design artefacts specified to document the output of the Framework.

This is further supported by the transaction metaphor, which is in effect passive (the transaction has to have taken place successfully in the past for it to exist), and this supports a goal-directed system as the graphs that have been projected onto the TM are a specification for 'success'. If anything is missing from the model then the transaction cannot take place. Similarly if the transaction is too ambiguous, even though it is valid conceptually, then further specialisation is required. Consequently a transaction is an excellent framework for the specification of agent goals, and the intentions (plans of tasks) can be declared in sufficient detail for the agents.

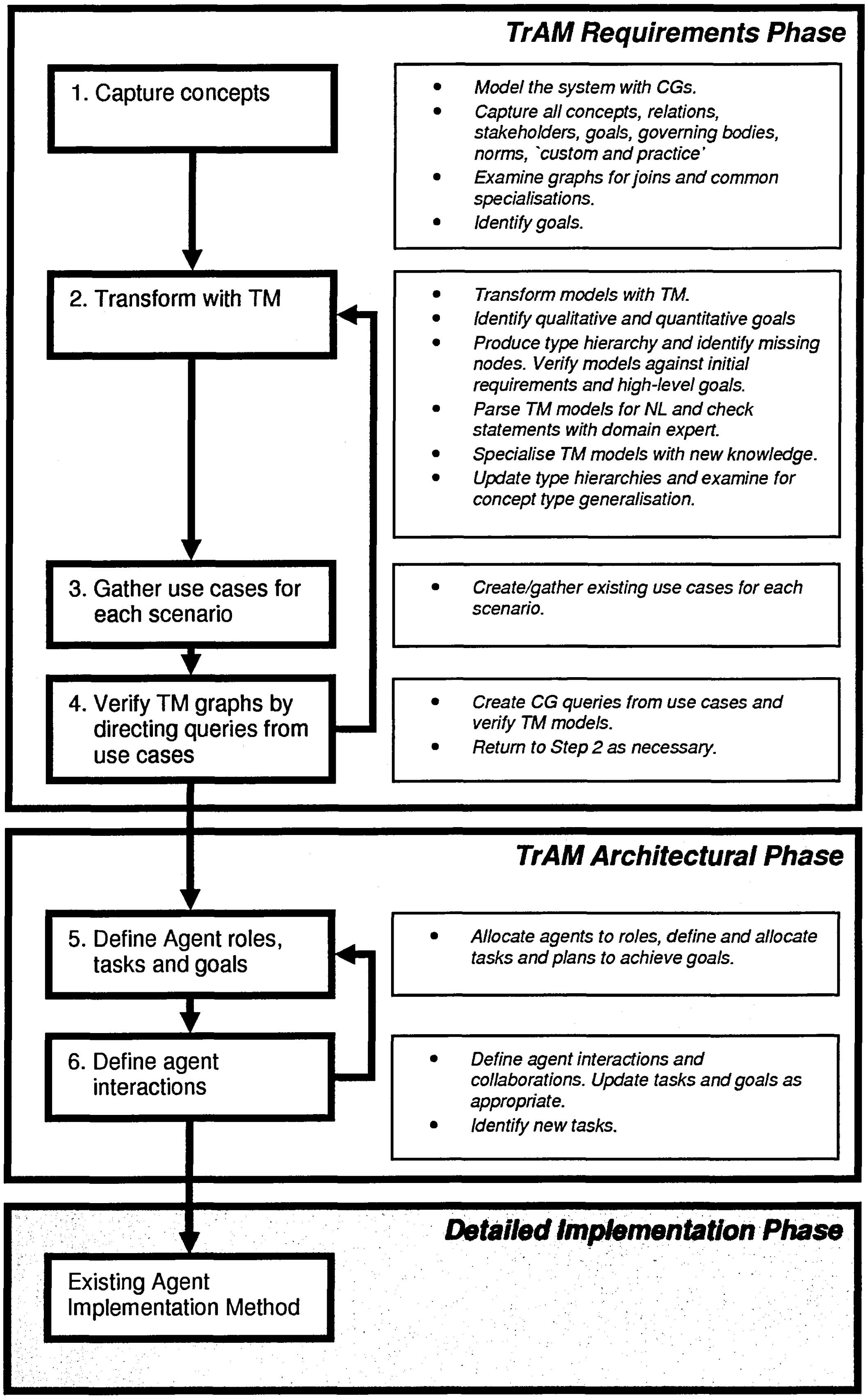

TrAM Requirements Phase
1. Capture concepts
• Model the system with CGs.
• Capture all concepts, relations, stakeholders, goals, governing bodies, norms, `custom and practice'
• Examine graphs for joins and common specialisations.
• Identify goals.
2. Transform with TM
• Transform models with TM.
• Identify qualitative and quantitative goals
• Produce type hierarchy and identify missing nodes. Verify models against initial requirements and high-level goals.
• Parse TM models for NL and check statements with domain expert.
• Specialise TM models with new knowledge.
• Update type hierarchies and examine for concept type generalisation.
3. Gather use cases for each scenario
• Create/gather existing use cases for each scenario.
4. Verify TM graphs by directing queries from use cases
• Create CG queries from use cases and verify TM models.
• Return to Step 2 as necessary.
TrAM Architectural Phase
5. Define Agent roles, tasks and goals
• Allocate agents to roles, define and allocate tasks and plans to achieve goals.
6. Define agent interactions
• Define agent interactions and collaborations. Update tasks and goals as appropriate.
• Identify new tasks.
Detailed Implementation Phase
Existing Agent Implementation Method

Figure 5.1: The TrAM Framework.

5.3 Issues with TrAM

The previous chapter illustrated the development of some agent design artefacts under the guidance of the TrAM Framework. So far, the following important characteristics have been demonstrated:

1. The use of conceptual modelling to capture and represent high-level concepts;
2. The generation of types and relations to support the creation of an ontology;
3. The use of the Transaction Model as a design metaphor.

Of these, items (1) and (2) in particular deserve more consideration. Firstly, whilst the use of CGs permits high-level concepts to be captured, and with the use of Peircian Logic, subsequently analysed, the elicitation of goals is not explicit. Goal specification, or the analysis of hard and soft goals (as per Tropos), is not mandated by the framework; rather it is assumed that the agent system designer will exploit the flexibility and richness of the notation to explore such issues.

It would be more useful if TrAM provided guidance for the elicitation and analysis of goals, in the same way that Prometheus supports this important activity (Chapter 2). Goals are a fundamental concept of agents, and their discovery is crucial to the success of the system design. Since TrAM provides the notation for capturing concepts, it would seem that the framework should also provide the guidance necessary to ensure that the fundamental concepts are accommodated.

The community healthcare case study also demonstrated a reliance upon UML use case modelling, which as a notation itself can be used to model qualitative scenarios. There is a *method* element that is lacking however, unlike Tropos where a clear process is defined for the capture of hard and soft goals. Whilst the CGs were used to verify the use case models during the early requirements capture phase, there is an assumption that the requirements capture process in place is satisfactory - which is what this research is attempting to address and improve upon.

Secondly, the potential power of producing hierarchies of types whilst developing conceptual models, is marred by the fact that the resulting artefacts still require translation into another format, such as RDF or OWL. Again the framework would be more useful if a representation was available that illustrated the mapping from concept to ontology. This would help the agent system designer by providing 'prompts', whilst also addressing the constant need for consistency.

Item (3) has shown how a design metaphor can assist the production of agent models, however the eventual artefact produced can suffer from the inherent generic abstraction; a more specialised graph, that incorporates core agent concepts, could assist the process considerably. With reference to Chapter 2, more explicit links to Belief-Desire-Intention (BDI) concepts (Georgeff et al., 1999) would provide not only extra support when populating the conceptual models, which is fundamental for realistic actor to agent mappings, but it would also make more agent-specific declarations in the ontology. As a result, agent-literate ontologies are more likely to be re-used and designers could take the ontology as a basis for new systems, knowing that the core BDI concepts are included.

Additionally, the initial TrAM Framework does not explicitly make reference to domain norms or policies. Again the flexibility of the notation and the TM graph is such that these concepts can be appended to the models, but there is a reliance upon domain expertise.

In summary there are three key areas for improvement:

1. Goal discovery and analysis;

2. Recognition of agent mental aspects;

3. Explicit inclusion of domain policies.

Since both the explicit declaration of goals and mental aspects are core concepts of the BDI model, the TrAM Framework shall now be developed further to accommodate these features. Similarly, the consideration of policies will also demonstrate not only how TrAM can be refined, but also the ease with which the models can be adapted for specific purposes, without compromising the flexibility of the early requirements capture stage. First of all, a brief recap of the pertinent BDI concepts will be described.

5.4 A Recap of Agent BDI Concepts

For the TrAM Framework to demonstrate 'usefulness' to the agent system designer, it must be able to accommodate agent specific concepts. The Belief-Desire-Intention model of agency (Georgeff et al., 1999) describes three core concepts:

- *Belief* - a *fact* or collection of facts about the world that an agent believes to be true;

- *Desire* - is something that is false, that an agent wishes were true. These manifest themselves as *goals* for an agent, which may or may not conflict depending upon the current circumstances;

- *Intention* - is a means of realising a desire (goal), by way of a *plan*, which may be a list of ordered *tasks*.

BDI refers to the mental aspects of an agent, and serves to simplify the design, specification and subsequent coding of agents. Similarly, for collections of agents in a MAS, who work together to achieve a common purpose, it is useful to be able to consider abstract representations such as *organisation*. Similarly a *society* is a collection of organisations and agents that collaborate to promote their own goals. From such concepts we can begin to consider (and model) the effect of organisational guidelines (*norms*, often expressed as rules) upon a particular society.

Thus if the TrAM Framework could accommodate BDI concepts, they would by nature be made explicit and therefore become a mandatory part of the process. Figure 5.2 illustrates a CG representation of a BDI Agent.

`Belief`, `Desire` and `Intention` concepts have been appended to the Agent concept, which has now been specialised to become `BDIAgent`. The object (`Obj`) of `Intention` is a `Plan`, the content (`Cont`) of which is `Action`. The `Desire` concept has four characteristics (Perich et al., 2004):

1. AchievableDesire - It is likely that an agent will have many desires, but only some of them will be achievable at any given time.
2. NonAchievableDesire - is a desire that cannot be achieved at present.
3. ConflictingDesire - is a desire that conflicts with another desire, norm, action or personal belief.
4. NonConflictingDesire - a desire that has no other conflicts.

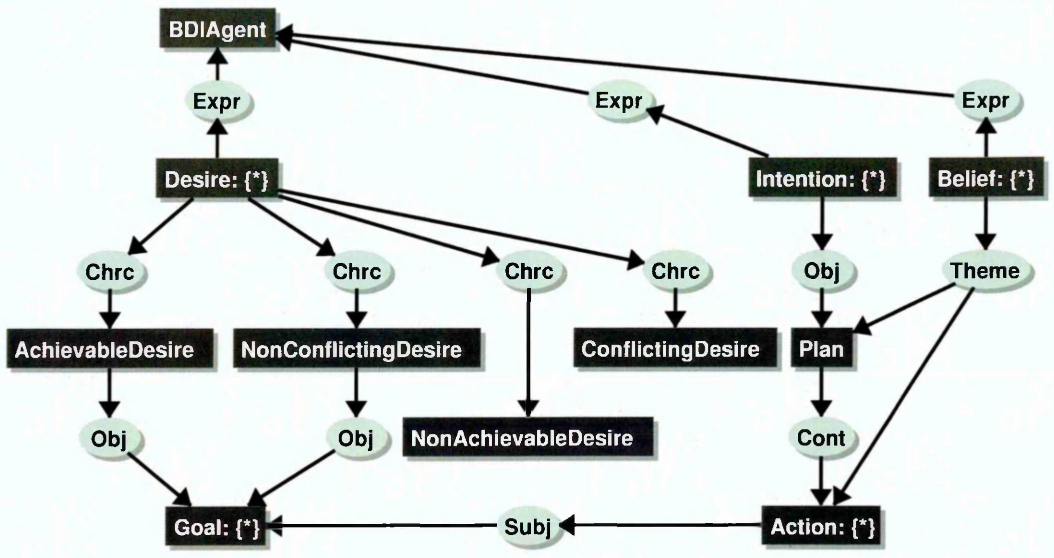


Figure 5.2: CG Representation of a BDI Agent.

As such, `AchievableDesire` and `NonConflictingDesire` can both be sub-classed as a `Goal`. A type hierarchy can be deduced from Figure 5.2 to derive Figure 5.3. Without adding any rules, constraints or cardinality, a simple translation into OWL gives Figure 5.4 and the listing in Appendix A, section A.1.

5.4.1 Norms and Policies

Institutional norms can often appear as qualitative concepts, such as politically-charged mandates, and using CGs they can be modelled as has been described earlier. However, the capture of such concepts does not guarantee their successful translation into an agent design specification, and organisations typically express their norms in the form of policies. Of course, there are norms which 'exist' but are not written down, or formally recognised. The advantage of conceptual modelling for early requirements is that there is no discrimination between formal and informal norms; they can both be specified as policies.

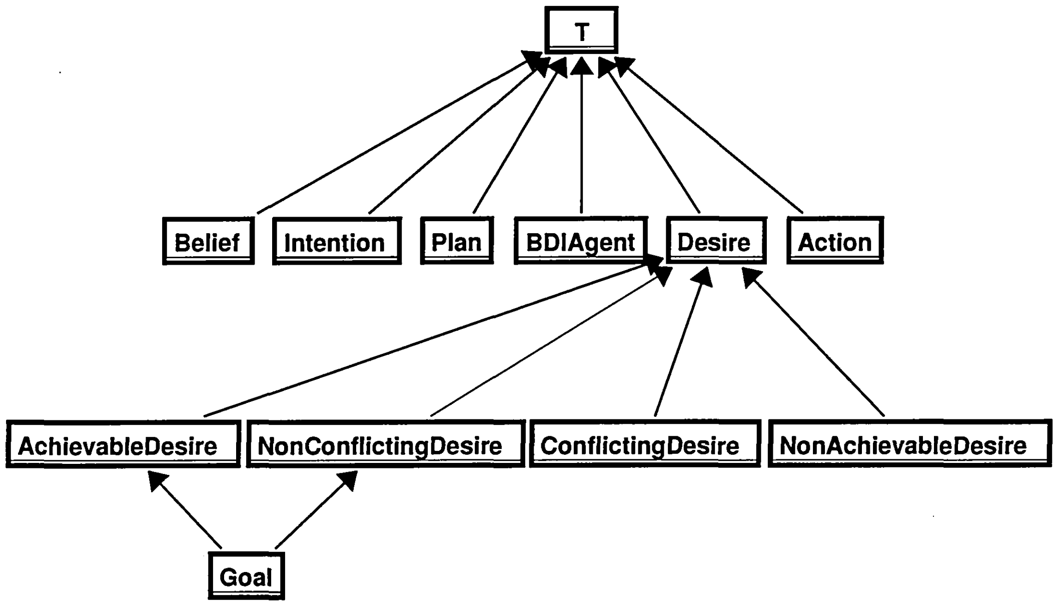


Figure 5.3: Type Hierarchy of BDI concepts (Absurdity Type omitted).

With reference to the MoGATU BDI Ontology of Perich et al. (2004), *policies* are used to represent the concepts gathered by declaring the *pre* and *post* conditions of an agent's *action.* The amended type hierarchy is shown in Figure 5.5. Again, the types can be transferred into OWL relatively easily.

5.5 Towards a Refined Framework

So far, the TrAM Framework has been developed to accommodate BDI concepts via the use of ontologies. Additionally, the use of an existing ontology from the MoGATU project Perich et al. (2004) also illustrates the simplicity of mapping agent concepts back into the TrAM models. For completeness, an ontological representation of the Transaction Model is required. This will ensure that the framework provides comprehensive support for all aspects of the agent requirements gathering process. As a result of this work it is likely that the over-arching process introduced in Chapter 4, and critiqued at the

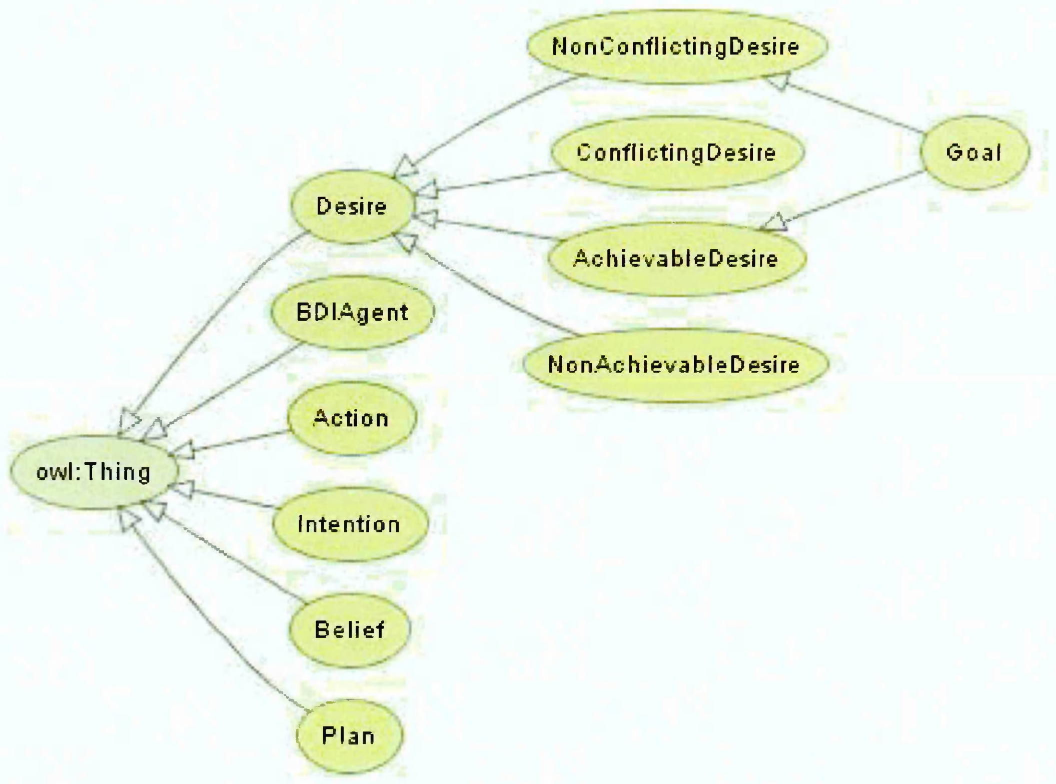


Figure 5.4: Visualisation of OWL file, translated from the type hierarchy.

beginning of this chapter, will also require rework. First, the development of a transaction ontology is described.

5.5.1 A Transaction Ontology

In its current form, the TM graph serves as an aid to structuring the early requirements efforts by providing a convenient metaphor. This also means that the concept types are defined in readiness for the generation of an ontology. It should be noted that the requirement for specifying *relation* names in the CG models implicitly creates relationship properties for an ontology, saving design time and supporting consistency in modelling.

The hierarchy of types from the generic TM is shown in Figure 5.6. Since

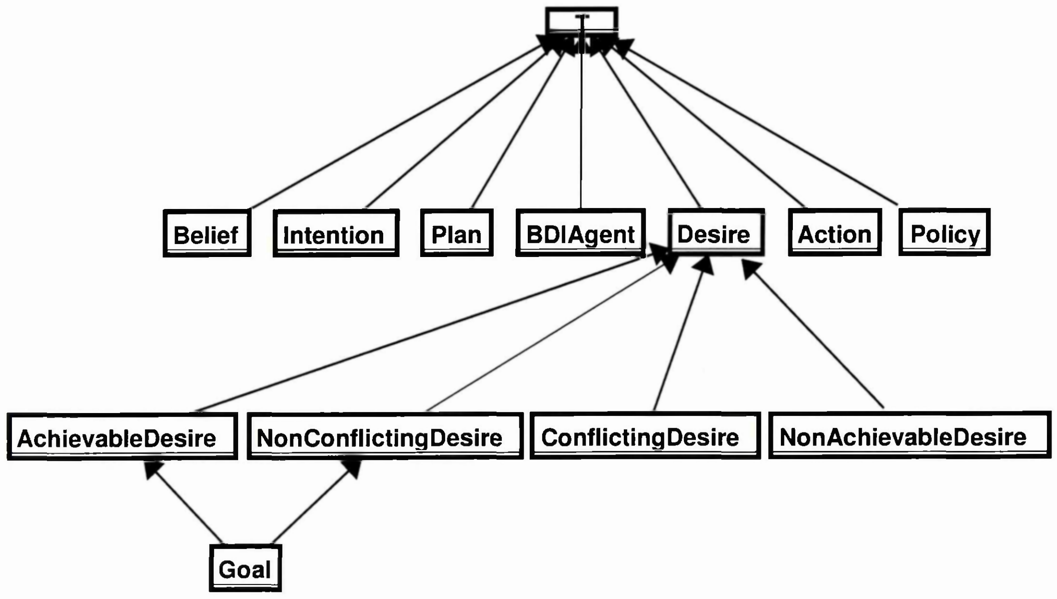


Figure 5.5: Amended type hierarchy to include `Policy` concept.

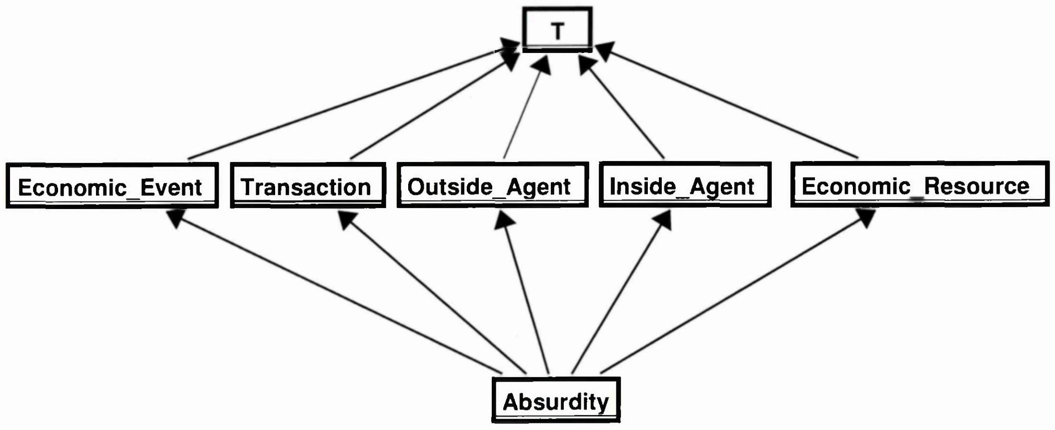


Figure 5.6: Type hierarchy from the generic Transaction Model.

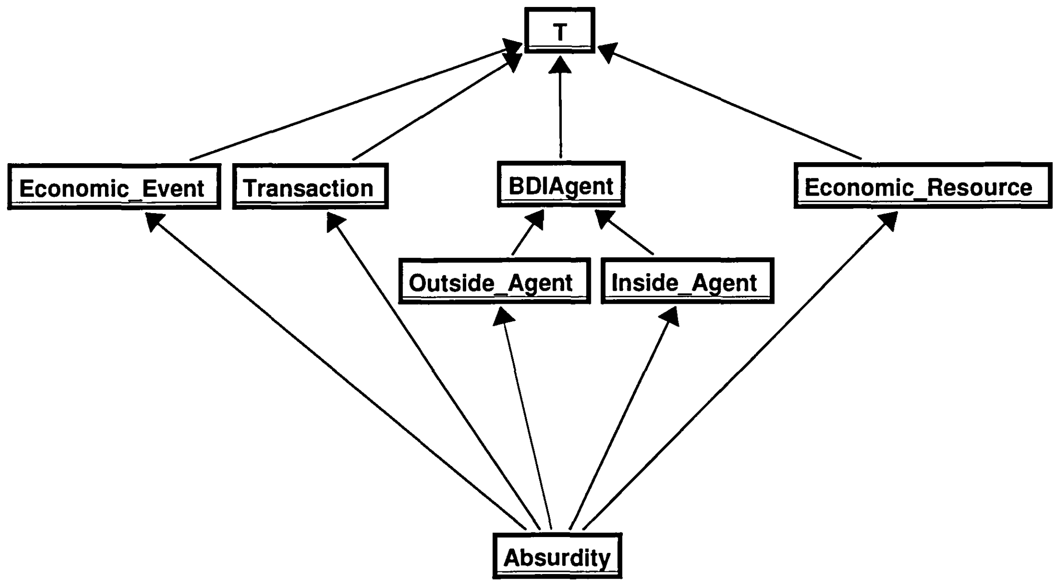


Figure 5.7: Refined TrAM Type Hierarchy.

both the `InsideAgent` and `OutsideAgent` are specialisations of a `Type:  Agent`, this relationship can be generalised. To reuse the ontology discussed in the previous section, the type has been generalised to `BDIAgent`, as in Figure 5.7. The *classes* in OWL are concrete implementations of *concepts* so:

`[Transaction]` becomes `<owl:Class rdf:ID=''Transaction''>`

Similarly OWL *properties* map to *relations*. However, from the TM CG we have the `Part` relationship between `Transaction` and `Economic Event`.

`[Transaction]->(Part)->[EconomicEvent]`.

To satisfy the OWL 'property', an inverse relationship has also to be declared. this results in two relationships:

`hasPart` and `isPartOf`

Giving:

`Transaction hasPart EconomicEvent`, and `EconomicEvent isPartOf Transaction`. The relevant OWL is as follows:

```
<owl:ObjectProperty rdf:about="#hasPart">
```

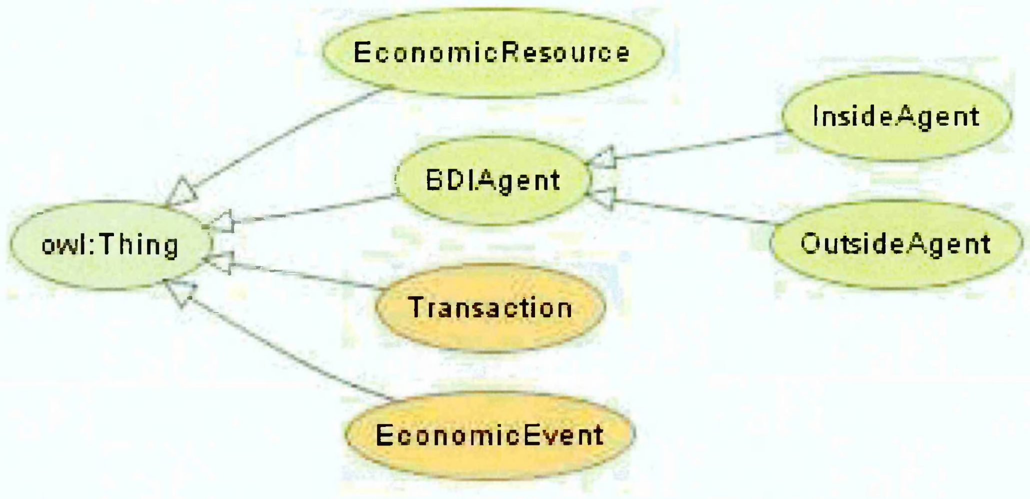


Figure 5.8: Visualisation of TM ontology.

```
    <rdfs:range rdf:resource="#EconomicEvent"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="isPartOf"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Transaction"/>
  </owl:ObjectProperty>
```

The corresponding visualisation of the OWL ontology is shown in Figure 5.8, and the entire OWL listing can be found in Appendix A, section A.2.

5.6 An Improved Process

This chapter has developed the TrAM Framework in order that it can explicitly mandate the elicitation of relevant agent concepts, whilst also mapping the models to an existing ontology. Furthermore, the Transaction Model has been mapped to an ontology to support the whole process.

However, reflecting critically upon the case study in Chapter 4, the whole of the analysis was predominantly driven by UML use cases, and made use of domain expertise that had already been acquired prior to modelling. Whilst the TM was used to refine the use case information, there was little 'early' requirements gathering and as such, the potential of CGs and the TM were not demonstrated fully.

Additionally, Chapter 4 recognised that a degree of familiarisation with CGs helps the process enormously and it is probable that most domain experts will neither have the time nor the inclination to study another notation.

John Sowa approached the CG notation from the perspective of *natural language* (NL) and CGs can be parsed into NL. A previous example is illustrated in Figure 5.9. The associated NL (parsed by the CharGer tool, Delugach (2006a)) is as follows:

```
There is a Proposition where
betw Person Mum and Person Dad is Person Daniel
```

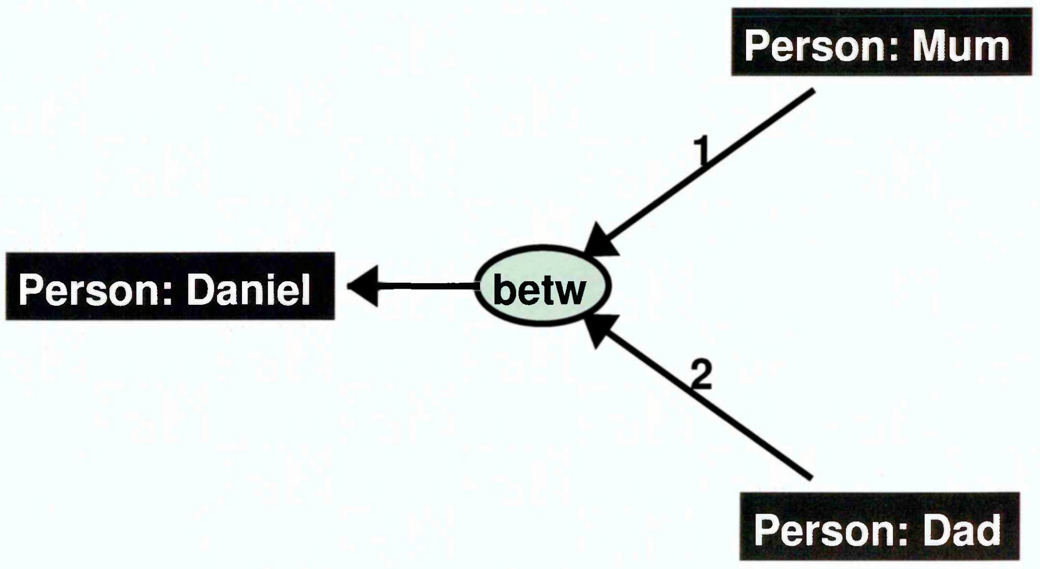


Figure 5.9: Example of a display form graph to be parsed into natural language.

A more pertinent example is shown below, with the associated graph in Figure

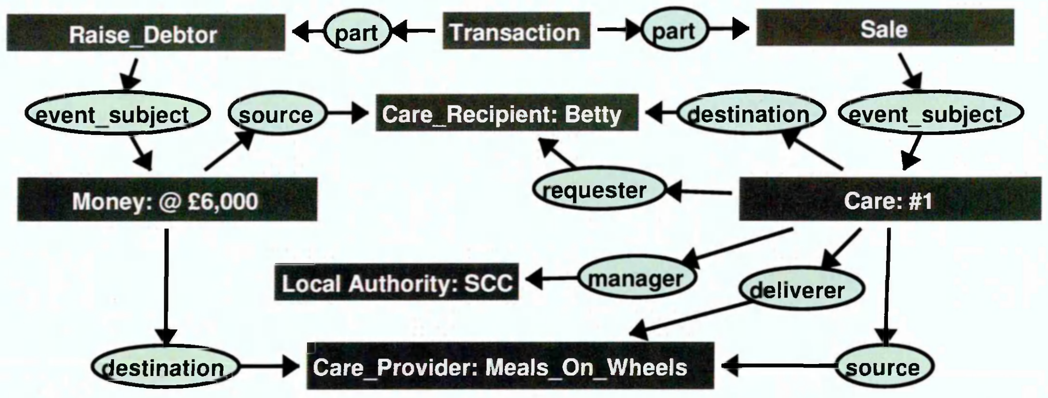


Figure 5.10: Specialised healthcare TM graph.

5.10.

```
There is a Proposition where

manager of Care #1 is Local_Authority SCC

deliverer of Care #1 is Care_Provider Meals_On_Wheels

part of Transaction is Sale

part of Transaction is Raise_Debtor

requester of Care #1 is Care_Recipient Betty

source of Money @ 6,000 is Care_Recipient Betty

destination of Care #1 is Care_Recipient Betty

event_subject of Sale is Care #1

event_subject of Raise_Debtor is Money @ 6,000

destination of Money @ 6,000 is Care_Provider Meals_On_Wheels and

source of Care #1 is Care_Provider Meals_On_Wheels
```

The parsed output depends upon at which concept the reader attempts to interpret the graph. This can be mandated with an LF graph (since we tend to read from left to right, and from top to bottom), in contrast to a DF graph. In this respect, the following issues are important:

- LF requires the concept order to be read correctly, which is absent from a DF CG. Thus we need to represent where a DF CG 'starts', and maybe direct the order in which the concepts are evaluated. This deals with the criticism of a lack of process (order) that DF graphs have.

- CharGer produces an 'English' output from DF graphs that ideally needs transforming into LF (in order that the 'start' concept might be identified), after which it might be further transformed back into NL for the domain expert. This would reduce the intellectual distance between model and NL for the domain expert, simplifying knowledge capture and accuracy.

- Some of the relation names from DF do not read very well in LF - Sowa has attempted to define a conceptual dictionary (Sowa, 1984), with models that are based upon NL. Translating from NL to LF, then DF might make the LF more readable for the expert, or at least be more sensible as an input for conversion back into NL for a domain expert to understand.

In essence TrAM overcomes this by providing the TM metaphor; this simplifies the comprehension of the graph as it uses a vocabulary that is sufficiently abstract to accommodate a wide variety of concepts, yet is straightforward enough to comprehend in terms of a perceived organisational activity. This restricted vocabulary also provides guidance as to the 'fit' of possible domain terms, aiding the agent system designer and domain expert alike.

However, since LF and NL assist the framing of questions for the domain expert to check the validity of the model, the use of CGs and the TM do not preclude the use of NL at the outset for requirements capture, though this is beyond the scope of this research.

5.7 Discussion

This chapter has described the refinement of the TrAM Framework and whilst generic features such as high-level conceptual modelling, ontology generation and model-checking have been illustrated through the healthcare case study, the TrAM Framework shall now be evaluated with reference to Section 2.8 in Chapter 2, in order to critically assess the usefulness of this approach.

5.7.1 Desired Characteristics

Chapter 2 established a set of desirable characteristics for an agent design framework, thus describing a mandate for this research. Each of these characteristics shall now be considered in relation to TrAM by using the ranking model proposed by Sturm and Shehory (2003):

1. *A clearly defined process that describes how the framework is applied together with the details of any implicit process.* The TrAM Framework process describes the steps required to perform modelling and analysis of requirements capture for a MAS. To supplement this, a series of design artefact documents illustrate how the models are processed and refined in an iterative way (Appendix C). Much of the model analysis could be automated, and the current lack of an automated tool means that this particular criteria is not yet comprehensively satisfied.
   Ranking = 5.

2. *An ability to manage differing levels of abstraction, from the highest down to the most detailed descriptions.* Both of the case studies and associated prior work (Hill et al., 2006a,b) have demonstrated the wide variety of levels of abstraction that the CG notation can represent, from individual

agent goal and task analysis through to societal motivation concepts. Unlike Gaia however, TrAM does not specify an organisational meta-model, even though the notation can support it. This is less of an issue for an experienced agent system designer, but the inclusion of a meta-model would improve comprehension considerably, and offer guidance when dealing with complex domain problems.

Ranking = 5.

3. *An ability to capture and model high-level qualitative concepts at an* 'embryonic' *requirements stage.* The representation and analysis of qualitative concepts is a key strength of the TrAM approach, enabling high-level concepts to be scrutinised and included within the resulting system. This reduces the temptation to compromise system functionality in order to successfully implement an a MAS application. Prometheus gives considerable support for the decomposition of goals, though it assumes that the 'early' requirements have been established already. Tropos is much better in this respect, as it explicitly addresses early requirements. The treatment of qualitative goals with Tropos is less obvious, though it is likely that 'hard' and 'soft' goal analysis will prompt the designer sufficiently that the necessary qualitative analysis is performed.

   Ranking = 7

4. *A guide to the elicitation of stakeholders and their goals, and be able to manage the discovery of system goals.* The consideration of high-level concepts with TrAM means that societal stakeholders can be elicited. It is conceivable that prior experience with other agent design methodologies may cause the agent system analyst to 'rule out' societal or strategic

stakeholders, restricting the potential benefit of utilising TrAM. In contrast to Tropos and Prometheus however, goal elicitation is performed by iterative analysis, rather than the process of AND/OR decomposition. Therefore TrAM is more flexible in its representation, though it is possible to be less disciplined.

Ranking = 5

5. *A mechanism for eliciting and deriving pertinent agent and domain concepts, allowing the representation and open expression of agent concepts such as: belief, desire, intention, role, society, task.* Again, the CG notation permits the widest variety of concepts to be represented. TrAM formalises the representation of BDI concepts in CG notation, by making reference to a BDI ontology and producing the Transaction Model type hierarchy and resulting ontology of types. The connection between concepts, type hierarchies and ontology, within the framework of the Transaction Model graph, means that TrAM specifically supports the evolution of a domain ontology as the models are iterated and refined. This process is currently performed manually, and would benefit from automation, thus improving speed of analysis and also provide consistency checking.

   Ranking = 6

6. *A process that includes an implicit model check to verify the elicitation of key domain concepts at the earliest opportunity. This process must be able to enable checking of the model's consistency, ideally with tool support.* Use of the TM graph ensures that 'balance' checks upon opposing concepts is implicitly performed. Any missing nodes in the model result

in an incomplete graph, and consequently the ontology is lacking also. The use of CG notation means that the very first TM graph produces a rudimentary domain ontology, thus supporting the production of different views of a system at the earliest opportunity. System analysts can use the CG type hierarchies to refine their models, and they may also check the appropriateness of domain terms with a domain expert. Additionally the simple parsing of CG models into natural language is also a convenient vocabulary check for both system analyst and domain expert. The use of tools is important since the complexities of agent systems offer many opportunities for inconsistencies to present themselves, and whilst CharGer (Delugach, 2006a) supports the maipulation of CGs and Protégé (Stanford Medical Informatics, 2006) the manipulation of ontologies, there is currently no automated interoperability between these tools. Since ontologies produced with Protégé can be utilised with agent programming APIs such as Jade (Bellifemine et al., 2001), and the transformations between CGs and ontologies are now mapped under the TrAM Framework, this is clearly an area that requires work.
Ranking = 5

7. *A process whereby focus is directed upon inconsistencies or parts of the model that are ambiguous.* Model generation with TrAM is focused almost entirely upon the realisation of 'problematic' concepts. The inherent balance check of the TM forces qualitative and difficult-to-realise concepts to the fore, resulting in the analysts' efforts being expended in the most challenging areas.
Ranking = 7

8. *A means by which domain terms, constraints and rules can be captured and represented in an ontology.* TrAM offers explicit support for ontology generation by providing a mapping from the TM model, that incorporates agent specific characteristics. The FOL underpinnings of CGs imposes a rigour upon the modelling process that together with the visuality of DF graphs and Peircian logic, presents a comprehensive means of building rules that can be used to query models and derive new knowledge. At present this aspect lacks a theorem prover to automate the process, though the work of Heaton and Kocura (1993) provides a basis for this operation.
   Ranking = 6

9. *A representation medium that permits the transfer of models across domains, and that serves to complement other agent design methodologies.* The use of TrAM enables models to be transferred in a variety of ways, and the FOL underpinnings and exchange formats such as CGIF (Delugach, 2006b) permit concepts, relationships and logic to be preserved. Use of the TM graph allows the abstract transaction concept to be used as a design metaphor, that is specialised with domain specific terminology, policies and rules. Consequently the two case studies illustrate both the transferable abstract qualities of TrAM, as well as the domain specifics that appear as a result of modelling with this approach. Additionally, the notion of early requirements capture means that TrAM can be used as a precursor to other agent design methodologies that require some initial documented requirements analysis as an input, such as Gaia.
   Ranking = 6

10. *A process that is intuitive and enables novices and experts to design agent models.* TrAM documentation artefacts support the description of the process, however, there are two issues that this research has exposed. Firstly, the use of Peirce logic with DF graphs requires familiarity with both CG notation and rule-base logic. This can be problematic for non-computing domain experts, and may limit the potential audience for TrAM. Secondly, the CG notation is very flexible, and by its very nature can be used without reference to the TrAM Framework process. One strategy would be to impose more stringent process models (akin to Prometheus), or even to derive a series of design patterns that provide pre-built models to populate, similar to the abstract TM.

    Ranking = 5

Characteristic	GAIA	Prometheus	Tropos	TrAM
1. Process	4	6	5	**5**
2. Abstraction	4	5	5	**5**
3. Early requirements	1	1	5	**7**
4. Goal discovery	1	5	5	**5**
5. Agent concepts	2	5	5	**6**
6. Consistency checking	2	3	4	**5**
7. Analysis by exception	2	3	3	**7**
8. Ontology support	1	2	2	**6**
9. Transferability	4	4	4	**6**
10. Intuitive	3	6	5	**5**

Table 5.1: Evaluation of TrAM against desired characteristics identified in Chapter 2.

5.8 Conclusions

This chapter has examined the limitations of the initial TrAM approach and has considered the lack of support for agent concepts such as BDI. The production of ontologies is regarded as a key advantage of this approach, yet the initial framework only produced type hierarchies and rules in the form of CGs. To make the approach more useful, a collection of ontologies have been mapped that recognise the core agent concepts and the TM, providing the necessary support for the TrAM Framework. Chapter 6 will demonstrate the improved framework in a second case study, in the m-Learning domain.

Chapter 6

Applying TrAM to MOBIlearn

6.1 Introduction

The refined framework introduced in Chapter 5 will now be applied to MOBIlearn[1] an EU Funded project in the m-learning domain. The TrAM Framework is used to produce a series of artefacts including:

- High-level conceptual models demonstrating qualitative and political influences upon the case study;
- Specialised Transaction Models (TM) illustrating duality relationships between events and resources;
- Hierarchies of concept types and an audit trail of key modelling decisions;
- Ontology development from the requirements models.

These results illustrate the extent to which each of the key criteria identified in Chapter 2 have been addressed. Areas of generic applicability are identified, as are domain specific aspects of the modelling process.

[1]EU Project IST-2001-37440

6.2 Rationale for Case Study

The MOBIlearn Project (more details in Appendix B Section B.1) is an attempt to improve access to knowledge for selected users, by retrieving learning materials from the Internet via mobile connections and devices, to "foster their life long learning and enhance their working experience". Three specific groups of users have been identified:

1. *Workers* - providing learning for continual, work-based skills and knowledge development;

2. *Citizens as members of a culture* - to enrich the learning and offer new possibilities for embracing cultural knowledge during a visit to a city;

3. *Citizens as family members* - to provide simple medical information on demand.

The aim of the project is to provide a set of requirements, pedagogical guidelines, best practices and an architectural framework to support mobile-learning (m-learning) (Haley et al., 2004). MOBIlearn is similar to the community healthcare case study discussed in Chapter 4 in that they are both inherently multi-agent systems, and they are both complex. The m-learning platform however, differs somewhat in that the students must engage with the technology and utilise mobile devices to assist their own learning process. In contrast the community healthcare domain abstracts the technology away from the recipient and attempts to make it as invisible to the user as possible. There is also a need to build interactive communities, amongst not only the managers and facilitators of learning (tutors), but also the learners themselves. This makes the process of gathering requirements more challenging as a much wider range of demands will need to be accommodated.

As such, MOBIlearn is a particularly suitable candidate for a second case study as the preliminary work packages concentrated upon requirements capture, and some significant issues have been identified with regard to the difficulties encountered during the categorisation and management of domain terms (Haley et al., 2004). Specifically, one of the 'lessons learned' was:

> "People have varying concerns and want to examine the requirements from different perspectives. These concerns change over time and during different stages of the project."

The key outcome was that a requirements management system needs to be able to permit ad hoc updating of categorisation criteria from the requirements models. The use of TrAM to build ontologies from high-level conceptual models is an attempt to address this.

6.3 MBA Scenario

The MBA scenario explores formal learning by highly motivated, busy professionals and first-year students. It investigates the use of new and emerging technologies as part of a time and cost optimised learning process.

6.3.1 Capture Concepts

The first step is to capture some concepts. From Appendix B, Section B.1, one of the objectives of the project is:

> "... to improve the knowledge level of individuals through cost and time optimisation of learning processes."

This is simply modelled as a CG, to define some of the concepts, and to start thinking about the relationships between concepts. Figure 6.1 illustrates the relevant graph. Such is the richness of the notation that two individuals may produce two different graphs; this is typical during requirements gathering exercises and the TrAM approach is no exception. The debating process can start early, and it typically focuses attention upon a particular area for further analysis. Additionally a parsed English version of the graph is as follows:

```
There is a Proposition where
Level of Knowledge is Increase
Provider of Learning is MOBIlearn_service
Srce of Money {*} is Person {*}
Srce of Commitment is Person {*}
Subj of Commitment is Time
Chrc of Cost is Money {*}
optimise are Cost and Time and Process
Expr of Learning is Person {*}
Inst of Learning is Process
Chrc of Learning_materials is Cost
Rslt of Learning is Knowledge and
Obj of Learning is Learning_materials
```

This process is repeated until all of the concepts seem to be captured. Unlike other agent modelling approaches however, it is anticipated that designers will not have to immediately derive process-level use cases. It is also preferable to model high-level concepts such as, society, government, economy and culture. For instance, the system has a *duty of care* to ensure that the students' experience is maintained or improved. Figure 6.2 gives an example of how this might be modelled. Immediately, `Meas` (measure) relationships

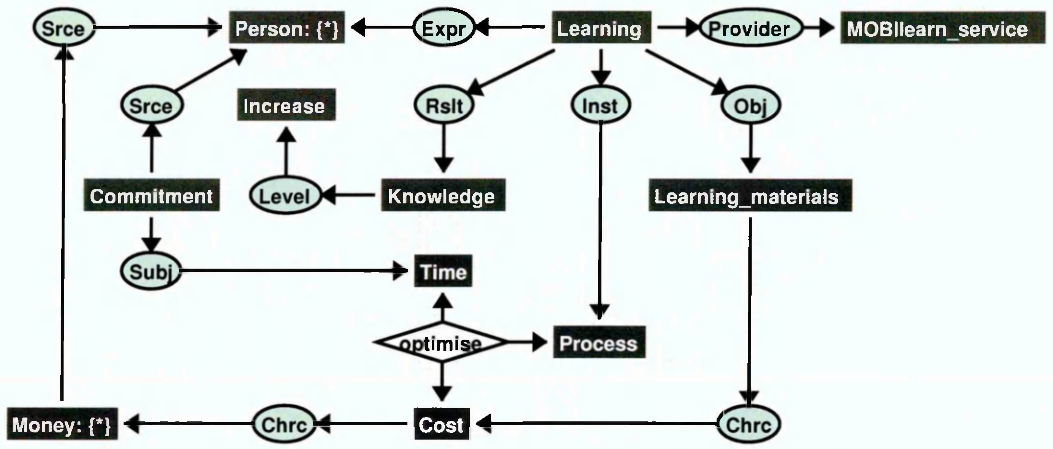


Figure 6.1: Modelling a MOBIlearn objective with CGs.

identify probable qualitative concepts, in this case `Quality` and `Feedback`. Looking specifically at the MBA scenario, Figure 6.3 describes the basic char-

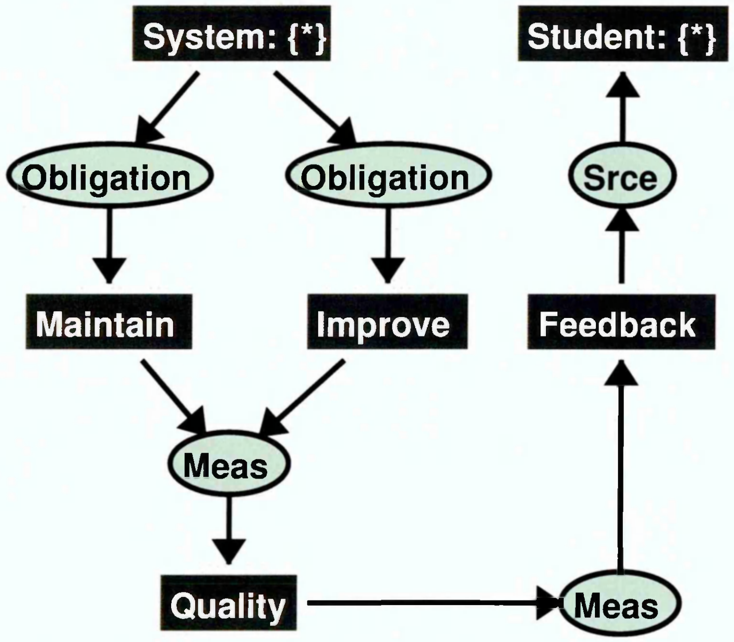


Figure 6.2: The M-Learning platform's duty of care towards students.

acteristics of the MBA student. Again, concepts are being brought to the fore ready for debate. The services offered by the MOBIlearn platform are also considered and the model begins to show some of the stakeholders of

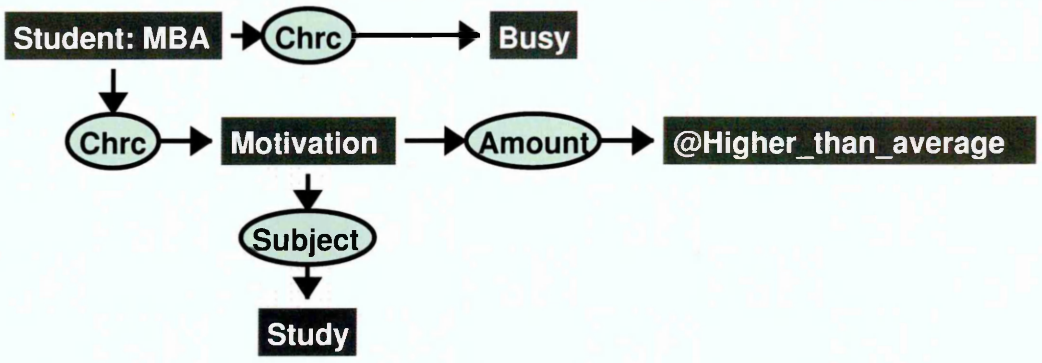


Figure 6.3: CG representing an MBA student.

the system, together with the roles they play. For instance, the `Student` will request `Learning_Opportunities` in order to acquire knowledge. The `Materials` will be authored by a `Tutor`. The `Tutor` will also facilitate various `Learning_Opportunities` such as `Tutorial_Activity` and `Group_Working`, and deliver a `Lecture` or `Presentation`. Whilst these models are being created, several other elements of the process are also occurring:

- The analyst is thinking about domain terms - both concept names and relation names. These will eventually become part of the classes and properties of an ontology.
- Analysis is continually focused on modelling that is either too abstract, qualitative or just too difficult to represent with the current knowledge. This concentrates effort upon the areas that need the most work.
- '*Why*' questions are being asked of the model, as well as the more typical *what* and *how*. This helps build up a rationale for the modelling decisions, which is implicitly documented as part of the process artefacts.
- The analyst will also be thinking about how this system will be realised in terms of goals - what are they, where are they and how they will be

expressed.

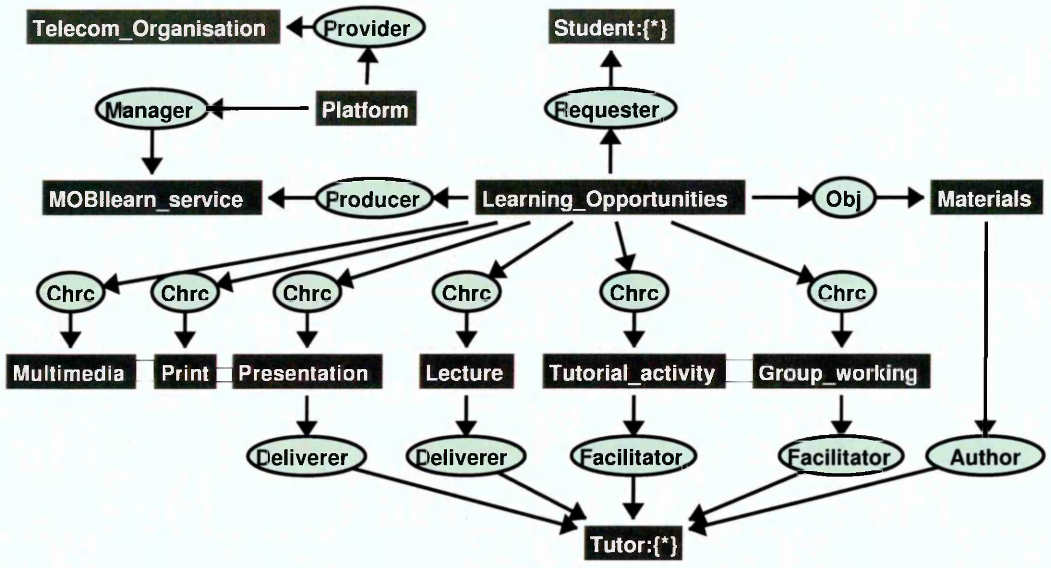


Figure 6.4: CG model of the MOBIlearn service characteristics.

Once a number of graphs have been created, recurring concepts will start to emerge. At this point it becomes possible to join graphs to make a more comprehensive model of the system. Figures 6.5 and 6.6 show graphs that can be joined. The graphs remain separate for legibility, however the coreferent concepts `[Student:{*x}]` and `[Learning_Materials:{*y}]` illustrate where the graphs will join. It should be noted that the graph of Figure 6.4 contains the concept `Materials`, whereas Figure 6.1 contains `Learning_Materials`.

Clearly they are referring to the same entity and this is one such example of how domain concepts are reconciled throughout the process of modelling. As the concepts become more grounded, the analyst can begin to consider the desires that each stakeholder would like to pursue.

For instance, the `Learning_provider` would like a course that is recognised as `Prestigious`, whilst also offering `Value_for_money`. Similarly a `Student` also desires `Value_for_money`.

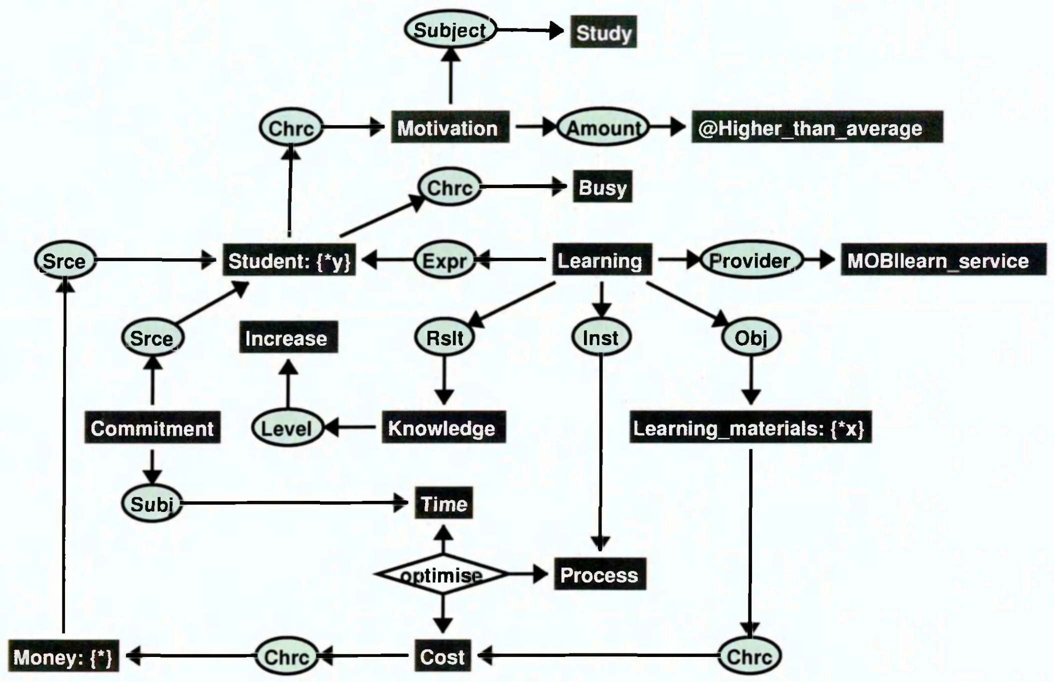


Figure 6.5: Initial overall model Part 1.

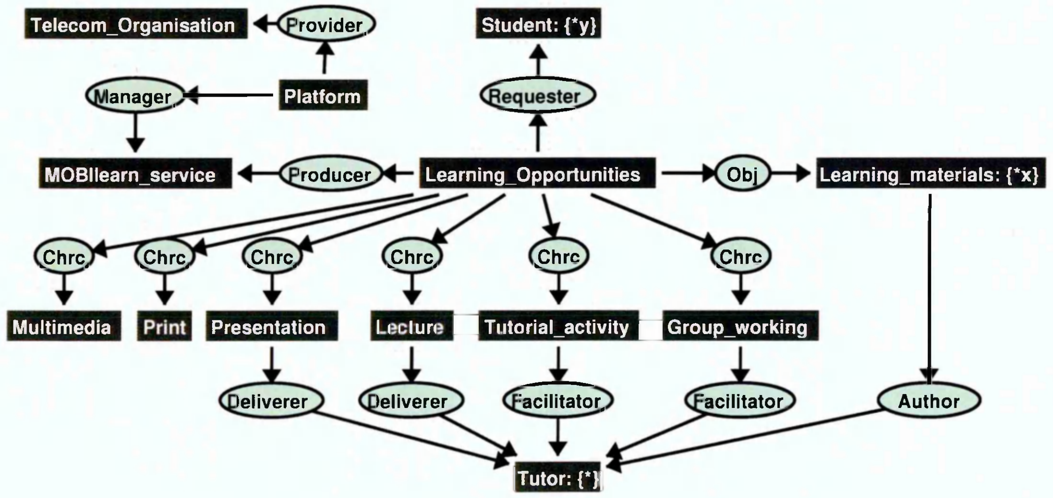


Figure 6.6: Overall model part 2 (note coreferent links to Part 1 model).

The Learning_provider also wants academic integrity, in both Students and Tutors. Immediately conflicting goals such as 'recruit more students' and 'raise course fees' present themselves as part of the elicitation process. Figure 6.7 illustrates some of these desires. Using the preliminary model in Figure

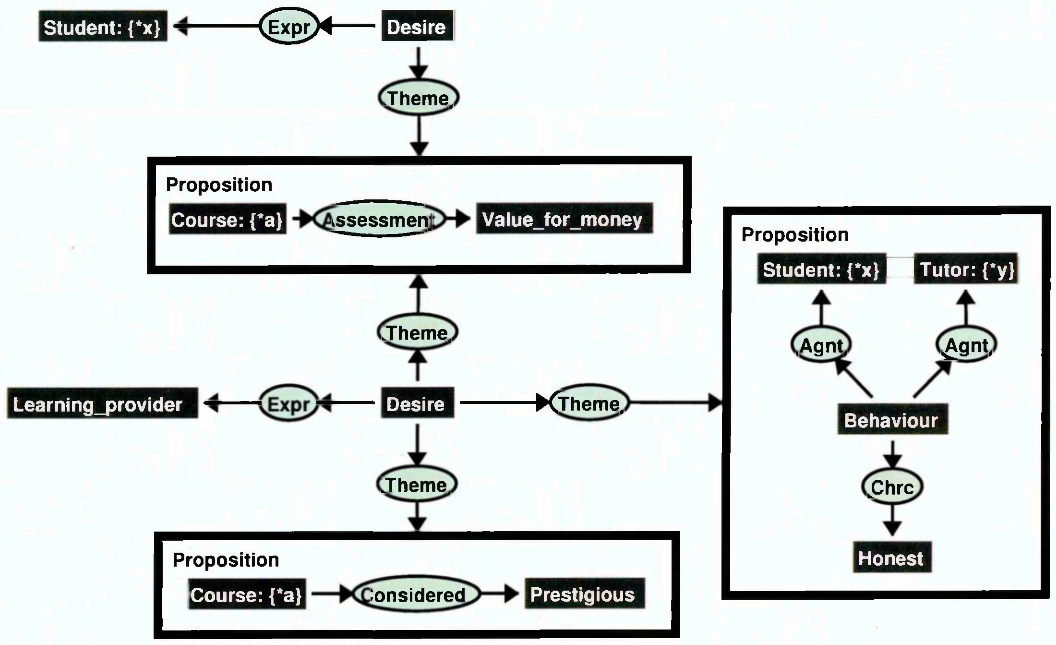


Figure 6.7: Some high level desires of the stakeholders.

6.7, analysis is now conducted upon the individual stakeholders. From Figure 6.8 we can see that the high level desires of the Student can be broken down further. In order to pass the course, the Student must attain a Total_Mark of at least 40%. To achieve this, it is necessary to engage with the learning opportunities and participate in group activities. Improved_Prospects are also a goal, as is the acquisition of new and relevant Knowledge. Repeating the exercise for the Tutor, we derive Figure 6.9. The high-level goals for Student and Tutor are summarised in Table 6.1.

Desires such as *enjoy course* and *obtain better prospects* are of course examples of qualitative goals that would give an agent little indication of the

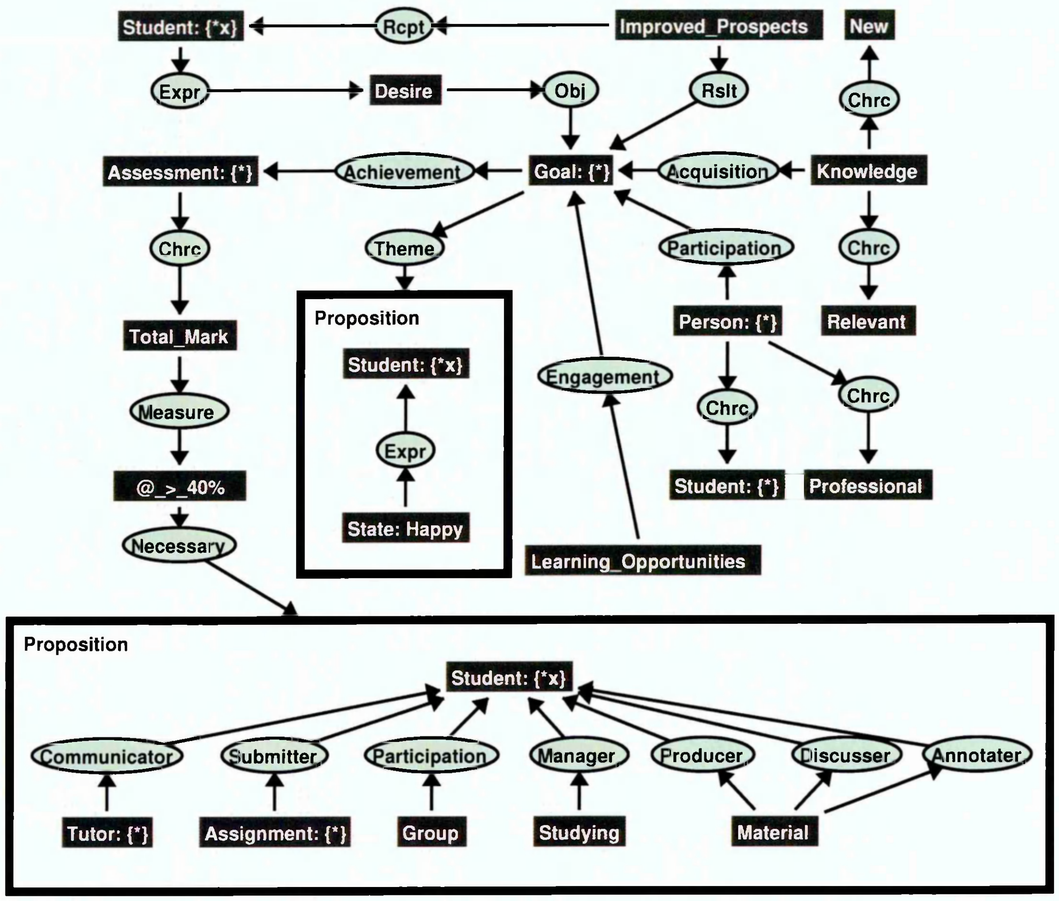


Figure 6.8: Iterated CG model of Student desires.

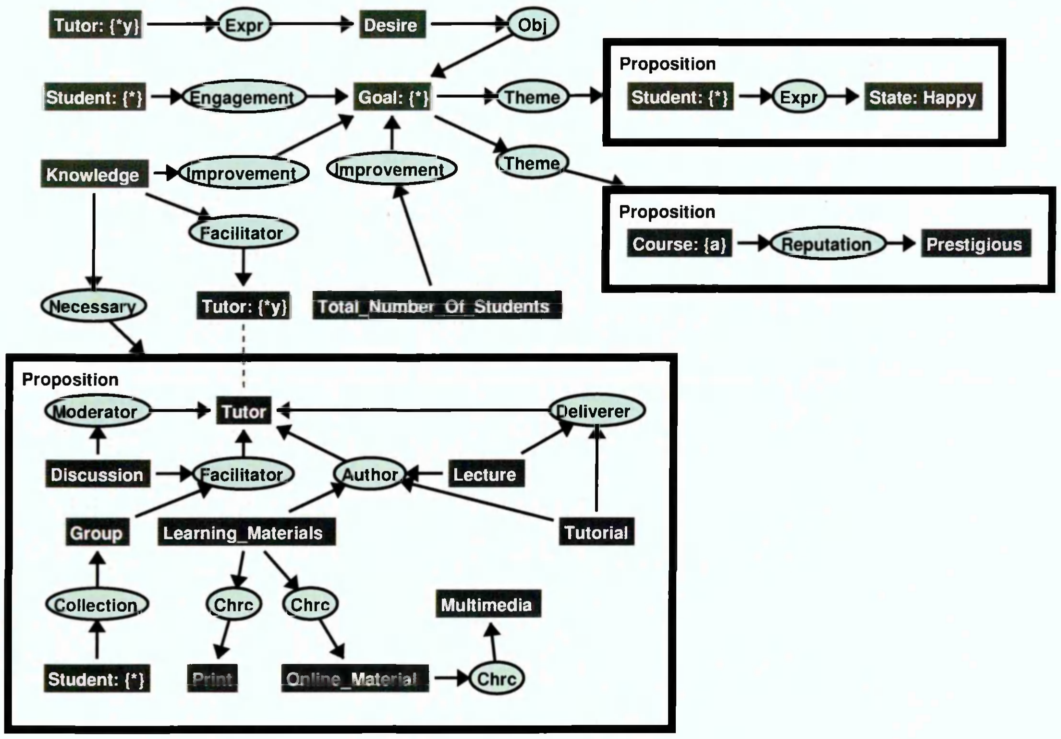


Figure 6.9: Iterated CG model of Tutor desires.

intention (plan) required, and as such require more scrutiny before the system is implemented.

If *enjoy course* is considered, then the analyst is directed towards defining an output or set of results that would indicate that the goal has been successfully achieved. Some indication of the `Students`' state could be gleaned via feedback mechanisms, the result of which is some data that might be used for a performance indicator (PI). Figure 6.10 illustrates how the `Measure` relationship is utilised to achieve this.

6.3.2 Transform with TM

From the models gathered so far the CGs are reviewed to determine how they fit in to the generic TM (repeated in Figure 6.11).

Stakeholder	**Goal**
Student	Enjoy course
	Pass course
	Acquire new knowledge
	Acquire relevant knowledge
	Obtain better prospects
	Engage with learning opportunities
Tutor	Facilitate knowledge acquisition
	Facilitate student group discussion
	Author learning materials
	Deliver lectures and tutorials
	Moderate student discussion
	Engage student
	Increase number of students on course
	Facilitate a prestigious reputation for the course

Table 6.1: Some of the high-level stakeholder goals from the CG models.

The TM denotes that `Acquire_knowledge` is a transaction that arises due to the occurrence of two complementary economic events, namely `Sale` and `Raise_Debtor` as shown in Figure 6.12.

These are considered economic events because they illustrate the demand upon a limited resource. The `Raise_Debtor` requires limited resources of the `Learning_provider`, who has to make provision for this cost at the potential expense of other events such as developing new materials, marketing courses or investing in new infrastructure, upon which their finances could be spent.

Similarly the `Sale` calls upon the `Tutors` priorities, in terms of potentially being required elsewhere or more simply the 'contact-time' spent with a student. Hence the `Learning_provider` needs to manage (*source*) some optimally cost-effective time. Clearly if money was no object then an infinite number of tutors could be employed and the time available would be maximised, and as a result the greatest opportunity for learning would take place.

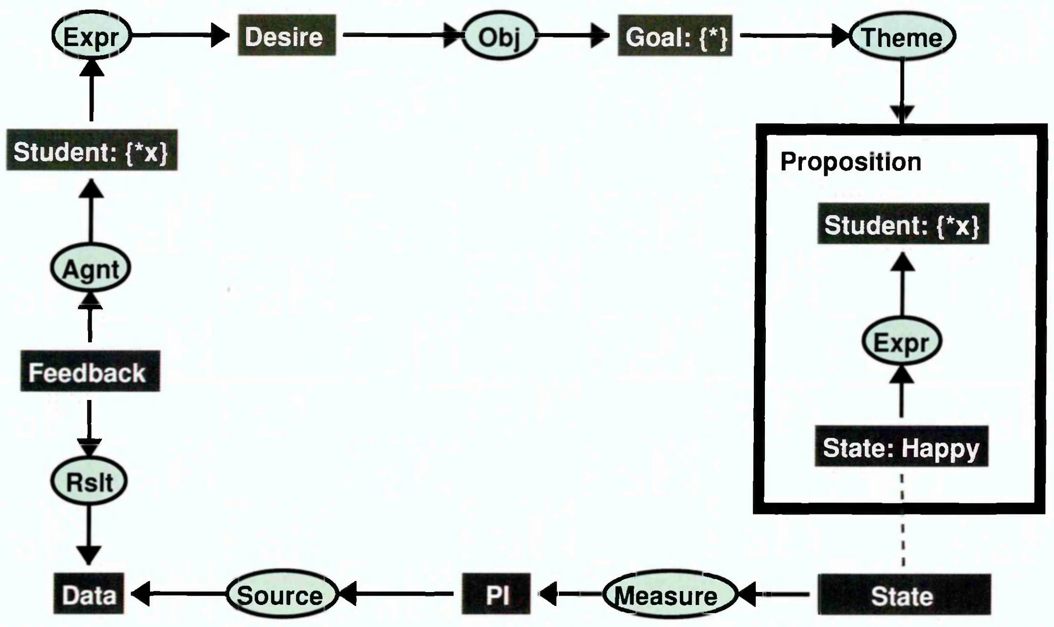


Figure 6.10: Exploring the *enjoy course* goal.

Since the `Learning_providers` finances are limited, and there are competing prioritising demands for that finance to be spent elsewhere, money is an economic resource that is scarce. This in turn makes an unrestricted number of `Tutors`' hours impossible, and as a consequence `Time` is an economic resource. This is demonstrated by the very fact that `Time` denotes a restriction caused by competing demands (e.g. another `Student` being attended to by the `Tutor`) or the geographical timezone of the Tutors' location when the Sale is made. The corresponding benefit for these costs is the economic resource of the `Learning`, the `Destination` of which is the `Student`.

The consideration of `Time`, `Money` and `Learning`, brings the realities of the `Learning_provider` as a business to the fore. The cost-benefit trade-off for the Learning_provider is that there may be sacrifices that are too great to make, such as high `Tutor` to `Student` ratios. Since this transaction depends quite heavily upon the optimal 'spend' of time against money, there is a focus towards considering efficiency gains that might make the transaction more profitable.

The graph of Figure 6.12 should also show a transaction between the `Student` and the `Learning_provider`. However this is difficult to ascertain since the graph appears to indicate that the `Learning_provider`, like the `Student`, is the `Destination` of the `Learning`. This clearly does not model the relationship between the concepts in a realistic manner, and therefore it is necessary to represent the economic resource with a meaningful measure such as a performance indicator (as shown before in Figure 6.10). This PI would then be used to measure the effectiveness of the `Learning_provider` in providing learning opportunities. This measure thereby offers the focus for a relevant quantifiable concept upon which the `Learning_provider` and `Tutor` can make the most informed decisions, as indeed would their software agents. The following CG in Figure 6.13 captures these dimensions, therefore demonstrating that intangible qualitative economic resources need to have a characteristic of being measurable.

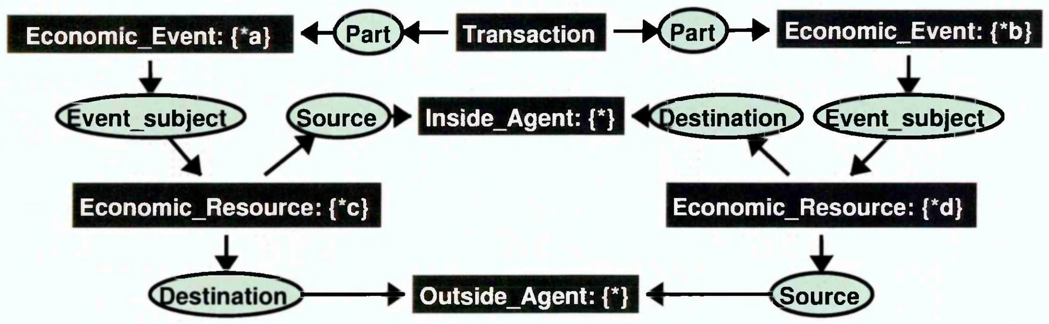


Figure 6.11: The generic Transaction Model.

We can repeat this activity for other transactions. One such pertinent case is the investment of time to study at the cost of time spent with the family (Figure 6.14).

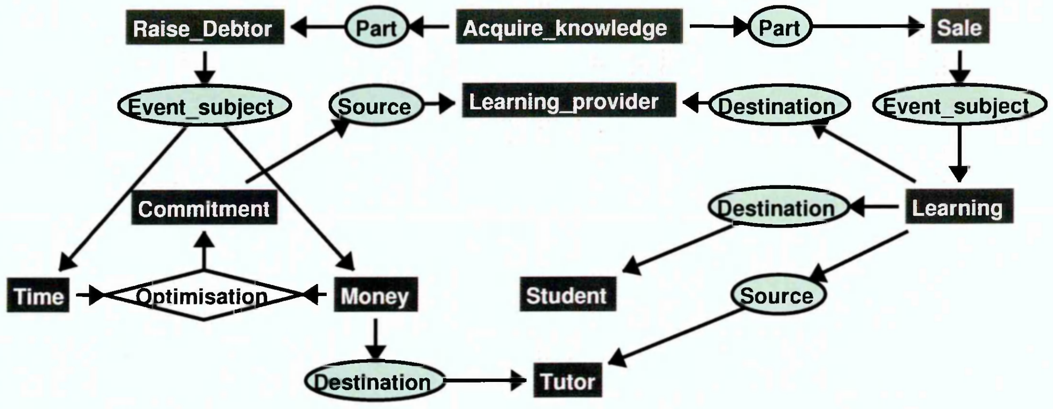


Figure 6.12: MOBIlearn scenario Transaction Model.

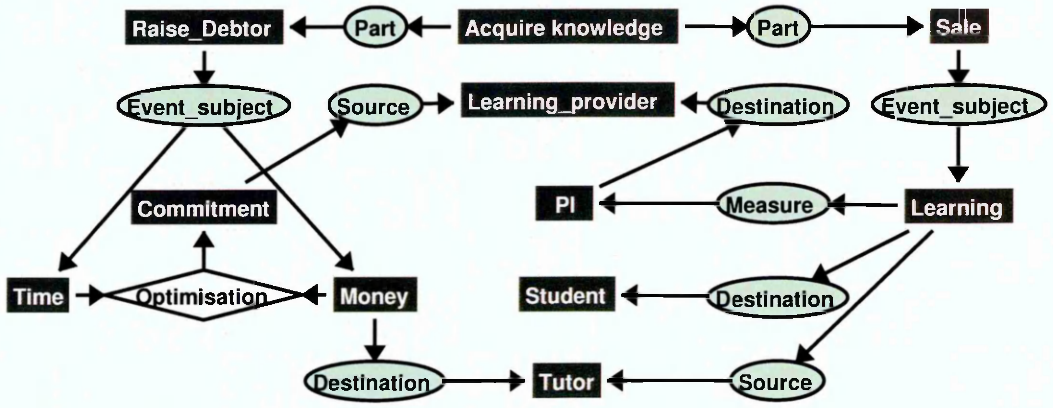


Figure 6.13: Amended Transaction Model.

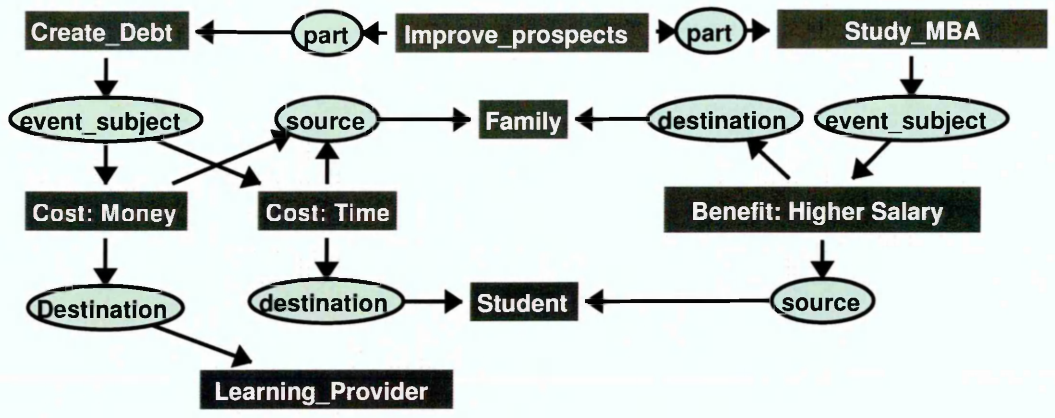


Figure 6.14: A trade-off between studying and spending time with the family.

Figure 6.15 illustrates another transaction, where the `Student`, by taking the course, produces performance indicator (`PI`) data. Therefore a potential student could infer the aggregate quality of student care by comparing this data with respect to a particular benchmark. As before in the previous case

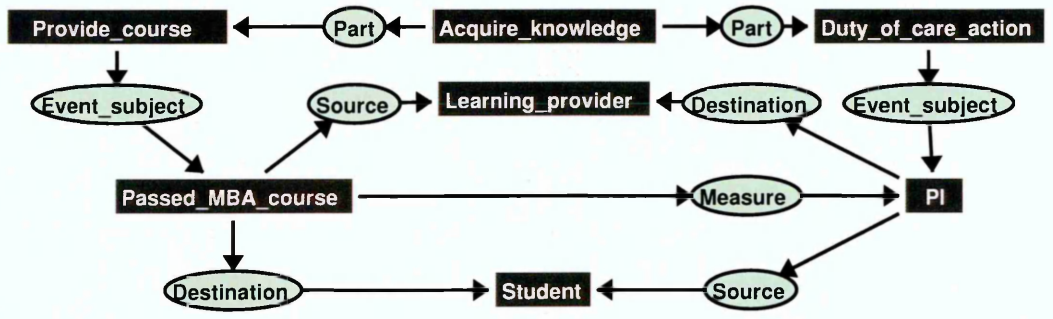


Figure 6.15: Capturing a student transaction.

study, (Chapter 4) transformation with the TM graph enables a type hierarchy to be mapped from the concepts derived (Figure 6.16).

The models are repeatedly iterated until the obvious missing concepts are derived. A useful operation at this stage is to check the Natural Language representation of a graph. This produces statements that:

1. Have domain concept names that can be verified in terms of grammar,

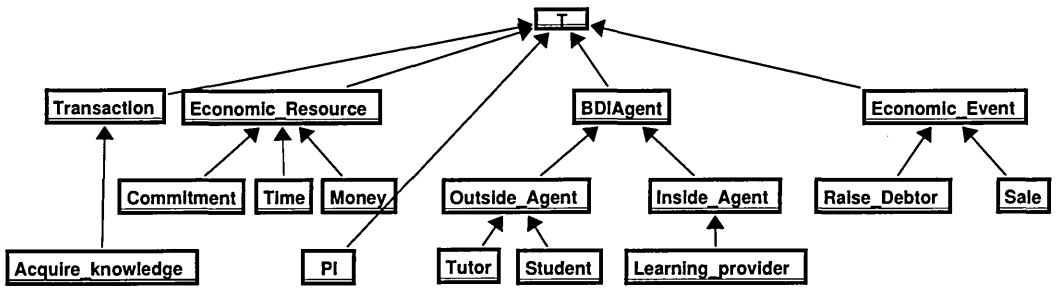


Figure 6.16: Type hierarchy from TM.

and;

2. Are verified in relation to the domain context.

For example, the following has been generated from Figure 6.13. Most of the statements appear to make sense. If we consider however, `Measure of Learning is PI`, it might be more appropriate to use a domain-related term such as `Examination_mark` or `Overall_mark`. If this is judged to be necessary, then the type hierarchy can be amended accordingly. This is a suitable point to consult a domain expert, since a considerable amount of analysis has already been performed purely on the high level concepts. The creation of the models is such that they serve as a framework for domain specifics such as terms, and the TM and resulting type hierarchies are easily modified. NL is an important step as it enables the analyst to perform a rudimentary check of the work conducted so far, whilst presenting the analysis in a straightforward way for domain experts.

```
There is a Proposition where
Measure of Learning is PI
Destination of Learning is Student
Destination of Money is Tutor
Optimisation of Time and Money is Commitment
```

```
Part of Acquire_knowledge is Sale

Part of Acquire_knowledge is Raise_Debtor

Source of Commitment is Learning_provider

Destination of PI is Learning_provider

Event_subject of Sale is Learning

Event_subject of Raise_Debtor are Money and Time and

Source of Learning is Tutor
```

The graphs are now amended to reflect any new knowledge that has been derived.

6.3.3 Gather Use Cases

With reference to the MBA use case scenario text in Appendix B, section B.2, Figure 6.17 illustrates the top-level use case model. The following stakeholders are shown as actors:

- Teacher,
- Student Administrator,
- Student, and
- Group, a generalisation of the Student actor.

The use case provide the necessary process logic that is absent from the TM graphs produced so far, whilst also enabling both model types to be iterated into a cohesive requirements model. Consequently the next step is to verify and refine the TM graphs.

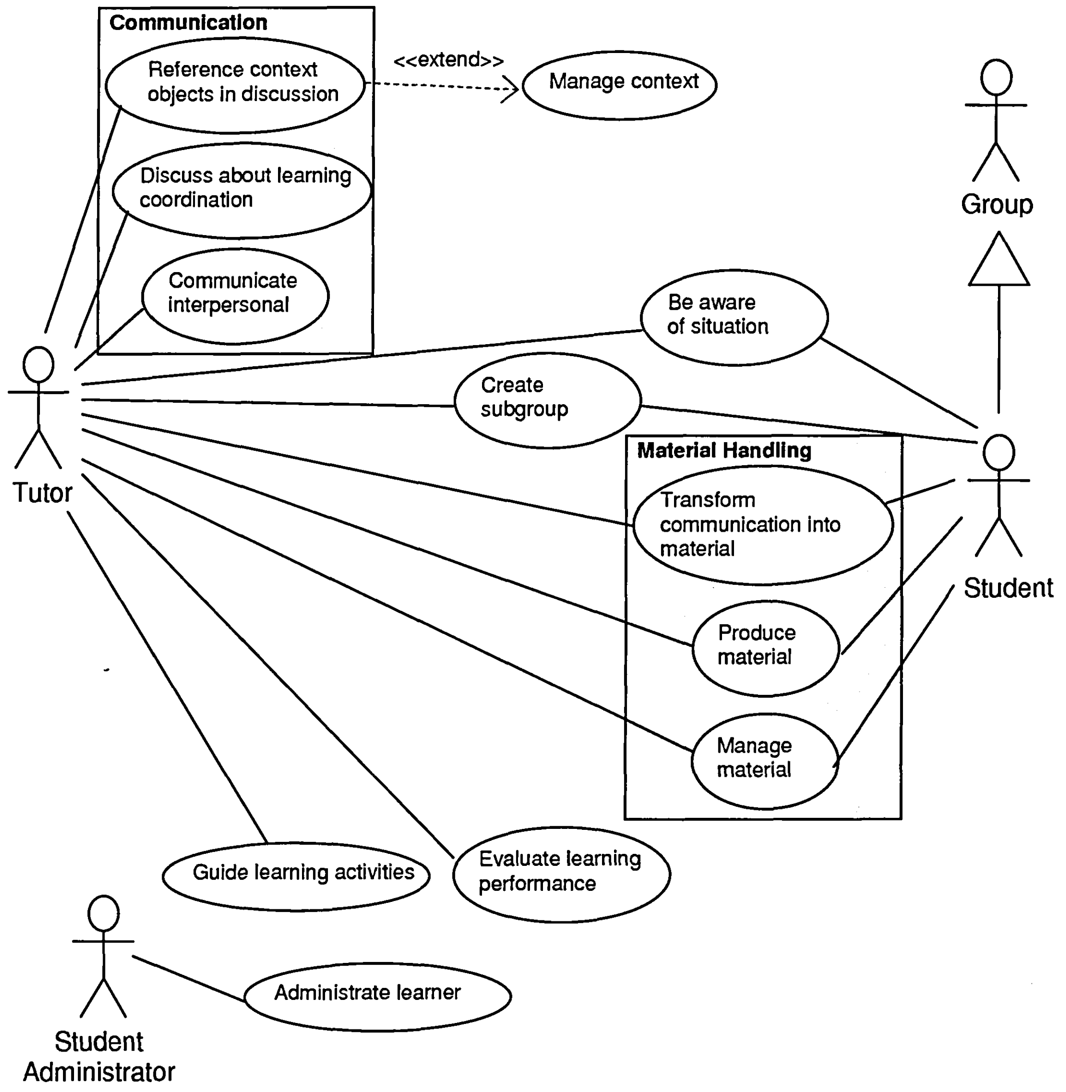

Communication
Reference context objects in discussion
<<extend>>
Manage context
Discuss about learning coordination
Communicate interpersonal
Group
Be aware of situation
Create subgroup
Tutor
Material Handling
Transform communication into material
Produce material
Manage material
Student
Guide learning activities
Evaluate learning performance
Administrate learner
Student Administrator

Figure 6.17: MBA top-level scenario use case model.

6.3.4 Verify TM Graphs

Upon producing the use case models, it is clear that some inconsistencies already exist. First, the domain term `Teacher` appears, rather than `Tutor`. This can be accommodated within the eventual ontology. Secondly, a `Student Administrator` actor is specified in the use case model. Whilst no such stakeholder exists in the TM graphs, the `Student Administrator` is a role within the remit of the `Learning Provider` and is therefore a specialisation. From the original TM of Figure 6.13, the two simple amendments are demonstrated in Figure 6.18.

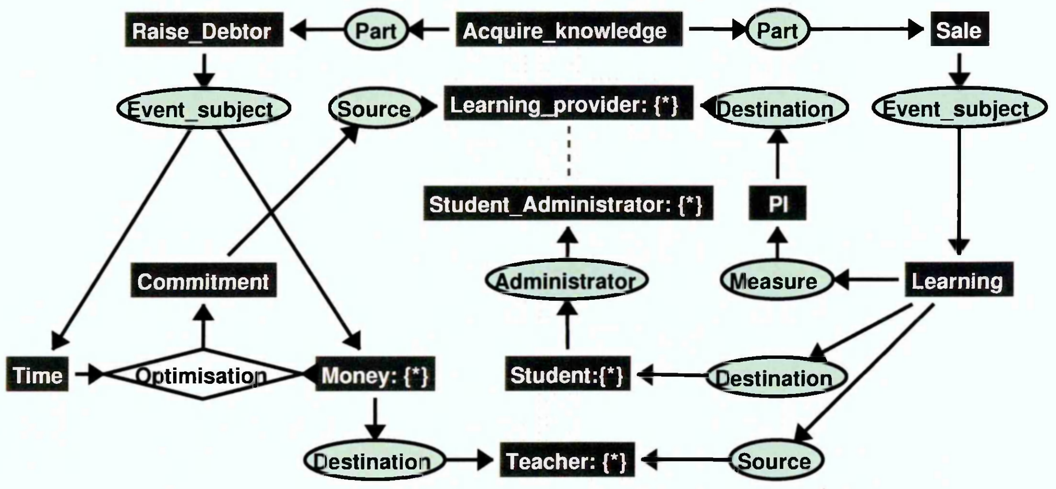


Figure 6.18: Updated TM graph from MBA use cases.

Further iterations refine the individual goal models of the stakeholders, as the focus is directed upon more of the detail. For instance, the desires expressed in the graph of Figure 6.8 has yet to offer sufficient detail to document the concept of `Assessment`. Whilst this graph shows that the `Student` must achieve a `Total_Mark` in excess of 40%, the components within that assessment are not articulated. Figure 6.19 shows the extra facets that are derived during modelling.

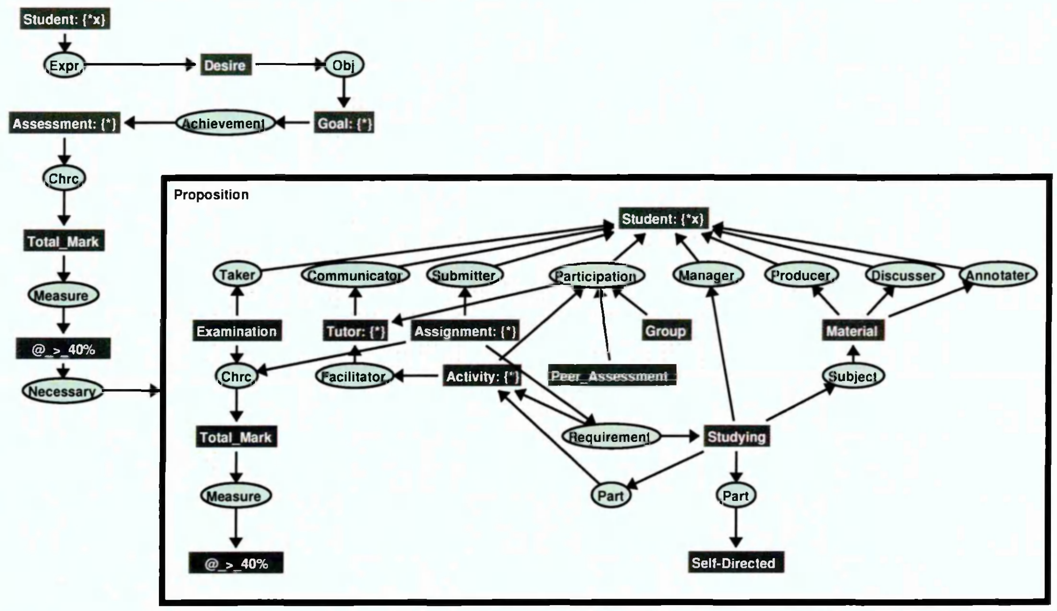


Figure 6.19: Refined model of Student desires.

The next stage is to perform some analysis upon the models to ascertain any inconsistencies, whilst also verifying the requirements gathered from the use cases.

Querying the Model

Using Peirce logic (described in Chapter 3) the models are queried and amended where necessary to take account of deficiencies in the modelling so far. This is performed by directing queries at the TM in the form of rules. For example:

The MOBIlearn system employs a pedagogical approach to facilitate mobile learning.

The linear form CG would be:

```
¬[[Learning: {*x}]->(Delivery)->[MOBIlearn]-
¬[[Mobile]<-(Chrc)<-[Learning: {*x}]-
->(Approach)->[Pedagogy]]].
```

The display form graph is shown in Figure 6.20. The following rules are further examples and are not an exhaustive list:

1. *Students may participate as individuals, as a member of a group or both.* Figure 6.21.

2. *All learning content must be administered and managed remotely.* Figure 6.22.

3. *The Local Authority pays for the education in full where it is deemed that the student is eligible.* Figure 6.23.

4. *The Student pays for the education in full where it is deemed that the Student is ineligible for financial assistance.* Figure 6.24.

5. *A Student may be eligible for financial assistance if they are female and between the ages of 18 and 65 years old.* Figure 6.25.

6. *The eligibility is determined by reference to current educational policy.* Figure 6.26.

As each rule is scrutinised, the TM can be appended with the new knowledge in order to specialise it further. This serves to establish the conditions required for a transaction to successfully occur, whilst building the required ontology of domain terms. In order to assess the viability of the model, it is then tested by using domain-specific situations. This stage is important as it assists the verification of consistency with the application domain, as shortcomings in the model are easier to elucidate with a concrete example.

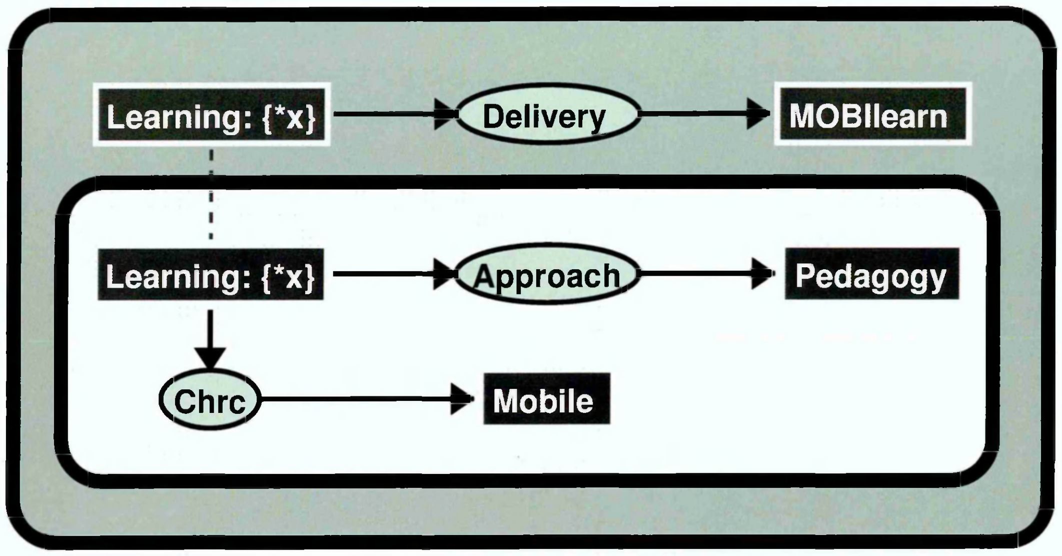


Figure 6.20: The MOBIlearn system employs a pedagogical approach to facilitate mobile learning.

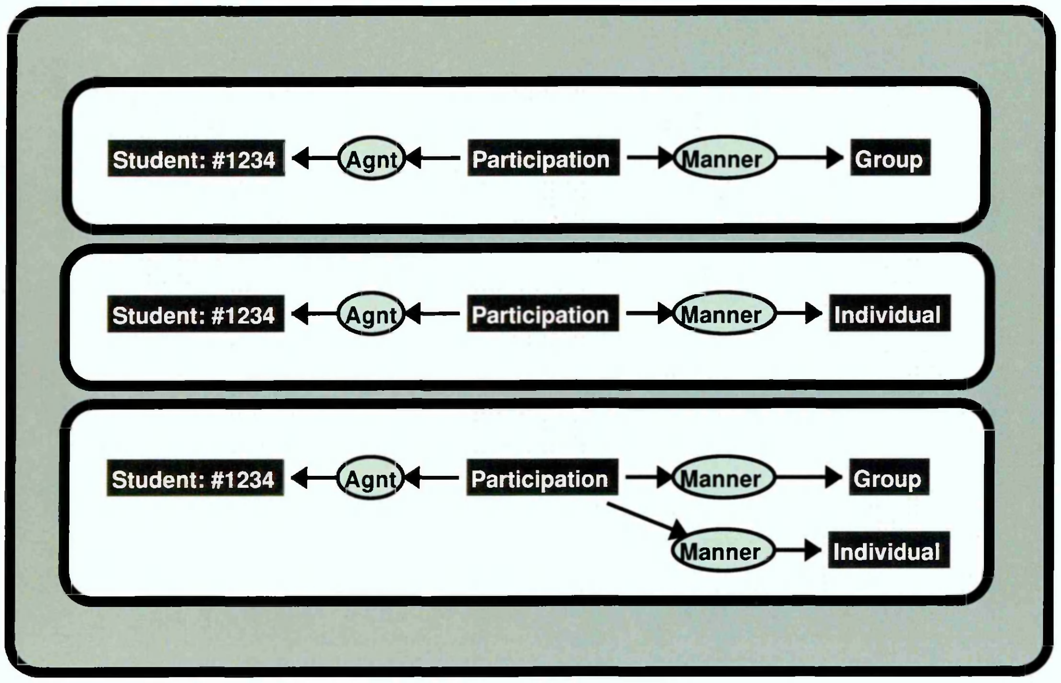


Figure 6.21: Students may participate as individuals, as a member of a group or both.

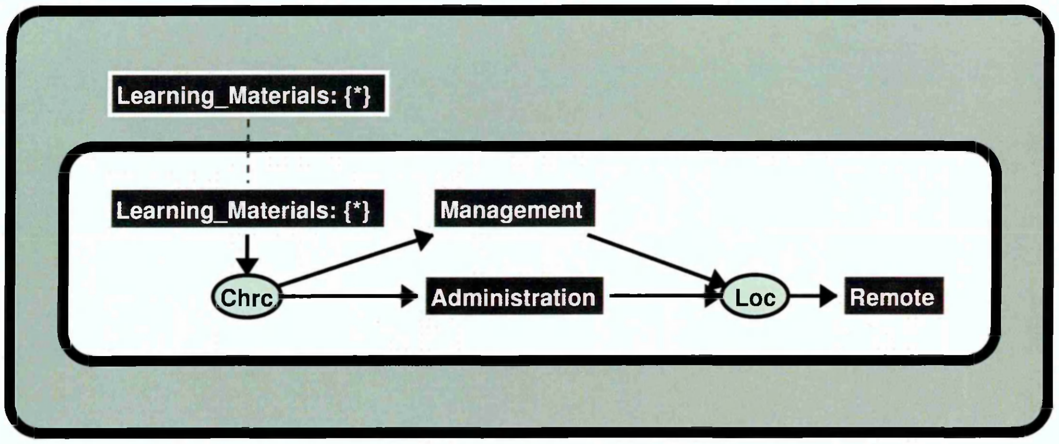


Figure 6.22: All learning content must be administered and managed remotely.

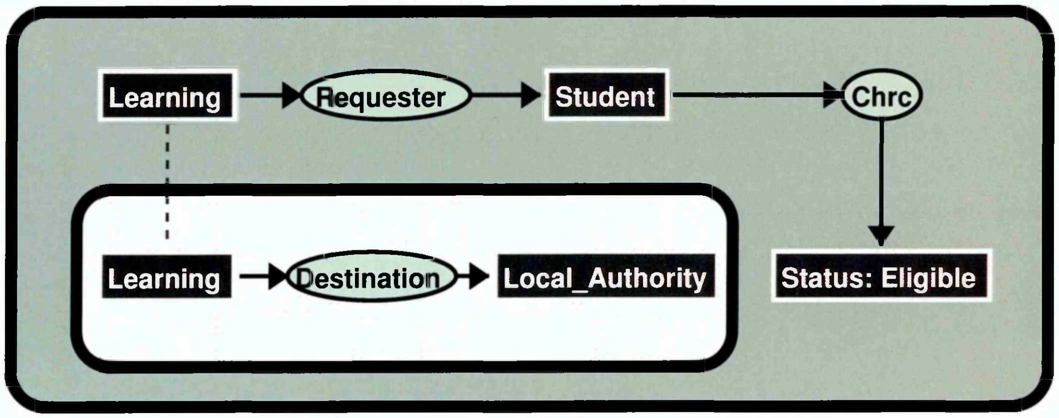


Figure 6.23: The Local Authority pays for the education in full where it is deemed that the student is eligible.

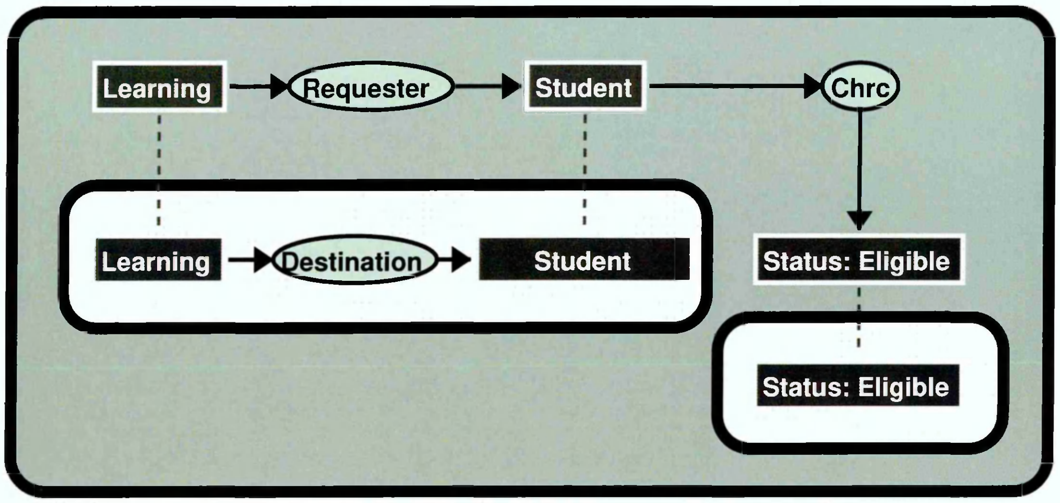


Figure 6.24: The Student pays for the education in full where it is deemed that the Student is ineligible for financial assistance.

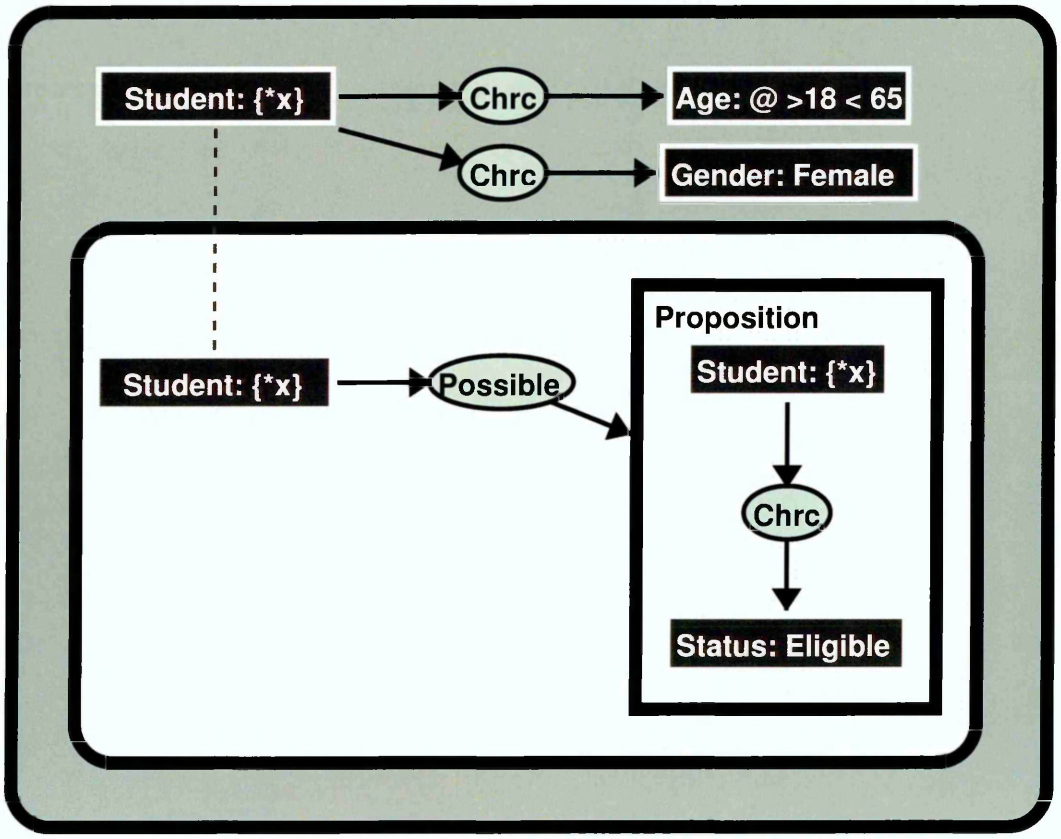


Figure 6.25: A Student may be eligible for financial assistance if they are female and between the ages of 18 and 65 years old.

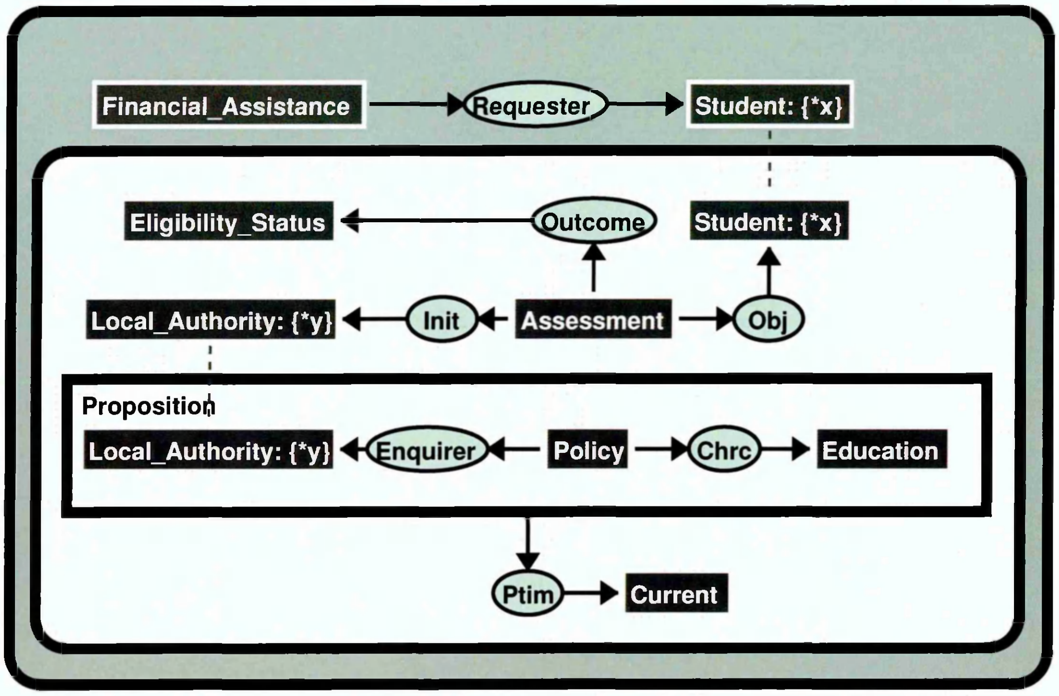


Figure 6.26: The eligibility is determined by reference to current educational policy.

6.3.5 Allocate Agents

Once the models have been checked, Agents are allocated to each of the roles that have been identified. These roles are summarised in Table 6.2. As before in the community healthcare exemplar (Chapter 4), each agent is now allocated tasks. For brevity only some of the tasks for the Student and Teacher agents are illustrated in Table 6.3. The last stage is to examine the interactions between the agents, in order to build a collaboration model. Once this has been completed, the design artefacts are ready for use by an existing agent design methodology such as Tropos or Gaia.

Agent Type	**Role**
Student Agent	The representation of the Student within the system.
Teacher Agent	The representation of the Teacher within the system.
Learning Provider (LP) Agent	This agent represents the provider of the learning environment to support work-based learning, in this case the MBA scenario.
Student Administrator (SA) Agent	Responsible for all aspects of student-related administration such as enrolment, processing of results, etc.
Local Authority (LA) Agent	The body that represents the local face of government, which may provide a means of assistance to the student in terms of learning facilities or financial support.
Presentation Agent	A role that manages the provision of learning content via different access mediums such as personal computers, personal digital assistants, tablet PCs and smartphone devices.
Learning Materials (LM) Agent	An information agent that marshalls learning materials repositories.
Schedule Agent	An agent that manages the provision of schedule information.
Student Records (SR) Agent	The agent that oversees the administration and management of student records.

Table 6.2: MOBIlearn agent roles.

6.4 Conclusions

This chapter has described the use of TrAM in the m-learning domain, and has incorporated the refinements introduced in Chapter 5. After producing a series of design artefacts that includes high-level conceptual models, a generic TM, a specialised TM for the MBA Scenario, query graphs (rules), together with the associated type hierarchies and OWL ontologies, the framework demonstrates how the criteria identified in Chapter 2 are addressed.

Agent Type	**Task**
Student Agent	Access learning materials. Take examination. Complete coursework. Manage materials. Transform communications into materials. Manage group work. Evaluate own performance. Find materials.
Teacher Agent	Produce learning materials. Transform communications into materials. Set coursework. Mark coursework. Moderate coursework marks. Create coursework marking scheme. Set examination. Mark examination. Moderate examination marking. Create examination marking scheme. Manage student groups. Moderate discussions. Evaluate learning performance.

Table 6.3: Task allocation for the Student Agent.

Chapter 7

Conclusions and Further Work

7.1 Hypothesis

The hypothesis of this research is: "Conceptual modelling is a useful activity for the early part of gathering requirements for agent-based information systems." For the purposes of this thesis, 'usefulness' is characterised by the following:

1. An opportunity to reduce the need for input from domain experts;
2. A means by which system models are tested earlier in the requirements capture process;
3. An ability to capture abstract domain terms as concepts;
4. The elicitation of an ontology that reflects the domain more faithfully;
5. An approach that complements other MAS design methodologies and;
6. An approach that is sufficiently abstract to be generally applicable in the wider context.

The use of TrAM has illustrated how high-level concepts can be captured in the community healthcare and m-learning domains, and demonstrates the process by which qualitative concepts are quantified and used to populate a hierarchy of types prior to ontology generation. From the earliest stage, concept types, relations and domain terms can be qualified with domain experts. TrAM offers the significant advantage of being able to focus in on areas that require concentrated analysis, thus guiding the agent system analyst, whilst also concentrating the efforts of the domain expert. The capture, representation and subsequent analysis of early requirements is also supported by TrAM, and since the framework explicitly supports BDI concepts the resulting design artefacts can be used as a precursor to detailed implementation with existing agent design methodologies. Finally, the TrAM approach conveniently uses a transaction metaphor that is sufficiently abstract to be domain independent. As such, it is established that conceptual modelling is a useful activity and therefore the hypothesis is believed to be true.

7.2 Research Approach

The choice of a case study approach might be contentious in some quarters since there is a view that case study research is only suitable for either pilot studies or for generating hypotheses (Abercrombie et al., 1984). For this research the approach has provided two significant advantages:

1. The use of case studies has enabled authentic, realistic models to be developed that capture context-specific details. Models based upon theory however, rely upon general rules that may apply in the wider domain, thereby restricting the depth to which a scenario can be explored.

2. The opportunity for learning from a scenario is maximised when the investigator is immersed within the particular case. Furthermore, the detailed examination of a specific scenario enables real-life issues to be captured and included within a model. In particular the consideration of social interaction, which is a characteristic of a multi-agent environment, requires deep understanding. Such understanding is difficult to achieve from general theories.

Upon reflection the use of a case study to develop the framework in Chapter 4 enabled the 'nuances' of a real-life situation to be considered. An additional benefit was the assistance of domain expertise available when problems inevitably occurred. Such expertise aided verification of the process steps, particularly when an attempt was made to establish the most appropriate sequence of activities. Indeed the subsequent development of the framework in Chapter 5 was underpinned by prior detailed work upon an exemplar. Subsequent work with the second case study (Chapter 6) enabled the framework and its process to be refined further, facilitating the test of a problematic domain which contains many qualitative aspects.

One difficulty encountered during the research was the process of summarising the results. It is tempting to seek generalisations from the specific scenarios, and to expect that the results will somehow be validated by increasing the number of cases introduced. Rather than producing a large data set in an attempt to summarise the cases, the ensuing process required to generate the models was abstracted away from the domain-specific detail of the scenario. As such the TrAM Framework describes the process, whereas each case describes an instance of a real-life scenario for an agent-based system.

Since deeper understanding will be developed by applying TrAM to other domains, the selection of new cases is very important. In fact case study choice can have a significant impact on the ability to generalise results. Thus, further work to develop and refine TrAM must consider scenarios that have the potential to polarise a result; cases that are either likely to support or falsify a particular hypothesis.

7.3 Contributions

In summary, the primary contributions of this research are as follows:

1. *Use of the Transaction Model to impose a rigour upon the requirements elicitation process for agent-based systems.* The respected Event Accounting model of Geerts and McCarthy (1991) has been utilised as a metaphor for the design of an agent based system. The TM is used as a business metaphor to elicit the pertinent qualitative concepts and assist the agent system designer discover quantitative metrics for the implementation, and introduces a balance check in order that the conceptual models are checked prior to further analysis. The TM graph provides the guidance necessary for the TrAM framework, permitting rich modelling activity, yet within the constraints of a suitable organisational representation. In particular, the work of Polovina (1993) has been extended to include BDI concepts. The TM has been translated into a generic ontology, and specialisations have produced domain specific ontologies for community healthcare payment systems and an m-learning scenario, using OWL.

2. *Use of Conceptual Graphs type hierarchies for ontology construction.*

Each CG has an associated hierarchy of types. Use of the TM enables a rudimentary ontology to be created much earlier than with other agent design approaches, which is used in conjunction with more iterations of the TM to refine the domain terms and their relations. In particular, the TM promotes the verification of domain terms, specifically when ambiguous qualitative concepts exist, and its use has demonstrated how new terms and revised relationships were derived. The CG notation provides sufficient abstraction to be able to model at the societal level.

3. *A means to check the transaction models using graphical inferencing with Peirce Logic.* TrAM offers three aspects of model checking:

    (a) TM balance check - TM models remain incomplete until the transaction is satisfied.

    (b) Consistency check - Type hierarchies and NL parsing enable the TM to be verified in terms of the suitability of domain terms, and the associated super/sub type relations. This work can also be conducted with a domain expert if required.

    (c) Graph querying with Peirce logic - Once the generic TM has been populated, specific scenarios can be modelled and used as test queries for the TM. This checks the suitability of the model whilst also deriving new knowledge, resulting in a more specialised model.

    Since the models can be queried graphically with IF-THEN rules, including AND/OR reasoning where applicable - this can be used to demonstrate the behaviours of the system being modelled and check whether the intended specifications will be met.

4. *Providing a method for the elicitation and decomposition of soft goals.* Typically, guidance for agent requirements capture is limited to *identify stakeholders*, *identify goals*, and then suggest that a goal hierarchy is produced. The resulting AND/OR decomposition can derive *hard* goals from *soft* goals, but this does assume that the *identify goals* activity has been sufficiently comprehensive. TrAM improves upon this by ensuring that:

   (a) Goal names are correctly defined - the goal must fit with the rest of the model and it must describe the concept accurately, in a way that is commonly understood (for the type hierarchy and subsequent ontology).

   (b) The TM metaphor enables these high level goals to be scrutinised within the discipline of a particular graph. Balanced models that contain goals which are too abstract cannot be realised until the concepts are grounded.

   TrAM provides more guidance at the beginning of the requirements capture process and provides mechanisms for the capture and analysis of system goals.

5. *The TrAM process for agent system requirements elicitation.* The TrAM process forces concepts to be considered so that the models can be completed. It may be possible to populate the TM with a particular concept name, and though the hierarchy of types is completed, the concept may still be too abstract or qualitative. Effort is then focused upon this concept, representing the term in a measurable, quantitative way.

Furthermore the early requirements capability of TrAM enables methodologies without this capability to be extended, improving discovery of stakeholders and qualitative goals. Models produced with TrAM introduce more formality at the outset by specifying a notation for the capture of requirements. For instance, Tropos may produce goals such as *enjoy visit* and *provide cultural service*. The equivalent in TrAM would be:

```
[Visit]->(Exp)->[State: Enjoy].
[Provide]->(Obj)->[Service]->(Chrc)->[Cultural].
```

This simple example demonstrates how the TrAM approach specifies domain terms and relationships at the earliest opportunity, enabling ontologies to be built iteratively, in conjunction with the modelling and analysis activities. Since TrAM can be used at the highest levels of abstraction, it is possible to include the *what* and *who* questions for stakeholders and goals, and also the *why*. Why is this a relevant issue? Why does the stakeholder regard this goal as important? Why is this policy in place?

7.4 Further Work

The key areas for further work as a result of this research are:

- *Automation.* Much of this research has exploited the transformation of one formalism to another and as such there is much work to be done with regard to the automation of these repetitive tasks. One particular candidate is the automation of the Peirce logic inferencing, which may present difficulties with its intended audience; as the author's experience with postgraduate and final year undergraduate students illustrates, it can be challenging to comprehend. An alternative approach might be to simplify the model querying stage by utilising graph projections only; business and ontology rules would be built up by specialisations and then graphs would be projected until a desired set of conditions is obtained. This makes the process of graph specialisation much easier for potential users, since rule building is a convenient metaphor for domain experts and it is therefore a primary research activity for the future. Tool support is already available for some elements of the TrAM process (Charger, (Delugach, 2006a), Protégé, (Stanford Medical Informatics, 2006)), but there needs to be better interoperability of tools for the process to be mechanised. This would improve consistency, and also enable measures of consistency to take place.

- *Metamodels.* The flexibility of the CG notation is such that it is possible to build models that stray from the TrAM process, and it may be useful to supplement the framework with a metamodel or collection of metamodels that describe a variety of abstractions such as organisation and society.

- *Patterns.* A series of design patterns may emerge that represent some of the more convoluted organisational transactions, assisting the system analyst compile a solution from tested solutions to common problems. In particular, it is feasible that domain specific patterns may emerge, supported by a relevant ontology.

- *Semantic interoperability.* This thesis has not considered the need for semantic representations in agent communication languages, but the development of a framework that can capture and create ontological representations from a domain means that this is an area worthy of exploration.

The use of a controlled language (or at least a defined vocabulary) can assist the mapping of graphs to agent models and it would be useful to examine the extent to which such a vocabulary would be beneficial. The first stage is to define some simple semantics that can be applied to the graphs. The use of `(Agnt)` and `(Obj)` relations helps define graph concepts considerably. The second stage is to incorporate interaction protocols into the framework, by concentrating on more of the detailed agent design. This activity may be provided by an existing agent design approach. The next stage is then to investigate the communication semantics demanded by the BDI approach and then provide a means by which these can be generated.

References

Aalborg University (2006). "Online Course in Knowledge Representation using Conceptual Graphs". Online. Accessed 30th July 2006.
URL http://www.huminf.aau.dk/cg/

Abercrombie, N., Hill, S., and Turner, B. S. (1984). *Dictionary of Sociology.* Penguin, 3rd edition.

Alsinet, T., Bjar, R., Fernanadez, C., and Many, F. (2000). "A Multi-Agent System Architecture for Monitoring Medical Protocols". In C. Sierra, M. Gini, and J. Rosenschein, eds., "Proceedings of the Fourth International Conference on Autonomous Agents", pp. 499–505. ACM-AAAI, ACM Press. ISBN 1-58113-230-1.

Andronache, V. and Scheutz, M. (2004). "Integrating Theory and Practice: The Agent Architecture Framework APOC and its Development Environment ADE". In "Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)", ACM Press, New York, USA.

Ardissono, L., Goy, A., and Petrone, G. (2003). "Enabling Conversations with Web Services". In "Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)", pp. 819–826. ACM Press. ISBN 1-58113-683-8.

Austin, J. L. (1962). *How To Do Things With Words.* Oxford University Press, Oxford.

Bauer, B. (2001). "UML Class Diagrams and Agent-Based Systems". In "Proceedings of the Fifth International Conference on Autonomous Agents", pp. 104–105. ACM Press, Montreal, Quebec, Canada. ISBN 1-58113-326-X.

Bauer, B., Müller, J., and Odell, J. (2001). "Agent UML: A Formalism for Specifying Multi-Agent Interaction". In Ciancarini and Wooldridge, eds., "Agent-Oriented Software Engineering", volume 1957, pp. 91–103. Springer-Verlag.

Beer, M. D., Anderson, I., and Huang, W. (2001). "Using Agents to Build a Practical Implementation of the INCA (Intelligent Community Alarm) System". In "Proceedings of the Fifth International Conference on Autonomous Agents", pp. 106–107. ACM Press, Montreal, Quebec, Canada. ISBN 1-58113-326-X.

Beer, M. D., Bench-Capon, T., and Sixsmith, A. (1999). "Some Issues in Managing Dialogues between Information Agents". In "Proceedings of Database and Expert Systems Applications '99", volume 1677 of *Lecture Notes in Computer Science (LNCS)*, pp. 521–530. Springer, Berlin.

Beer, M. D. and Hill, R. (2004). "Teaching Multi-Agent Systems in a UK New University". In "Teaching Multi-agents Workshop: Proceedings of the

Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)", IEEE Computer Society, New York, NY, USA. ISBN 1-58113-864-4.

Beer, M. D. and Hill, R. (2005). "Integrating Multi-Agent Systems into the Wider Computing Curriculum". In "Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AA-MAS) - Teaching Agents Workshop", Autonomous Agents and Multi-Agent Systems, Utrecht, The Netherlands.

Beer, M. D. and Hill, R. (2006a). "Building a Community Care Demonstrator with JADE Semantic Agents". *International Transactions on Systems Science and Applications*, volume 1(1):pp. 1–14. ISSN 1751-1461.

Beer, M. D. and Hill, R. (2006b). "Building a Community Care Demonstrator with Semantic Agents". In "Proceedings of the Second International Workshop on Multi-Agent Systems for Medicine, Computational Biology, and Bioinformatics (MAS*BIOMED2006)", Future University, Hakodate, Japan.

Beer, M. D., Hill, R., Huang, W., and Sixsmith, A. (2002). "Using Agents To Promote Effective Coordination In A Community Care Environment". In Y. Ye and E. F. Churchill, eds., "Agent Supported Collaborative Work", chapter 3, pp. 53–77. Kluwer Academic Publishers. ISBN 1-4020-7404-2.

Beer, M. D., Hill, R., and Sixsmith, A. (2003a). "Building an Agent-Based Community Care Demonstrator on a Worldwide Agent Platform". In "Agents and Healthcare", Multi-Agent Systems, pp. 19–34. Whitestein.

Beer, M. D., Hill, R., and Sixsmith, A. (2003b). "Deploying an Agent-Based

Architecture for the Management of Community Care". In "Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)", pp. 932–933. ACM Press, Melbourne, Victoria, Australia. ISBN 1-58113-683-8.

Bellifemine, F., Poggi, A., and Rimassa, G. (2001). "JADE: a FIPA2000 Compliant Agent Development Environment". In "Proceedings of the Fifth International Conference on Autonomous Agents", pp. 216–217. ACM Press, Montreal, Quebec, Canada. ISBN 1-58113-326-X.

Bergenti, F. and Poggi, A. (2001). "A Development Toolkit to Realize Autonomous and Interoperable Agents". In "Proceedings of the Fifth International Conference on Autonomous Agents", pp. 632–639. ACM Press, Montreal, Quebec, Canada. ISBN 1-58113-326-X.

Bergenti, F., Poggi, A., Rimassa, G., and Turci, P. (2002). "CoMMA: a Multi-Agent System for Corporate Memory Management". In "Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)", pp. 1039–1040. ACM Press, Bologna, Italy. ISBN 1-58113-480-0.

Berners-Lee, T. (1999). *Weaving the Web.* Orion Business, London.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004). "TROPOS: An Agent-Oriented Software Development Methodology". *Journal of Autonomous Agents and Multi-Agent Systems*, volume 8:pp. 203–236.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2001). "A Knowledge Level Software Engineering Methodology for

Agent-Oriented Programming". In J. P. Müller, E. Andre, S. Sen, and C. Frasson, eds., "Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS-01)", pp. 648–655. ACM Press, Montreal, Quebec, Canada. ISBN 1-58113-326-X.

Bretier, P. and Sadek, D. (1997). "A Rational Agent as the Kernel of a Cooperative Spoken Dialogue System: Implementing a Logical Theory of Interaction". In J. P. Müller, M. Wooldridge, and N. R. Jennings, eds., "Intelligent Agents III", volume 1193 of *Lecture Notes in Artificial Intelligence (LNAI)*, pp. 189–204. Springer, Berlin.

Busetta, P., Ronnquist, R., Hodgson, A., and Lucas, A. (1999). "JACK - Components for Intelligent Agents in Java". Technical Report 1, Agent-Oriented Software Pty. Ltd., Melbourne, Australia.
URL `http://www.agent-software.com`

Castelfranchi, C. (1998). "Modelling Social Action for AI Agents". *Artificial Intelligence*, volume 103(1).

Cernuzzi, L., Juan, T., Sterling, L., and Zambonelli, F. (2004). "The Gaia Methodology". In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, eds., "Methodologies and Software Engineering for Agent Systems", chapter 4, pp. 69–88. Kluwer Academic Publishers. ISBN 1-40208-057-3.

Chopra, A. K. and Singh, M. P. (2004). "Commitments for Flexible Business Protocols". In "Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)", pp. 1360–1361. IEEE Computer Society, New York, NY, USA. ISBN 1-58113-864-4.

Cohen, P. R. and Levesque, H. J. (1990). *Rational Interaction as the Basis for Communication.* MIT Press.

Compton, P. and Jansen, R. (1990). "A Philosophical Basis for Knowledge Acquisition". *Knowledge Acquisition*, volume 2:pp. 241–257.

Compton, P., Peters, L., Edwards, G., and Lavers, T. (2006). "Experience with Ripple-Down Rules". In A. Macintosh, R. Ellis, and T. Allen, eds., "Applications and Innovations in Intelligent Systems XIII: Proceedings of the Twenty-fifth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence", pp. 109–121. Springer, Cambridge, UK. ISBN 1-84628-223-3.

DAML (2001). "The DARPA agent markup language". Online. URL `http://www.daml.org`

Dardenne, A., van Lamsweerde, A., and Flickas, S. (1993). "Goal-Directed Requirements Acquisition". *Science of Computer Programming*, volume 20(1-2):pp. 3–50.

Dasgupta, P. R. and Hashimoto, Y. (2004). "Multi-Attribute Dynamic Pricing for Online Markets Using Intelligent Agents". In "Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)", IEEE Computer Society, New York, NY, USA.

Dastani, M. (2004). "Programming Multi-Agent Systems". Technical forum group meeting, AgentLink-III, Rome, Italy.

Dastani, M., de Boer, F., Dignum, F., and Meyer, J.-J. (2003a). "Programming Agent Deliberation: An Approach Illustrated Using the 3APL Language".

In “Proceedings of the second international joint conference on Autonomous Agents and Multi-Agent systems”, pp. 97–104. ACM Press. ISBN 1-58113-683-8.

Dastani, M., Dignum, V., and Dignum, F. (2003b). “Role-Assignment in Open Agent Societies”. In “Proceedings of the second international joint conference on Autonomous Agents and Multi-Agent systems”, pp. 489–496. ACM Press. ISBN 1-58113-683-8.

Dastani, M., Hulstjn, J., Dignum, F., and Meyer, J.-J. C. (2004). “Issues in Multi-Agent System Development”. In “Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)”, pp. 920–927. ACM Press, New York, New York, USA.

Dau, F. (2003). *The Logic System of Concept Graphs with Negation and its Relationship to Predicate Logic*, volume 2892 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg.

DeLoach, S. A. (1999). “Multi-Agent Systems Engineering: A Methodology and Language for Designing Agent Systems”. In “Proceedings of Agent-Oriented Information Systems”, pp. 45–57.

DeLoach, S. A. and Wood, M. (2000). “Developing Multi-Agent Systems with AgentTool”. In “Intelligent Agents VII. Agent Theories, Architectures and Languages. 7th International Workshop 2000, Boston USA.”, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

Delugach, H. (2006a). “CharGer - Conceptual Graph Editor”. Online. Accessed 30th July 2006.
URL `http://sourceforge.net/projects/charger/`

Delugach, H. (2006b). "Common Logic Standard". Online.
URL `http://cl.tamu.edu/`

Depke, R., Heckel, R., and Kuster, J. M. (2001). "Improving the Agent-Oriented Modeling Process by Roles". In "Proceedings of the fifth international conference on Autonomous agents", pp. 640–647. ACM Press. ISBN 1-58113-326-X.

Dickinson, I. and Wooldridge, M. (2003). "Towards Practical Reasoning Agents for the Semantic Web". In "Proceedings of the second international joint conference on Autonomous agents and Multi-Agent Systems", pp. 827–834. ACM Press. ISBN 1-58113-683-8.

Do, T., Kolp, M., and Pirotte, A. (2003). "Social Patterns for Designing Multi-Agent Systems". In "Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03)", .

Ehrler, L. and Cranefield, S. (2004). "Executing Agent UML Diagrams". In "Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)", pp. 904–911. ACM Press, New York, New York, USA.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic.* Academic Press, London.

Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., and Arcos, J. L. (2004). "AMELI: An Agent-Based Middleware for Electronic Institutions". In "Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)", New York, USA.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M., McKay, D., Fritzson, R., Shapiro, S., Pelavin, R., and McGuire, J. (1993). "Specification of the KQML Agent-Communication Language – plus example agent policies and architectures".

FIPA (1999). "Specification part 2 - agent communication language".

FIPA (2002). "FIPA Contract Net Interaction Protocol Specification". Online. Las accessed 31st August 2006.
URL `http://www.fipa.org/specs/fipa00029/`

FIPA (2006). "FIPA Agent Communication Language Specification". Online. Last accessed 10th June 2006.
URL `http://www.fipa.org/repository/aclspecs.html`

Fuxman, A., Kazhamiakin, R., and Pistore, M. (2004). "Formal Tropos: language and semantics".

Fuxman, A., Pistore, M., Mylopoulas, J., and Traverso, P. (2001). "Model Checking Early Requirements Specifications in Tropos". In "Proceedings of the 9th IEEE International Requirments Engineering Conference", IEEE, IEEE, Toronto, Canada.

Garcia-Ojeda, J. C. and Arenas, A. E. (2004). "Extending the Gaia Methodology with Agent-UML". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", ACM Press, New York, New York, USA.

Geerts, G. L. and McCarthy, W. E. (1991). "Database Accounting Systems". In B. Williams and B. J. Sproul, eds., "Information Technology Perspectives in Accounting: and Integrated Approach", pp. 159–183. Chapman and Hall.

Geerts, G. L. and McCarthy, W. E. (1997). "Modelling Business Enterprises as Value-Added Process Hierarchies with Resource-Event-Agent Object Templates". In J. Sutherland and D. Patel, eds., "Business Object Design and Implementation", pp. 94–113. Springer-Verlag.

Geerts, G. L., McCarthy, W. E., and Rockwell, S. R. (1996). "Automated Integration of Enterprise Accounting Models throughout the Systems Development Life Cycle". *Intelligent Systems in Accounting, Finance and Management*, volume 5:pp. 113–128.

Genesereth, M. R. and Fikes, R. E. (1992). "Knowledge Interchange Format". Version 3.0 Reference Manual logic-92-1, Computer Science Department, Stanford University.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (1999). "The Belief-Desire-Intention Model of Agency". In J. Müller, M. P. Singh, and A. S. Rao, eds., "Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)", volume 1555, pp. 1–10. Springer-Verlag: Heidelberg, Germany.

Giorgini, P. (2003). "Agent-Oriented Software Engineering Report on the 4th AOSE Workshop (AOSE 2003)". *SIGMOD Rec.*, volume 32(4):pp. 117–119. ISSN 0163-5808.

Giorgini, P., Perini, A., Mylopoulos, J., Giunchiglia, F., and Bresciani, P. (2001). "Agent-Oriented Software Development: A Case Study". In "In Proc. of the 13th Int. Conference on Software Engineering & Knowledge Engineering (SEKE01)", Buenos Aires, Argentina.

Giovannucci, A., Rodriguez-Aguilar, J. A., Reyes, A., Noria, F. X., and

Cerquides, J. (2004). "Towards Automated Procurement via Agent-Aware Negotiation Support". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", ACM Press, New York, USA.

Griffiths, N. and Luck, M. (2003). "Coalition Formation Through Motivation and Trust". In "Proceedings of the second international joint conference on Autonomous Agents and Multi-Agent systems", pp. 17–24. ACM Press. ISBN 1-58113-683-8.

Gruber, T. (1993). "A Translation Approach to Portable Ontologies". *Knowledge Acquisition*, pp. 199–220.

Grüninger, M. and Fox, M. S. (1994). "The Role of Competency Questions in Enterprise Engineering". In "Proceedings IFIP WG5.7 Workshop on Benchmarking - Theory and Practice", Trondheim, Norway.

Guessoum, Z., Ziane, M., and Faci, N. (2004). "Monitoring and Organizational-Level Adaptation of Multi-Agent Systems". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", pp. 514–521. ACM Press, New York, New York, USA.

Haigh, K. Z., Phelps, J., and Geib, C. W. (2002). "An open agent architecture for assisting elder independence". In "Proceedings of the first international joint conference on Autonomous agents and Multi-Agent systems", pp. 578–586. ACM Press. ISBN 1-58113-480-0.

Haley, D., Nuseibeh, B., Sharp, H. C., and Taylor, J. (2004). "The Conundrum of Categorising Requirements: Managing Requirements for Learning on the

Move." In "12th IEEE International Conference on Requirements Engineering (RE 2004), Kyoto, Japan.", pp. 309–314. IEEE Computer Society 2004, ISBN 0-7695-2174-6.

Harper, L. and Delugach, H. S. (2003). "Using Conceptual Graphs to Capture Semantics of Agent Communication". In A. de Moor, W. Lex, and B. Ganter, eds., "Conceptual Structures for Knowledge Creation and Communication: Proceedings of the 11th International Conference on Conceptual Structures (ICCS 2003)", volume 2746, pp. 392–404. Springer-Verlag.

Heaton, J. E. and Kocura, P. (1993). "Presenting a Pierce Logic Based Inference Engine and Theorem Prover for Conceptual Graphs". In "ICCS '93: Proceedings on Conceptual Graphs for Knowledge Representation", pp. 381–400. Springer-Verlag, London, UK. ISBN 3-540-56979-0.

Hendler, J. (2001). "Agents and the Semantic Web". *IEEE Intelligent Systems*, volume 16(2):pp. 30–37.

Hill, R. (2007). "Capturing and Specifying Multi-Agent System Requirements for Community Healthcare". In H. Yoshida, A. Jain, A. Ichalkaranje, L. Jain, and N. Ichalkaranje, eds., "Advanced Computational Intelligence Paradigms in Healthcare", volume 48 of *Studies in Computational Intelligence*, chapter 6, pp. 121–158. Springer-Verlag.

Hill, R., Polovina, S., and Beer, M. D. (2004). "Towards a Deployment Framework for Agent-Managed Community Healthcare Transactions". In "The Second Workshop on Agents Applied in Healthcare, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)", pp. 13–21. ECCAI, IOS Press, Valencia, Spain.

Hill, R., Polovina, S., and Beer, M. D. (2005a). "From Concepts to Agents: Towards a Framework for Multi-Agent System Modelling". In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, eds., "Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 05)", pp. 1155–1156. ACM Press, Utrecht, The Netherlands. ISBN 1-59593-093-0.

Hill, R., Polovina, S., and Beer, M. D. (2005b). "Managing Community Healthcare Information in a Multi-Agent System Environment". In "Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) - BIOMED Workshop", pp. 35–49. Utrecht University, Netherlands.

Hill, R., Polovina, S., and Beer, M. D. (2005c). "Managing Healthcare Workflows in a Multi-Agent System Environment". In "Proceedings of the Third Workshop on Agents Applied in Healthcare, International Joint Conference on Artificial Intelligence (IJCAI)", IJCAI, Edinburgh.

Hill, R., Polovina, S., and Beer, M. D. (2006a). "Improving AOSE with an Enriched Modelling Framework". In J. Müller and F. Zambonelli, eds., "Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering (AOSE-2005)", volume 3859 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Utrecht, The Netherlands.

Hill, R., Polovina, S., and Shadija, D. (2006b). "Transaction Agent Modelling: From Experts to Concepts to Multi-Agent Systems". In "Proceedings of the Fourteenth International Conference on Conceptual Structures (ICCS '06): Conceptual Structures: Inspiration and Application", volume 4068 of *Lecture Notes in Artificial Intelligence (LNAI)*, pp. 247–259. Springer-Verlag, Aalborg, Denmark.

Hirsch, T., Forlizzi, J., Hyder, E., Goetz, J., Kurtz, C., and Stroback, J. (2000). "The ELDer Project: Social, Emotional, and Environmental Factors in the Design of Eldercare Technologies". In "Proceedings of the 2000 Conference on Universal Usability", pp. 72–79. ACM Press. ISBN 1-58113-314-6.

Huang, I., Jennings, N. R., and Fox, J. (2001). "An Agent-Based Approach to Healthcare Management". *International Journal of Applied Artificial Intelligence*, volume 9:pp. 173–184.

Huang, W., Beer, M. D., and Hill, R. (2003). "Community Care System Design and Development with AUML". In "Proceedings of the 9th International Conference on Information Systems Analysis and Synthesis (ISAS '03)", Orlando, Florida, USA.

Huget, M.-P. and Odell, J. (2004). "Representing Agent Interaction Protocols with Agent UML". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", ACM Press, New York, USA.

Ijiri, Y. (1967). *The Foundations of Economic Accounting.* Prentice Hall, Englewood Cliffs, NJ.

Jennings, N. R. (2000). "On Agent-Based Software Engineering". *Artificial Intelligence*, volume 117:pp. 277–296.

Jennings, N. R. (2001). "An Agent-Based Approach for Building Complex Software Systems". *Communications of the ACM*, volume 44(4):pp. 35–41. ISSN 0001-0782.

Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Wooldridge, M. J., and Sierra, C. (2001). "Automated Negotiation: Prospects, Methods and Challenges". *Group Decision and Negotiation*, volume 10(2):pp. 199–215.

Juan, T., Pearce, A., and Sterling, L. (2002). "ROADMAP: extending the gaia methodology for complex open systems". In "Proceedings of the First ACM International Joint Conference on Autonomous Agents and Multi-Agent Systems", pp. 3–10. ACM Press. ISBN 1-58113-480-0.

Klugl, F., Bazzan, A. L. C., and Wahle, J. (2003). "Selection of Information Types Based on Personal Utility: a Testbed for Traffic Information Markets". In "Proceedings of the second international joint conference on Autonomous agents and Multi-Agent systems", pp. 377–384. ACM Press. ISBN 1-58113-683-8.

Kurbel, K. and Loutchko, I. (2003). "Towards Multi-Agent Electronic Marketplaces: What is There and What is Missing?" *The Knowledge Engineering Review*, volume 18(1):pp. 33–46.

Labrou, Y., Finin, T., and Peng, Y. (1999). "Agent Communication Languages: the Current Landscape". *IEEE Intelligent Systems*, volume 14(2):pp. 45–52.

Luck, M., McBurney, P., and Preist, C. (2004). "A Manifesto for Agent Technology: Towards Next Generation Computing". *Autonomous Agents and Multi-Agent Systems*, volume 9(3):pp. 203–252.

Massonet, P., Deville, Y., and Nave, C. (2002). "From AOSE Methodology to Agent Implementation". In "Proceedings of the first international joint conference on Autonomous agents and Multi-Agent systems", pp. 27–34. ACM Press. ISBN 1-58113-480-0.

Mavetera, N. and Kadyamatimba, A. (2003). "A Comprehensive Agent: Mediated e-Market Framework". In "Proceedings of the 5th international conference on Electronic commerce", pp. 158–164. ACM Press. ISBN 1-58113-788-5.

Mayfield, J., Labrou, Y., and Finin, T. (1996). "Evaluation of KQML as an Agent Communication Language". In M. Wooldridge, J. P. Müller, and M. Tambe, eds., "Proceedings on the IJCAI Workshop on Intelligent Agents II : Agent Theories, Architectures, and Languages", volume 1037, pp. 347–360. Springer-Verlag: Heidelberg, Germany. ISBN 3-540-60805-2.

Mazouzi, H., Seghrouchni, A. E. F., and Haddad, S. (2002). "Open Protocol Design for Complex Interactions in Multi-Agent Systems". In "Proceedings of the first international joint conference on Autonomous agents and Multi-Agent systems", pp. 517–526. ACM Press. ISBN 1-58113-480-0.

McCarthy, W. E. (1979). "An Entity-Relationship View of Accounting Models". *The Accounting Review*, pp. 667–686.

McCarthy, W. E. (1982). "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment". *The Accounting Review*, pp. 554–578.

McCarthy, W. E. (1999). "Semantic Modeling in Accounting Education, Practice and Research: Some Hierarchies with Resource-Event-Agent Object Templates". In P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, eds., “Conceptual Modeling: Current Issues and Future Directions”, pp. 144–153. Springer-Verlag.

Mellouli, S., Mineau, G. W., and Pascot, D. (2002). “The integrated modeling of Multi-Agent systems and their environment”. In “Proceedings of the first international joint conference on Autonomous agents and Multi-Agent systems”, pp. 507–508. ACM Press. ISBN 1-58113-480-0.

Microsoft (2004). “Microsoft .NET home page”. www.
URL `http://www.microsoft.com/net/`

Moreno, A. and Isern, D. (2002). “A first step towards providing health-care agent-based services to mobile users”. In “Proceedings of the first international joint conference on Autonomous agents and Multi-Agent systems”, pp. 589–590. ACM Press. ISBN 1-58113-480-0.

Mouratidis, H., Giorgini, P., and Manson, G. (2003). “Modelling secure Multi-Agent systems”. In “Proceedings of the second international joint conference on Autonomous agents and Multi-Agent systems”, pp. 859–866. ACM Press. ISBN 1-58113-683-8.

Munroe, S., Luck, M., and d’Inverno, M. (2003). “Towards a motivation-based approach for evaluating goals”. In “Proceedings of the second international joint conference on Autonomous agents and Multi-Agent systems”, pp. 1074–1075. ACM Press. ISBN 1-58113-683-8.

Mynatt, E. D., Rowan, J., Craighill, S., and Jacobs, A. (2001). “Digital family portraits: supporting peace of mind for extended family members”. In “Proceedings of the SIGCHI conference on Human factors in computing systems”, pp. 333–340. ACM Press. ISBN 1-58113-327-8.

Ndumu, D. T., Nwana, H. S., Lee, L. C., and Collis, J. C. (1999). "Visualising and debugging distributed Multi-Agent systems". In "Proceedings of the third annual conference on Autonomous Agents", pp. 326–333. ACM Press. ISBN 1-58113-066-X.

Nguyen, T. D. and Jennings, R., Nicholas (2004). "Coordinating multiple concurrent negotiations". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", ACM Press, New York, USA.

Nilsson, N. (1971). *Problem Solving Methods in Artificial Intelligence.* McGraw Hill.

Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). "ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems". *Applied Artificial Intelligence Journal*, volume 13(1):pp. 129–186.

Nwana, H. S., Rosenschein, J., Sandholm, T., Sierra, C., Maes, P., and Guttmann, R. (1998). "Agent-Mediated Electronic Commerce: Issues, Challenges and Some Viewpoints". In "Proceedings of the second international conference on Autonomous agents", pp. 189–196. ACM Press. ISBN 0-89791-983-1.

Odell, J., Parunak, H. V. D., and Bauer, B. (2001). "Representing Agent interaction protocols in UML". *Agent-Oriented Software Engineering - Proceedings of the First International Workshop AOSE-2000, LNCS*, volume 1957:pp. 121–140.

OMG, O. M. G. (2005). "UML Resource Page".
URL `http://www.uml.org`

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004). "Environment-based coordination for intelligent agents". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", ACM Press, New York, USA.

Padgham, L., Thangarajah, J., and Winikoff, M. (2005). "Tool Support for Agent Development using the Prometheus Methodology." In "QSIC", pp. 383–388.

Padgham, L. and Winikoff, M. (2002). "Prometheus: A Methodology for Developing Intelligent Agents". In "Proceedings of the Third International Workshop on Agent-Oriented Software Engineering", AAMAS.

Pan, J., Cranefield, S., and Carter, D. (2003). "A lightweight ontology repository". In "Proceedings of the second international joint conference on Autonomous agents and Multi-Agent systems", pp. 632–638. ACM Press. ISBN 1-58113-683-8.

Patil, R., Fikes, R. F., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T., and Neches, R. (1992). "The DARPA Knowledge Sharing Effort: Progress Report". In B. Nebel, C. Rich, and W. Swartout, eds., "Proceedings of the Third International Conference on Knowledge Representation, KR'92. Principles of Knowledge Representation and Reasoning", pp. 777–788. Morgan Kaufmann, San Mateo, California.

Perich, F., Finin, T., Joshi, A., and Yesha, Y. (2004). "MoGATU BDI Ontology". Online.
URL `http://mogatu.umbc.edu/bdi/`

Piaget, J. (1973). *Sociological Studies.* Routledge, London.

Poggi, A. and Rimassa, G. (2000). "An agent model platform for realizing efficient and reusable agent software". In "Proceedings of the fourth international conference on Autonomous agents", pp. 74–75. ACM Press. ISBN 1-58113-230-1.

Polovina, S. (1993). *The Suitability of Conceptual Graphs in Strategic Management Accountancy.* Ph.D. thesis, Loughborough University.
URL `http://www.polovina.me.uk/phd`

Polovina, S. and Heaton, J. (1992). "An Introduction to Conceptual Graphs". *AI Expert*, volume 7(5):pp. 36–43.

Polovina, S., Hill, R., and Beer, M. D. (2005). "Enhancing the Initial Requirements Capture of Multi-Agent Systems through Conceptual Graphs". In H. Pfeiffer, K. E. Wolff, and H. S. Delugach, eds., "Proceedings of the Thirteenth International Conference on Conceptual Structures. Conceptual Structures at Work: Contributions to ICCS 2005", LNAI, pp. 439–452. Springer.

Polovina, S., Hill, R., Crowther, P., and Beer, M. D. (2004). "Multi-Agent Community Design in the Real, Transactional World: A Community Care Exemplar". In H. Pfeiffer, K. E. Wolff, and H. S. Delugach, eds., "Conceptual Structures at Work: Contributions to ICCS 2004 (12th International Conference on Conceptual Structures)", pp. 69–82. Shaker Verlag. ISBN 3-8322-2950-7, ISSN 0945-0807.

Preece, A. D., ying Hui, K., Gray, W. A., Marti, P., Bench-Capon, T. J. M., Jones, D. M., and Cui, Z. (2000). "The KRAFT architecture for knowledge fusion and transformation". *Knowledge Based Systems*, volume 13(2-3):pp. 113–120.

SAP (2004). "SAP Global Home Page". www.
URL `http://www.sap.com/`

Sauvage, S. (2004). "Agent oriented design patterns: a case study". In "Proceedings of the third international joint conference on Autonomous agents and Multi-Agent systems (AAMAS)", p. 23. ACM Press, New York, USA.

Searle, J. R. (1969). *Speech Acts: an Essay in the Philosophy of Language.* Cambridge University Press.

Shehory, O. and Sturm, A. (2001). "Evaluation of modeling techniques for agent-based systems". In "Proceedings of the fifth international conference on Autonomous agents", pp. 624–631. ACM Press. ISBN 1-58113-326-X.

Sixsmith, A., Hawley, C., Stilwell, J., and Copeland, J. (1993). "Delivering 'Positive care' in Nursing Homes". *International Journal of Geriatric Psychiatry*, volume 8(5):pp. 407–412.

Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley.

Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical and Computational Foundations.* Brooks-Cole.

Stanford Medical Informatics (2006). "The Protégé Ontology Editor and Knowledge Acquisition System". Online.
URL `http://protege.stanford.edu/`

Sturm, A., Dori, D., and Shehory, O. (2003). "Single-model method for specifying Multi-Agent systems". In "Proceedings of the second international joint conference on Autonomous agents and Multi-Agent systems", pp. 121–128. ACM Press. ISBN 1-58113-683-8.

Sturm, A. and Shehory, O. (2003). "A Framework for Evaluating Agent-Oriented Methodologies". In P. Giorgini and M. Winikoff, eds., "Proceedings of the 5th International Bi-Conference Workshop on Agent-Oriented Information Systems", pp. 60–67.

Sun (2004). "Java Technology web page". www.
URL `http://java.sun.com/`

Taylor, J., Mistry, V., Sharples, M., Bo, G., and Ahonen, M. (2002). "MOBIlearn WP 2 - Evaluation Framework, Open University UK, D2.2 Evaluation Methodology". Technical report, MOBIlearn.
URL `http://www.mobilearn.org/download/results/public_deliverables/MOBIlearn_D2.2_Final.pdf`

Uschold, M. and Grüninger, M. (1996). "Ontologies: Principles, Methods and Applications". *The Knowledge Engineering Review*, volume 11(2):pp. 93–155.

Van Dyke Parunak, H. and Odell, J. (2001). "Representing social structures in UML". In "Proceedings of the fifth international conference on Autonomous agents", pp. 100–101. ACM Press. ISBN 1-58113-326-X.

van Lamsweerde, A. (2001). "Goal-Oriented Requirements Engineering: A Guided Tour." In "5th IEEE International Symposium on Requirements Engineering (RE 2001)", p. 249. IEEE Computer Society, Toronto, Canada.

Vasconcelos, W., Robertson, D., Sierra, C., Esteva, M., Sabater, J., and Wooldridge, M. (2004). "Rapid prototyping of large Multi-Agent systems through logic programming". *Annals of Mathematics and Artificial Intelligence*, volume 41:pp. 135–169. Kluwer Academic Publishers.

W3.org (1986). "ISO 8879. Information Processing – Text and Office Systems - Standard Generalized Markup Language (SGML)".

W3.org (2004a). "Extensible Markup Language (XML) 1.0 (Third Edition) Recommendation". www.
URL `http://www.w3.org/TR/2004/REC-xml-20040204/`

W3.org (2004b). "Web Ontology Language (OWL)". www.
URL `http://www.w3.org/2004/OWL/#specs`

Willmott, S., Beer, M., Hill, R., Greenwood, D., Calisti, M., Mathieson, I., Padgham, L., Reese, C., Lehmann, K., and Scholz, T. (2005). "NETDEMO: openNet Networked Agents Demonstration". In "AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems", pp. 129–130. ACM Press, New York, NY, USA. ISBN 1-59593-093-0.

Wooldridge, M., Fisher, M., Huget, M.-P., and Parsons, S. (2002). "Model checking Multi-Agent systems with MABLE". In "Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems", pp. 952–959. ACM Press. ISBN 1-58113-480-0.

Wooldridge, M. and Jennings, N. R. (1995). "Intelligent agents: theory and practice". *The Knowledge Engineering Review*, volume 10(2):pp. 115–152.

Wooldridge, M. and Jennings, N. R. (1998). "Pitfalls of agent-oriented development". In "Proceedings of the second international conference on Autonomous agents", pp. 385–391. ACM Press. ISBN 0-89791-983-1.

Wooldridge, M., Jennings, N. R., and Kinny, D. (1999). "A methodology for agent-oriented analysis and design". In "Proceedings of the third annual conference on Autonomous Agents", pp. 69–76. ACM Press. ISBN 1-58113-066-X.

Wooldridge, M. J. (2002). *An Introduction to Multiagent Systems.* Wiley, 1 edition.

Wooldridge, M. J., Jennings, N. R., and Kinny, D. (2000). "The Gaia Methodology for Agent-Oriented Analysis and Design". *International Journal of Autonomous Agents and Multi Agent Systems*, volume 3(3):pp. 285–312.

Xueguang, C. and Haigang, S. (2004). "Further extensions of FIPA Contract Net Protocol: threshold plus DoA". In "Proceedings of the 2004 ACM symposium on Applied computing", pp. 45–51. ACM Press. ISBN 1-58113-812-1.

Yu, E. S. K. (1995). *Modelling Strategic Relationships for Process Reengineering.* Ph.D. thesis, University of Toronto.

Yu, E. S. K. (1997). "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". In "Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)", pp. 226–235. Washington D.C., USA.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). "Developing

Multi-Agent Systems: The Gaia Methodology". *ACM Trans. Softw. Eng. Methodol.*, volume 12(3):pp. 317–370. ISSN 1049-331X.

Zhang, L., Ahn, G.-J., and Chu, B.-T. (2002). "A role-based delegation framework for healthcare information systems". In "Proceedings of the seventh ACM symposium on Access control models and technologies", pp. 125–134. ACM Press. ISBN 1-58113-496-7.

Appendix A

OWL Listings

A.1 Belief-Desire-Intention Ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/Ontology1155291197.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1155291197.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Goal">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="AchievableDesire"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="NonConflictingDesire"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Action"/>
  <owl:Class rdf:ID="Plan"/>
  <owl:Class rdf:ID="Belief"/>
  <owl:Class rdf:about="#NonConflictingDesire">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Desire"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="NonAchievableDesire">
    <rdfs:subClassOf rdf:resource="#Desire"/>
  </owl:Class>
  <owl:Class rdf:ID="ConflictingDesire">
    <rdfs:subClassOf rdf:resource="#Desire"/>
  </owl:Class>
  <owl:Class rdf:ID="BDIAgent"/>
  <owl:Class rdf:ID="Intention"/>
  <owl:Class rdf:about="#AchievableDesire">
    <rdfs:subClassOf rdf:resource="#Desire"/>
  </owl:Class>
</rdf:RDF>
```

A.2 Transaction Model Ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.owl-ontologies.com/Ontology1155310434.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
xml:base="http://www.owl-ontologies.com/Ontology1155310434.owl">
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="Transaction">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasPart"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#int"
      >2</owl:minCardinality>
    </owl:Restriction>
  </owl:equivalentClass>
  <owl:disjointWith>
    <owl:Class rdf:ID="EconomicEvent"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="EconomicResource"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="OutsideAgent"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
```

```
        <owl:Class rdf:about="#EconomicEvent"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasPart"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource=
  "http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:comment rdf:datatype=
  "http://www.w3.org/2001/XMLSchema#string"
  >the satisfactory exchange of scarce resources between
   two agents via opposing events</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:ID="InsideAgent"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#OutsideAgent">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="BDIAgent"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype=
  "http://www.w3.org/2001/XMLSchema#string"
  >the perspective of the transaction from the agent</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:about="#InsideAgent"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Transaction"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#EconomicResource"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#EconomicResource"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isDestinationOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#OutsideAgent"/>
      <owl:onProperty>
        <owl:TransitiveProperty rdf:ID="isSourceOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#EconomicEvent"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#EconomicResource">
  <rdfs:comment rdf:datatype=
  "http://www.w3.org/2001/XMLSchema#string"
  >the scarce resource to be exchanged</rdfs:comment>
  <owl:disjointWith rdf:resource="#Transaction"/>
  <owl:disjointWith rdf:resource="#OutsideAgent"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#InsideAgent"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#EconomicEvent"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#InsideAgent">
  <owl:disjointWith>
    <owl:Class rdf:about="#EconomicEvent"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#EconomicResource"/>
      <owl:onProperty>
        <owl:TransitiveProperty rdf:about="#isSourceOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Transaction"/>
  <rdfs:subClassOf rdf:resource="#BDIAgent"/>
  <owl:disjointWith rdf:resource="#OutsideAgent"/>
  <owl:disjointWith rdf:resource="#EconomicResource"/>
  <rdfs:comment rdf:datatype=
  "http://www.w3.org/2001/XMLSchema#string"
  >the perspective of the transaction from the agent</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#EconomicResource"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isDestinationOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#EconomicEvent">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasSubject"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#EconomicResource"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#OutsideAgent"/>
  <rdfs:subClassOf rdf:resource=
  "http://www.w3.org/2002/07/owl#Thing"/>
  <owl:disjointWith rdf:resource="#InsideAgent"/>
  <owl:disjointWith rdf:resource="#Transaction"/>
  <owl:disjointWith rdf:resource="#EconomicResource"/>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
```

```
        <owl:ObjectProperty rdf:about="#hasSubject"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasPart">
  <rdfs:range rdf:resource="#EconomicEvent"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isPartOf"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Transaction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isEventSubjectOf">
  <rdfs:domain rdf:resource="#EconomicResource"/>
  <rdfs:range rdf:resource="#EconomicEvent"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasSubject"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isPartOf">
  <rdfs:domain rdf:resource="#EconomicEvent"/>
  <rdfs:range rdf:resource="#Transaction"/>
  <owl:inverseOf rdf:resource="#hasPart"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isDestinationOf">
```

```
  <owl:inverseOf>
    <owl:TransitiveProperty rdf:about="#isSourceOf"/>
  </owl:inverseOf>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#OutsideAgent"/>
        <owl:Class rdf:about="#InsideAgent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#EconomicResource"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasSubject">
  <rdfs:range rdf:resource="#EconomicResource"/>
  <rdfs:domain rdf:resource="#EconomicEvent"/>
  <owl:inverseOf rdf:resource="#isEventSubjectOf"/>
</owl:ObjectProperty>
<owl:TransitiveProperty rdf:about="#isSourceOf">
  <owl:inverseOf rdf:resource="#isDestinationOf"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#OutsideAgent"/>
        <owl:Class rdf:about="#InsideAgent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
```

```
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#EconomicResource"/>
  </owl:TransitiveProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.2, Build 331)
http://protege.stanford.edu -->
```

Appendix B

MOBIlearn Case Study

B.1 MOBIlearn Case Study Background

The integration of new technologies (e.g., personalisation, multimedia, ambient intelligence, haptic interactions, mobile devices) in education and training is basically a culturally driven process with the need to bring about change not only in people, but in the entire learning environment. This is a part of the comprehensive eEurope Action Plan for European uptake of digital technologies, in which a basic objective is for education systems to use developments in information and communication technology (ICT). Another important part of MOBIlearn is the free circulation of knowledge, in forms that are appropriate for individual users. In the last decades political and social progresses have underlined the importance of the free circulation of knowledge as the most advanced answer to the increasing needs of new skills related to new technologies and new socio-economic models brought by the Information Society. On these social and technological premises, MOBIlearn aims at improving access to knowledge for selected target users (such as mobile workers and learning citizens), giving them ubiquitous access to appropriate (contextualised and personalised) learning objects, by linking to the Internet via mobile connections and devices, according to innovative paradigms and interfaces. The project will focus, in fact, on the target markets (individuals or small groups of people spread Europe-wide in many and various sites, willing to access knowledge on demand, just in time and in the field to foster their life long learning and enhance their working experience). The final objective is to improve the knowledge level of individuals through cost and time optimisation of learning processes. This maximises the opportunities of three representative groups:

- Workers, to meet their job requirements and to update their knowledge continually;
- Citizens as members of a culture, to improve the learning experience while visiting a cultural city and its museums;
- Citizens as family members, to have simple medical information for everyday needs.

The MOBIlearn system will allow acquisition of ways to meet user needs and build knowledge spaces. Impacts of the solution on self-learning will be explored in three selected and very representative applications for mobile learning (m-learning), namely:

1. Master in Business Administration (MBA) schools, where international MBA institutes (partners of MOBIlearn) will extend the reach and scope of their current blended-learning offering, by providing learners with personalised and tailored subscriptions to content on mobile networks;
2. A European city famous worldwide for its art (Florence), where Firenze Musei (not a partner, but a member of the MOBIlearn Special Interest Users' Group), a consortium managing all the European historical and

cultural heritage locations of the city, will improve its offerings enabling learning citizens to access context sensitive art, historical and cultural knowledge with mobile devices while visiting museums and galleries;

3. Access to basic medical knowledge to enable support for anywhere and anytime interventions.

The certified knowledge basis is provided by the European Resuscitation Council (not a partner, but a member of the MOBIlearn Special Interest Users' Group), which already trains non-specialised citizens in basic medical procedures (such as Basic Life Support), with quick reference, audiovisual procedural guides and VR simulations. Nevertheless the solution could be applied in many other business sectors and knowledge domains and applications for many kinds of learning and many circumstances and areas. The MOBIlearn project contributes to breaking traditional barriers to learning for many people, which exist for them now due to their limited access to information, limited time for learning and isolated environment. It should be borne in mind that these application areas are selected to provide a diverse set of user requirements and technical challenges, to draw upon previous EU-funded projects, and to allow consideration of a broad range of user activities. The MOBIlearn project has international relevance by proposing the conception, population and experimentation and exploitation of new models of learning and information use, via next-generation mobile networks, through:

- creation of pedagogical paradigms to support learning in a mobile environment (such as collaborative learning, organisational learning, dynamic knowledge creation in a group);

- new architectural layouts to support creation, brokerage, delivery and tracking of learning and information contents on the mobile network, which extend existing systems;

- selection and adaptation of existing eLearning contents for mobile devices, enabling automatic multi channel and multi device versioning;
- realization of new business models, based on existing success-cases (e.g. DoCoMo iMode), for the self sustainability and deployment of the conceived solutions beyond the research timeframe within Europe's Knowledge Society framework for the third Millennium.

The goal of MOBIlearn is the creation of a virtual network for the diffusion of knowledge and learning via a mobile environment where, through common themes, it is possible to demonstrate the convergence and merging of learning supported by new technology, knowledge management, and new forms of mobile communication. This also creates a virtual point of mobile access to content that could be used at a European and International level. A subsidiary goal is to develop deeper understandings of the social processes and interactions that arise when connectivity reaches a critical point, so that we are alert to the possible emergence of "ambient intelligence" equivalents of the widespread take-up by users of SMS. The objectives and scope of MOBIlearn appear to be very challenging, yet achievable thanks to the multi facet and innovative layout of the proposed architecture and model specifically addressing the variety of pedagogical, social and working contexts that a typical European mobile worker and learning citizen might experience.

B.1.1 Objectives

The specific objectives and challenges of the MOBIlearn are:

B.1.2 On Pedagogical Issues

The definition of theoretically-supported and empirically-validated models for:

- Effective learning/teaching/tutoring in a mobile environment;
- Instructional design and eLearning content development for mobile learning.

B.1.3 On Human Interaction and Technical Issues

The development of a reference mobile-learning architecture that is attractive to key actors in Europe and beyond, and that supports:

- Human interfaces adaptive to the mobile device in use and the nature (e.g., bandwidth, cost) of the ambient intelligence that is available in a given location;
- Context-awareness tools for exploiting context and capturing learning experience;
- Integration of mobile media delivery and learning content management systems;
- Collaborative learning applications for mobile environments.

B.1.4 On Business Issues

The conception of a business model for future deployment, starting from:

- A study of existing business models and market trends;

- An appraisal of the external environment (e.g., to take into account the business tactics of large non-European organisations entering EU mobile markets).

To achieve these objectives, MOBIlearn aims:

- To define new pedagogical models and guidelines for learning and teaching and for effective instructional content design for mobile environment. Since research in this field goes far beyond the MOBIlearn lifecycle, the definition of roadmaps for further research on pedagogical aspects of mobile learning is essential;

- To conceive, design and implement a mobile-learning reference architecture that supports the flexibility needed for the effective deployment of new pedagogical and business paradigms for knowledge access and sharing in mobile environments;

- To influence international standards and specifications bodies (i.e. ISO, IEC JTC1, SC36, ADL SCORM, CEN/ISSS WSLT, IEEE LTSC, XML, 3GPP, DVB-MHP) for extensions and integrations for mobile-learning requirements;

- To verify proposed models and solutions with real life scenarios and user trials, namely in the business administration education, in accessing cultural heritage knowledge, and basic medical information.

MOBIlearn will develop a significant and innovative mobile learning architecture. This will have elements (layers) that reflect the needs of each constituency represented by the Consortium partners and Special Interest Groups. Those constituencies include end-users (in each of the test markets), pedagogical experts, 3G mobile operators, mobile devices manufacturers (mobile phones, laptops, and PDA's), content providers with large Digital Repositories, and technology providers (integrating and extending pre-existing technologies, such as Learning Content Management, Media Streaming, collaborative software). The project will foster architectural integration and upgrades to satisfy new methodologies for mobile learning environments. These will include practical implementations and trials using learning materials in selected contexts (i.e. business administration and management education for the mobile worker, art and cultural heritage information access for the learning citizen, basic medical knowledge for everyday life). There are many aspects of learning that mobile technology could address (such as support of informal learning, mobile conversational learning, mentoring of mobile learners, outdoor science learning experiments). We envisage exchanging results with projects that will be addressing those aspects specifically. Our primary focus, however, is on an aspect of mobile learning that is of immediate economic significance: content delivery for adult learning and professional development enabled with collaborative spaces, context awareness and adaptive human interfaces. The value of the "content delivery" model of learning has been widely debated and it is particularly appropriate for well-motivated learners (e.g. adult professionals, people on cultural trips) to address a clearly defined learning need. And these are exactly the typology of learners that MOBIlearn addresses, as indicated also by the selected user trials. MOBIlearn shall not, therefore, be addressing all the emerging areas of mobile learning in this project, but it explores the chosen aspects in terms of all its different components (pedagogy, technical and human interaction, business). Furthermore, according to this approach, and following a recommendation of the EC report on "Next Steps in Learning

Futures", MOBIlearn research has been based on a multi-disciplinary approach taking into account joint pedagogical, technological and organisational aspects of learning in mobile environment. As far as mobile devices are concerned, even if the conceived architecture will be open for any device, MOBIlearn will use leading-edge laptops, mobile phones and PDA's as test-beds for development and for user trials. The company manufacturers of these devices are partners of the MOBIlearn project, and, if research proves it is necessary, it will be possible to access even low-level specifications to implement middleware (e.g. using MHP, Multimedia Home Platform standard) or to improve existing microbrowsers.

http://www.mobilearn.org

B.2 Case Study: Description of MBA Use-Case Scenario

Hans Beerli is a manager of Finance Suisse and participating in the Executive MBA. In the course of two years, Hans takes a total of 80 contact days, mostly structured into three-day modules. The class size is 30 students. On Tuesday March, 9th 2004 he will start the module on Information Management. The previous week he has received his course preparation pack with a printed case study "Printpro's odysee through E-Business". As he had been busy working, he can only open the package on Saturday: it contains a printed version of the case and his personalised prepaid course card[1]. He reads the case and is fascinated by the similarities between his own experiences at Finance Suisse

[1]This course card pays for all conversation and interactions in the MBA-learning community and identifies the user to all the course resources

and Printpos.

On Tuesday there are mainly classical lecture classes in the University lecture halls. He uses his PhonePDA to annotate the PowerPoint presentation of the slides and to link the relevant part of them to his notes on the case study. He also very much liked the example of a process analysis presented to the class in a film. As he has the feeling that others are puzzled, too, he requests to view it again. After a short discussion with the teacher, the control over the projection device is transferred to his PDA and he rewinds the film to the critical section. Having control over the shared media, he is now able to lead the class discussion on the open issues.

On Wednesday the group has to work on the case study. They meet in a University electronic meeting room for face-to-face collaboration. First the teacher asks one student to summarize the main points of the case and then the group is engaged in an electronic discussion on the underlying problems of Printpro. Some people link their PDAPhone directly to the electronic moderation toolset; the others prefer to attach it to the tablet PCs available in the room. The group identifies possible problems, and selects and structures the most important problems again using the electronic moderation toolset. Still, the outcome appears fragmented and often superficial. After a break, the teacher then presents applicable theories in order to give them a more solid foundation for the analysis.

After lunch-break, the group is split in 6 subgroups with 5 persons each. Each subgroup receives the task to analyse the case using a different perspective (marketing, financial, strategy, IS-Architecture...). As a resource they receive a shared electronic desk. The teacher has prepared specific information in an electronic library and a set of tools for each group. They use their

PDAPhone for adding information to the shared environment as well as for controlling it. After two hours of intensive work in subgroups, the group reconvenes and each subgroup presents its results to the plenary on public electronic displays. Using the Phone PDA as a remote control and annotation tool intensively, the group members are able to link the different perspectives to a comprehensive picture.

Next each subgroup has to provide a strategy for Printpro and a concept for solutions to the problems identified. The students are explicitly asked to link their subgroups proposals to their companies E-Business approaches. In a final lecture, the teacher provides the students with an overview over applicable concepts for the solutions.

During the rest of the week, Hans Beerli spends considerable time in finding out Finance Suisse's E-Business strategy. He uses his PhonePDA to support his interviewing and to exchange intermediate results with his subgroup's members. A virtual group room is used to collect immediate results and serves as a context for asynchronous group discussions and chats. A virtual classroom supports the information exchange and discussion in the plenary. As Hans has been elected leader of his subgroup, he has a longer tele-meeting with the teacher on the subgroup's progress. Twice the subgroup meets for an hour in a restaurant and during an elaborate lunch they assemble each subgroup members to a comprehensive solution. To support these activities they create a shared environment linking applications on their PhonePDA.

Next Tuesday, the subgroups present and discuss their results in a similar way as on Wednesday afternoon. A general background lecture on E-Business gives them a comprehensive overview over E-Business aspects not covered so far. In the closing electronic questionnaire the participants indicate that they were happy with most aspects of the course. The ad-hoc evaluation on the public screen however shows that the group is split on the issue whether more anonymous participation would have been useful. The teacher reserves some time for an oral discussion to get more input on this issue. The participants quickly note that the preference for anonymity depends on their companies attitude towards criticism.

All group output has been electronically documented. The teacher promises to support the electronic course community as long as there is still activity. As Hans Beerli returns home, he still downloads the most important material to his computer. He is determined to use it to improve Finance Suisse's E-Business approach.