Design Methodology for Ontology-based Multi-Agent Applications (MOMA)

Weir Ying

Submitted in the total fulfilment of the requirements for the degree of

Master of Philosophy (MPhil)

March 2009

School of Information Systems, Technology and Management

University of New South Wales

Australia

ORIGINALITY STATEMENT

Abstract

Software agents and multi-agent systems (MAS) have grown into a very active area of research and commercial development activity. There are many current emerging real-world applications spanning multitude of diverse domains. In the context of agents, ontology has been widely recognised for their significant benefits to interoperability, reusability, and both development and operational aspects of agent systems and applications. Ontology-based multi-agent systems (OBMAS) exploit these advantages in providing intelligent and semantically aware applications.

In addressing the lack of support for ontology in existing methodologies for multi-agent development, this thesis proposes a design methodology for the building of such intelligent multi-agent applications called MOMA. This alternative approach focuses on the development of ontology as the driving force of the development process. By allowing the domain and characteristics of utilisation and experimentation to be dictated through ontology, researchers and domain experts can specify the agent application without any knowledge of agent design and lower level programming. Through the use of a structured ontology model and the use of integrated tools, this approach contributes towards the building of semantically aware intelligent applications for use by researchers and domain experts.

MOMA is evaluated through case studies in two different domains: financial services and e-Health.

Acknowledgements

First and foremost, I wish to express my deepest gratitude to A/Prof. Pradeep Ray, my supervisor, for his encouragement, support and valuable guidance to every stage of my research. His devotion is sincerely appreciated.

I would also like to thank the students and research fellows from the Asia-Pacific Ubiquitous Healthcare Research Centre (APuHC), Dr. Subhagata Chattopadhyay, Dr. N. Parameswaran, Alfred Wong and Fred Yip. Their criticisms, advice and help with my thesis have proven to be extremely valuable.

My appreciation extends to Angalee Sujanani and Jaminda Wimalosiri. Their research provided inspiration and a basis for the case studies in this thesis.

Last but not least, I would like to thank my family and friends who have provide d much needed support and understanding throughout my research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACL	Agent Communication Language
AOSE	Agent Oriented Software Engineering
CCS	Clinical Computer Systems
CORBA	Common Object Requesting Broker Architecture
DAML	DARPA Agent Markup Language
DARPA	Defence Advanced Research Projects Agency
EHR	Electronic Healthcare Record
EMR	Electronic Medical Record
FIPA	Foundation for Intelligent Physical Agents
GT	Grounded Theory
HL7	Health Level Seven
IP	Interaction Protocol
KQML	Knowledge Query Manipulation Language
MAS	MAS Multi-Agent System
OBMAA	Ontology-based Multi-Agent Application
OBMAS	Ontology-based Multi-Agent System
OIL	Ontology Inference Layer or Ontology Interchange Layer
OOP	Object Oriented Programming
RIM	Reference Interchange Model
SUMO	Suggested Upper Merged Ontology
SWRL	Semantic Web Rule Language

Chapter 1. Introduction

This chapter provides a brief background on ontology, the agent paradigm and their usefulness. In so doing, we reveal the motivations for the design methodology for ontology-based multi-agent applications called MOMA. This chapter will then highlight the research objects, motivation and its significance. A detailed description of the research methodology of this thesis will then be provided. Finally, an outline of the thesis is presented.

1.1 Background and context

Agent technology has become one of the most active and promising areas of research and development activity in computing in recent years. Agents are highly autonomous, situated, interactive software entities that are seen as the backbones for the next generations of mainstream software systems (Fan 2000). Agent technology has drawn on the diversity of computing areas, including software engineering, distributed computing, networking, mobile computing, collaborative computing, security and robotics (Sundsted 1998; Honavar 1999).

The potential of agent technology is revealed through Multi-Agent Systems (MAS). MASs are computational systems in which two or more agents are interacting or working together to achieve a set of goals (Fan 2000). The coordination between agents possessing diverse knowledge and capabilities would enable the achievement of global goals that could not be otherwise achieved by a single agent working in isolation (Nwana & Wooldridge 1996). These characteristics of MASs have made it extremely useful for the running of simulations and information retrieval systems in a wide variety of domains such as finance, telecommunications and e-Health. The term Multi-Agent (MA) application in this thesis will refer to the application of MAS in specific real world domains.

Ontology is a specification of objects, concepts and entities that exist in a domain of interest and the relationships among them. They have been used successfully in the fields of artificial intelligence, information retrieval, natural language processing and knowledge engineering. In the context of MASs, ontology has been acknowledged as being beneficial to various MAS development activities in addition to operational aspects of MAS. Ontological modelling of agent knowledge is regarded as essential to operations of MAS, particularly to the communication between system components and reasoning of agents. Ontology brings both a degree of interoperability and reusability of system design for MAS (Uschold & Gruninger 1996; Chandrasekaran et al. 1999; Falasconi et al 1996). As a result, Ontology-based Multi-Agent Systems (OBMAS) have gained popularity in terms of research and applications. In a wider context, ontology allows the sharing of common understanding of structure of information among people. The nature of ontology also allows the reuse of domain knowledge. A MAS system is ontology-based when its design specification explicitly includes ontology, and ontology is used by the agents at the run-time to facilitate operation.

The growing popularity of agent-based technologies leads traditional software engineering methodologies to evolve into a set of Agent Oriented Software Engineering methodologies (AOSE). The role of the AOSE methodologies is to assist in all the phases of the life cycle of an agent-based application, including its management.

While small development projects such as these may be acceptable for applying informal software engineering principles to, the absence of specialised AOSE methodologies for MAS construction will generally result in cumbersome, error-prone, and time-consuming application development (Eurescom 2001b; Lind 2000b). The disregard for AOSE methodologies is seen as the main reason for the failure of many past MAS development experiences (Fan 2000). Therefore it is accepted that AOSE methodologies are needed to guide developers in the creation of multi-agent applications.

1.2 Research problem and motivation

The numerous advantages for the use of ontology in MAS are widely acknowledged (Falasconi et al. 1996; Malucelli & Oliveira 2004; Yuan 1999) Ontological modelling of agent knowledge is also regarded as essential to the operation of MAS, particularly to the communication between system components (e.g. between agents) and the reasoning of agents. Reusability of system design through ontology has been recognised in single agent knowledge based systems (Uschold & Grunninger 1996; Chandrasekaran et al. 1999; Falasconi et al. 1996). However, the majority of AOSE methodologies do not support the use of ontology-based MAS development. As a result, existing AOSE methodologies do not provide, or provide to a lesser extent, the various important capabilities that an ontology-based development methodology can, such as support for interoperability and reusability.

The AOSE methodologies show that there is a conceptual level for analysing the agent-based systems, no matter what the agent theory, agent architecture or agent languages are. The lack of a standard agent architecture and agent programming languages are a problem for the implementation of these methodologies. Since there is no standard agent architecture, the design of MAS needs to be customised for each of the existing agent architectures.

One implicit assumption that is made by all these AOSE methodologies is that the user of the methodology must be knowledgeable in the field of agent technology, or at least software engineering processes, techniques and tools. In domains such as finance and medicine, we cannot expect those experts and researchers to have knowledge in agent oriented software engineering. In these cases we will assume that outside help needs to be sought. From here onwards, experts and researchers in their perspective domains will be referred to as domain experts.

1.3 Research Objectives

The objective of this research is to develop a practical methodology for the development of an ontology-based multi-agent application (OBMAA). This methodology, called Design Methodology for Ontology-Based Multi-Agent Applications or MOMA will concentrate on the development of ontology and the use of this ontology to drive the implementation of the agent application. Full MOMA methodology includes the design and development of the ontology, agents and their integration. However, the agent development methodology is borrowed from existing AOSE methodologies. This thesis will focus only on the design and development of ontology parts of the methodology. The other parts of the methodology will be treated as black boxes for implementation and evaluation purposes. This is where that part of the methodology is assumed to be there and fully working.

The main objectives of this methodology are to (Refer to Section 3.1 for further details):

1. Provide a structured meta-model for the development of ontology for agent application development.
2. Allow the ontology model to define behaviour of agents.
3. Facilitate the use of tools to drive development, conceptual testing and implementation of ontology for agent systems.
4. Provide support for reuse and sharing of the developed ontology.
5. Distinguish the roles of domain expert and agent developer in the development process of MOMA.
6. Work towards a methodology that can be used by domain experts (and researchers) without the expertise of an agent developer.

1.4 Research Questions

A software engineering methodology, defined by Henderson-Sellers et al. (1998), contains the following:

- A software engineering **process** to conduct the system development
- **Techniques** to assist the process; and
- **Definition of the work product.**

The process contains activities and tasks (Henderson-Sellers et al. 1998). Activities are large scale descriptions of what needs to be done, such as “requirements engineering” activity, “design” activity, “implementation” activity and “testing” activity. A task on the other hand refers to smaller scale that is associated with each activity in the process. Techniques are linked to each task which provides away of carrying out each task. Tasks will be referred to as steps if they are sequential in nature. Since the work product will be ontology based multi-agent application, these definitions will be the models on which this application will be built.

The development of the MOMA methodology conforms to the definition provided above. Several research questions must be answered:

1. What are the **models** required for ontology in an ontology-based multi-agent application?

   The term "model" here refers to the class of models or meta-models which is used to produce ontology and the system design by the domain experts during the development process (Section 3.2.2.1-3.2.2.2). This question will satisfy the work product definition.
2. What **process** is needed to develop OBMA applications?

   The process will be the activity and tasks of the methodology. That is, the steps of the methodology.
3. What **tools and techniques** can be used in assisting the development of OBMA applications?

   This question will satisfy the "techniques" part of the definition. Tools and techniques will be used to assist the performance of the activities and tasks in the process.

1.5 Significance of research

This research provides an alternative lightweight approach to the traditional AOSE methodologies for the development of OBMAS. It will allow domain experts (with help) to quickly develop OBMAA.

The inclusion of ontology allows us to separate domain knowledge from the underlying agent implementation, to a certain degree. Capturing the domain knowledge in ontology would keep the agent system as generic as possible. This will greatly facilitate the reuse of ontology and agent systems. This moves us one step closer to the eventual goal of having the domain knowledge and generic MAS platforms interchangeable.

Ultimately, the demonstration of the significant advantages of ontology and its use in both AOSE design process and agent operations will foster the widespread development of ontology based agent systems, hence contributing to the growing maturity of both ontology and agent technology.

1.6 Research design methodology

The work of this thesis can be classified as design science, one of the two core paradigms that characterise much of the research in the Information systems discipline, the other being behavioural science (Hevner et al 2004; March and Smith 1995). Behavioural science research paradigm seeks to develop and verify theories that explain or predict human/organisational behaviour surrounding the development and use of information systems, while design science paradigm seeks to create innovative artifacts through which the development and use of information can be effectively and efficiently accomplished. Artifacts can be broadly classified as methods (i.e. set of steps, guidelines or algorithms), models (i.e. abstractions and representations), constructs (vocabularies and symbols) and implementation (i.e. prototype systems) (Hevner et al. 2004). This thesis aims to create two of these artifacts: methods and models. The method will be the MOMA process itself, while the models will be the set of ontology meta-models that accompany the MOMA methodology.

March and Smith (1995) identified that a typical design science research should comprise of two basic processes: build and evaluate. Build refers to the constructions of artifacts – i.e. the model and the method of MOMA. The evaluation process refers to the use of appropriate evaluation methods to assess the artifacts' performance. The evaluation of designed artifacts typically uses methodologies available in the knowledge base. A summary of evaluation methods is shown in Table 1-1.

1. Observational	• Case Study: Study artifact in depth in business environment • Field Study: Monitor use of artifact in multiple projects
2. Analytical	• Static Analysis: Examine structure of artifact for static qualities (e.g., complexity) • Architecture Analysis: Study fit of artifact into technical IS architecture • Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behaviour • Dynamic Analysis: Study artifact for dynamic qualities (e.g. performance) during use.
3. Experimental	• Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability) • Simulation: Execute artifact with artificial data
4. Testing	• Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects • Structural (White Box) Testing: Perform coverage testing of some metric (e.g. execution paths) in the artifact implementation
5. Descriptive	• Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the

	artifact's utility • Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

TABLE 1-1: DESIGN EVALUATION METHODS (HEVNER ET AL. 2004)

The evaluation method used for this research will through case studies. Case studies of the MOMA methodology will be conducted in both Financial Services and e-Health domains. Each case study however will be evaluated using a combination of black box and scenario testing.

1.7 Organisation of thesis

The thesis will be presented in the following chapters:

Chapter 1 - Introduction: provides a brief background to establish context and an overview of the research motivation, objective and significance.

Chapter 2 – Background: gives detailed background information on the domains of both ontology and multi-agent systems. This chapter continues with a review of current AOSE methodologies and their limitations.

Chapter 3 – MOMA Methodology: Details of the MOMA methodology, including model, process, tools and techniques as well as examples.

Chapter 4 – Case study in finance domain using MOMA.

Chapter 5 – Case study in e-Health domain using MOMA.

Chapter 6 – Discussion, evaluation, conclusions and future works.

Chapter 2. Background

2.1 Introduction

Agent technology has become one of the most active and promising areas of research and development activity in computing in recent years. Agents are highly autonomous, situated, interactive software entities that are seen as the backbones for the next generations of mainstream software systems (Fan, 2000). The potential of agent technology is revealed through MASs which are computational systems in which two or more agents are interacting or working together to achieve a set of goals. The coordination between agents possessing diverse knowledge and capabilities would enable the achievement of global goals that cannot be otherwise achieved by a single agent working in isolation (Nwana, 1996). Originating from artificial intelligence, agent technology has progressively draw on a diversity of computing areas, including software engineering, distributed computing, networking, mobile computing, collaborative computing, security and robotics (Honavar, 1999). As a result, Agent-Oriented Methodologies emerged as an extension to traditional software engineering methodologies and became widely accepted as needed for agent technology to become widespread commercial success (Flores-Mendez 1999; Sycara 1998; Zambonelli Organisational Abstractions for the Analysis and Design of Multi-Agent Systems, 2000).

In this chapter, definitions of Agent and Multi-Agent systems will be provided along with their respective advantages and limitations. Ontology and its benefits to MAS will then be explored. Using these advantages as a measure, existing AOSE methodologies are reviewed. Limitations identified will provide motivation for the MOMA methodology in Chapter 3.

2.2 Agent and multi-agents

2.2.1 Definition of Agent

The term “Agent” in our case refers to a software agent. Software Agents are entities or piece of software that acts on behalf of its user to accomplish a given task (Mountzia, 1996). There is however a variety of proposed definition offering different opinions on what constitutes an agent (Mountzia 1996; Nwana 1996; Wooldridge M. 1999). As mentioned in Wooldridge, a universal definition of the term “software agent” may be impossible since attributes that characterize agency may vary across domains. Nwana describes an agent as a software abstraction, an idea, or a concept. The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. Of the various proposed definitions of agents, commonly agreed upon concepts include:

- *persistence* - code is not executed on demand but runs continuously and decides for itself when it should perform an activity;
- *autonomy* - agents are capable of task selection, prioritization, goal-directed behaviour and decision-making without human intervention;
- *social ability* - agents are able to engage other components through some sort of communication and coordination and may collaborate on a task;

- *reactivity* - agents perceive the context in which they operate and react to it appropriately.

Even though agents may assume other attributes such as mobility, adaptability and personality, the above attributes characterize the core notion of intelligent agency.

2.2.2 Definition of MAS

A MAS is a computational system, or a loosely coupled network in which two or more agents interact or work together to perform a set of tasks or to satisfy a set of goals. Each agent is considered as a locus of a problem-solving activity which operates asynchronously with respect to the other agents (Lesser, 1996). MAS can be used to solve problems which are difficult or impossible to solve for an individual agent or a monolithic system.

Agents in a MAS exhibit several important characteristics (Wooldridge M. 2002):

- Autonomy - the agents are at least partially autonomous
- Local points of view - no agent has a full global view of the system.
- Decentralisation – there is no controlling agent (otherwise the system is effectively reduced to a monolithic system).

MAS themselves manifest self-organising and complex behaviours even when the individual strategies of their agents are simple. Individual agents communicate through a common language such as the Knowledge Query Manipulation Language (KQML) or FIPA's Agent Communication Language (ACL).

2.2.3 Motivation for agents and MASs

Agents are believed to represent the next advancement in software engineering. They offer a notably more powerful and natural abstraction for modelling and developing systems than conventional methods such as procedural abstraction, abstract data types and object oriented abstraction (Wooldridge M. 1999). The concept of software components, capable of flexibly interacting with each other to satisfy their objectives is a familiar concept to software engineering. For example, in an electronic trading application, it is natural to model participants in trade transactions as agents which buy and sell stock on behalf of their end users. This allows the use of agent paradigm to easily model complex interactions in existing systems (Wooldridge M. a., 2000).

The power of agents and MASs are particularly realised in the engineering of open systems, which are often dynamic in structure. Their system components are usually not known in advance, highly heterogeneous and capable of change over time. Thus, the ability to engage in flexible and robust interaction among the system components is crucial. Agents exhibit this ability through negotiation and coordination capabilities. These capabilities are done through the use of agent communication languages (KQML or FIPA-ACL). The core attributes of agents such as autonomy, pro-activeness and reactivity allows them to deal with dynamic and unpredictable environments. Agents can continually monitor their environment revise their goals and proactively adopt new goals (Jennings, 1995).

Another important contribution of agents and MASs is in the engineering of distributed systems. In such systems, it is difficult to specify a simple point of control because the systems are built out of distributed components, each of which may possibly attempt to achieve conflicting individual goals (Jennings, 1995). It is therefore natural to map the distributed entities onto autonomous problem solving agents, which negotiate and coordinate in a versatile manner to resolve conflicts and achieve the global goals. In addition, the pro-activeness of the agents makes it possible to abstract away from the control issue, thereby dealing with the decentralisation of control. If the system incorporates distributed resources, agents can be used to “wrap” around these resources to create “active resources”. Tasks can then be performed directly at the remote resource sites, limiting the need for communication across the network and reducing network traffic (Horlait, 2003).

Agents also provide benefits of the conventional OO paradigm such as modularity and reusability. When a problem is too complex or unpredictable, the most effective way to address it is to develop a number of modular agents, each of which specialises at solving a particular aspect of the problem (Jennings, 1995). A MAS, however, represents more than a modular object-based system. As discussed earlier, agents can interact and coordinate in an autonomous, flexible and context-dependent manner so as to ensure that the tasks are properly managed (Sycara, 1998). Reusability is supported simply by reusing the design or coding of similar agents in previous MAS development experiences.

2.2.4 Limitations of Agents and MAS

Although the agent paradigm offers many exciting opportunities, it does have some shortcomings. For many applications the sophistication of agents is simply not needed (Eurescom, 2001). For example, a software entity that engages in a relatively small amount of reasoning and simple communications can sensibly be modelled as an object rather than an agent. A MAS is also not suitable for systems where global constraints have to be maintained when risk is too high to give agents absolute trust and delegations (Jennings, 1995).

2.3 Ontology

2.3.1 What is ontology?

Ontology is a very old concept that generally been confined to the philosophical domain in the past, since the time of Aristotle. However since the 1990s, ontology has become increasingly attractive to various computing areas such as knowledge engineering, knowledge management, natural language processing, information retrieval and integration, cooperative information systems and agent based system design (Gamper, 1999).

In the context of computing, ontology is defined as an “explicit specification of a shared conceptualisation” (Fensel, 2001). “Conceptualisation” refers to an abstract model of phenomena in the real world. It defines the relevant concepts or entities that exist in the universe of discourse and the relations that hold amongst them. The “shared” characteristic of ontology implies that ontology should capture consensual knowledge, i.e. it is not restricted to an individual but accepted by a group. “Explicit” means that ontology should be clearly defined. In the context of MAS, this means that ontology used by agents need to be explicitly stated and no remain implicit within the agent codes. Finally “formal” refers to the fact that ontology should be machine-readable. Different degrees of formality are possible. Ontology like WordNet provides a thesaurus for natural language terms explained in natural language where as Cyc (Cyc 2009) provides formal axioms for knowledge (Fensel 2001).

2.3.2 Benefits and motivation for the use of ontology

The importance of ontology in areas such as knowledge engineering, information retrieval and database design has been widely discussed (Uschold, 1996). This research focuses on the importance of ontology in the context of MAS. Ontology has been widely recognised for its significant benefits to interoperability and reusability.

One of the major benefits of ontology is that it provides a degree of interoperability. Interoperability refers to the ability of heterogeneous components to interact and work with each other to achieve shared or individual goals. Interoperability involves not only communication between the heterogeneous components, but also the ability of these components to use exchanged information. In the context of MAS, the interoperability problem can be divided into two major issues:

Semantic heterogeneity – this is a case of semantic interoperability. When the knowledge base, each agent or information of each resource uses a different vocabulary to express the same information (e.g. “Price” versus “Cost”) and/or uses the same vocabulary to express different information (e.g. “Stock” in one agent/resource means shares, but another agent refers stock as items of inventory), then there is an inconsistency in the meaning of the knowledge. Another example of this is where the same concept refers to different scales or reference of measurement (e.g. “Price” maybe measured in dollars in one instance and euros in another).

Structural heterogeneity – This occurs when the knowledge of each agent or resource, uses a different conceptual schema to represent its data. For example, the concept "Stock-Name" is represented as an object in one instance, but used as an attribute under another concept in another instance.

Both heterogeneity issues can be addressed by the use of ontology. When the knowledge bases of heterogeneous agents or resources are explicitly conceptualised by ontology, the structural and semantic interoperability between these agents or resources can be achieved by mapping between these ontologies. This is done through a method known as ontology mapping, which specifies the semantic correspondences between the concepts of one ontology with another (Madhavan, 2002).

Another major benefit of the use of ontology is its capability to enhance reuse. Ontology can be used to capture knowledge elements of a system. For example, in the case of a problem-solving system, the methods for domain independent problem-solving methods and the domain knowledge can be kept in two separate components. This modularity in knowledge modelling would allow different problem domain and the reuse of domains knowledge across different problems (Uschold 1996).

Another factor that enables ontology to enhance reusability is its readability. Software reuse is typically promoted by the readability of the software design and/or codes. Ontology enhances readability by offering a structured, explicit human-readable mechanism for representing knowledge. They help the system developers to easily comprehend, inspect and reuse this knowledge for future applications.

2.3.3 Motivation for the use of Ontology in MAS

Literature is currently rich with discussion of ontology’s importance (Uschold, 1996), such as in the areas of knowledge engineering (Shave 1997), information retrieval (Ding, 2001) database design (Sugumaran & Storey 2001) and the semantic web (Davies Fensel & Van Harmelen 2003). The focus on the importance of ontology in this thesis is in the context of MAS. Ontology has been widely recognised for their significance and benefits to interoperability, reusability, MAS development and MAS operation (Falasconi, Lanzola, & Stefanelli, 1996; Malucelli & Oliveira, 2004; Knoblock, Arens, & Hsu, 1994). Interoperability and reusability were mentioned briefly in the previous section. This section will look at the benefits of ontology to MAS development activities and operations.

2.3.4 Benefits of ontology to MAS development

Two major activities in MAS development that can be facilitated by the use of ontology are system analysis and agent knowledge modelling.

System analysis involves the formulation of the problem to be solved and/or the representation of the application’s domain knowledge (Girardi & deFaria, 2004). The availability of an ontology which holds explicit, comprehensive knowledge about the target domain will greatly promote the developer’s understanding of the application, thereby facilitating his elicitation of the system goals and responsibilities. This importance of ontology has been realised and exploited by the Knowledge Engineering community in the engineering of knowledge-based systems (Shave, 1997; Chandrasekaran, Josephson, & Benjamins, 1999). Ontology offers a structured, explicit, human-readable mechanism for representing domain knowledge. These characteristics promote the readability of an ontology, hence enhancing its reusability in terms of representation.

Given these benefits to system analysis, various methodological frameworks for developing MASs and knowledge-based systems have exploited ontology to facilitate their problem-elicitation process such as "GRAMO" (Girardi & deFaria, 2004) and "CommonKADS" (Schreiber et al. 1994).

Agent knowledge modelling refers to the specification of local knowledge of each agent in a MAS, including problem-solving knowledge and local domain-related knowledge. Just as for an application's domain knowledge, an ontology can be used as an effective representation mechanism for agent's local domain-related knowledge (Mukherjee, Dutta, & Sen, 2000). Different ontology can be assigned to different agents to represent the agent's different views of the world (Falasconi, Lanzola, & Stefanelli, 1996). In addition, ontology offers a mechanism for decoupling the modelling of agent domain-related knowledge from its problem-solving knowledge, hence promoting the reuse of agent and knowledge modules. Since the local domain-related knowledge of each agent is extracted from the application's domain knowledge, the use of ontology to represent the application's domain knowledge during system analysis would facilitate the use of ontology to represent agent's local knowledge during agent knowledge modelling.

2.3.5 Benefits of Ontology to MAS operations

Ontology is beneficial to two major aspects of MAS operations: inter-agent communication and agent reasoning.

Inter-agent communication occurs when messages are passed from one agent to another. Even though sharing a common ACL will allow agents to exchange messages (common syntax and protocols), it does not ensure that the communicating agents will interpret the exchanged messages in a uniform and consistent manner, that is, to share the same semantics or meaning of the message (Uschold 1996; Falasconi Lanzola & Stefanelli 1996). Successful agent communication requires an agreement between agents to share ontology during communication. This shared ontology provides the agents with a set of common vocabulary for formulating and interpreting the content of the exchanged messages. For example, if agent A sends agent B the following message (written in FIPA-ACL):

```
Inform
:sender AgentA
:receiver AgentB
:language KIF
:ontology FruitDomainOntology
:content (>(price fruit X) (price fruit Y))
```

Then both agents need to commit to the Fruit Domain ontology (stated in the field ":ontology") where concepts "price" and "fruit" are defined. This means that the local knowledge of each agent should contain the common ontology that is used for communication. This requirement indicates the inter-dependency between ontology's role in agent communication at run-time and modelling of agent knowledge at design time.

Agent reasoning at run-time uses the problem-solving knowledge of the agent. The domain-related knowledge held by the agents is also use as inputs. (Benjamins, de Barros & Valente 1996). If the domain-related knowledge has been modelled as ontology during agent knowledge modelling, with all the relevant domain concepts and relationships being explicitly defined, the agent reasoning process can easily utilise this knowledge. By using representation language such as OWL (W3C), reasoning engines can be used directly on the ontology. Axiom languages such as Semantic Web Rule Language (SWRL) can be imposed on the ontology to provide logic to the reasoning.

2.4 Agent-Oriented Software Engineering

The role of agent-oriented methodologies is to assist in all phases of the life cycle of an agent-based application, including its management. While for small development projects, it may be acceptable to apply informal software engineering principles for the development of MASs, the absence of specialised AOSE methodologies for MAS construction will generally result in cumbersome, error prone, and hence expensive, development process (Lind 2000).

Even though research in AOSE is still less developed than other conventional software engineering paradigms such as the OO paradigm, a number of AOSE methodologies have been proposed to assist the analysis and design of MASs. These methodologies vary significantly in their scope, approach, processes, modelling concepts and modelling notations as well as their intended purpose and domain. To avoid building these methodologies from scratch researchers on agent-oriented methodologies have followed the approach of extending existing methodologies to include the relevant aspects of agents. Summaries of the commonly used methodologies in each category are presented in the following section.

2.4.1 AOSE Methodologies

MaSE

The Multi-agent Systems Engineering (MaSE) methodology is a general purpose methodology for developing heterogeneous multi-agent systems (Deloach, Wood & Sparkman 2001). MaSE uses a number of graphically based models to describe system goals, behaviours, agent types and communication interfaces. MaSE also provides a way to specify architecture-independent detailed definition of the internal agent design. MaSE uses conventional OO modelling techniques such as OMT and UML. An overview of the MaSE methodology is illustrated in Figure 2-1.

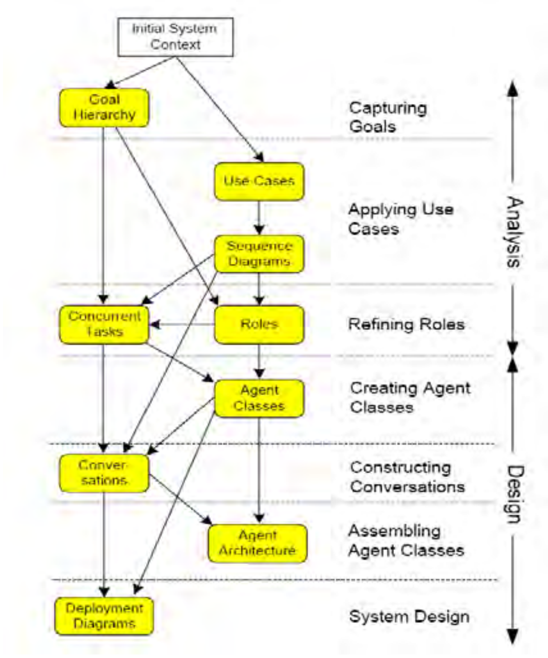


FIGURE 2-1: OVERVIEW OF MASE

The development phase of MaSE consists of Analysis and Design phases. The Analysis Phase involves three steps.

1. "Capturing Goals" step firstly identifies goals of the target system and organises them into Goal Hierarchy Diagram. An example is illustrated in Figure 2-2.

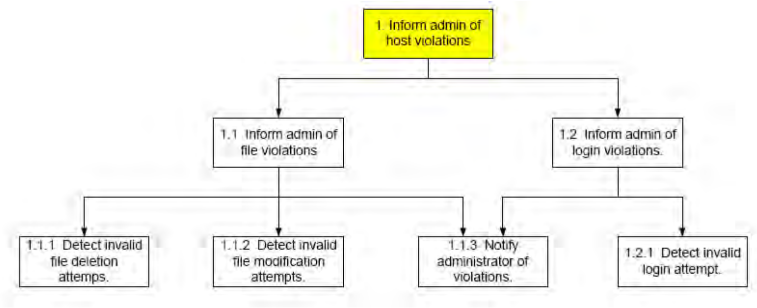

1. Inform admin of host violations
1.1 Inform admin of file violations
1.2 Inform admin of login violations.
1.1.1 Detect invalid file deletion attemps.
1.1.2 Detect invalid file modification attempts.
1.1.3 Notify administrator of violations.
1.2.1 Detect invalid login attempt.

FIGURE 2-2: MASE GOAL HIERACHY DIAGRAM

2. "Applying Use Cases" step produces Use Cases from the system requirements and elaborates on them in the form of Sequence Diagrams. An example is illustrated in Figure 2-2.
3.

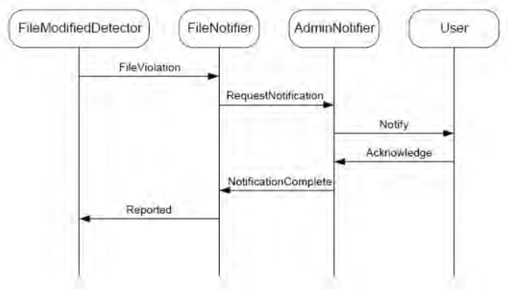


FIGURE 2-3: MASE SEQUENCE DIAGRAM

4. "Refining Roles" step identifies roles from system goals and actors, thereby developing a Role Model. This model shows all the roles in the system, their corresponding goals and the communication paths between roles (Figure 2-4). The developer may further elaborate on the Role Model by defining tasks to be performed by each role and the communications

between tasks. A concurrent Task Diagram, which is basically a state transition diagram, can be developed to provide a detailed definition of each task.

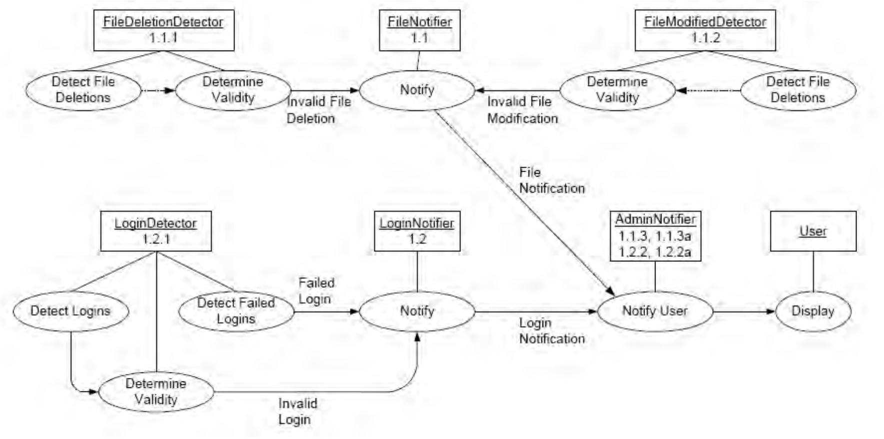


FIGURE 2-4: MASE ROLE MODEL

The Design Phase of MaSE transforms the preceding Analysis models into constructs necessary for the actual implementation of the MAS system. The phase consists of four steps.

“Creating agent Classes” step identifies agent classes for the target system by applying one-to-one mappings between roles and agents. Multiple roles, however, can be combined into a single agent class if the size and frequency of inter-role communications are high. An Agent Class Diagram is produced to show the identified agent classes, their corresponding roles and conversation paths between agent classes. An example of an Agent Class Diagram is illustrated in Figure 2-5.

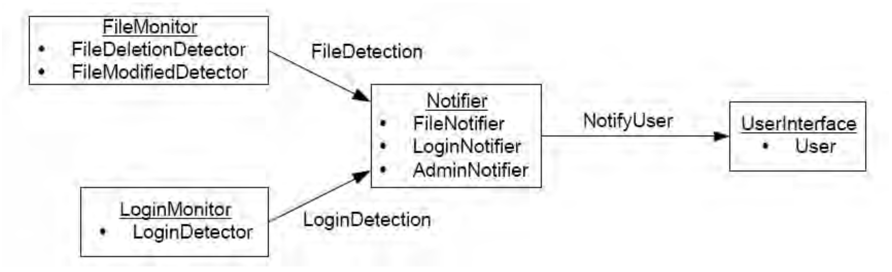


FIGURE 2-5: AGENT CLASS DIAGRAM

“Constructing conversations” step defines coordination protocols between agents. Each conversation is described by two Communication Class Diagrams, each specifying the state transitions of each agent participant during the conversation. An example of Communication Class Diagram is illustrated in Figure 2-6.

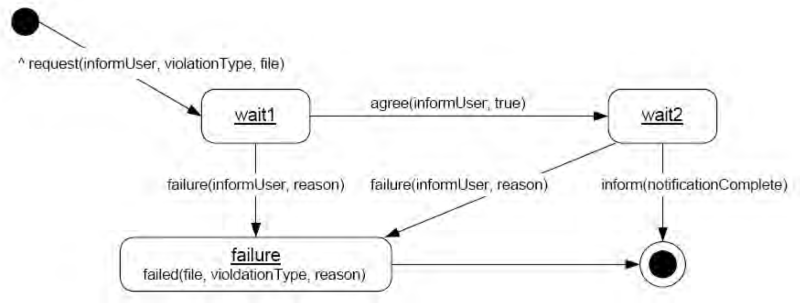

^ request(informUser, violationType, file)
wait1
agree(informUser, true)
wait2
failure(informUser, reason)
failure(informUser, reason)
inform(notificationComplete)
failure
failed(file, violdationType, reason)

FIGURE 2-6: MASE COMMUNICATION CLASS DIAGRAM

“Assembling Agent Classes” step identifies and constructs the internal components of each agent class. The developer can either reuse a pre-defined agent architecture and internal components, or retrieve pre-defined components and assemble them into a user-defined architecture, or define both internal components and agent architecture from scratch.

“System Design” step instantiates agent classes with actual agent instances and allocates these instances to nodes. A Deployment Diagram is developed to show the number, types, locations and communication paths between agent instances. An example of Deployment Diagram is illustrated in Figure 2-7.

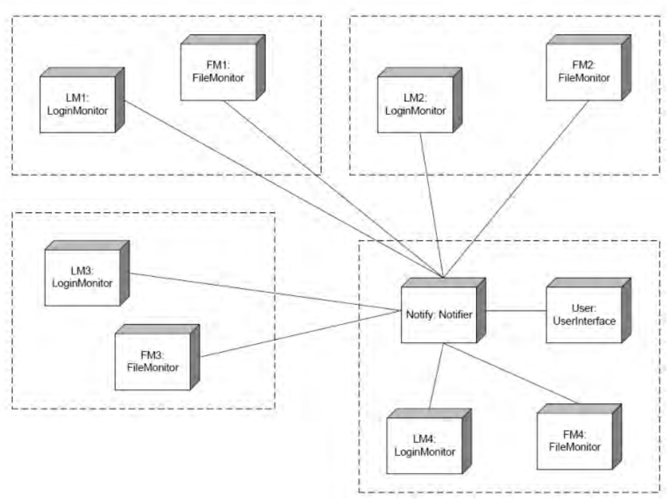


FIGURE 2-7: MASE DEPLOYMENT DIAGRAM

In 2002, MASE was expanded to provide support for ontology-based MAS development (DiLeo, Jacobs, & DeLoach, 2002). Ontology was introduced as a mechanism to model the application domain. An additional step – "Building ontology" – has accordingly been added to the Analysis phase (Figure 2-8). This step constructs the domain ontology by identifying the scope of the ontology, collecting data about the domain, forming the initial ontology, and finally refining, validating and maturing the ontology into a complete version. Once the domain ontology is constructed, parameters passed between agents during the execution of tasks or during conversations are specified in accordance with the ontology. Specifically, the data type of each exchanged parameter is defined using the concepts defined in the ontology. Step "Assembling agent classes" of MASE has also been extended to support the specification of ontology for individual agents. This specification is needed if the agent requires a knowledge model that is different from the other agents and/or from the overall domain ontology. The developer should determine the mappings between these individual agents' ontologies in order to interoperate between the heterogeneous agents.

The step "Assembling agent classes" of MASE has also been extended to support the specification of ontology for individual agents. This specification is needed if the agent requires a knowledge model that is different from the other agents and/or from the overall domain ontology. The developer should determine the mappings between these individual agents' ontologies in order to interoperate between the heterogeneous agents.

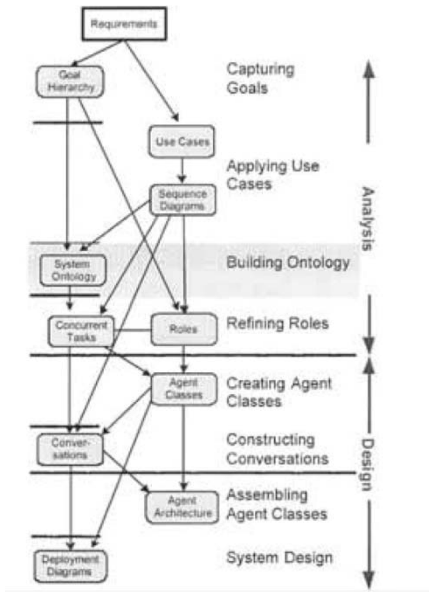


FIGURE 2-8: EXTENDED VERSION OF MASE 2002

MASSIVE

MASSIVE (Lind 2000) follows an "iterative view engineering process" for MAS development, which is a product-centred development process that combines Round-trip engineering and Iterative Enhancement. In the first cycle of the development process, the developer firstly produces a preliminary version of the development product, which is composed of seven different “views” of the system. These views are then implemented and refined if errors occur during implementation. The initial implementation is then tested and/or enhanced, which may result in enhancements to the views. If enhancements cannot be integrated into the views (e.g. because they are incompatible with some basic requirements of the views), the implementation must be changed. After this step, the next cycle is executed until the entire system is fully implemented.

SODA

SODA, “Societies in Open and Distributed Agent spaces” (Omicini, 2000) proposes a number of abstractions and techniques for the modelling of agent societies and environments. It does not aim to provide support for agent internal design, but rather focuses on inter-agent design. SODA’s development process is structured into Analysis and Design phases.

GAIA

This widely referenced methodology aims to guide the developer from a statement of requirements to a design that is sufficiently detailed to be implemented directly (Wooldridge, Jennings, & Kinny, 2000). GAIA has been extended to include new organisational abstractions that enable it to support the development of “open” MASs (Zambonelli, Jennings & Wooldridge 2003).

MESSAGE

MESSAGE (Eurescom 2001) adopts the Rational Unified Process lifecycle and extends UML to support the modelling of concepts such as “organisation”, “role”, "goal" and "task". The MESSAGE development process covers the Analysis and Design phases only.

Methodology for BDI Agents (BDIM)

Belief-Desire-Intention (BDI) is a prominent architectural model for agents. Each BDI agent is composed of beliefs (the agent's knowledge of the world), desires (the agent's motivations such as goals, objectives or allocated tasks) and intentions (the desires that the agent is committed to achieving at a certain point in time). The BDIM (Kinny, Georgeff, & Rao, 1996) is especially targeted at MASs that are based on the BDI paradigm.

In BDIM, models are classified into two levels of abstraction: external and internal. External models describe the target MAS from the system-level point of view, while Internal models define each agent class in terms of its internal components. Accordingly, the development process of BDIM is organised into two groups of steps: those for developing external models and those for developing internal models.

INGENIAS

INGENIAS (Pavon & Gomez-Sanz 2003) is built upon MESSAGE. It reconstructs and extends MESSAGE to include a new model (Environment Model), provide support for the BDI agent architecture and provide tools for documenting the system and for automatic code generation.

Methodology with High-Level and Intermediate Levels (HLIM)

HLIM (Elammari & Lalonde 1999) starts from a high-level view of the system and drills down to intermediate, implementable definitions of system design. Its development process is structured into two phases: Discovery and Definition.

Methodology for Enterprise Integration (MEI)

MEI (Kendall, 1999)is targeted at enterprise integration applications. It is based upon the IDEF approach in workflow modelling, CIMOSA framework in enterprise modelling and use-case approach in OO software engineering. MEI develops MAS by mapping various elements of the Use Case Model, IDEF/CIMOSA Functional Model and IDEF Information Model onto the design of agents, agent internal components and agent interactions.

PROMETHEUS

Prometheus (Padgham & Winikoff 2002; Winikoff & Padgham 2004) is well suited to the development of BDI-based MASs. The development process of Prometheus is structured into three phases: System Specification, Architectural Design and Detailed Design.

PASSI

A Process for Agent Societies Specification and Implementation (PASS) (Burrafato & Cossentino 2002; Cossentino & Potts 2002), offers a step-by-step requirement-to-code process for MAS development. It consists of twelve steps, grouped according to their outputs.

ADELFE

ADELFE (Bernon et al. 2002) is a methodology dedicated to adaptive MASs, which are MASs that can adapt themselves to unpredictable, evolutionary and open environments. At the core of ADELFE is the AMAS theory, which postulates that the global behaviour of a MAS emerges from the collective behaviour of the different agents composing it. Agents designed by ADELFE are equipped with an ability to deal with cooperation failures know as “non cooperative situations”.

CoMoMAS

CoMoMAS (Glaser N. 1997;) is built upon CommonKADS – a methodology for developing knowledge-based systems (Schreiber et al. 1994). CommonKADS proposes a set of seven models for specifying various types of knowledge required by a knowledge-based system: Organisation, Task, Expertise, Decomposition Expertise, Design, Communication and Agent Models. CoMoMAS adapts CommonKADS to the development of MAS by including MAS-specific knowledge structures, taking into account the reactive, cognitive, cooperative and social competencies of autonomous agents.

MAS-CommonKADS

MAS-CommonKADS (Iglesias et al. 1996; Iglesias et al. 1998) is also based on CommonKADS. However, the methodology also takes advantages of various OO techniques such as the use of case analysis and CRC cards.

CASSIOPEIA

CASSIOPEIA (Collinot & Drogoul 1998; Collinot, Drogoul & Benhamou 1996) aims to support the development of problem-solving MASs, where agents work together to fulfil a specific collective task. The methodology process from the collective task to the design of MAS along three steps.

TROPOS

TROPOS (Castro et al. 2001; Castro, Kolp & Mylopoulos 2002; Bresciani et al. 2004) is based upon the organisational modelling framework proposed by Yu. It employs the concept of “actor”, “goal”, and “dependency” to represent system requirements, MAS architecture and MAS detailed design. The development process of TROPOS is structured into four phases.

2.4.2 Support for Ontology-Based MAS Development

As mentioned in Section 2.3, ontology is widely acknowledged in literature for its significant benefits to interoperability, reusability, MAS development and operations. However, a majority of the existing AOSE methodologies do not recognise and implement these ontology’s benefits, including MASSIVE, SODA, GAIA, BDIM, INGENIAS, HLIM, MEI, PROMETHEUS, ADELFE, COMOMAS, CASSIOPEIA and TROPOS. These methodologies neither mention the use of ontology in their MAS development process, nor integrate ontology into their MAS model definitions. The AOSE methodologies review that showed some consideration for ontology includes: MAS-CommonKADS, MESSAGE, MASE and PASSI.

In MAS-CommonKADS, ontologies are used to represent the knowledge of the application’s domain and the agents’ local domain-related knowledge. Accordingly, MAS-CommonKADS illustrates the use of ontologies for knowledge representation in system analysis and agent knowledge modelling respectively. However, MAS-CommonKADS does not recognise the essential role of ontologies in agent communication. In particular, it overlooks the importance of ontology-sharing by communicating agents, and the need for the messages exchanged to be formulated in terms of shared ontological concepts. It is also unclear whether, and how, MAS-CommonKADS can enable agent reasoning at run-time to utilize agents' ontology-based knowledge, since no reference to ontologies is made during the specification of agents' problem-solving knowledge. Moreover, MAS-CommonKADS completely overlooks the capability of ontologies to support interoperability. The methodology does not consider the possibility of agents possessing heterogeneous ontologies, or of MAS incorporating heterogeneous non-agent resources, and how the heterogeneity issues between these components can be solved. As a result, MAS-CommonKADS' support for reusability is also limited, since the methodology cannot show how legacy (heterogeneous) system components can be reused.

Similar to MAS-CommonKADS, MESSAGE uses ontologies as the representation mechanism for modelling application's domain knowledge and agents' local domain related knowledge. Thus, it exercises the use of ontologies to support system analysis and agent knowledge modelling. However, unlike MASCommonKADS, MESSAGE makes it possible for agent reasoning to utilize ontology-based knowledge at run-time. The specification of agents' behavioural knowledge at design time in MESSAGE refers to the domain-related knowledge of agents (which is modelled in ontologies) as providing the context for, and the input information to, the agents' behavioural knowledge. Nevertheless, MESSAGE does not recognise the importance of ontologies in agent communication. It neglects the requirement of ontology-sharing between the communicating components, and the need for formulating exchanged messages using the shared ontological concepts. MESSAGE also does not exploit ontologies to support interoperability. The potential existence of heterogeneous MAS components and how these components can be interoperated are not discussed.

The extended version of MASE (DiLeo, Jacobs, & DeLoach, 2002) exploits ontologies to facilitate system analysis and agent knowledge modelling, by using ontologies as the representation mechanism for application's domain knowledge and agents' local domain-related knowledge. MASE outperforms MESSAGE and MAS-CommonKADS in that it recognises the essential role of ontologies in agent communication. In particular, it requires the developer to formulate the exchanged messages in term of the concepts obtained from an ontology shared between the communicating agents, through the "datatyping" of the exchanged parameters with these concepts. MASE also exploits ontologies to support interoperability. It considers the case of agents committing to heterogeneous ontologies (e.g. when the agents wrap around heterogeneous information sources) and highlights the need for ontological mappings between these local ontologies. MASE' support for reusability is thus enhanced, since it allows the legacy (heterogeneous) system components to be reused. However, the benefits of ontologies to agent reasoning cannot be realised in MASE, since MASE does not address how agents' behavioural knowledge (such as agents' plans and actions) relates to agents' ontology-based knowledge. Without an explicit indication of this relationship, MASE cannot illustrate whether, and how, the agent reasoning process can utilize the ontology-based domain knowledge.

In PASSI, ontologies are used in system analysis and agent knowledge modelling to represent the application's domain knowledge and agents' local domain-related knowledge. The importance of ontologies to agent communication is also acknowledged by PASSI. The developer is required to identify, for each agent conversation, the ontology that needs to be shared by the communicating agents, and to define the exchanged messages in term of the shared ontological concepts. However, PASSI fails to provide clear support for the use of ontology-based knowledge by agent reasoning at run-time, since no reference to ontologies is made during the specification of agents' problem-solving knowledge.

Even though the above four AOSE methodologies excise the use of ontology in their MAS development process and product, they do not comprehensively acknowledge and implement all of those diverse roles of ontology in MASs, namely those identified in Section 2.3.4. More specifically, although all four methodologies exploit ontology to facilitate their system analysis and agent knowledge modelling activities, none of them, can illustrate the use of ontology to support interoperability, reusability, agent communication and agent reasoning altogether by itself.

Table 2-1 summarises the steps in all the methodologies reviewed. The four methodologies that utilise ontology are indicated by step 5, the utilisation of domain concepts. The extent of the support for ontology discussed earlier is summarised in Table 2-2.

Steps	MASE	MASSIVE	SODA	GAIA	MESSAGE	INGENIAS	BDIM	HLIM	MEI	PROMETHEUS	PASSI	ADELFE	CASSIOPEIA	CoMoMAS	MAS-CommonKADS	TROPOS
1. Identify System Functionality	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X
2. Specify use case scenarios	X					X		X	X	X	X	X			X	
3. Identify roles	X	X	X	X	X	X	X	X			X		X			
4. Identify agent classes	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
5. Model domain concept utilisation	X				X						X				X	
6. Specify acquaintances between agent classes	X	X		X	X	X	X	X	X	X	X	X	X		X	X
7. Define interaction protocols	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X
8. Define content of exchanged messages	X				X	X	X	X		X	X	X			X	X

18. Instantiate agent classes	17. Specify agent inheritance and aggregation	16. Specify agent-environment interaction mechanism	15. Model MAS environment	14. Specify organisational Structure/inter-agent authority relationship	13 Specify system architecture	12. Define agent behaviour Constructs	11. Define agent information structure	10. Specify agent architecture	9. Specify agent communication language
X							X	X	
		X	X	X	X			X	
			X						
X	X		X	X		X			
X			X	X	X		X	X	X
		X	X	X	X		X		
X	X					X	X		
				X		X	X		
		X					X		
X		X	X		X	X	X	X	
					X	X	X	X	
	X	X	X		X	X	X	X	X
				X					
				X			X		
X	X		X	X	X	X	X	X	
			X	X			X		

19. Specify agent instance deployment	X									X						

TABLE 2-1: SUPERSET OF STEPS IDENTIFIED IN AOSE METHODOLOGIES

Extent of ontology support	**MAS-CommonKADS**	**MESSAGE**	**MaSE**	**PASSI**
Ontology domain model	Expertise model includes the domain knowledge	Domain model relevant to the target application only	Ontology used as representation of the application's domain knowledge and used as agent domain related knowledge	Apart of Agent Society Model
Ontology communication model	No	No	No explicit model – exchange messages are formulated in terms of concepts	Yes – exchanged messages are in terms of shared ontological concepts
Notation used for ontology	CommonKADS notation	Extended UML	UML	Class diagrams Object

				constraint language
Support for Use of existing generic ontology	No	No	No	No
Defined ontology structure	No	No	No	No
Ontology reasoning at runtime	No	Yes	No	No

TABLE 2-2: METHODOLOGIES WITH SUPPORT FOR ONTOLOGY

2.4.3 Implementation for reviewed methodologies

The AOSE methodologies show that there is a conceptual level for analysing the agent-based systems, no matter the agent theory, agent architecture or agent language. The lack of standard agent architecture and agent programming languages is a problem for the implementation of these methodologies. Since there is no agent architecture, the design of these methodologies needs to be customised for each agent architecture. As a result very little is mentioned about the agent architecture and implementation. This problem is difficult to address without proper standards. The examples presented in Chapter 3 will be using the de facto standard of Foundation for Intelligent Physical Agents (FIPA) as a guideline.

2.5 Summary

This chapter has defined the terms “Agent”, “Multi-Agent System” and “Ontology”. It also discussed the potentials of Agent Technology and MAS, and the benefits of ontology to MAS development and MAS operation.

This chapter also provided a review of some existing AOSE methodologies for MAS analysis and design. It describes each methodology and highlights the general limitations of each method. These limitations include those relating to the general analysis and design activities of MAS, and those relating particularly to the support for ontology-based MAS development. Limitations also extend to the lack details on agent implementation, system architecture and deployment. Although this is majorly due to a lack of standardisation, the reviewed methodologies do no mention any details of implementation specific to any architecture.

The reviewed AOSE methodologies do not mention the splitting of tasks and roles. Although it is assumed that users of these methodologies will be the designers and developers of the agent system, we cannot assume that agent developers possess expertise knowledge of the domain of every application they are developing. Some roles can be done by one person. However, it is useful to explicitly state the roles for tasks of a methodology to allow greater flexibility in planning.

These limitations provide the motivation for the MOMA methodology presented in the next chapter.

Chapter 3. Design Methodology for Ontology-based Multi-Agent Applications (MOMA)

3.1 Introduction

This chapter will present the MOMA methodology. MOMA is intended for use by domain experts for creating smart agent applications without needing to have intricate agent development knowledge. MOMA attempts to do this by having the domain knowledge and business logic specified in the ontology as much as possible. Together with guided procedures and a set of tools, MOMA aims to distinguish the roles of its users and address some of the limitations of existing AOSE methodologies regarding ontology support.

In the last chapter we established the usefulness of ontology and multi-agent systems, in particular the benefits of ontology-based multi-agent systems. We explored AOSE methodologies and their usefulness in designing multi-agent based systems. We saw that there was very limited support for ontology in these methodologies and even lower support for ontology at runtime. These methodologies also do not provide details on architecture specific implementation as a part of the methodology. Although this cannot be seen as a limitation per se, developers using this methodology will need to look elsewhere to find methods of implementation depending on the agent architecture they are going to use.

The Design Methodology for Ontology-Based Multi-agent Applications (MOMA) presented in this chapter intends to address the above mentioned limitations to some extent. MOMA will try to:

1. Provide a structured meta-model for the development of ontology for agent application development. An established model is essential for the further support of reuse and sharing.
2. Allowing the ontology model to define behaviour of agents. In a sense, we are trying bring the business logic from the underlying agent implementation to an abstract conceptual level.
3. Facilitate the use of tools to drive development, conceptual testing and implementation of ontology for agent systems. Tools are essential throughout the MOMA process.
4. Provide support for reuse and sharing of the developed ontology. Although ontology inherently supports reuse and sharing, without the establishment of some kind of modularisation and structure, reuse and sharing will be very limited.
5. Distinguish the roles of domain expert and agent developer in the development process of MOMA. We will be assuming that the domain expert does not have expertise in agent development and that the agent developer has very little knowledge of the domain.
6. Work towards a methodology that can be used by domain experts (and researchers) without the expertise of an agent developer.

3.1.1 Scope and Limitations

MOMA is not a formal and comprehensive methodology and has some limitations. Before any use or evaluation, this section will detail some of the foreseeable limitations of MOMA. This section will also scope the coverage of this thesis on the MOMA methodology.

Firstly, this thesis does not address the design of agents and agent “societies”, nor the interaction and behaviour of agents with established agent theories. There will also be minimal discussion of agent implementation. The scope of this thesis will focus on ontology development at the core. The AOSE methodologies reviewed in Chapter 2 used agent theory for the designing and development of agents and multi-agent systems. Further work will be required to incorporate these methodologies with MOMA. Established agent theories will allow MOMA to ultimately translate the ontology into an agent application. However, this will not be covered in this thesis.

MOMA uses the Java Agent DEvelopment Framework (JADE), an agent framework implemented in Java. This limits MOMA to a single architecture. However, this also has an advantage. By limiting to a single architecture, MOMA can detail some of the implementation and deployment of the agent system itself.

Although MOMA introduces support for adding logic in ontology through rules and axioms, it does not contain a methodology to formulate these from the requirements.

The ontology exists at a conceptual level at design time only. Once the ontology is complete it will be consumed in the implementation process and will be embedded with the agent code at runtime. This means that changes in the ontology structure will result in a proportional change in the agent code.

Ultimately, we envision MOMA being a part of or integrated with a formal and complete methodology that allows design and development of both ontology and agents for the construction of ontology-based multi-agent systems in any architecture.

3.1.2 Overview of MOMA process

MOMA is driven by ontology development. MOMA methodology can be broken down into two main parts: Ontology Development and Agent Development. Ontology Development is performed by the domain expert. This is where the ontology of the world is modelled. The resulting ontology is then used in the Agent Development part. The agent development part is performed by the Agent Developer. Agent Development involves the implementation of the world modelled by the ontology through the use of agent theories. The result of the methodology (the two parts) is an agent application, which can then be tested and used. Any changes that need to be made to the agent application will result in beginning the process again. The process is illustrated in Figure 3-1.

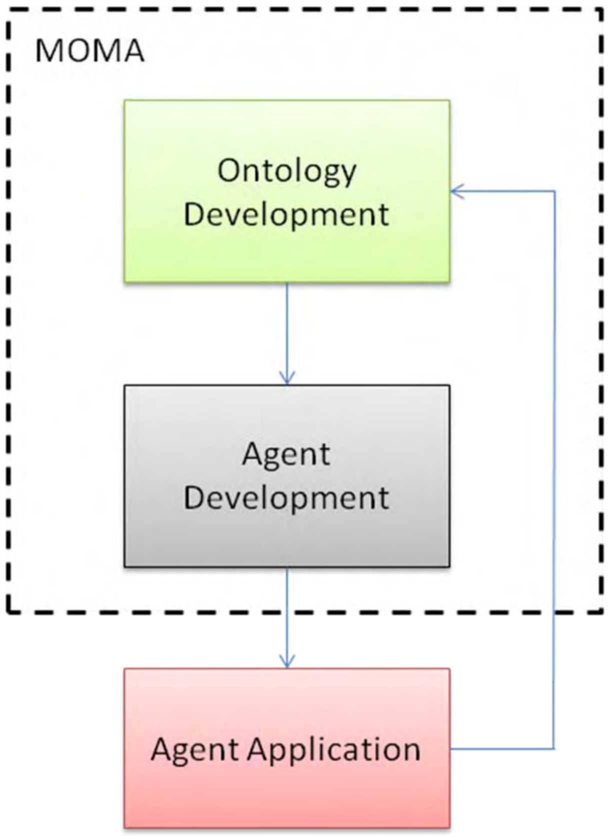


FIGURE 3-1: PARTS OF MOMA

Due to the magnitude of the topics and research areas that can be covered for the components of the methodology, the scope of this thesis will only be focused on the Ontology Development part. The Agent Development part will be briefly explained and possible processes proposed. However for the purpose of implementation and evaluation, the Agent Development part will be treated as a black box parts where only input and output is known.

The rest of the chapter will be split into the two components, Ontology Development and Agent Development, in Sections 3.2 and 3.3 respectively. This is followed by a summary in Section 3.5.

3.2 Ontology Development

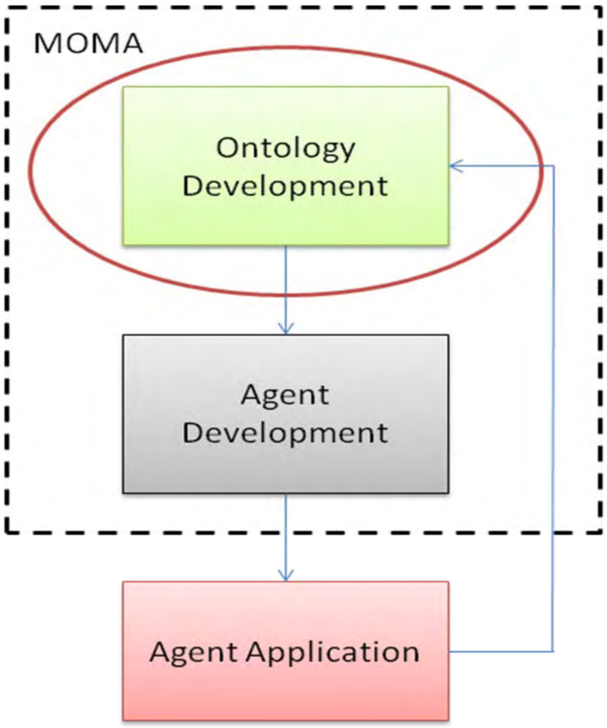


FIGURE 3-2: ONTOLOGY DEVELOPMENT PART OF MOMA

This section details the Ontology Development part of the MOMA methodology (outlined in red in the figure above as a part of the overall process).

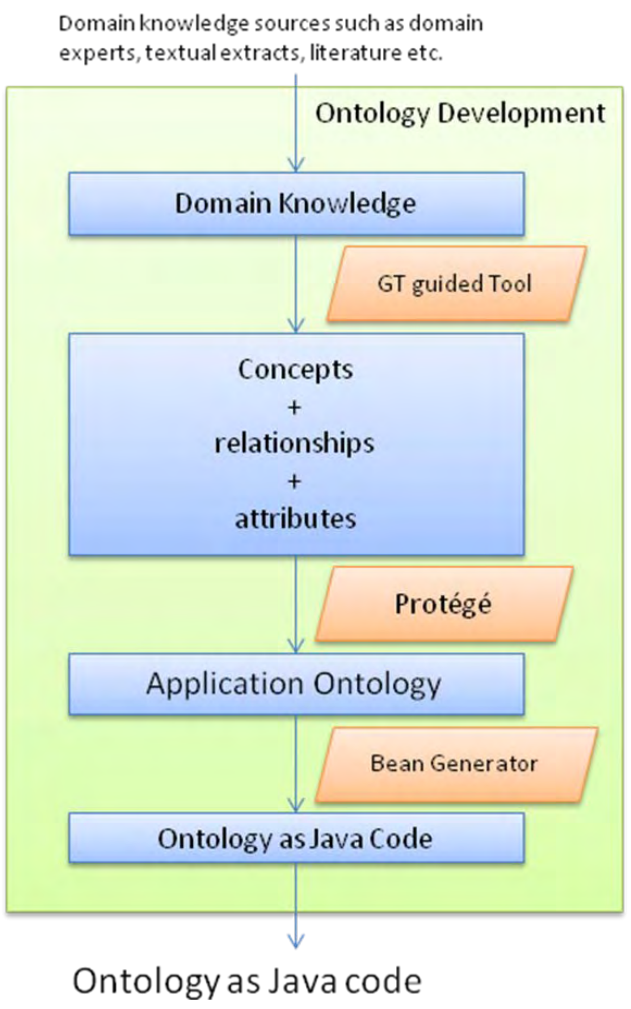


FIGURE 3-3: ONTOLOGY DEVELOPMENT PART

MOMA concentrates on the development of ontology for agent implementation. The purpose of the Ontology Development part is to model the domain knowledge and application world as ontology and output it in the form of code which can be integrated and implemented by the Agent Development part. The modelling is performed by the domain expert. This means that the domain knowledge is explicitly modelled (in the form of ontology) so that it does not have to be defined in lower level code. Conceptually, this also separates some of the logic from the underlying agent code. Figure 3-3 above shows the states of domain knowledge as it transitions into application ontology as java code. In this diagram, the tools are used to make each of the transitions. Initially we have the input, sources of knowledge that are consolidated into the domain knowledge. Using the Ground Theory guided tool, concepts, their relationships and attributes are then extracted from the domain knowledge. These concepts are then used to for modelling into the domain ontology through the use of Protégé ontology development IDE. This ontology can then be translated into Java code through a protégé plug-in called Bean Generator. The output of the entire component is the Java code that represents the domain ontology. The concepts identified may also be agents or agent actions. For example, “Trader” and “Buy” are concepts which can be implemented as agents and agent actions. The domain expert does not need to know this.

To achieve this, the Ontology Development Component is broken down into three main steps:

1. **Concept Identification** – domain expert identifies the concepts, their relationships and attributes about domain of the intended application.
2. **Ontology Modelling** – domain expert models the domain in which the application exists.
3. **Code Generation** – code is generated from the ontology to be implemented in Agent Development part.

3.2.1 Step 1: Concept Identification

The purpose of this step is let the domain expert identify concepts, their relationships and attributes in the domain and world in which the agent application will exist. The input of this step is domain knowledge. This source of knowledge could be in the form of text extracts, literature or even the domain expert themselves. A list of concepts and their relationships and attributes will be the outcome of this step. This knowledge will be used in the next step when modelling the ontology.

Possible agents and agent actions may also be identified as concepts in this step. However, the domain expert will not need to be aware of this as agents will be chosen and implemented in the Agent Development part.

Identification of concepts for the purpose of ontology modelling can be a very time-consuming task. It also follows a very implicit and intuitive process. To make it easier for domain experts (who might not have expertise in knowledge engineering), a more methodological approach is needed. Hence, for the purpose of identification of concepts and relationships, MOMA is guided by the principles of Grounded Theory (GT) (Strauss 1994; Strauss 1998). GT facilitates the production of core categories and relationships from data through a systematic method of constant comparison where new data is continuously compared to existing data. Although GT originates in the social sciences; it has been proven to valuable when applied to ontology construction (Kuziemsky 2007). The key points are marked with a series of codes, which are extracted from the text. The codes are grouped into similar concepts, in order to make them more workable. From these concepts categories are formed. In the context of ontology, the codes are extracted from requirements and domain information. Concepts and sub-concepts are the results. GT has three systematic coding cycles: open, axial and selective coding, all of which are described below.

Open Coding – involves refactoring, breaking down, examining, comparing, conceptualisation and categorising data to identify discrete concepts, which are the basic units of analysis in grounded theory (Strauss 1998). Once concepts are identified they are grouped together to establish preliminary categories. Open coding is continuous and as new data or knowledge is gathered and concepts and categories are identified they go through a continuous cycle of comparison to existing concepts and categories in a process called constant comparison.

Axial Coding **–** extends the initial level concepts and categories from open coding by establishing connections in new ways between categories and sub-categories (concepts and sub-concepts in context of ontology) (Strauss 1998). During axial coding Strauss recommends using a "paradigm model" that establishes a framework by linking data by condition context, action/interaction and consequences (Strauss 1994). This "paradigm model" will be our extended ontology meta model.

Selective coding – involves consideration of the multiple concepts and sub-concepts that emerge from the axial coding and identifying one or two core categories to which all the other child concepts or sub-concepts relate (Strauss 1994). The core concepts become the means for building a conceptual from which to develop the ontology.

GT is a general methodology that uses an interpretive approach for deriving theory. Although we describe our approach as GT guided, we are only using some of the principles from GT. Our approach is derived from the works of (Kuziemsky 2007).

Data collection

GT does not specify where the data should come from. However, it is important to identify source of where knowledge and information is coming from. This is because we need to know what format the data will be in. For example, dealing with text from literature and an interview with a domain expert will require different methods to extra the data. Hence our approach will introduce data collection as a first step before the coding process. Collecting data will involves identifying possible sources of knowledge. The source of knowledge will then be converted to a common format and centralised. The preferred format is text (which we have built a tool for), but other formats in audio or visual will also work. For example, we have literature as one source and domain expert knowledge as another. Since literature is already in text format, we do no need to convert. For domain expert however, one way of extracting knowledge is in the form of an interview or discussion, during which notes can be taken. These notes can then be easily converted to text (if they are not already in text).

Opening coding or Initial concept gathering

After we have centralised all our source data and converted them into a similar format, we can start indentifying the preliminary concepts. This stage also known as open coding, involves going through the data (text) that was gathered and identify relevant concepts (open codes). These are usually keywords or important domain related terms. Open coding is continuous and as new data or knowledge is gathered and concepts and categories are identified. For example, in the domain of Finance, Stock, Equity or Shares may be relevant concepts in right context.

Axial Coding or concept refining

Once our initial concepts have been identified, we will refine these concepts through axial coding. Axial coding in GT uses a "paradigm model" that establishes a framework by linking data by condition context, action/interaction and consequences (Strauss 1994). Because we are using GT for the identification of concepts for ontology construction we will use an ontology paradigm model. We will link the concepts using properties or attributes of the concept and relationships between concepts. This means that we will refine concepts in terms of their properties and relationships. For example, we have Stock, ASX Code and Portfolio. Refining concepts will be broken down into two stages. First we identify the properties. This is done first because relationships are normally not identified as initial concepts and we will need to add them in later. In the example earlier, we can see that ASX code can be a property of stock. For identification of relationships, the domain expert will need to identify the relationship first and then connects the two concepts using that relationship. For example, Stock is a part of portfolio. The domain expert needs to identify the relationship "apart" first and then link the two concepts. There might also be a inverse relationship. i.e. Portfolio "contains" Stock. This can be treated as another case of axial coding.

The output of axial coding stage is a list of concepts, their properties and relationships. The figure below illustrates the overall process.

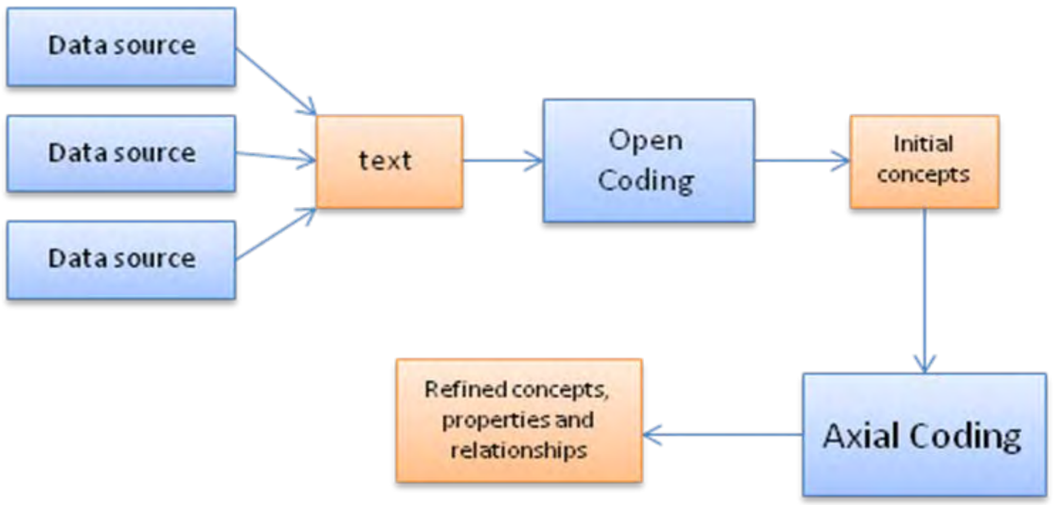


FIGURE 3-4: GT GUIDED CONCEPT IDENTIFICATION

The next stage of GT is selective coding which involves identifying core categories. This is so that concepts can be sub-categorised into a hierarchy. For example, Dollars and Euros could be sub-categorised under Currency. Selective coding however will not be used, as it overlaps with ontology modelling in the next step.

The process of identifying concepts, their properties and relationships can be very subjective, where different domain experts will get different sets of concepts as output. The objective of this however is not to get consistent results, but to provide the domain expert with a structured method for identifying concepts based on principles of GT. A tool for this process is illustrated in Section 3.2.4.1.

3.2.2 Step 2: Ontology Modelling

The purpose of this step is to model the world in which the agent application resides as an ontology. Once again, the domain expert does not require knowing about agents in this step. Their goal is to explicitly model the world and any details that are required to simulate the world in which they want to implement the agent application. The input for this step is the concepts, relationships and attributes identified in the last step. The output of this step is the resulting ontology after being modelled.

3.2.2.1 The generic meta-model

The aim of the ontology meta-model is to assist the creation of ontology for the analysis and design phase of MAS development. The model presents knowledge at several levels as well as from different agent perspective. In particular, it defines the concepts, relations and logic the agents need to know and share about the application domain and its tasks.

The ontology model should be structured in such a way that it provides modularity, separating purpose and task of each different ontology. The model should also be layered in terms of generality, such that each layer captures its own level of knowledge and perspective of the application domain. Layering the ontology also promotes reuse and easier maintenance of the ontology.

A common taxonomy for classifying ontology is by their level of generality (Guarino 1997; Falasconi et al. 1996; Fensel 2001; van Heijst et al. 1997; Gamper et al. 1999). These are broken down into Generic ontology, Domain ontology, Task Ontology and Application Ontology shown in the diagram below.

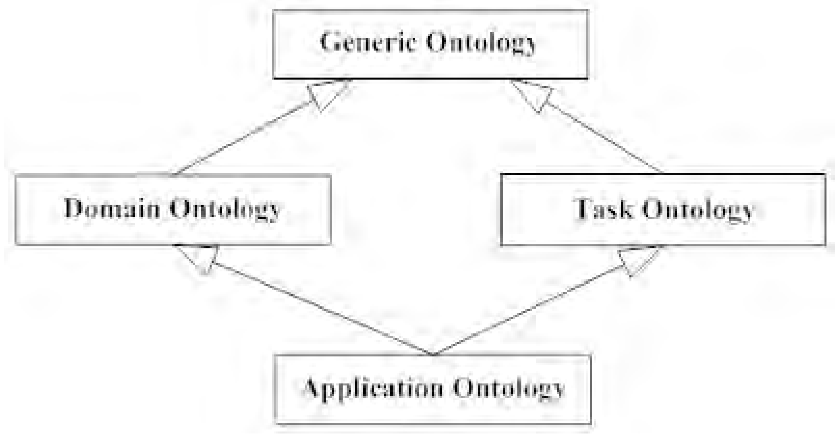


FIGURE 3-5: TYPES OF ONTOLOGY (GUARINO 1997)

Generic ontology or **Foundational ontology** defines very general concepts about the world such as “Time”, “Object”, “Entity”, “Action”, “Event” etc. These concepts are independent of domains and tasks and thus can be reused across applications. One example is CYC (Lenat & Guha 1990), a generic ontology that provides thousands of concepts and millions of axioms for formalizing commonsense knowledge for reasoning. Another example is SUMO (Suggested Upper Merged Ontology) (Niles 2001), the largest formal public ontology in existence today developed by the IEEE Standard Upper Ontology Working Group (http://suo.ieee.org).

Domain Ontology defines concepts that are specific to particular domains. For example, the Accounting domain defines concepts such as “Debit”, “Credit”, “Asset”, “Liability”, etc. While a Medical domain ontology will define concepts such as “Disease”, Symptom”, “Medication”, “Surgery” etc. Domain ontologies may be reused across applications that belong to the same domain. For example, the Unified Medical Language System (UMLS) ontology offers numerous biomedical and health related concepts that can be reused across biomedical and e-Health systems. Domain ontologies can be thought of as extensions of Foundational Ontologies. A purpose suggested by Valente 1995 for the use of domain ontologies is to act as abstract core concepts that play a pivotal role in reasoning, and that they may be a source for constructing special inference services in spatial and temporal reasoning. For example, Valente 1995 developed a formalism and inference engine for reasoning with (legal) norms, as a part of legal core ontology for law. As a general guideline, concepts in the domain ontology are usually nouns.

Task Ontology defines domain independent concepts that are related to generic tasks (e.g. negotiation task, diagnosis task) or problem-solving methods (e.g. propose and revise method, board-game method). For example, a Negotiation Task Ontology may define concepts such as “Offer” and “Reserved Price”, while an Inventory Management Ontology may define concepts such as “Order” and “Inventory Count”. Task ontology can be reused in similar tasks across different applications. As a general guideline, concepts in the domain ontology are usually verbs, actions or related nouns.

Application Ontology defines concepts that are specific to the application. Since each application is typically characterised by a particular domain(s) and a particular task(s), Application Ontology are basically a synthesis of Domain ontologies and Task Ontology that have been specialised to model the application's specific knowledge needs. For example, an application ontology of a Real Estate Agency MAS may define concepts such as "House-offer-price", which is the specialisation of concept "House-price" from a Real Estate Ontology and the concept "Offer" from a Negotiation Task Ontology. Application ontology normally cannot be reused across applications because each different application normally engages in a different combination of domain and tasks as well as numerous custom concepts particular to that application (Ying 2006).

3.2.2.2 The extended meta-model

The ontology taxonomy in the previous section represents a very general view of structure of ontology. However, for the purpose of MAS development this model is not sufficient. Elements that are missing or not explicit in this model are:

Communication ontology – According to FIPA standards, communications consist of the speech acts (Searle 1969). Agent communication languages such as ACL and KQML provide a standard for agent communication. These languages enable an agent to specify the intention and the content of a message as well as the protocol, the language, and the ontology that are used. Without a these elements (protocol, language and ontology) the message would not be understood. So there is a necessity to use association ontology to link these elements. The communication ontology will help define the syntax in which the agents communicate with. For example, an agent A wants to say "hi" to agent B. In this case a formally defined Greetings concept will specify the protocol "inform" (FIPA ACL), the language as RDF, and ontology of the English language containing the word "hi".

The communications ontology should be constructed while modelling the agents and establishing their interactions.

Mediation Ontology - Mediation ontology are used for heterogeneous MASs that makes use of external entities. It provides a layer of abstraction to those external sources. The idea is very similar to that of a Mediator Pattern in OOP for subsystems. These entities can be:

Information sources – Repository of information or data. For example, a database. The mediation ontology should capture concepts and relations that conceptualise the information stored inside the resource or those that needs to be used and accessed by the MAS. It may be derived from the information source's conceptual schema (Guarino 1997).

Application Systems or services – The corresponding mediation ontology will capture all the concepts relating to the operational interface of the source, including all accessible resources and services. In the case of other MASs, the ontology would also include communication protocols for interaction with those agents.

In these systems, only the agents that are directly interfacing with the external sources will need to hold knowledge of the Mediation ontology, since only these agents are required to know about the conceptualisation of the resources' applications. The figure below illustrates how Mediation ontology is used.

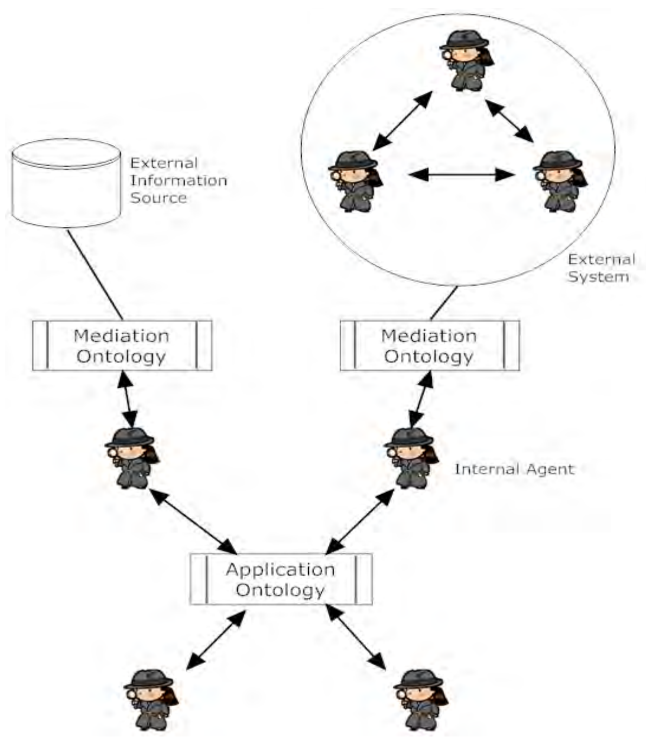


FIGURE 3-6: MEDIATION ONTOLOGY

Specialised Domain Ontology and Specialised Task Ontology– Every MAS will require its own specialised ontology which is modelled in the application ontology. Although the domain ontology should provide the application ontology all the concepts that it needs, sometimes it might be missing details or information that might be specific to the application. This is especially the case when the domain ontology is not developed for the particular MAS and existing ontology is being reused. For example, an animal ontology would have a concept called “Dog” and have attributes such as “Breed”, “Fur Colour” and “Weight”. While an ontology used for a Pet shop Application may require extra attributes such as “Diet”, “Number of walks required per week” etc. This way the specialised domain ontology allows us to add another layer of abstraction and allows us to extend the domain ontology for specific applications while at the same time keeping the domain ontology to be generic as possible so that it can be reused. Similarly, the Specialised Task Ontology has the same rationale. Specialised Ontology will also include action actions.

Rules and Axioms – Ontology in most MAS systems today are used as a medium for communication and understanding between the agents. The reasoning of the agents are designed and implemented within the agents themselves. Another approach to this is to have the rules and axioms to accompany the ontology so that the agents can use it for reasoning through existing reasoning engines without having to implement its own. Although ontology supports reasoning, through languages such as OWL, it is very hard to model complicated logic within the ontology itself. This is the reason for the introduction of rule languages such as SWRL (A Semantic Web Rule Language) (SWRL 2008). A simple use of these rules would be to assert that the combination of the hasParent and hasBrother properties implies the hasUncle property. Informally, this rule could be written as:

$$hasParent(?x,?y) \wedge hasBrother(?y,?z) \Rightarrow hasUncle(?x,?z)$$

From this rule, if John has Mary as a parent and Mary has Bill has a brother then John has Bill as an uncle.

Extending the common structure of ontology with additional elements, we arrive at a more complete ontology model for MAS. In terms of implementation of the ontology for agent consumption, the Generic, Domain and Task Ontology are not required. However, for the purpose of completeness and additional semantics, they are included in the model. The relation between the ontology above is illustrated in the diagram below presented in UML.

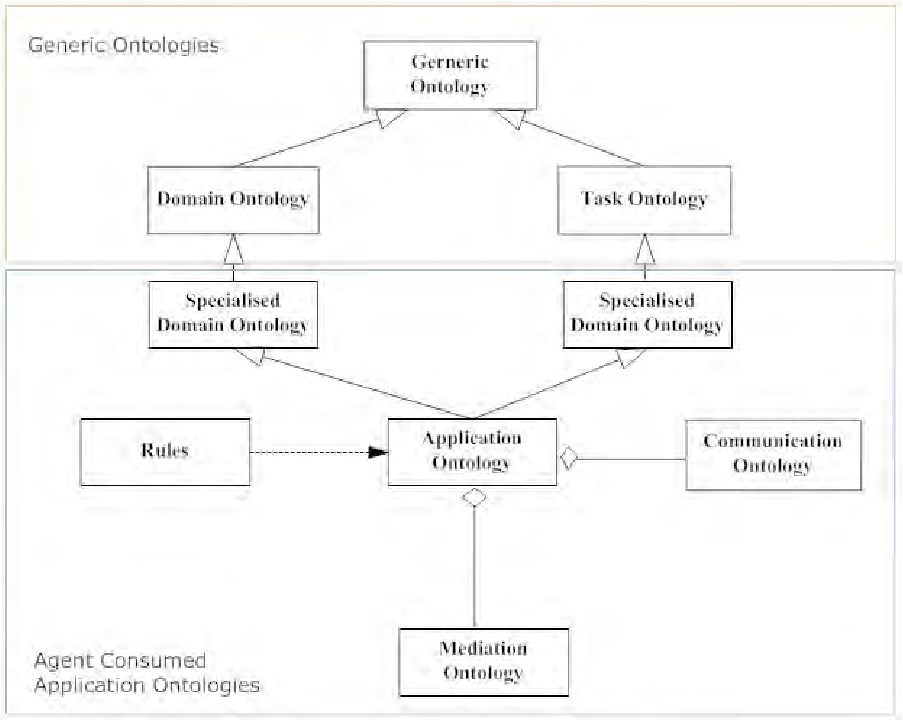


FIGURE 3-7: ONTOLOGY META-MODEL

3.2.2.3 Ontology Development Process

Ontology development methodology is a series of steps set for defining a process in which ontology could be created systematically. Because of the myriad factors such as purpose, intention and domain for ontology, finding a common methodology for the purpose of ontology engineering is difficult. Currently there are several major methodologies that could be considered such as the Mike Uschold and King's skeletal methodology, Bernarass et alia, TOVE and SENSUS and those surveyed in (Cristani 2005), all of which either do not provide details of building steps or are domain specific. The surveyed methodologies also do not consider the ontology for the purpose of agent consumption in the context of AOSE. The scope of this thesis does not cover methodology for the modelling of ontology. An adapted ontology engineering methodology will be considered as future work. Therefore the domain expert should use methodologies mentioned above in conjunction with the guidelines provided below for the modelling of the ontology in the meta-model. For the purpose of the case study and its evaluation, we have used an iterative method based on (Ying 2006).

For the purpose of illustration and documentation, UML is used as a graphical representation of ontology and ontological concepts (see Section 3.2.2.4.1). Ontology is modelled using an Integrated Development Environment (IDE) for ontology modelling called Protégé (see Section 3.2.2.4.2).

3.2.2.3.1 Constructing the ontology

The developer has two choices in this step. He/she can either find and use existing ontology or construct each ontology themselves. The reuse of ontology involves matching the required concepts to existing ontology such as SUMO or Cyc. The developer can then either use the entire ontology or just parts of the ontology where only the required concepts are present. The advantage of using existing ontology is that it provides compatibility which bridges the syntactic gap with existing standardised generic ontology. This will make problems such as the ontology mapping (Kalfoglou 2003) task easier. The disadvantage of using of existing ontology is that it may be time-consuming to identify all the concepts that are required in the existing ontologies.

The developer has the option of creating their own Generic, Domain and Task Ontology. The advantage of this is that it can be much faster and the ontology will only contain relevant concepts. The disadvantage of this is that it becomes a custom ontology. Although the concepts may be generic in nature, the combination of these concepts will have very little reuse value. In essence, it will become a part of the application ontology.

A simple example of Generic, Domain and Task Ontology for the Computer Shop example is illustrated in Figure 3-8 below.

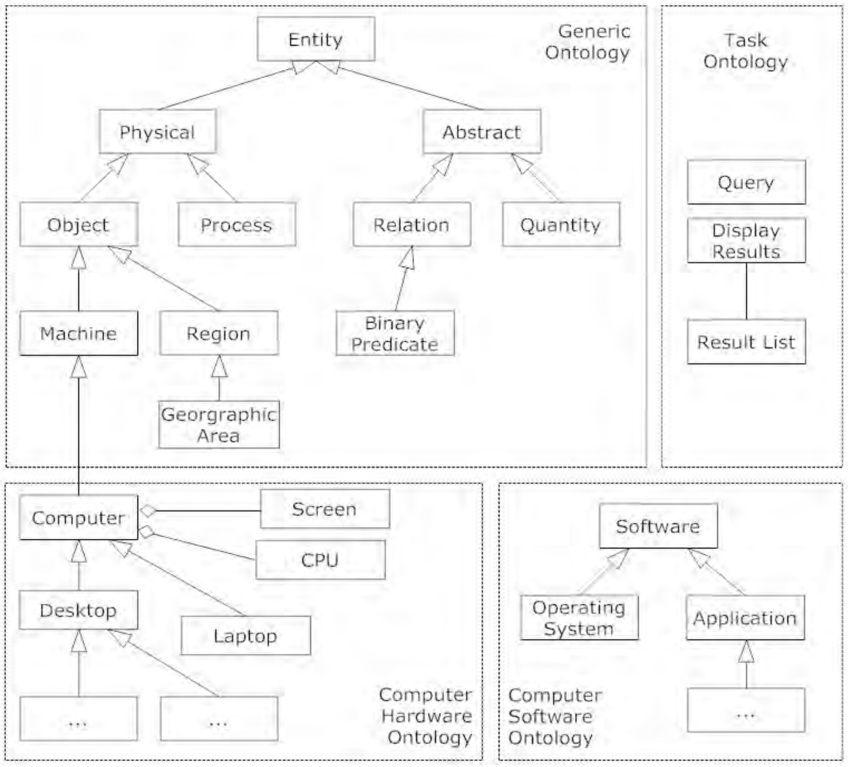


FIGURE 3-8: AN EXAMPLE OF GENERIC ONTOLOGY

3.2.2.3.2 Customising Domain and Task Ontology for Application Ontology

Application ontology needs specific attributes and relationships specific to the MAS. Since Generic, Domain and Task ontology should be as generic as possible, they are not necessarily used by the application ontology. For example, a concept such as “Laptop” may include an attribute called “User Rating” which is only relevant to the MAS application.

3.2.2.3.3 Modelling domain and task concepts

If Generic Ontology has already been constructed then, the Specialised Domain and Task Ontology are simply adding attributes and relationships that are specific to the MAS application. The developer will need to identify the Domain and concepts from the requirements documentation. For example, the requirement may allow the agent to accept user queries and perform searches. This indicates the need to know about the information retrieval domain, which involves concepts such as “Query”, “Keyword” and “Hit”.

Similarly a Specialised Task Ontology may need to reuse or specialise concepts from the Task Ontology. For example, consider a Computer parts inventory management MAS which may involve “calculate stock order”. Specialisation of Task Ontology concepts such as “Calculate”, “Inventory Count” and Domain Concepts such as “Stock” maybe required creating the specialised Task Ontology concept called “Calculate-number-of-stock to order”.

3.2.2.3.4 Modelling specialised attributes of concepts

Attributes in terms of MAS applications have similar functions to that of attributes of Classes in Object-Oriented Programming (OOP). That is, they store information about the particular concept or class. They allow instances of a concept with a unique set of attributes to be created.
After the identification of all the concepts, the developer should inspect these concepts in further detail as some concepts may be attributes of other concepts. For example, “CPU manufacturer” would become “manufacturer” attribute in the concept “CPU”. Below are examples of attributes.

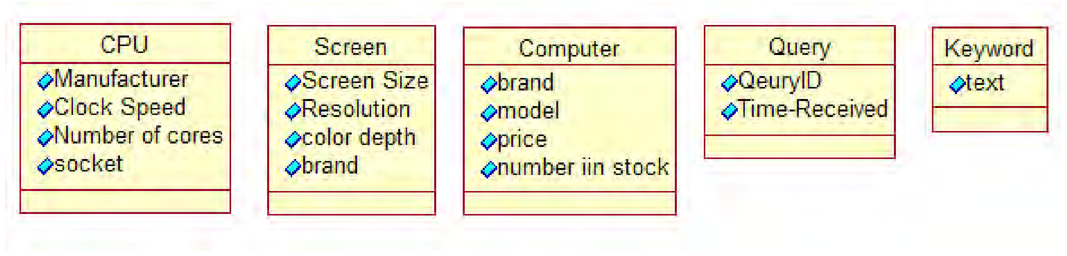


FIGURE 3-9: CONCEPT ATTRIBUTES

3.2.2.3.5 Modelling relationships between concepts

Relationships between two concepts are a type of link between the corresponding instances of concepts. Some intuition and knowledge of the domain may be required in identifying relationships between concepts. Concepts can be related using three UML standard relationships:

- **Generalization:** it permits the generalization/specialization relationship between two concepts that is one of the fundamental operators for constructing ontology.

- **Association:** it models the existence of a logical relationship between two concepts. It is possible to specify the role of the involved entities in order to clarify the structure.

- **Aggregation**: it can be used to construct sets where value restrictions can be explicitly specified; in the W3C RDF standard three types of container object are enumerated: the bag (an unordered list of resources), the sequence (an ordered list of resources) and the alternative (a list of alternative values of a property). We choose to represent a bag as an aggregation without any explicit restriction, while a sequence is qualified by the ordered attribute and the alternative is identified with the only one attribute of the relationship.

Below is an example of relationships between concepts illustrated in UML.

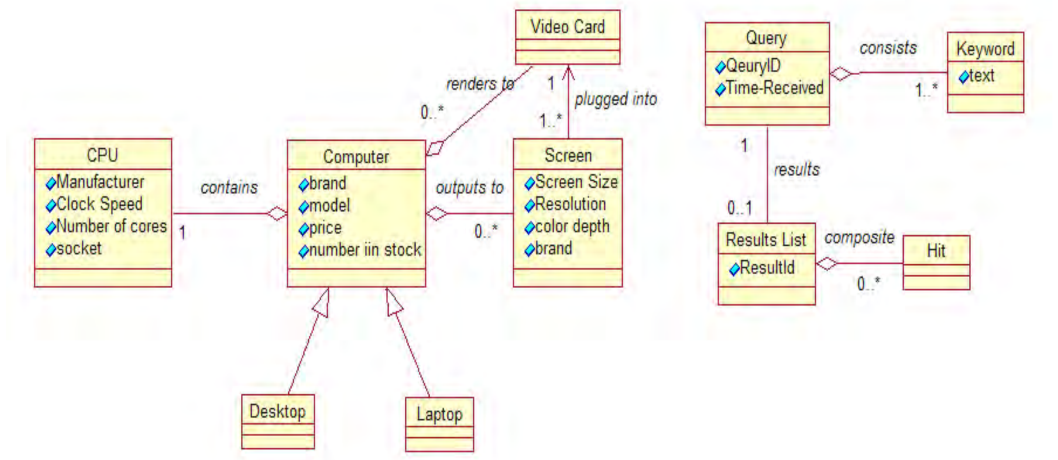


FIGURE 3-10: EXAMPLE OF CONCEPT RELATIONSHIPS IN UML

Note that all knowledge regarding the tasks and knowledge may not be apparent until later stages of the development process (agent development part), thus indicating the need for iterative refinements of both Specialised Domain and Task Ontology.

3.2.2.3.6 Building the Mediation Ontology

If the target MAS contains external resources, the developer needs to extend the Ontology Model to include Mediation ontologies that conceptualise the applications provided by these resources.

- If the resource is a **processing application system** (e.g. a legacy system), its Mediation ontology should capture all the concepts and relations that conceptualise the domains and tasks/services provided by the resource; and

- If the resource is an **information source** (e.g. a database), its Mediation ontology should capture all the concepts and relations that conceptualise the information stored in the resource. This Mediation ontology can be derived from the conceptual schema of the resource, e.g. database schema.

Generally, each resource in a MAS should be conceptualised by a separate Mediation ontology. The developer is referred to other research work on external resource ontology development, e.g. Hwang (1999), Pazzaglia & Embury (1998), Mars et al. (1994), Decker et al. (1999) and FIPA (2001b).

3.2.2.3.7 Specify ontological mappings between mediation ontology and MAS application ontology.

Ontological mappings between Mediation ontologies and MAS Application ontologies are necessary because:

- They enable wrapper agents to translate ACL messages (formulated in MAS Application ontologies' vocabulary) into resource-level queries (formulated in Mediation ontologies' vocabulary), and from resource-level information back to ACL messages; and
- They allow the interoperability between heterogeneous resources. For example, information retrieved from different resources can be integrated using MAS Application ontology.

If each heterogeneous resource is wrapped by a different agent class, each resource's ontology would need to be mapped against the corresponding wrapper agent's ontology. The different wrappers will then communicate with each other to exchange the information/services obtained from the resources. Otherwise if the heterogeneous resources are wrapped by the same agent class, it is most efficient for each resource's ontology to be mapped against the agent class's ontology, which acts as the common ontology.

3.2.2.3.8 Building the Communication Ontology

FIPA standards define agent communication in terms of speech acts (Searle 1969). These are grouped by FIPA in several interaction protocols. FIPA standards also require agent communication to have a language and ontology. The communication ontology defines these concepts in terms of ontology. It can be thought of as a medium used to communicate domain concepts. Agent communication languages such as ACL and KQML provide a standard for agent communication. These languages enable an agent to specify the intention and the content of a message as well as the protocol, language, and ontology that is used. Without these elements (protocol, language and ontology) the message would not be understood. So there is a necessity to use association ontology to link these elements.

Each communication concept will contain at least three attributes: protocol, language and ontology. Protocol refers to the FIPA Interaction Protocols (IPs). IP specifications deal with pre-agreed message exchange protocols for ACL messages. Refer to (FIPA 2002) for details. Language is the representation language of the domain ontology. For example, OWL or RDF could be used for modelling the ontology. Finally, ontology refers to the name of the domain ontology or concept that is referred to in the communication. Figure 3-11 below illustrates communications between an Inventory Agent and a Sales Agent.

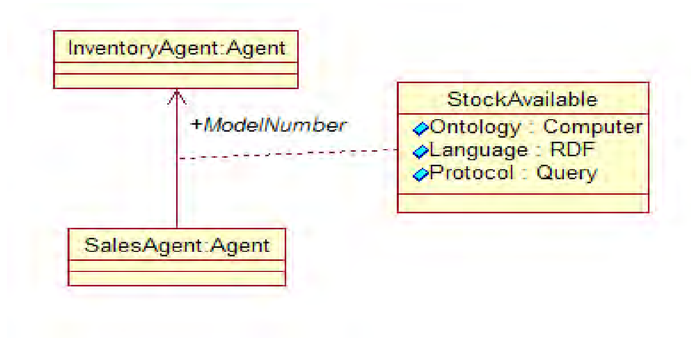


FIGURE 3-11 COMMUNICATION ONTOLOGY EXAMPLE

In the above diagram, the Sales Agent starts a conversation with the Inventory Agent. The conversation contains the Computer ontology, the Query protocol and the RDF language. This means that the Sales Agent wants to perform a speech act based on the FIPA's query protocol in order to ask the Inventory Agent on how much stock is available(number of stocks) provided by the Computer ontology.

The Communication Ontology requires knowledge of agents. Therefore, this will be deferred to the Agent Development Part. However, the concepts such as Sales and Inventory and stock availability will still be modelled in the ontology. Hence the domain expert is not required to be aware of agents.

3.2.2.3.9 Adding logic through Rules and Axioms

Although logic is already modelled into the ontologies through description logic in languages such as OWL, complex business logic is hard to model. For this reason, the use of rules and axioms to accompany the ontology will fill this gap. For example, a simple rule that asserts that Laptops are computers. Laptop(?x) $\Rightarrow$ Computer(?x), can be modelled using the subclass facility in OWL as:

Class(Laptop partial Computer)

or

Subclassof(Laptop Computer)

The Equivalent Semantic Web Rule Language (SWRL) rule for such a rule would be:

Implies(Antecedent(Laptop(I-variable(x)))
Consequent(Computer(I-variable(x))))

For complicated logic however, would be much harder to model in ontology languages such as OWL. For example:

Computer(?x) & (=?y numberOfScreens) (?x) $\Rightarrow$ (>=?y numberOfVideocards) (?x)

This means that a computer must have equal or more video cards than screens.
By modelling business logic in the ontology, the ontology will dictate the behaviour of the agents. This allows knowledge to be specified at a higher level in ontology as appose to low level programming languages used to implement the agents. The modelling of business logic in rules and axioms however is out of the scope of this thesis and will not be covered in detail.

3.2.2.3.10 Specifying Ontological mappings between Application Ontologies

When developing the application ontology the developer should also consider the specification of ontological mappings between ontologies. Ontological mapping is a semantic correspondence between two concepts of two different ontologies (Madhavan 2002). Research in linguistics, logics and psychology has proposed much potential semantic correspondence between concepts (Winston et al. 1987). Winson et al. (1987) presents a taxonomy of semantic correspondence that pertain to the part-whole relationships. Storey (1993) suggested seven major semantic correspondences between concepts: “inclusion”, “possession”, “attribution”, “attachment’, “synonym”, homonym” and “case”. They can therefore adopt whichever semantic correspondence suits that mapping of the target MAS application ontology. However, the developer should consider the following three basic semantic correspondence, which covers the majority of the possible semantic associations between concepts (Parent & Spaccapietra 1998):

Equivalence - this is where two concepts are semantically equivalent. For example, concept “LCD Screen” is the computer ontology is equivalent to the “LCD Display” in the Entertainment Systems domain ontology.

Subsumes – this is where one concept semantically includes another concept, either in terms of whole-part, specialisation or instantiation. For example, the concept “Computer” in the Computer Domain Ontology subsumes “Calculator” in the Electronics Domain Ontology.

Intersects - this is where one concept overlaps partially in semantics with another concept. For example, “Computer System” in the computer ontology intersects with the concept “Home Entertainment system” in the Entertainment Systems ontology.

The related MAS Application ontology can either be mapped against each other, or against a common ontology. Normally, when there are more than two ontologies to be mapped amongst themselves, the second approach should be favoured over the first. The common ontology to be used in the second approach may be one of the existing MAS ontology itself, or built from scratch from existing application ontologies.

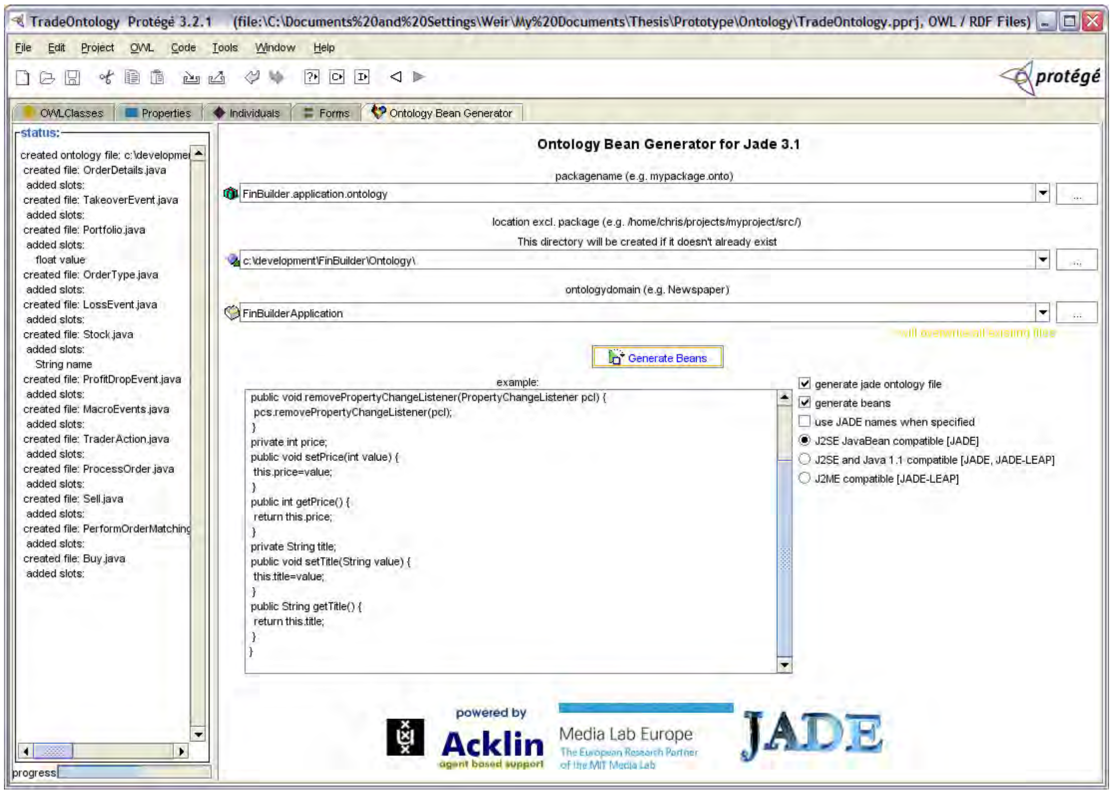
Generating ontological mappings is a time consuming and error prone task. There are several research works in performing this activity. Some of them include: Ehrig and Sure (2004), Kalfoglou & Schorlemmer (2003), Calvanese et al. (2001) and Madhavan et al. (2002). The domain expert may use one of these as a guide for completing this task.

3.2.3 Step 3: Code Generation

The purpose of this step is to convert the ontology into code that will assist agent implementation in the agent development part. Because the agent development part is still work in progress, the sophistication of the generated code from this step is essential for the determination of the processes that will be required in the agent development part.

Bean Generator was used to generate Java code for the ontology modelled in protégé. Below is a screenshot of the generator.

The ontology bean generator plug-in is a Protégé Tab widget which generates java files representing an ontology that can be used with the JADE environment. With the bean generator tool you can generate FIPA/JADE compliant ontologies from RDF(S), XML and Protégé projects.


TradeOntology Protégé 3.2.1 (file:\C:\Documents%20and%20Settings\Weir\My%20Documents\Thesis\Prototype\Ontology\TradeOntology.pprj, OWL / RDF Files)

File Edit Project OWL Code Tools Window Help
protégé
OWLClasses Properties Individuals Forms Ontology Bean Generator

status:

```
created ontology file: c:\developme
created file: OrderDetails.java
 added slots:
created file: TakeoverEvent.java
 added slots:
created file: Portfolio.java
 added slots:
   float value
created file: OrderType.java
 added slots:
created file: LossEvent.java
 added slots:
created file: Stock.java
 added slots:
   String name
created file: ProfitDropEvent.java
 added slots:
created file: MacroEvents.java
 added slots:
created file: TraderAction.java
 added slots:
created file: ProcessOrder.java
 added slots:
created file: Sell.java
 added slots:
created file: PerformOrderMatching
 added slots:
created file: Buy.java
 added slots:
```

progress

Ontology Bean Generator for Jade 3.1

packagename (e.g. mypackage.onto): FinBuilder.application.ontology

location excl. package (e.g. /home/chris/projects/myproject/src/)
This directory will be created if it doesn't already exist: c:\development\FinBuilder\Ontology\

ontologydomain (e.g. Newspaper): FinBuilderApplication

example:

```
public void removePropertyChangeListener(PropertyChangeListener pcl) {
 pcs.removePropertyChangeListener(pcl);
}
private int price;
public void setPrice(int value) {
 this.price=value;
}
public int getPrice() {
 return this.price;
}
private String title;
public void setTitle(String value) {
 this.title=value;
}
public String getTitle() {
 return this.title;
}
}
```

- [✓] generate jade ontology file
- [✓] generate beans
- [ ] use JADE names when specified
- ● J2SE JavaBean compatible [JADE]
- ○ J2SE and Java 1.1 compatible [JADE, JADE-LEAP]
- ○ J2ME compatible [JADE-LEAP]

FIGURE 3-12: SCREENSHOT OF ONTOLOGY BEAN GENERATOR

Bean Generator is fairly simplistic and does not include features for generating Rules and axioms. Bean Generator also does not generate any agent related code. A tool that can generate more agent relevant code and those that can handle rules and axioms can be thought of as future work. Refer to Appendix A2 and Appendix B3 for examples of generated java code.

3.2.4 Tools and techniques

This section presents tools and techniques used for the ontology development process.

3.2.4.1 GT Guided Tool

We have adapted the GT process for identification and grouping of concepts. The identification of concepts is an intuitive and time consuming task that does not have a clear guided process. The aim of this tool is to provide the user with a methodological process to do this. The implementation of this tool is done as a proof of concept. Some features have not been implemented and will be considered future work. Refer to Appendix C1 for implementation source. The process is shown below in Figure 3-13.

It is a linear process where the output of one step is used as the input for the next.

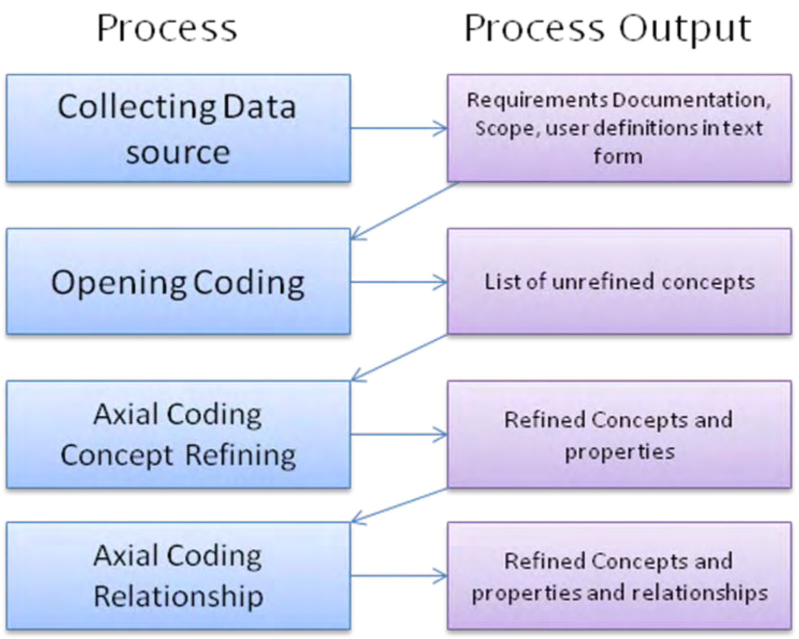

Process
Process Output
Collecting Data source
Requirements Documentation, Scope, user definitions in text form
Opening Coding
List of unrefined concepts
Axial Coding Concept Refining
Refined Concepts and properties
Axial Coding Relationship
Refined Concepts and properties and relationships

FIGURE 3-13: GT GUIDED PROCESS FOR CONCEPT INDENTIFICATION

The process begins with data collection. This can be in the form of requirements documentation, scope or thoughts and even concepts in the entered text. The tool requires some form of text as in input data source. Figure 3-14 shows a screenshot of the interface of the tool that that allows user to input the data source in the form of existing file. The user also has the option to manually input text in the text area. Although the source needs to be entered as text, other sources such as notes from discussions or interviews with domain experts can also be translated to text and entered.

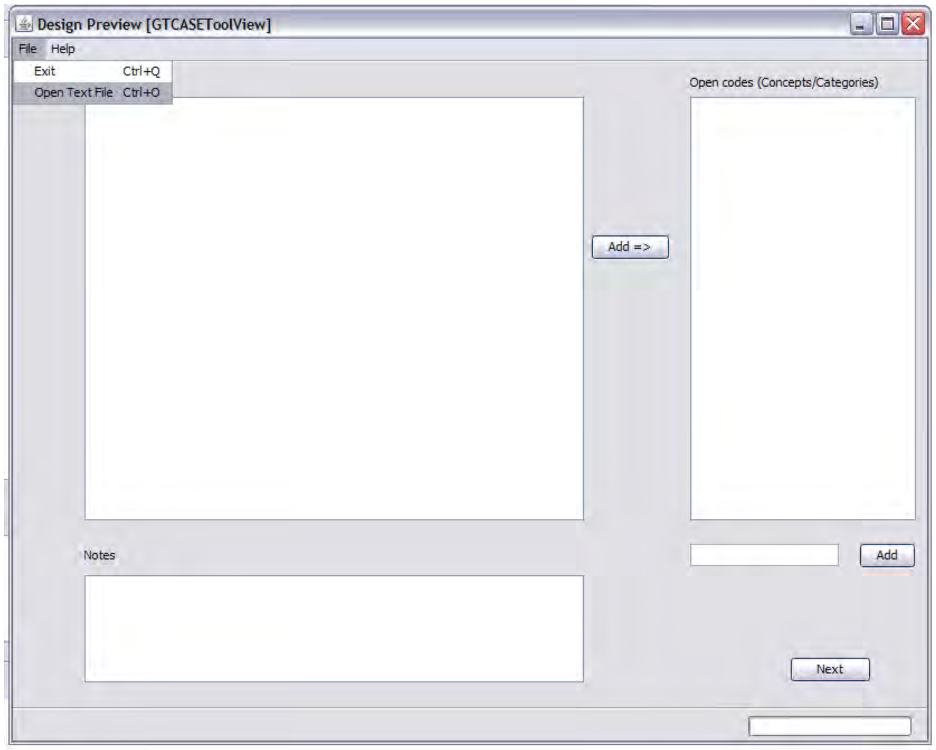
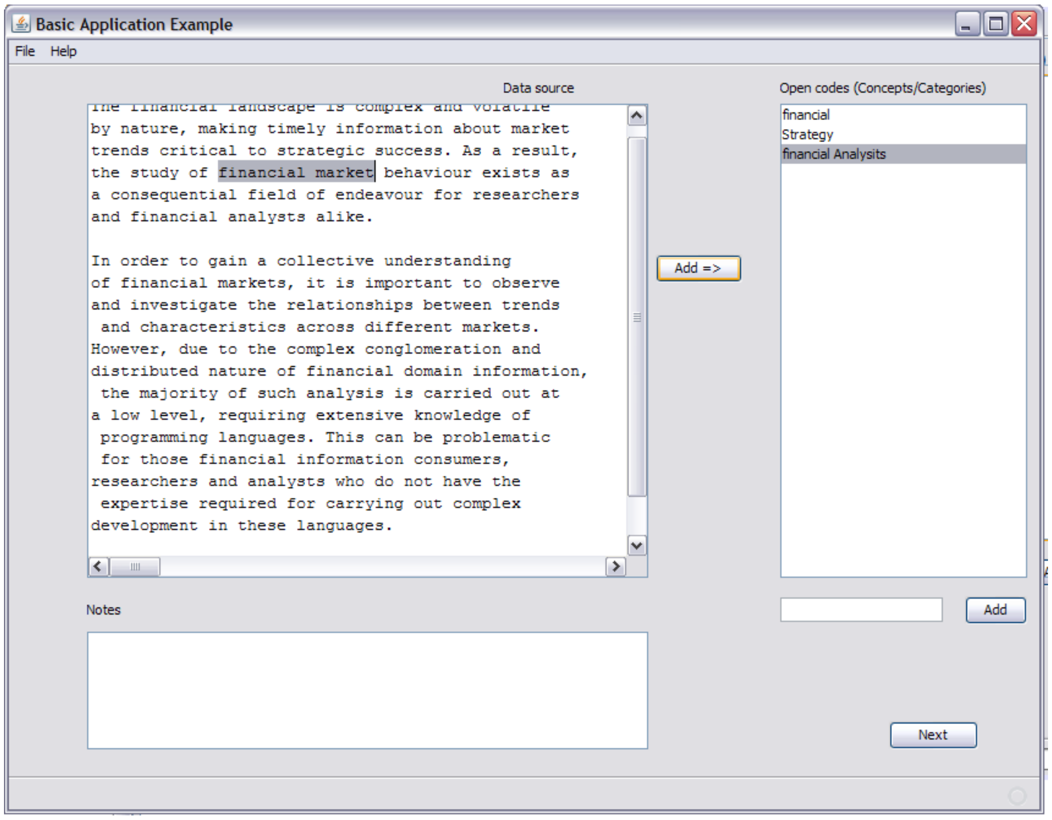

Design Preview [GTCASEToolView]
File
Help
Exit Ctrl+Q
Open Text File Ctrl+O
Open codes (Concepts/Categories)
Add =>
Notes
Add
Next

FIGURE 3-14: GT GUIDED TOOL FOR COLLECTING DATA SOURCE

Once the data source as been entered, the user may go through and select concepts and/or categories from the next step as a part of open coding (Figure 3-15). The open codes are listed on the right hand side. The user may also add concepts that are not identified in the input text. Another feature would be the tool parsing the input text and counting occurrences of keywords and suggesting possible concepts in a separate list. This however has not been implemented yet.



```
The financial landscape is complex and volatile
by nature, making timely information about market
trends critical to strategic success. As a result,
the study of financial market behaviour exists as
a consequential field of endeavour for researchers
and financial analysts alike.

In order to gain a collective understanding
of financial markets, it is important to observe
and investigate the relationships between trends
 and characteristics across different markets.
However, due to the complex conglomeration and
distributed nature of financial domain information,
 the majority of such analysis is carried out at
a low level, requiring extensive knowledge of
 programming languages. This can be problematic
 for those financial information consumers,
researchers and analysts who do not have the
 expertise required for carrying out complex
development in these languages.
```

```
financial
Strategy
financial Analysits
```

FIGURE 3-15: GT GUIDED TOOL FOR OPEN CODING

Once the user is done identifying concepts in open coding, the "Next" button will bring them to the first part of axial coding (Figure 3-16). Figure 3-15 above is a screenshot of the tool where the user is allowed to refine the open codes selected in the previous step. The codes are split into concepts, categories or properties. The user still has the option to add additional concepts and properties. When the concept is selected in the top right, the "concept properties" listed directly below will automatically update with the properties of that concept. This can be edited by adding in concepts using the "Add" button or by selecting from the list of open codes on the left.

The "Next" button will end this step and bring the user to the second part of axial coding. In this step, the user identifies the relationships between each concept. Right now the tool supports one to one relationships (Figure 3-17) and two-way relationships (Figure 3-18). Many-to-many relationships will be considered as future implementations.

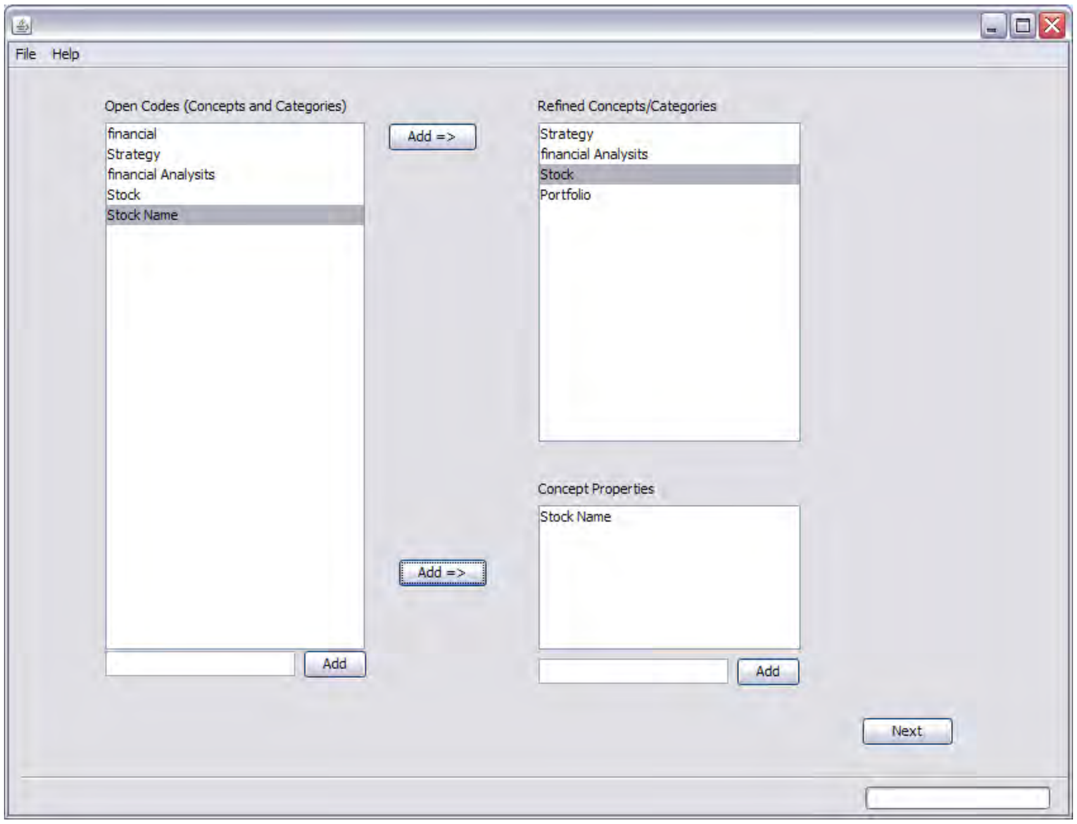


FIGURE 3-16: GT GUIDED TOOL FOR AXIAL CODING AND CONCEPT REFINING

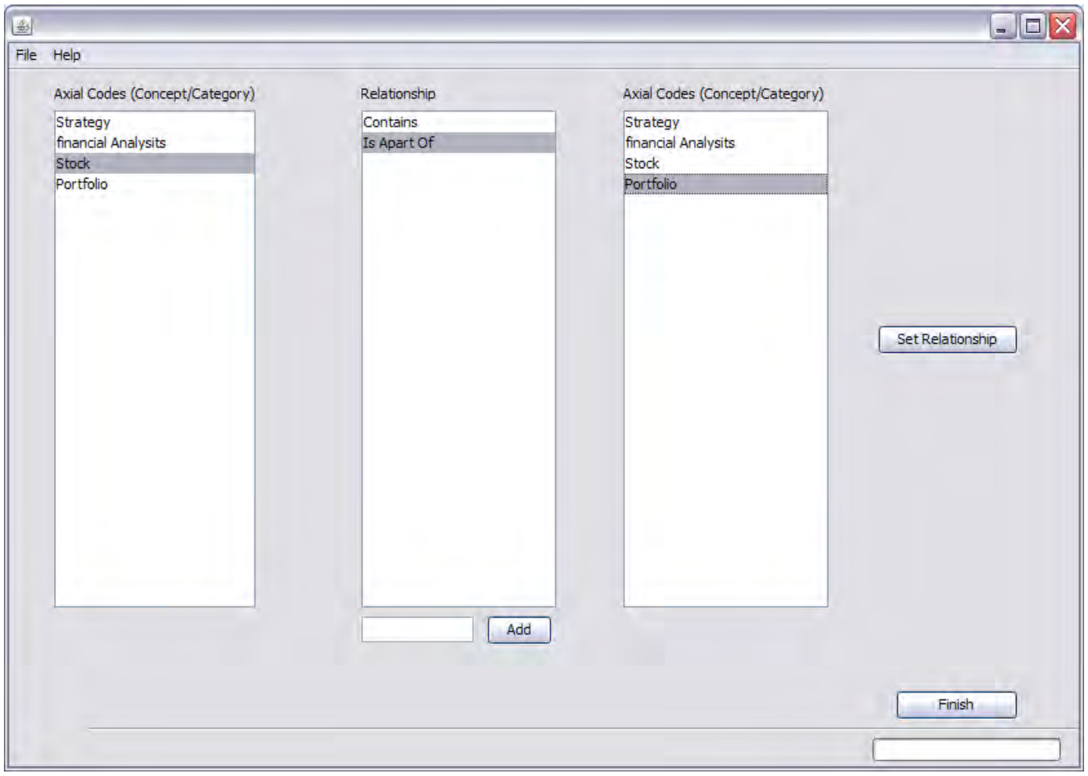


FIGURE 3-17: GT GUIDED TOOL FOR AXIAL CODING AND RELATIONSHIP DEFINITION

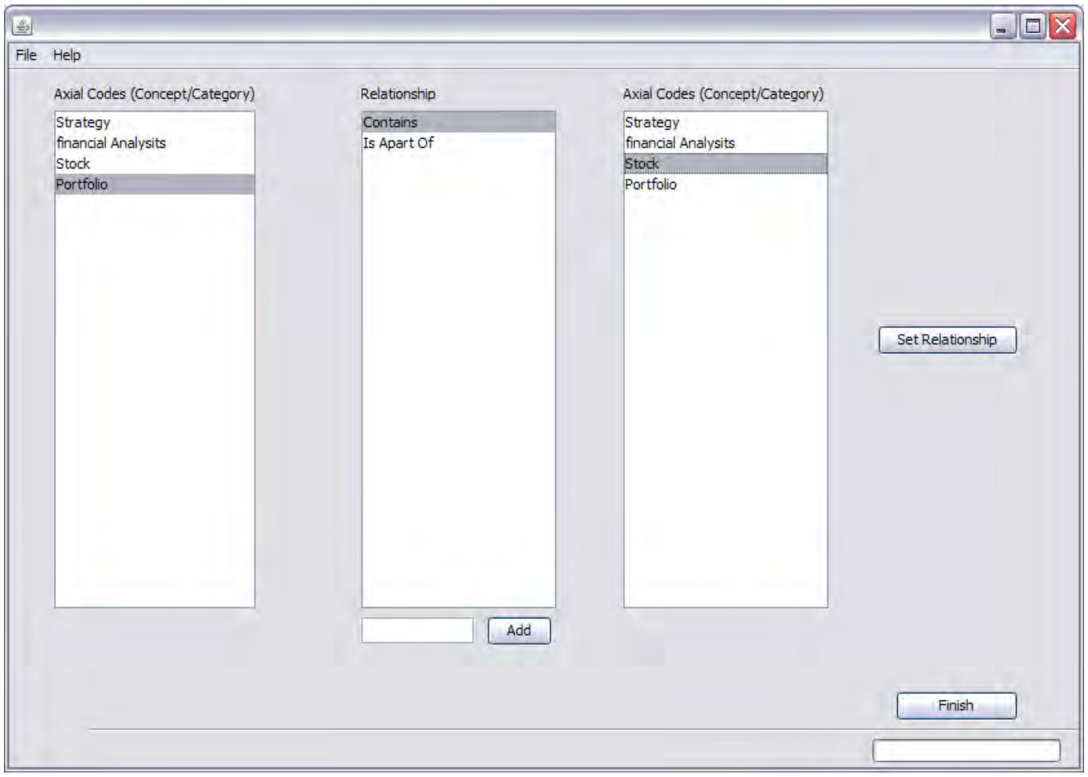
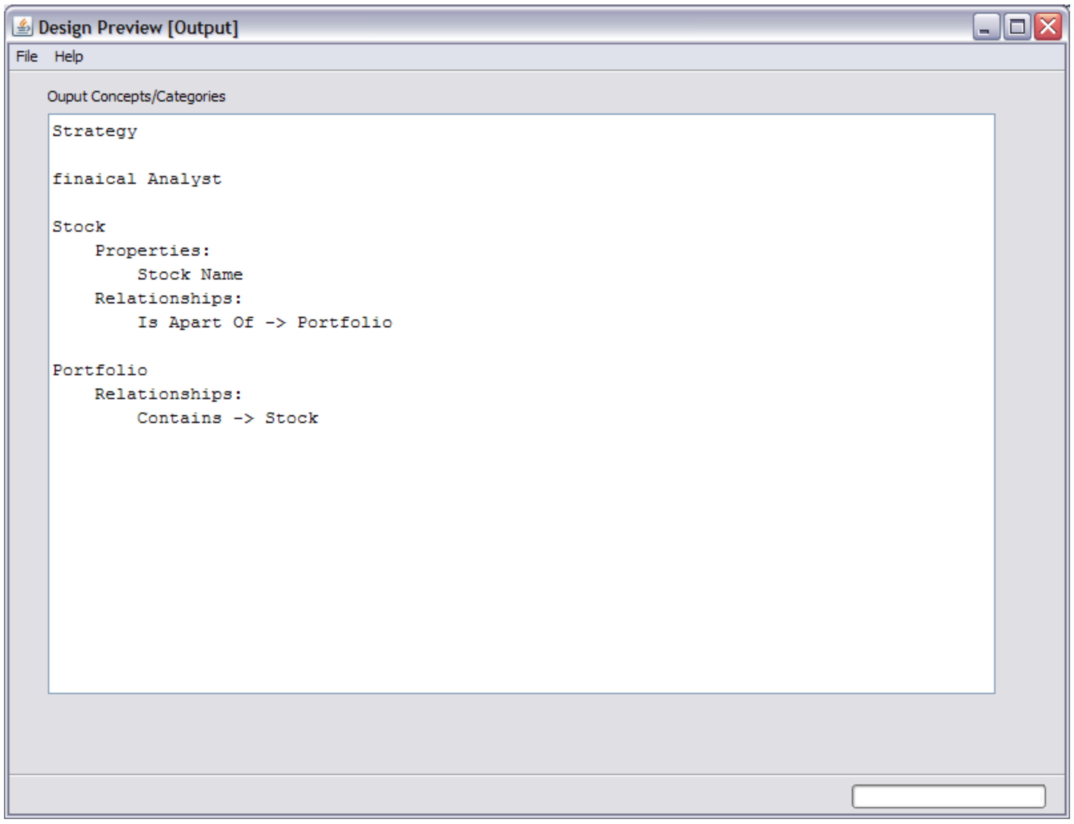


FIGURE 3-18: GT TOOL FOR AXIAL CODING AND RELATIONSHIP DEFINITION - 2 WAY RELATIONSHIPS

Once the axial coding steps are complete, the “Finish” button will bring the user to the result page. This is the output of the concepts, properties and relationships the user has defined from the original text input. This output can then be used for modelling the ontology in OWL or other formats through tools such as Protégé (Section 3.2.2.4.2). A screenshot of the output is shown in Figure 3-19.



```
Strategy

finaical Analyst

Stock
    Properties:
        Stock Name
    Relationships:
        Is Apart Of -> Portfolio

Portfolio
    Relationships:
        Contains -> Stock
```

FIGURE 3-19: GT GUIDED TOOL FINAL OUTPUT

3.2.4.2 Ontology languages

Currently there are many presentation languages for ontologies. For the purpose of documentation, a graphical language for ontology modelling is recommended to facilitate communication and provide a visual aid. Some examples include UML, IDEF5 Schematic Language and LINGO. However, if the graphical language is not powerful enough in terms of expression for the ontology, textual languages such as CycL, KIF, KL-ONE and DAML+OIL can be used. Alternatively, XML based languages such as OWL can used as a visual representation for ontology (Separate visualisation tools will be needed such as Protégé and VisOWL plugin). For modelling in the design and analysis stage, UML and Object Constraint Language (OCL) will be used. Examples in the case studies will also use this.

Using UML, the ontological **concepts** are represented as UML classes. **Attributes** of the concepts are represented by attributes of classes. Operations/methods of the classes are not modelled because ontology only captures the structure of the concepts, not their behaviour (Bergenti and Poggi 2002).

Relationships between concepts are represented as relationships between classes. UML allows for the representation of the following types of relationships between concepts (Object Management Group 2003):

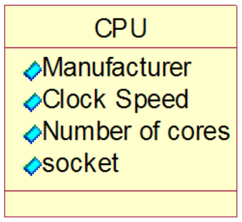


FIGURE 3-20: CONCEPT AND ATTRIBUTES IN UML

Generalisation – concept A is a type of concept B. For example, "Laptop" is a type of "Computer".

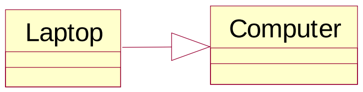


FIGURE 3-21: GENERALISATION OF CONCEPT IN UML

Aggregation – concept A is a part of concept B. For Example, "CPU" is a part of "Computer".

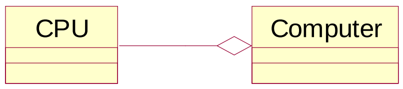


FIGURE 3-22: AGGREGATION OF CONCEPTS IN UML

Composition – a stronger type of aggregation. When A is part of concept B, then A only exists if B exists. For example, "Single Result" is a part of "Search Results".

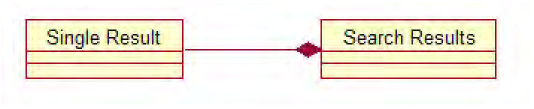


FIGURE 3-23: COMPOSITION

Association – when concept A is related to concept B. An association relationship may be described by a predicate, which is basically an ontological concept itself (Bergenti & Poggi 2002). For example, “Employee” is related to “Company” through “Job”.

Each relationship between concepts should be annotated with **cardinalities**, which indicates the number of potential instances of each concept that may be involved in the relationship.

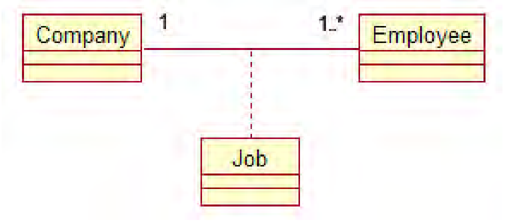


FIGURE 3-24: ASSOCIATION WITH CARDINALITY IN UML

FIGURE 3-25: LEGEND OF CARDINALITY

Axioms, rules or other **assertions** that specify constraints on the ontological concepts, attributes and relationships are modelled by OCL. OCL constraints are represented as notes in UML.

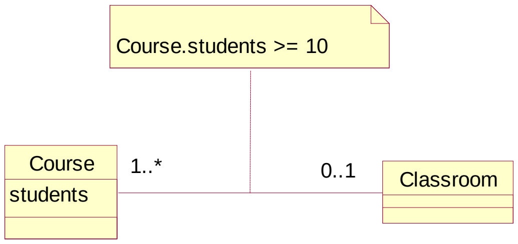


FIGURE 3-26: CONSTRAINS IN UML

Ontological mappings use an extension of Dependency or Instantiates of UML. The corresponding semantic correspondence (equivalent, subsumes or intersect) is used as a label. If the mapping is bi-directional, then the arrow can be double-headed.

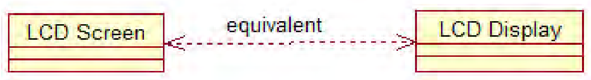


FIGURE 3-27: ONTOLOGICAL MAPPING IN UML

3.2.4.3 Ontology modelling tool

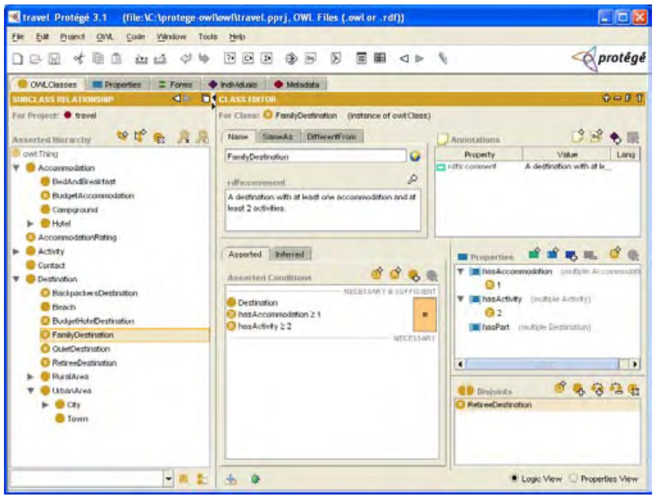
Protégé – OWL

The Protégé-OWL editor is an extension of Protégé that supports the Web Ontology Language (OWL). OWL is the most recent development in standard ontology languages, endorsed by the World Wide Web Consortium (W3C) to promote the Semantic Web vision. "An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specify how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms" (see the OWL Web Ontology Language Guide).

The Protégé-OWL editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

Protégé-OWL's flexible architecture makes it easy to configure and extend the tool. Protégé-OWL is tightly integrated with Jena and has an open-source Java API for the development of custom-tailored user interface components or arbitrary Semantic Web services. Below is a screen shot of the Protégé –OWL development environment.



Name: FamilyDestination

rdfs:comment: A destination with at least one accommodation and at least 2 activities.

Property	Value	Lang
rdfs:comment	A destination with at le...	

Asserted Conditions:
Destination
hasAccommodation ≥ 1
hasActivity ≥ 2

Properties:
hasAccommodation (multiple Accommodati
1
hasActivity (multiple Activity)
2
hasPart (multiple Destination)

Disjoints: RetireeDestination

Figure -3.3.9 Protégé-OWL development environment

3.3 *Agent Development*

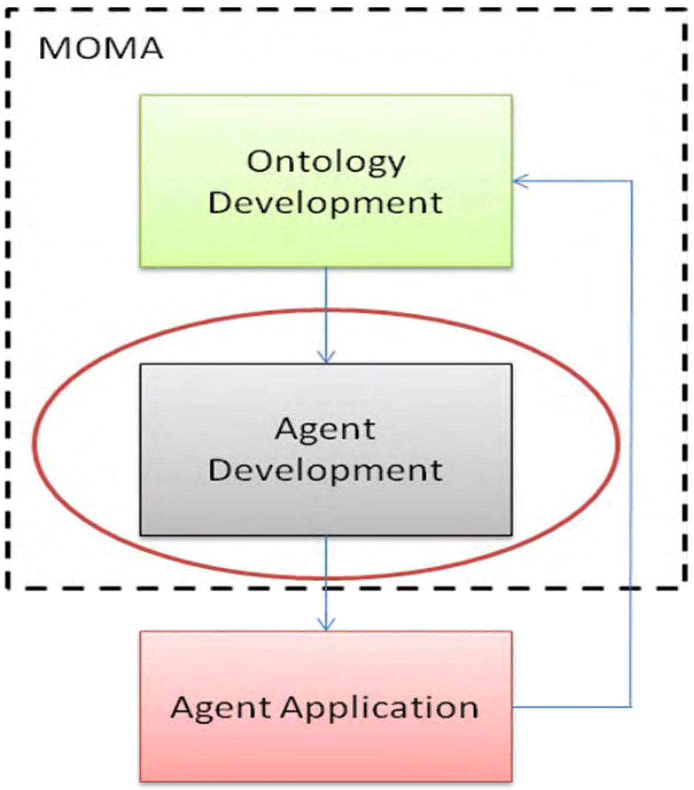


FIGURE 3-28: ONTOLOGY PART OF THE MOMA PROCESS

This section details the Agent Development part of the MOMA methodology (outlined in red in the figure above as a part of the overall process).

The purpose of the Agent development Part is to implement the ontology and code generated from the ontology into the agent application. The ontology should contain the domain knowledge. It is up to the agent developer to identify, design and code the agent "societies".

This part is performed by the agent developer or software engineer. With the intended requirements of the agent application and the code generated from the ontology development part, the agent developer will need to identify the agents as well as define their logic. Some of this may already be generated as code and the agents modelled as concepts. It is the agent developer's task to implement the agent application given the knowledge (in the form of code) from the domain expert.

This thesis does not cover the methodology for designing agents that use those methodologies reviewed in chapter 2.

This part varies in difficulty depending on the sophistication of the code generated in Ontology Development part. For the purpose of the case studies in Chapters 4 and 5, this part will be seen as a black box.

3.3.1.1 JADE agent platform

JADE (Java Agent Development Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. JADE is most used agent platform in agent-related scientific projects. It is available under Open Source License.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching based. The full FIPA communication model has been implemented and its parts have been clearly distinct and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and, finally, transport protocols. The transport mechanism, in particular, is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, and IIOP are currently used, but more protocols can be easily added and integration of HTTP has been already achieved. Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application dependent behaviour of each state of the protocol. SL and agent management ontology have been implemented already, as well as the support for user defined content languages and ontologies that can be implemented, registered with agents, and automatically used by the framework.

In the past work, ontology has had to be manually translated into more redistricted machine readable formats such as XML, database schema, or object oriented schema in order to bridge the communication gap between software agents and the ontology. JADE agent platform can also through Protégé, take advantage of and use the FIPA compliant ontology in Java that is generated directly from Java ontology Bean Generator.

3.3.1.2 Ontology Management

FIPA recommends MAS store ontology on an ontology server(s), which is exclusively controlled by an “ontology manager” agent. Other agents in the system that wish to obtain, access or update ontology would need to communicate with the Ontology Manager.

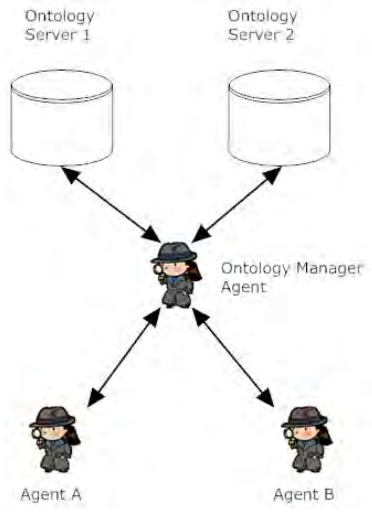


FIGURE 3-29: ONTOLOGY MANAGER

Potential Tasks of the “ontology manager” agent are:

- To perform all necessary reasoning, inferences or ontology-mapping activities to answer ontology-related queries posed by other agents.
- To distribute copies of ontology to authorised agents
- To control the update of ontology
- To inform the other agents of changes in the ontology

The use of a specialised “Ontology Manager” agent is useful in that it helps to relieve the workload from the other agents by taking care of all ontology-related reasoning and mapping activities. It also helps to ensure security by checking whether a particular agent is authorised to obtain a requested ontology.

The “Ontology Manager” Agent is an application-independent component that is generally provided by the implementation framework. FIPA-based platforms such as JACK, JADE, FIPA-OS and ZEUS all provide this. The developer therefore does not have to design one from scratch, but can customise the provided specification of the provided “Ontology Manager” agent to suit the application.

The alternative choice for the developer is to let the agents have direct access to the ontology without using the “Ontology Manager” agent. This will provide a much simpler design. However, the drawbacks are that the system loses access control and management of the ontology. The agents would also need to perform all the reasoning and mapping activities themselves.

3.4 Testing

Testing and evaluation will exist on several levels. The tests can be carried out at different levels of development. The testing and evaluation refers to the testing of the application to be developed and not the methodology itself. The testing and evaluation of the methodology is through case studies in Chapter 4 and 5.

Concept testing

Concept uses tools to verify that the ontology is correct before moving on to implementation. Concept testing involves two steps.

Finding missing concepts – this is can be done while stepping through the scenarios. While going through the scenarios, the developer will check if all the concepts that were used in the scenario that are needed by the agents have been modelled in the ontology. For example, for the scenario used above, while stepping through the scenario, the developer might realise that there is only a concept for reserved price, but not minimal price. The developer would then add this concept into the ontology at the appropriate place.

Scenario testing – Scenario testing can be done before any modelling or implementation. The aim is to run through scenarios to logically verify whether the system will work. By developing full scenarios, problems can be identified early before any development has begun. This stage of testing would also set the ground and generate ideas for modelling and implementation. For example, consider an application involving trading agents. A scenario might be a simple negotiation between two trading agents A and B. The scenario starts with agent A offering agent B amount z for an item. Agent B will consider the offer by following the logic:

If offer z >x where x is the reserved price, then accept offer.

Else if offer y < z < x where y is the minimum acceptable value, then make counter offer of value n where y < n <x.

Else decline the offer.

Once agent A receives a response of accept or decline then the trade ends here, but if agent A receive a counter offer, then agent A would use a similar logic as above to that of agent B, to make another offer or decline or accept the offer. Once the scenario is set out, the developer would step through it with different values to verify that the logic is sound.

Consistency testing – this is where the complete ontology is modelled in a tool such as Protégé and by exploiting tools associated with ontology representation languages (e.g. OWL/DL), perform automatic checking of consistency using a reasoner.

3.5 Summary

This chapter presented the MOMA methodology used to construct ontology for multi-agent consumption. The MOMA process strings together several tools and methodologies in creating agent applications. MOMA is broken down into two parts, Ontology Development and Agent Development. Ontology Development involves the domain expert identifying concepts and relationships and modelling it as ontology. This is then generated as Java code to be used by Agent Development. In the Agent Development part, agent developers implement the stub code generated from the ontology code produced by Ontology Development part. The result is the agent application. Agent Development was not discussed in this thesis.

In the next chapter, a detailed case study in the financial services domain is conducted using MOMA.

Chapter 4. Case study in the financial services domain

The financial landscape is complex and volatile by nature, making timely information about market trends critical to strategic success. As a result, the study of financial market behaviour exists as a consequential field of endeavour for researchers and financial analysts alike.

In order to gain a collective understanding of financial markets, it is important to observe and investigate the relationships between trends and characteristics across different markets. However, due to the complex conglomeration and distributed nature of financial domain information, the majority of such analysis is carried out at a low level, requiring extensive knowledge of programming languages. This can be problematic for those financial information consumers, researchers and analysts who do not have the expertise required for carrying out complex development in these languages.

The purpose of this case study is for the evaluation of the MOMA methodology presented in Chapter 3. This case study involves the development of an agent application that simulates stock market trading called OntoMarketSim. This case study is based on FINBuilder, a process our team has been working on since 2003 (Sujanani 2005). This case study was accepted as a journal paper, "The development of ontology driven multi-agent systems: A case study in the financial services domain" and will appear in the Computing and Informatics Journal, Volume 3, 2009 (Ying 2009). Through this case study, we will evaluate the benefits of MOMA. Namely, it will allow researchers and domain experts in the field of finance to create smart agent applications without the need for extensive agent design and lower level programming. By specifying the logic and domain knowledge in the ontology, the reuse and sharing nature of ontology can be exploited, allowing parts of the ontology to be shared and reused.

This chapter will be organised as follows: Section 4.1 will give an introduction and background to the case study as well as motivations behind the case study. Section 4.1 will also give an overview of the architecture of the end result. Section 4.2 and 4.3 present the use of MOMA methodology for the case study for Both Ontology Development and Agent Development parts respectively.
Section 4.4 will discuss testing and running of the resulting application. Section 4.5 will contain discussion and evaluation of the MOMA methodology for this case study. Section 4.6 will give a summary of the chapter.

4.1 Introduction

This case study will look at the process undertaken using the MOMA methodology in the design and development of financial market simulator (OntoMarketSim) – an application that facilitates user customisation of agent interaction within the financial domain. The introduction of an ontology-driven infrastructure for use in the financial domain would enable experimentation activities such as pre-trade and behavioural analysis of financial markets to be conducted by end-users at a more intuitive level than is currently possible.

4.1.1 Financial Multi-agent systems

Conventionally, research into financial markets has involved analytical frameworks described mathematically. Traditional mathematical methods used to study financial market behaviour such as statistical analysis have been identified as having shortcomings such as the following (Vergara et al. 2003):

- **They are able to describe macroscopic properties of a system already in existence, but not the origin of these properties.** This type of analysis involves studying financial data from different financial markets and then identifying regular patterns or laws governing the statistics of the data. It usually does not include examining the imperatives and actions that produced the financial data to begin with.
- **They cannot be easily applied to situations where the assumptions behind mathematical equations no longer hold.** The majority of statistical methods and techniques developed to analyse data are applicable on the condition that the variables involved satisfy certain assumptions - for example, that the sample comes from a normal population. However, in cases where the assumptions do not hold, these methods cease to be valid and hence should not be used.
- **They do not handle heterogeneity in populations well.** Traditionally, the behaviours of traders have been described with mathematical models and their interaction with financial market analysed under equilibrium conditions. In reality, it is not always the case that financial markets and traders exhibit the rational behaviour reflected in mathematical models. Traders for example, display heterogeneity in their trade decision-making, interpretation of company announcements and market trends, and adaptive behaviours.

In these instances, alternatives such as agent-based models or non-parametric methods need to be considered. In dealing with the dynamics of collections of entities, agent-based models are better equipped to handle the different kinds of global dynamics that can result from these entities significantly impacting on each other through their interaction within changing environments.

However, of the financial agent systems described in the literature, we found that most of the agents in these models were intrinsically algorithmically linked, with mathematical functions dictating and modifying the agents' behaviours (Neuberg 2003). Furthermore, in the majority of the work surveyed, the agent infrastructures were closely coupled with the application domain knowledge required to dictate the agents' behaviours. By placing most of the explicit domain information within the agents themselves, any potential to re-use the multi-agent infrastructure in conjunction with different domains was destroyed.

With these problems in hand, OntoMarketSim will be developed using the MOMA methodology to evaluate the effectiveness of design of such a system for the purpose of financial market simulation. The main objectives of this application will be the separation of domain knowledge from underlying agent code through the use of ontology. This will hence allow domain experts and typical users of the system to dictate the behaviour of the application through high level concepts rather than low level programming language. Another beneficial side effect is the potential for sharing and reuse of the domain knowledge.

4.1.2 Architecture

Figure 4-1 illustrates the overall application architecture of OntoMarketSim after all implementation. Following the acquisition of financial market knowledge, the conceptual outline of the financial ontology was developed. This enabled the process of specifying the ontology in Protégé to be more accurate and allowed for easier verification of knowledge and concepts.

The OntoMarketSim tool was developed within an agent-based framework that interfaced with the ontology using MOMA. We investigated a number of multi-agent platforms and decided to use the JADE framework as it had the greatest amount of ontology support. JADE has a content reference model which enabled ontologies subscribing to its model to be accessed by its agents. The model required the inclusion of low-level ontological elements - predicates, terms, concepts and agent actions. Using the Bean Generator Tool from step 3 of the Ontology Development part of MOMA (Section 3.2.3), we were able to generate a FIPA/JADE compliant ontology automatically from the ontology modelled in Protégé through which agents could be implemented.

The financial domain also influenced the design of the behaviours of the OntoMarketSim agents - such as reactions to different financial events and trading decisions. To create the market environment, trading data was acquired and modified from market data sources and fed into OntoMarketSim by hard coding the data into the system. This was done for a controlled simulation. This is illustrated in Figure 4-1 by the external entity Trading Data Sources which produce the Market Data feeds. For the purpose of this case study these data sets were hard coded.

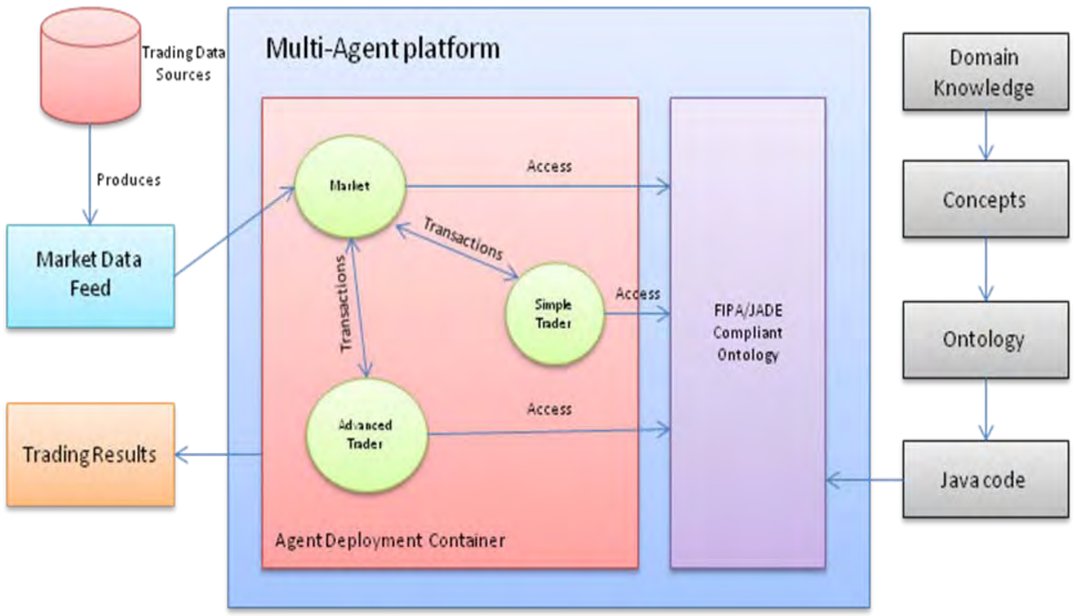


FIGURE 4-1: ONTOMARKETSIM OVERALL ARCHITECTURE

In the next section we will look at how we used MOMA in designing the whole application.

4.2 Ontology Development

A survey of current work discovered few financial domain ontologies, and none of those found had been written with the purpose of utilisation in multi-agent systems in the manner proposed by this case study.

For instance, in the stock market ontology of (Alfonso 2005), the low-level design details that describe the elements, relationships and rules of a stock market domain are presented. Though the paper focuses on the reusability of the ontology, it does not provide an application demonstrating how this could be made possible. In another two studies (Zhang 2000; Zhang 2003), the authors propose the use of financial domain ontologies within a multi-agent system. However, the ontologies developed were used only as a common semantic interface for agents where domain knowledge still resides within the agents. Our aim is to allow end-users (i.e. financial domain experts) to specify the system without needing the knowledge required to program agent behaviour in low-level languages. Other studies of financial market ontologies mainly focussed on ontology mapping such as mapping across different news sources or information formats (Snoussi 2003).

4.2.1 Step 1: Concept Identification

The application for this case study is not very complex and will not need a large number of concepts, hence it would easier to construct our own domain ontology. For identification of concepts, we used the GT Guided Tool (Section 3.2.1).

The source data used was partial selected extracts from Wikipedia and Douglas Mctaggart "Economics" fourth edition in the form of text. The extracts contain concept definitions and explanations of financial market mechanisms. These are used as input for the GT guided tool as a starting point for the extraction of concepts. The text extracts were entered into the tool as text.

Figure below depicts the process of using GT tool for identification of concepts.

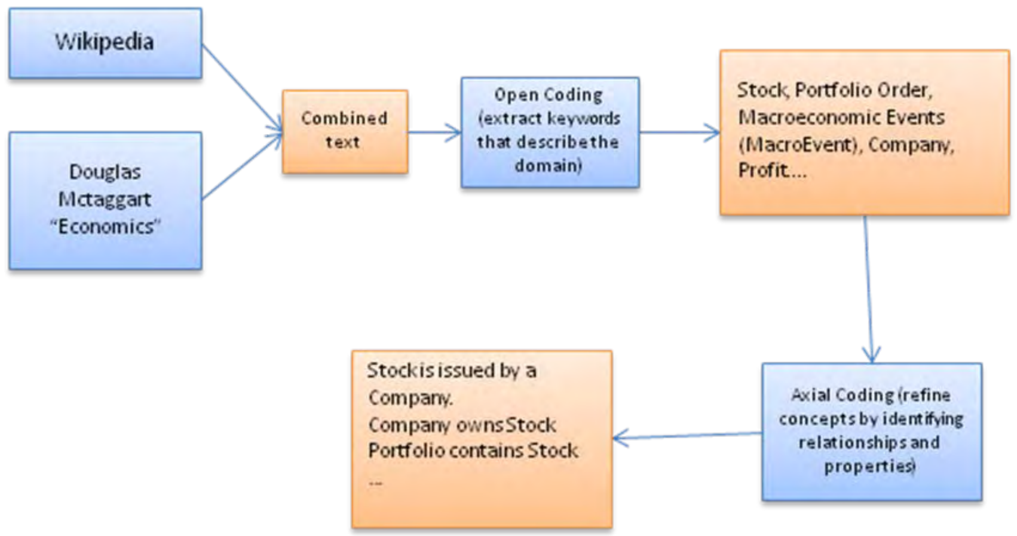

Wikipedia
Douglas Mctaggart "Economics"
Combined text
Open Coding (extract keywords that describe the domain)
Stock, Portfolio Order, Macroeconomic Events (MacroEvent), Company, Profit...
Axial Coding (refine concepts by identifying relationships and properties)
Stock is issued by a Company. Company owns Stock Portfolio contains Stock ...

FIGURE 4-2: CHAPTER 4 CONCEPT IDENTIFICATION

The resulting concepts we have identified:

Stock, Portfolio Order, Macroeconomic Events (MacroEvent), Company, Profit, Loss, Buy, Sell, Order Matching, Validate Order, Validate Portfolio, Take Over, Market, Trader, Order Type, Process Order, Invalid Order, Owns, Process Order Error, Amount, Price.

Note that concepts such as Amount and Price can be a part of the generic ontology, but here we will treat them as integers. The relationships are illustrated in the next section.

4.2.2 Step 2: Ontology Modelling

4.2.2.1 Customising Domain ontology for Application

Figure 4-3 presents a portion of our financial domain ontology where the concepts Portfolio, Stock, Company and MacroEvent are children of an abstract root concept called Concept. Additionally, the concepts TakeOverEvent, LossEvent and ProfitDropEvent are child concepts of MacroEvent. The dotted lines connecting concepts represent a non parent-child relationship. The relationships between the concepts Portfolio and Stock are 'containsStock' and the inverse 'isPartOfPortfolio'. Relationships for the concepts Company and Stock are 'isIssuedBy' and 'ownsStock'. For Company and MacroEvent the relationships are 'hasEvent' and 'eventBelongsTo'.

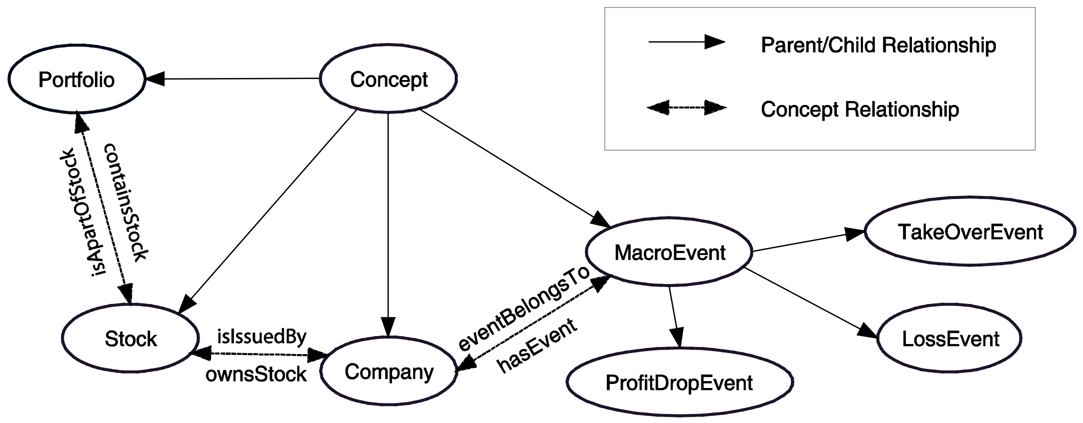


FIGURE 4-3: GRAPHICAL REPRESENTATION OF PARTIAL FINANCIAL DOMAIN ONTOLOGY

The Specialised task ontology for this case study Task ontology will only be “Perform Order Matching” and “Update Stock tables” for the order processing Agent. The Trader Agent will have its behaviour modelled as in the task ontology. Refer to Section 4.5. “The Event Agent Behaviour” and “Order Agent Behaviour” will be the task ontology for the Event Agent and Order agents respectively. Although the concepts Buy and Sell were considered as Tasks, they will be child concepts of TraderAction, which is implemented at code level. Please refer to Appendix A1 for the full ontology in RDF/OWL.

4.2.2.2 Building the Mediation Ontology

Mediation ontology would only apply for macroeconomic events if they are external to the system or if the market data was dynamically changing and have and external source. Since both of these are handled in the application itself by internal agents Mediation ontology is not required here.

4.2.2.3 Building the Communication Ontology

The communication ontology defines the concepts that are used when the agents interact with each other. The communication ontology for OntoMarketSim is show below.

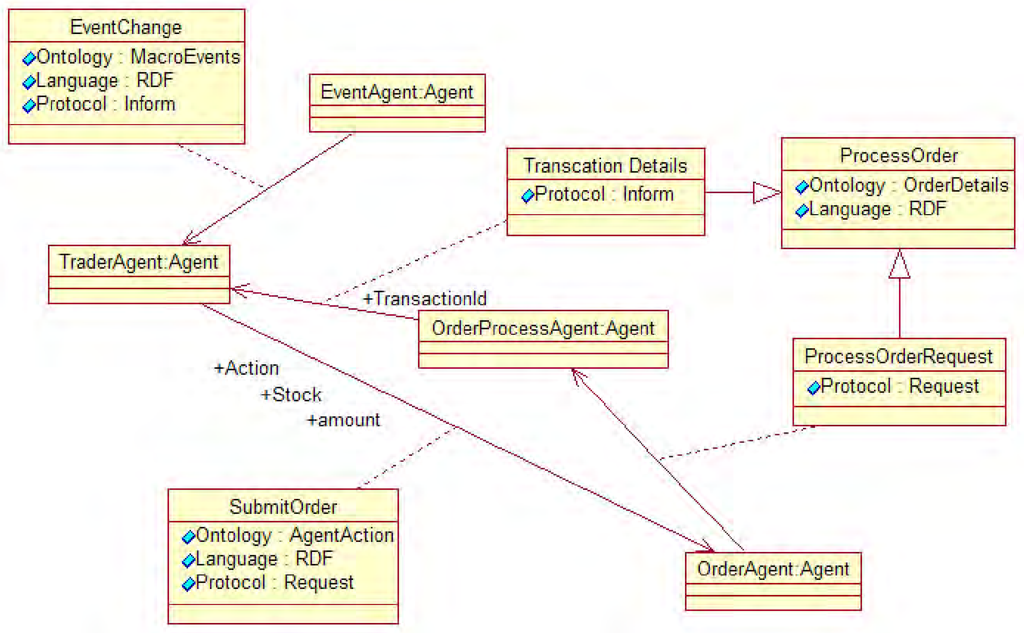


FIGURE 4-4: ONTOMARKETSIM COMMUNICATION ONTOLOGY

In the diagram above, there are four agents, the TraderAgent, OrderAgent, EventAgent and the OrderProcessAgent. The EventAgent informs the Trader of the change in macroeconomic events that are specified in the MacroEvents ontology. The TraderAgent submits an order depending on the updates it receives from the Event Agent. The other is specified through the ontology Agent Action, which can be the concept Buy or Sell. TraderAgent also provides the stock and the amount. The OrderAgent forwards the request to the OrderProcessAgent which processes the order and sends transaction details back to the TraderAgent for it to update its portfolio.

4.2.2.4 Adding logic through Rules and Axioms

An advantage of the use of ontologies is that the conditions can be classified through description logic reasoning. For a simplistic example using the ontology illustrated in Figure 4-3, we could create defined concepts GoodInvestmentCompany as a sub-concept of the concept Company. The GoodInvestmentCompany is defined as:

$$GoodInvestmentCompany \equiv Company \sqcap \exists\, hasEvent.TakeOverEvent$$

This simply means that a GoodInvestmentCompany is Company and is being taken over. We then create another concept called GoodStock as a sub-concept of Stock to define the stocks issued by instances of GoodInvestmentCompany.

$$GoodStock \equiv Stock \sqcap \forall isIssuedBy.GoodInvestmentCompany$$

Through the use of inferencing we could derive instances of the concepts 'GoodStock' which can be used by the trading agents. This however was not implemented. To implement this feature, an agent must either be setup to interface with a reasoner or the rules and axioms are generated or implemented in agent code. All other agents that require inference will need to interact with this agent to access the reasoner.

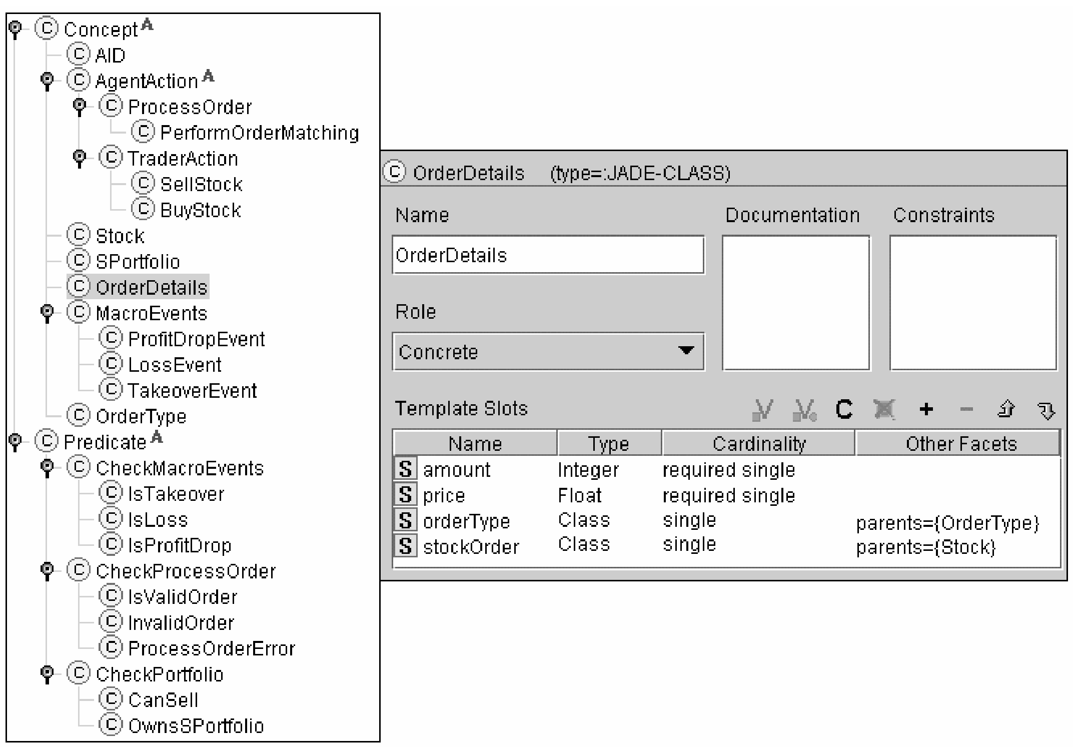
4.2.2.5 Specifying Ontological mappings between application ontology

Since the application only deals with one application ontology, this step is not required.

4.2.2.6 Modelling ontology in Protégé

Modelling of the ontology was done in Protégé. We used Protégé which stored the ontology internally as RDF for a number of reasons. Firstly, we felt the Protégé interface was both intuitive and user-friendly, not requiring a large amount of time to become familiar with. Secondly, it contained numerous plug-ins that enabled the user to extend the editor's core functionality. Some of the plug-ins that looked especially useful was the OntoViz Tab, which enabled the visualisation of Protégé ontologies and the XML Tab, which enabled Protégé ontologies to be extracted from XML files and XML files to be translated into Protégé ontologies. This could facilitate the depiction of the ontology in a more presentable manner.

There was a major benefit in exploit the tools associated with the ontology representation language (e.g. OWL/DL) to perform automatic checking of consistency. This is done by a reasoned (RacerPro) through a DIG interface in Protégé. Any defect will lead back to the conceptualization step resulting in a cyclic ontology development process. A development snap shot is show in Figure 4-5.



OrderDetails (type=:JADE-CLASS)

Name: OrderDetails

Role: Concrete

Name	Type	Cardinality	Other Facets
amount	Integer	required single	
price	Float	required single	
orderType	Class	single	parents={OrderType}
stockOrder	Class	single	parents={Stock}

FIGURE 4-5: PROTEGE ONTOLOGY DEVELOPMENT SNAPSHOT

4.2.3 Step 3: Code Generation

Using the Bean Generator Tool, we were able to automatically generate a FIPA/JADE compliant ontology automatically from the ontology specified in Protégé through which agents could be implemented. The code was generated using the Protégé plug-in, Bean Generator. Please refer to Appendix A2 for the output.

4.3 Agent Development

This section briefly details the development and implementation of the agents.

4.3.1 Agents and reasoning

It is the agent developer’s task to identify the agent from both the ontology and requirements. For example, the concept “Trader” is modeled as a concept in the ontology. The developer will identify this as being an agent and implement it as an agent class. Further he would subclass these further, into simple trader, intermediate trader and advanced.

Having established the agent environment, we developed some scenarios with which to test the OntoMarketSim tool. Agents were assigned different responsibilities and levels of sophistication for trading. These agents are identified from both the requirements and the ontology.

Order Agent: This agent carries out the role of the interface between the market and the traders. It is comparable to an electronic trading website that allows traders to submit buy and sell orders.

The frontend of OntoMarketSim includes an interface for entering external orders into the system. This is especially useful when we want to atomically test an agent's behaviour as it enables us to rapidly enter test orders into the system and observe the agent's reaction to the market state.

Order Processing Agent: In essence, this agent carries out the functions of a stock market trading engine. It communicates with the Order Agent in order to receive new orders and send back confirmations of order submissions. In addition, this agent updates the market buy and sell order tables by performing order matching. These tables display a continual listing of the current buy and sell orders - including the prices set for limit orders, the stock name and symbol, and the order quantity. Once a successful transaction is completed, the agent either removes the orders from the tables, or updates the buy and sell quantities displayed.

Event Agent: The prices of stocks in the simulated stock market are only affected by macroeconomic events. The Event Agent represents these events, and disseminates announcements relating to companies to all traders. Communication between this agent and the trader agents is carried out through messages conforming to the macroeconomic event ontology layer. Each trader agent's reaction to these events varies according to its level of sophistication. An alternative implementation would use the Mediation ontology, if macro-economic events were updated via an external information source.

Trader Agent: Trader agents comprise the main entities of interest in the prototype. Through their performance, the ability to simulate trading with the financial ontology can be evaluated. OntoMarketSim models the heterogeneity of stock market traders through three different trader agent types. These are:

- Simple Agent - exhibits primitive trading behaviour.
- Intermediate Agent - has moderately informed trading behaviour.
- Advanced Agent - possesses sophisticated trading behaviour.

In order for meaningful comparison of agent performance to take place, each agent is initialised with an ownership of the same number and valuation of stocks. Each agent is also provided with a list of stocks that they are interested in buying. This reflects real-world trading decisions to invest in technology stocks or blue chip stocks. For the purposes of better performance comparison, we decided to standardise the number of shares each agent buys or sells on each trade. Additionally, OntoMarketSim enables these settings to be defined at trader initialisation.

The main differences in behaviour arise from the agents' buying and selling strategies, and from their reaction to market macroeconomic events. For example, being the most primitive, the Simple Agent type is designed to ignore trend indications given by market macroeconomic events, while the Intermediate Agent and Advanced Agent behaviours react to these events. The reasoning behaviour of the agents is implemented through a series of conditional statements of the form:

$$C_1 \& \ldots \& C_n \rightarrow do: A_i, \ldots, A_j$$

for all C conditions and A actions. Agents evaluate each conditional statement as true or false by consulting the financial ontology. The statements vary depending on the sophistication of each trader agent. For example, an agent of intermediate intelligence incorporates the following conditions in its behaviour:

```
(company has loss) -->
            do: suspend trading for x time
( drop in profit ) -->
            do: suspend trading for x time
                  (company is being taken over) -->
            do: buy shares
(currently hold takeover target company shares) -->
            do: suspend trading for x time; sell shares
```

Each agent also has a trading portfolio, comprising of realised and unrealised profit tables. These can be viewed at any time during a OntoMarketSim simulation, and are dynamically updated every time an order transaction is successfully completed. The updating of the portfolio is carried out through the passing of information committed to the portfolio ontology layer. In addition, a graph of the profits over the total trading time can be viewed and is updated automatically.

An example of a scenario would start with a macroeconomic event agent creating an instance of TakeOverEvent concept. The instance of the TakeOverEvent concept is shown below by an OWL/RDF representation.

```
<TakeOverEvent rdf:ID="TakeOverByCompanyX">
  <isTakenOverBy rdf:resource="#CompanyX"/>
  <eventBelongsTo rdf:resource="#CompanyY"/>
</TakeOverEvent>
```

This is sent through the Macroeconomic Event Layer of the financial ontology. Because the trader agent shares the same ontology, it would immediately understand the concept and compute a response. Depending on the sophistication of the trader agent reasoning, conditional statements will be evaluated using the TakeOverEvent concept. A response by the trader agent will either be nothing or creation of an instance of OrderDetails with attributes representing sell or buy orders of certain quantities of stock. An example of an OrderDetails instance in OWL/RDF is shown below:

```
<OrderDetails rdf:ID="OrderDetailsInstance16">
  <amount rdf:datatype="&xsd;int">10</amount>
  <price rdf:datatype="&xsd;float">143.2</price>
  <orderType rdf:datatype="&xsd;string">Buy</orderType>
```

```
    <stockOrder rdf:datatype="&xsd;string">CompanyXStock</stockOrder>
  </OrderDetails>
```

The instance of OrderDetails is then sent to the order agent through the order layer of the ontology. Once matching and validation is complete, the instance of OrderDetails get passed onto the order processing agent which in essence updates our virtual stock market.

Although reasoning and logic is implemented in the underlying code, with the help of a better code generator or the use of agents that implement reasoner, MOMA intends to extract this logic from the lower level code and move it to the higher level ontology. This will be considered as future work.

4.4 Testing

As the application domain of OntoMarketSim did not already have a standard set of prescribed evaluation criteria, we developed an evaluation strategy based on a heuristic evaluation technique described in (Ray 2003). This involved both testing OntoMarketSim with predefined inputs and demonstrating it to a number of different individuals with varying knowledge and expertise in the fields of information technology and finance.

A number of discrete event simulations with varying macroeconomic event combinations were run using OntoMarketSim. Figure 4-6 shows the graph of the portfolio values of a Simple Agent and an Intermediate Agent, that both traded with an equal number of shares from the same company over a common time period.

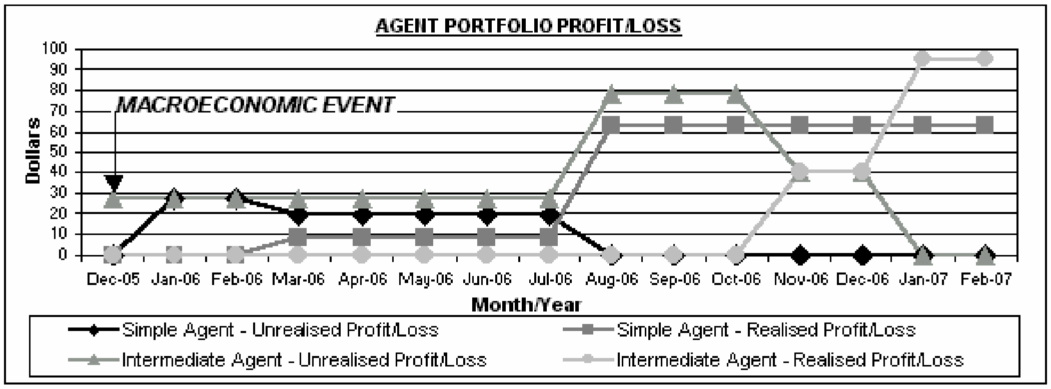


FIGURE 4-6: PORTFOLIO VALUES OF A SIMPLE AGENT TRADER AND AN INTERMETDIATE AGENT

The macroeconomic event that occurred during this period was an announcement that the company was the target in a takeover. This announcement occurred at the start of the trading period - in Dec-05.

As indicated in the graph, the Simple Agent, which was not responsive to the macroeconomic event, continued trading, as it normally would have. The black line on the graph goes to zero in Aug-06 as the agent has sold all its holdings, and has realised all its profits. The Intermediate Agent, on the other hand, was receptive to the company announcement through interaction with the financial ontology and reasoning. As a result it ceased trading for a short time to allow for market stabilisation, before re-commencing. In this instance, its strategy was successful. While the results obtained by each simulation were not always the same, they did show that OntoMarketSim successfully demonstrated the use of ontologies with heterogeneous agents within the financial domain.

4.5 Discussion and Evaluation

The initial problem of simulating a simple stock market environment for this case study was implemented successfully using the MOMA methods. We can see that the generated code from the ontology development part of MOMA useful in providing the knowledge, and building parts for the agent implementation. Without the generated code, the implementation would have taken much longer. Although the methodology itself did not be completely carried out by the domain expert themselves, further case studies and research will help devise a structured methodology for the Agent Development part of MOMA. The ontology development part however did not require agent or programming expertise at all. With a smart code generator, MOMA would give more control of the agent application logic. Ultimately, we envision the Agent Development part would contain implementation of generic code that would run on the generated code from ontology without modification.

In regards to reuse and sharing, the domain ontology in theory can definitely be reused for another agent application (even one with different requirements since the domain ontology would not change in this case). However, the agent implementation will not be able to be reused as it is customised to the specific requirements of this case study. Due to the small scale of this case study, further exhaustive case studies are required to verify the reuse and sharing capabilities of the ontology. Without a meta-model to guide the building of ontology, the ontology cannot be separated into the distinct modules and therefore will make it much harder to reuse.

OntoMarketSim was demonstrated on a Windows platform, however, it would be able to run on any platform that has the Java Runtime Environment. The JADE framework is also able to integrate with web browsers and Java Applets, so the application could be translated into a web service in the future, enabling greater flexibility. Similarly, due to the underlying JADE infrastructure, the prototype may be run on multiple computers with little complication and hence it has been assessed as scalable.

The prototype consisted entirely of the financial ontology layers, and agents. Hence its design was modular. In addition, coupling was loose, as agents communicated with each other through the sending and receiving of messages that subscribed to the ontology. Thus they did not necessarily need to know the names of the other agents to whom they were sending messages to as generic broadcasting techniques could be employed.

Interoperability - The specification of the ontology in Protégé enables sharing across applications and agents. The layered approach taken to the development means that concepts could be specified in separate smaller ontologies and then combined into a larger encompassing ontology.

Agent Communication - The ontology semantically unified agent communication. OntoMarketSim agents needed to merge information from diverse sources - other trading agents, market data and macroeconomic events. The financial ontology played a critical role here as it enabled agents to derive semantic understanding from the exchange of data. The most obvious advantage of inserting an ontology layer between the agent infrastructure and domain knowledge layers was that common domain knowledge was able to be specified in a single source, communicable to all entities that required it. Thus, while the agents were essentially heterogeneous in nature - having different behaviours and perceptions, the financial world they functioned within, was consistent across all agents.

The ontology plays a crucial role as agent communication is solely carried out through the passing of messages that subscribe to the ontology. This means that whenever an agent receives information for another agent - for example, when the Order Agent receives a sell order from a Trader Agent - no meaning translations are required to understand the communication. Thus, the need for the actual financial domain information to be coded at the infrastructure layer in order for all agents to understand is removed. Also eliminated is the need for human interpretation and supervision to facilitate agent reasoning and dictate behaviour.

Below is a table summarising stepwise the problems that were encountered in each step of the methodology and solutions where the problem was resolved.

Tasks	MOMA Problems and Issues	Solution
Identify domain and task ontology – doing so helped us model the domain of the agent application, which can then be used for the implementation of the application.	The use of Standard upper ontology such as SUMO or Cyc for small applications takes up more time than necessary (having to sort through a large ontology for only a few concepts).	The developer needs to make a compromise between the time and effort needed to build ontology off a standard one and the potential for the ontology to be reusable.
	Several Concepts were missed when performing this step.	Additional Concepts were discovered through scenario-based testing.
Customising domain ontology for application – this allowed generic ontology to be extended and customised as application ontology. By doing this, the	Does not support complex logic when modelling application ontology.	Rules and axioms can be achieved through the use of "defined" concepts in protégé using OWL. However, this proves to be difficult to implement in code. Also Bean Generator does not support generation of defined

generic ontology can be shared and reused.		concepts to code.
	Intermediate concepts were required to classify the concepts. E.g. buy and sell concepts would have a TraderAction parent. There was no sub-step for creating these concepts.	These concepts were added in, but it is noted that an additional sub-step is required to accommodate this.
Adding logic through Rules and Axioms – modelling the business logic of the agent application in ontology.	Hard to add logic through rules and axioms by generating into code.	Use an agent that is interfaced with a reasoner to perform reasoning for all other agents.

TABLE 4-1: CHAPTER 4 LESSONS LEARNED

After several iterations of the development steps, the ontology was complete. From the evaluation of the problems above, some limitations that were not foreseen were discovered. Namely, the need for development steps to needs to be iterative and the inclusion of a sub-step for discovering intermediate concepts.

In terms of satisfying the objectives of MOMA that were initially set out, the following table summarises the evaluation of each (refer to section 3.1 for details of each objective):

Objective	Evaluation
1. Structured meta-model for reuse and sharing	Although the meta model was designed to support reuse. The nature of the case study means that we could not test this fully.
2. Move business logic and domain knowledge from underlying agent code to higher	Domain knowledge is definitely been moved from the agent code to the ontology. However, some of the behaviour and business logic of the agents themselves still needs to be coded in the Agent Development Part. This is majorly due to the fact

level.	that generation of code for axioms and rules is not supported.
3. Facilitate the use of tools to accelerate development.	MOMA is driven by the use of tools as a part of its processes. The use of tools definitely sped up the development, especially for time consuming tasks such as concept identification.
4. Reuse of existing ontology.	Refer to 1.
5. Distinguish roles between domain expert and agent developer	There is a clear distinction between the roles of domain expert and agent developer. The two parts of MOMA separates these roles. However there is still requirement for the agent developer to request information from the domain expert.
6. Usability by domain expert without the agent developer	This objective has not been satisfied. Without the agent developer, at its current state MOMA cannot produce a working agent application. This is limited by the ability of the code generator. The ultimate goal is to have the ontology be generated into code that can be plugged directly into a generic agent framework.

TABLE 4-2: CHAPTER 4 EVALUATION OF MOMA METHODOLOGY

4.6 Summary

This chapter looked at a case study in the financial services domain which utilised the methodology presented in Chapter 3. A discussion of the case study was also presented. This chapter showed that there is potential for the MOMA methodology to be used by domain experts and researchers in the financial

domain without the knowledge and expertise of agent development and lower level programming. However, further research is required to fully exploit this benefit. The benefit of reusing agent application itself is not possible, however the reuse of the ontology at the state before it is turned into code can be reused for similar agent applications in the domain with slightly different requirements.
Overall this case study showed that the MOMA methodology can be used to build agent applications in the domain of financial services while meeting majority of its objectives set in chapter 3. The result of this case study showed gaps in, and parts of, the methodology that required ad hoc meeting the requirements and objectives. These gaps identified further areas of further research and future work.

The next chapter will present the second case study in the domain of e-Health.

Chapter 5. Case study in the e-Health domain

The term e-Health is a relative recent term for healthcare practice which is supported by electronic processes and communication. With an increase in development and research in areas such as telemedicine and Electronic Medical Record (EMR), the e-Health domain makes a good candidate for the introduction of agent applications in solving many of its problems.

The purpose of this case study is for the evaluation of the MOMA methodology presented in Chapter 3. This case study involves the development of an agent application that tries to solve the interoperability problem in data retrieval in the e-Health domain. This case study is based on previous work on OBMAS for e-Health (Wimalasiri 2003). This case study was submitted for publication as "An Ontology Driven Multi-agent Approach to Integrated e-Health Systems" for the Journal of Biomedical Informatics. The scenario of new born babies will be used due to the enormity of the scope of this application. The scenario is suggested as a part of a discussion with an experienced medical doctor. Although this was considered as a proof of concept, implementation was carried out for the Ontology Development part of MOMA.

MOMA with full agent implementation was illustrated in Chapters 3 and 4. Hence the agent part is not the focus of this thesis. The focus of the case study in this chapter focuses only on Ontology Development.

This chapter will be organised as follows: Section 5.1 will give an introduction and background to the case study as well as motivations behind the case study. Section 5.2 and 5.3 present the use of MOMA methodology for the case study for both Ontology Development and Agent Development parts respectively.

Section 5.4 will contain discussion and evaluation of the MOMA methodology for this case study. Section 5.5 will give a summary of the chapter.

5.1 Introduction

Increases in the quality of patient healthcare are dependent on transparent access to distributed patient information. The current healthcare industry finds healthcare consumers exercising their freedom to visit any number of healthcare providers who adopt Electronic Health Care systems that does not coordinate with other providers (HMT 2005). During each episode of care, an addition or a modification is made to the Electronic Health Record (EHR) that is stored with the respective healthcare providers. The quality of service provided by a healthcare provider is intrinsically dependent on the availability and the interoperability of this distributed patient health information (Katehakis 2001). In a clinical setting, each healthcare provider should be able to browse and query the patient’s healthcare record, irrespective of the locality or format of the EHR. Information held within these various EHRs may allow the healthcare provider to make a more informed diagnosis or recognize potential adverse drug interactions.

The need for collaboration between healthcare providers arises from the way healthcare consumers view and utilise the services of health care providers. Healthcare consumers generally visit a number of different healthcare providers over the course of their life, depending on their need or their proximity to the healthcare provider. An individual may visit his/her local GP for a mild influenza and/or vaccinations, but use the services of a specialist for treatment of a severe medical problem (e.g. cancer). Similarly, a patient may visit one healthcare provider when he is at home and another when he is away on business. As a result, patient records tend to be divided and dispersed amongst several different healthcare entities leaving each with incomplete information. Management of this medical information is crucial to both healthcare providers and consumers.

The need for collaboration is further bolstered by changes in today's society in terms of new ways of which doctors interact with patients. While the doctor-patient relationship has traditionally been hallowed turf, it is quickly becoming an impersonal one to the extent that they may no longer involve any face-to-face contact. The internet has further revolutionized the industry as more and more individuals consult online doctors to have their prescriptions filled and symptoms diagnosed. Each online transaction represents another detail in a patient record that is not to be accessible using the current healthcare system (Besell 2002). As a result of those developments, healthcare providers are less likely to be informed about their patient's entire treatment history. It is evident that the quality of healthcare could be improved substantially if all relevant information were available to each healthcare provider.

While there is an obvious need to share and exchange health information, lack of standards and coordination between different health information systems have resulted in isolated islands of information. Healthcare providers need to be presented with a unified view of a patient's medical record, transparent from its distributed nature.

This case study addresses semantic interoperability amongst these e-Heath systems using an Ontology-Based Multi-Agent application

5.1.1 Standardisation

Heterogeneity is a product of the intrinsic differences between healthcare providers. Since healthcare providers have a particular focus in service provision, their e-Health systems are similarly designed. For instance, a pharmacist may need only to store information on prescriptions, while a physician would store medical histories as well. Furthermore, the healthcare domain is always evolving, reflecting new treatments and guidelines. Diabetes diagnostic tests, for instance, have evolved from testing for the presence of glucose in urine, to measuring blood glucose levels, to the glycosylated haemoglobin A1c test. The results of each test have different thresholds and units, requiring an individually specific structure and schema to store (Ganguly 2005). As such, e-Health systems that adopt a new standard due to an update in both structure and schema for its data will be substantially different to e-Health systems that still adhere to old standards.

The ultimate aim in e-Health systems is to allow two healthcare providers to exchange patient health information seamlessly. One approach involves restricting e-Health systems to a standardised format or schema such as MML [http://www.medxml.net/], thus enabling interoperability with any supporting Clinical Computing Systems (CCS). This can be quite restrictive in terms of both implementation and extensibility of those e-Health systems if all systems had to model their internal data in the same way.

A second approach is to use standardized messaging protocols such as those specified by HL7 (Orguna 2005). These and other such standards and frameworks have been developed to enable the interoperability and integration of distributed and heterogeneous e-Health systems. Even if HL7 or some other standard messaging protocols are implemented, there is no guarantee that the semantics will match. For example, a date can easily be transferred a cross systems using such protocols where both systems will understand both the meaning and the data. However, depending on the implementation of the system, there might be confusion when trying to interpret the data. This date might refer to date or birth or the date the record was created. In this case, the semantics of the relationship was not clearly communicated by the messaging protocol.

Interoperability solutions are of four types: physical level (e.g. connectors), data level (e.g., schema designs), specification level (e.g. CORBAmed framework) and semantic level (e.g., semantic web). The problem of semantic interoperability is most difficult (Ganguly 2005). We believe standards translate the problem of semantic interoperability to specification level, a level at which most standard solutions exist. Our experience with CORBA and HL7 suggests that it is possible for applications to communicate with these standards, but semantic interoperability is still a problem. These methods allow semantic interoperability to a degree by establishing a standard messaging protocol.

While standards guarantee a certain level of interoperability (Wong 2003), an agreement on a high level schema – a prerequisite for effective co-operation between e-Health systems – is impossible to either establish or to maintain (Rector 1991)These standards only form a partial solution to the problem of syntactic and semantic interoperability. Furthermore, standards present the following problems:

- All participating healthcare providers in any transaction have to use the same standard. If two entities are using different standards, the two distinct pieces of software will not be able to communicate.
- Standards have to be maintained in a constantly changing environment.
- Clinical computing systems that are restricted to standards for internal data modelling or schema may not meet their particular business requirements

We realise that reliable interoperability is achievable only with a universally accepted standard. Given the complexities of the field of healthcare as demonstrated by the number of standards currently in existence, this prospect is highly unlikely. Integrating e-Health systems of different standards or indeed even non-standardized e-Health systems will be critical to the success of collaborative healthcare processes. Several CCSs attempt to address this need by creating multiple interfaces, one for each standard. This solution is obviously not scalable and does not accommodate generic ad hoc e-Health systems.

5.1.2 Agent applications for interoperability in e-health

Although standards such as HL7's Reference Information Model can also define a high level conceptual schema for communication, it does not support reasoning and is subject to a variety of logical flaws (Smith 2006). In addition, HL7's RIM is data oriented in its modelling and even though it supports abstract models, it is still at a schema level whereas ontology is at a higher level of conceptual abstraction. There are also other insurmountable obstacles and incoherencies of RIM which are addressed in (Smith 2006). In terms of maintenance, ontologies are able to evolve and adapt easily to the discovery of new concepts and relationships, where as a standard is more rigid in its evolution due to it being data focused. In addition, the creation of ontology relies to a greater extent on the knowledge of the domain of application – such as the medical domain – than on programming and data modelling knowledge.

In an attempt to solve the semantic interoperability problem between e-Health systems, we propose the use of an ontology-based multi-agent application. The high level of architecture of this solution is shown in Figure 5-1. In such architecture, agents communicate with each other through common domain knowledge. We believe ontologies could offer a solution for the management of information dissemination as the sharing of common domain concepts and relationships could bridge the different perspectives of the agents. This framework can be similarly applied to a Service Oriented Architecture and has been proven to be successful in telemedicine domains (Davis 2006).

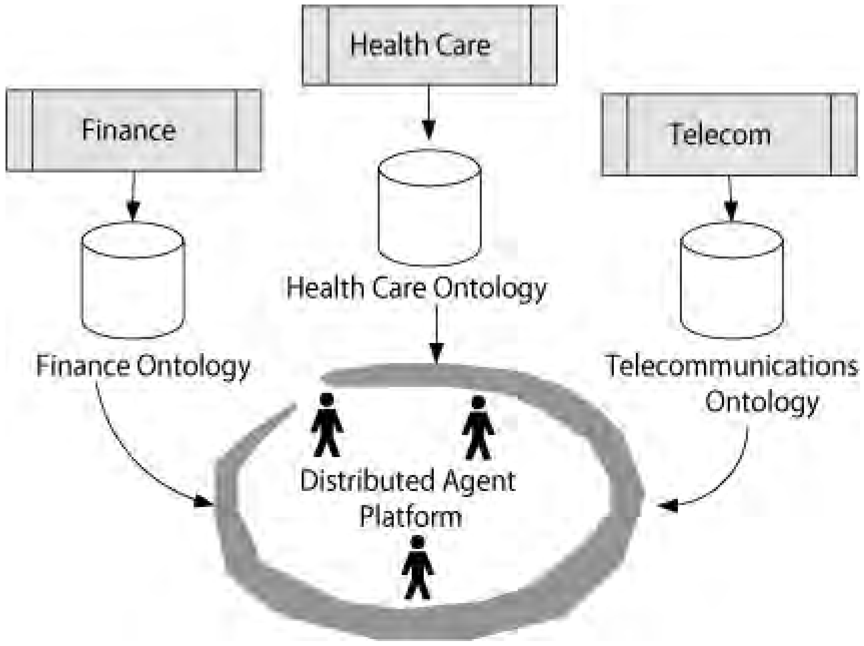


FIGURE 5-1: OBMAS ARCHITECTURE (SUJANANI 2005)

The hardware and data storage layers for the Clinical Computer Systems (CCS) are tightly coupled and are both represented in Figure 5-2. Heterogeneous e-Health systems will likely have its own implementation and can store data in different ways. An example of a storage technique is use of a relational database. Patient data will likely be stored in tables with schemas that are internal to the CCS implementation. It is therefore difficult for such systems to exchange data due to different interpretation and storage techniques. A possible representation of such a data store is illustrated in Figure 5-3, which presents two tables – Tables 3a and 3b use separate formats to store data for newborns which in turn may be used by two different EHR systems. An example of an interface for a CCS with this implementation would be a spread sheet and forms implemented in the end user interface.

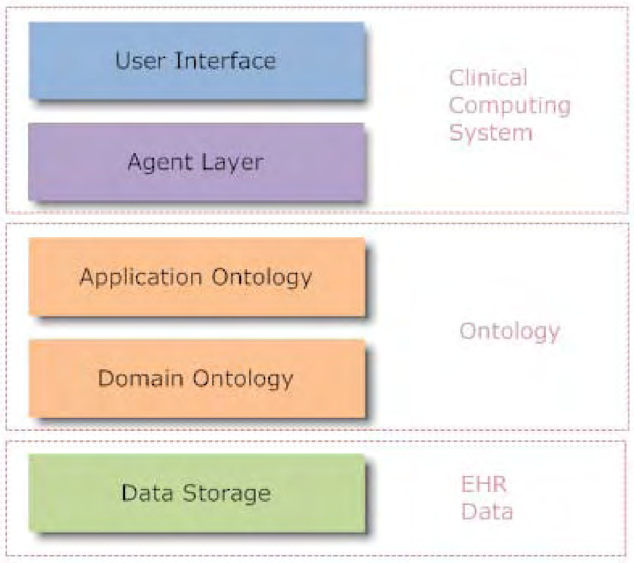
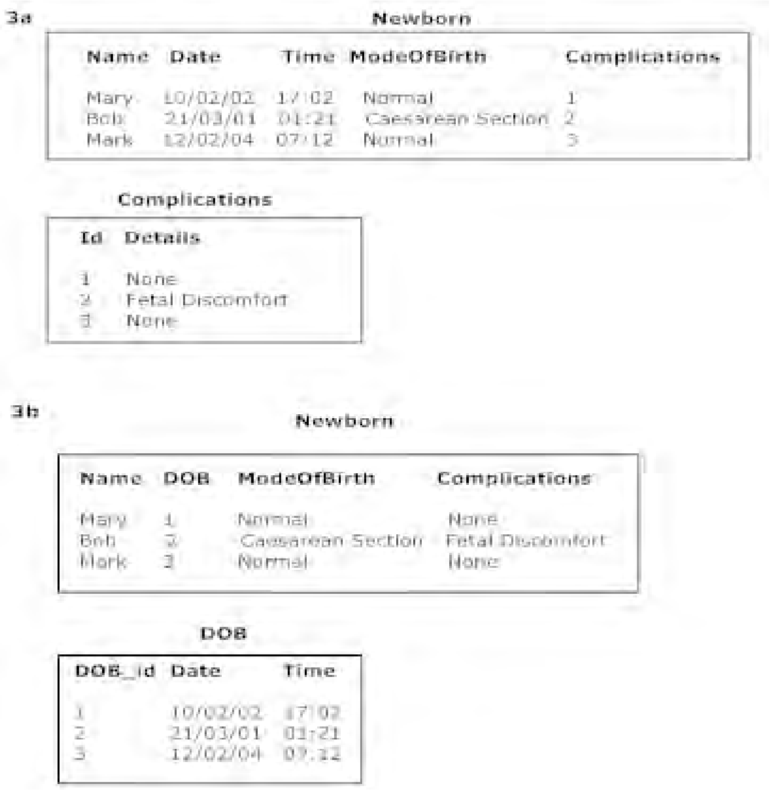


FIGURE 5-2: OBMAS LAYERED ARCHITECTURE


3a

Newborn

Name	Date	Time	ModeOfBirth	Complications
Mary	10/02/02	17:02	Normal	1
Bob	21/03/01	01:21	Caesarean Section	2
Mark	12/02/04	07:12	Normal	3

Complications

Id	Details
1	None
2	Fetal Discomfort
3	None

3b

Newborn

Name	DOB	ModeOfBirth	Complications
Mary	1	Normal	None
Bob	2	Caesarean Section	Fetal Discomfort
Mark	3	Normal	None

DOB

DOB_id	Date	Time
1	10/02/02	17:02
2	21/03/01	01:21
3	12/02/04	07:12

FIGURE 5-3: DATABASE SCHEMA AND DATA REPRESENTATION

Ontology helps us to overcome interoperability through defining a common vocabulary for entities that need to share information in a domain. This not only allows us to reuse the domain knowledge but also makes explicit assumptions and separates operational knowledge from domain knowledge (Natalya 2008). While ontologies have been used for sharing knowledge in many domains, this application extends this knowledge sharing to software agents that would help the semantic interoperability problem to some extent.

Due to the enormity of the scope, and use, of this agent application, we will be limiting the application to medical records of new-borns.

5.2 Ontology Development

5.2.1 Step 1: Concept Identification

Once again, as application for this case study is not very complex and a large number of concepts will not be required, and hence it would be easier to construct our own domain ontology. For the identification of concepts, we used the GT Guided Tool (Section 3.2.4).

There were multiple sources for of input data for the GT Guided Tool. These include:

Lowell Vizenor, Barry Smith, Werner Ceusters "Foundation for the Electronic Health Records" – used as a reference for EHR and possible record fields that were required. This was taken in the form of text.

Group Discussions and specific contributions from an experienced medical doctor with both concepts and scenarios. These were recorded as notes and diagrams and later converted into text for input into the GT guided tool.

Both sources were converted into text before entered into the GT guided tool. The figure below illustrates the process of this step.

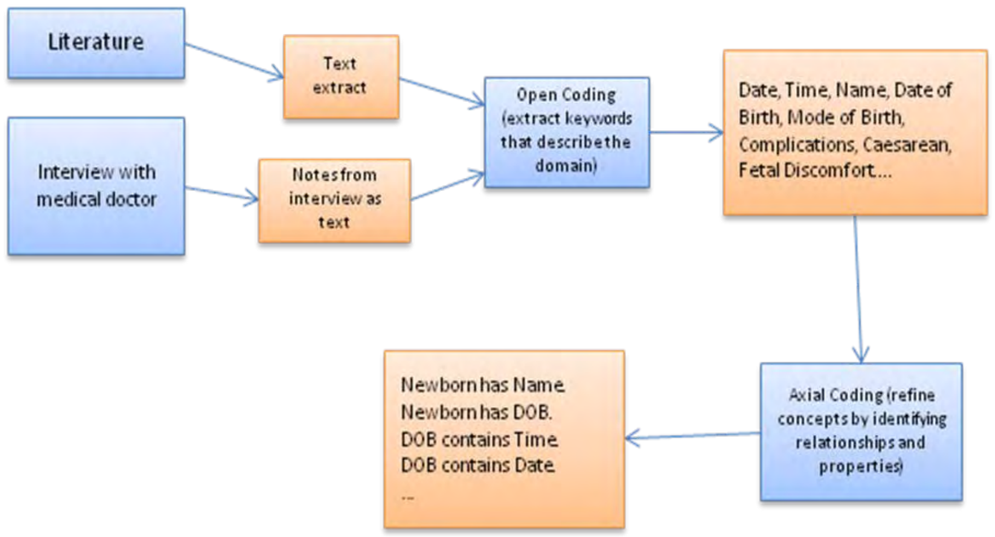

Literature
Text extract
Interview with medical doctor
Notes from interview as text
Open Coding (extract keywords that describe the domain)
Date, Time, Name, Date of Birth, Mode of Birth, Complications, Caesarean, Fetal Discomfort....
Axial Coding (refine concepts by identifying relationships and properties)
Newborn has Name
Newborn has DOB.
DOB contains Time
DOB contains Date
...

FIGURE 5-4: CHAPTER 5 CONCEPT IDENTIFICATION

The identified concepts include:

Date, Time, Name, Date of Birth, Mode of Birth, Complications, Caesarean, Fetal Discomfort

The relationships are illustrated in the next section.

5.2.2 Step 2: Ontology modelling

Domain specific ontology defines concepts in terms of semantics that are applicable to a certain area, which in our case, it's a subset of the medical domain. By defining ontology as such frameworks for specific requirements, ontology developers are able to reuse such frameworks and provide for application and information integration. An example of such ontology in the medical domain is the meta-thesaurus of UMLS (UMLS 2007). The Unified Medical Language System (UMLS) project develops and distributes multi-purpose, electronic "Knowledge Sources" and associated lexical tools for system developers. For the OBMAS framework to perform, we ideally assume that the e-Health systems use the same base ontology so that agents will have the same perspectives of the world when interacting and exchanging information. However, in cases where this underlying ontology differs, we would use ontology mapping techniques such as similarity based ontology mapping techniques (Wong 2003) to reconcile the two ontologies. These mapping techniques, however are beyond the scope of discussion in this paper.

For simplicity, partial domain ontology is shown in Figure 5-5. These concepts will be mapped to the lower level data within the different implementations such that the individual data will become instances of the concepts of the ontology. For example, the data instance “Mary” from Figure 3a will become an instance of the concept Name. The mapping strategies of the ontological concepts are out of the scope of this paper.

Once we have established a base ontology that provides the same domain perspective for all agents, it is possible to develop the application specific ontology as discussed next.

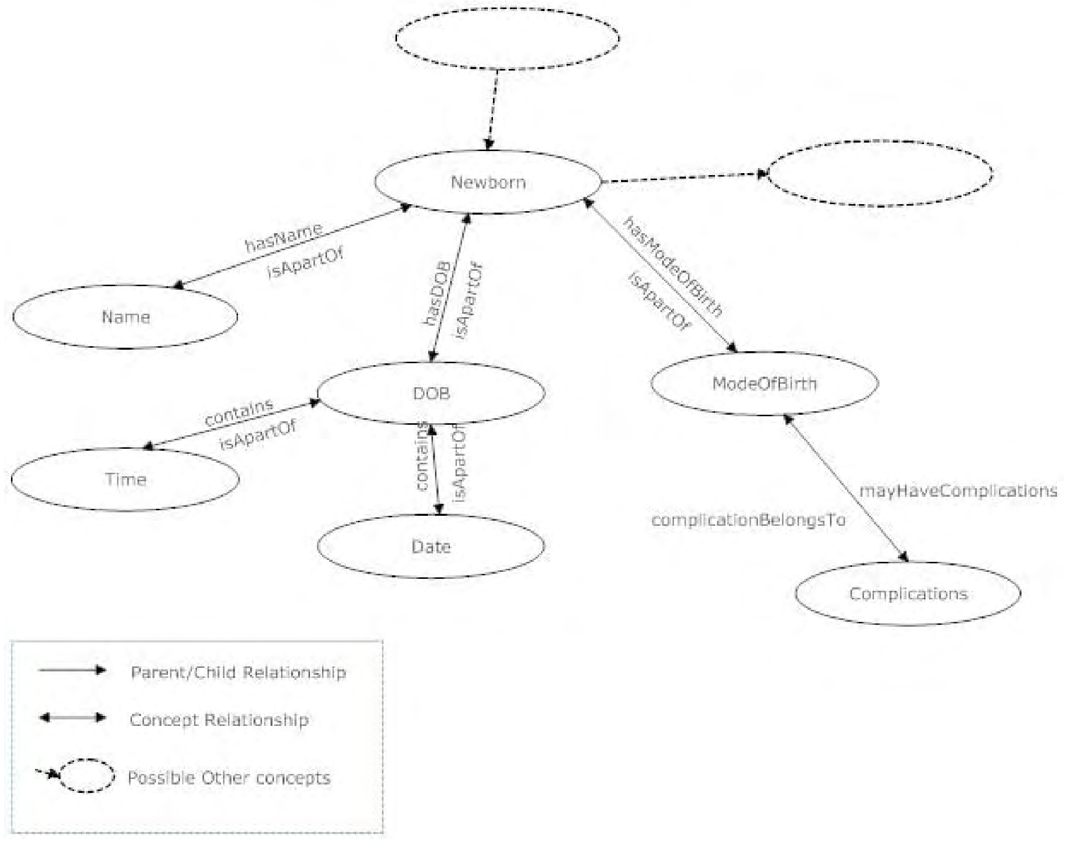


FIGURE 5-5: PARTIAL DOMAIN ONTOLOGY FOR NEWBORNS

5.2.2.1 Adding logic through Rules and Axioms

The application layer ontology will provide our framework with flexibility and customisation. This layer of ontology is built on top of (and using concepts from) the domain ontology. These individual ontologies are application specific and can be formalised as:

$$O:=\{(H,\{C\},\{R\}.\{r\}) \mid Ci \in D, Rj \in D, i=1...m, j=1...n, r \in (\{Ci\},\{Rj\})\} \qquad (1)$$

Where O is the ontology, H is the hierarchy, {C} and {R} are sets of concepts and relations belonging to the domain ontology D. r represents the constrains that are placed on the relationships of the concepts {C}.

The purpose of this layer of ontology is to provide each different e-Health system its own way of customising and constructing concepts. Because this ontology is built from the domain ontology that is shared by all other systems, all systems should be able to understand each other's application ontologies. For example, the doctor could create a concept called CaesareanBirth as a sub-concept of ModeOfBirth (shown in Figure 5-5). The concept refers to newborn babies that are born through a caesarean section due to a fetal discomfort. The CaesareanBirth concept is defined as:

$$\text{CaesareanBirth} \equiv \text{ModeOfBirth} \cap \exists\ \text{mayHaveComplications.FetalDiscomfort} \quad (2)$$

This simply means that the CaesareanBirth concept is a ModeOfBirth concept and has fetal discomfort as sub concept of Complication. We then could create another concept called SpecialCaseNewborn defined below:

$$\text{SpecialCaseNewborn} \equiv \text{Newborn} \cap \forall \text{hasModeOfBirth. CaesareanBirth} \quad (3)$$

Through the use of inferencing we could derive instances of newborns that are born through a caesarean section due to fetal discomfort. This can be done through the use of a reasoning engine such as Renamed Abox and Concept Expression Reasoner (RACER 2007).

5.2.2.2 Specifying Ontological mappings between application ontology.

The CCS may have several layers of user interface for creating the ontology. For system maintenance and administrators, there could be an ontology creation and editing tool such as protégé. For general purpose use, the CCS could just translate a search query into an ontology concept underneath and provide a layer of encapsulation for the end user. The ontology language used for modelling of these new concepts should be reasoner compliant such as OWL-DL, but again this is an implementation issue. We have used Protégé as the platform for our prototype ontology development.

A partial screenshot of the application ontology mentioned in the earlier section is shown in Figure 5-6. The RDF/OWL representation of the concept SpecialCaseNewborn is shown below:

```
           <owl:Class rdf:ID="SpecialCaseNewborn">
               <owl:equivalentClass>
                   <owl:Class>
                       <owl:intersectionOf rdf:parseType="Collection">
                           <owl:Restriction>
                               <owl:onProperty rdf:resource="#hasModeOfBirth"/>
                               <owl:someValuesFrom
       rdf:resource="#CaesareanBirth"/>
                           </owl:Restriction>
                           <owl:Class rdf:about="#NewBorn"/>
                       </owl:intersectionOf>
                   </owl:Class>
               </owl:equivalentClass>
           </owl:Class>
```

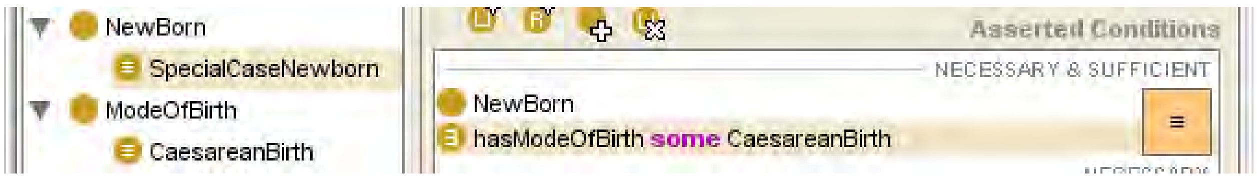


FIGURE 5-6: PROTÉGÉ DEVELOPMENT ENVIRONMENT SHOWING SPECIALCASENEWBORN CONCEPT

5.2.3 Step 3: Code Generation

Please refer to Appendix B2 for the output.

5.3 Agent Development

5.3.1 Agent reasoning and concept sharing

Agents are software entities that augment objects by being able to perform a task autonomously, or with minimal guidance. Agents are able to communicate in a high-level language based on the "speech-act theory" rather than on method-invocation, which results in increased robustness and flexibility. Agents use a knowledge base that contains information about themselves and their environment and are able to migrate from one host to another, interacting with other agents by exchanging messages using the agent communication language (ACL). In our case, the messages that are exchanged will encapsulate the ontology. Because the agents share the same domain ontology, the data exchanged between two agents will have a higher level abstraction and become instances of concepts. In dealing with the dynamics of collection of entities from which multiple data is exchanged, agent-based models are better equipped to handle different kinds of global dynamics that can result from these entities significantly impacting each other through their interaction within the changing environment.

Ontology facilitates the use of reasoning and can easily derive instances for data requests from other agents in the form of concepts. The agents themselves do not have to need reasoning capabilities, as the reasoning engine can be requested from a separate entity such as a server or another agent. When the concepts are in the application layer ontology, the interacting agent will derive the arbitrary application layer concept by referring to its domain ontology layer and once again establishing the same perspective of the world. This allows the agents and their underlying systems to share new concepts as well as data and bridge the gap of semantic interoperability.

5.4 Example scenario

We make use of a scenario to illustrate the agent application approach and how it attempts to solve the semantic interoperability problem between e-Health systems. The scenario is based on the examples in the earlier sections. We start with a basic OBMAS setup shown in Figure 5-7, in which there are three CCS implementations with their own EHR data stores and a common ontology. They all use the agent framework for communication. The scenario is a doctor wanting to find records of newborn babies that were born through a caesarean section due to fetal discomfort, for research purposes.

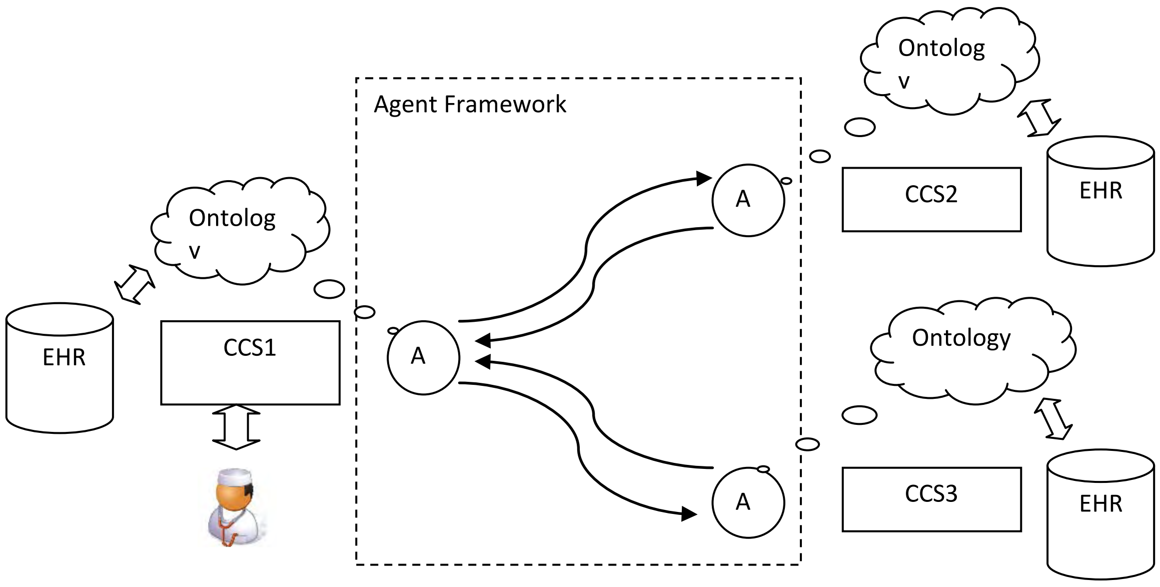
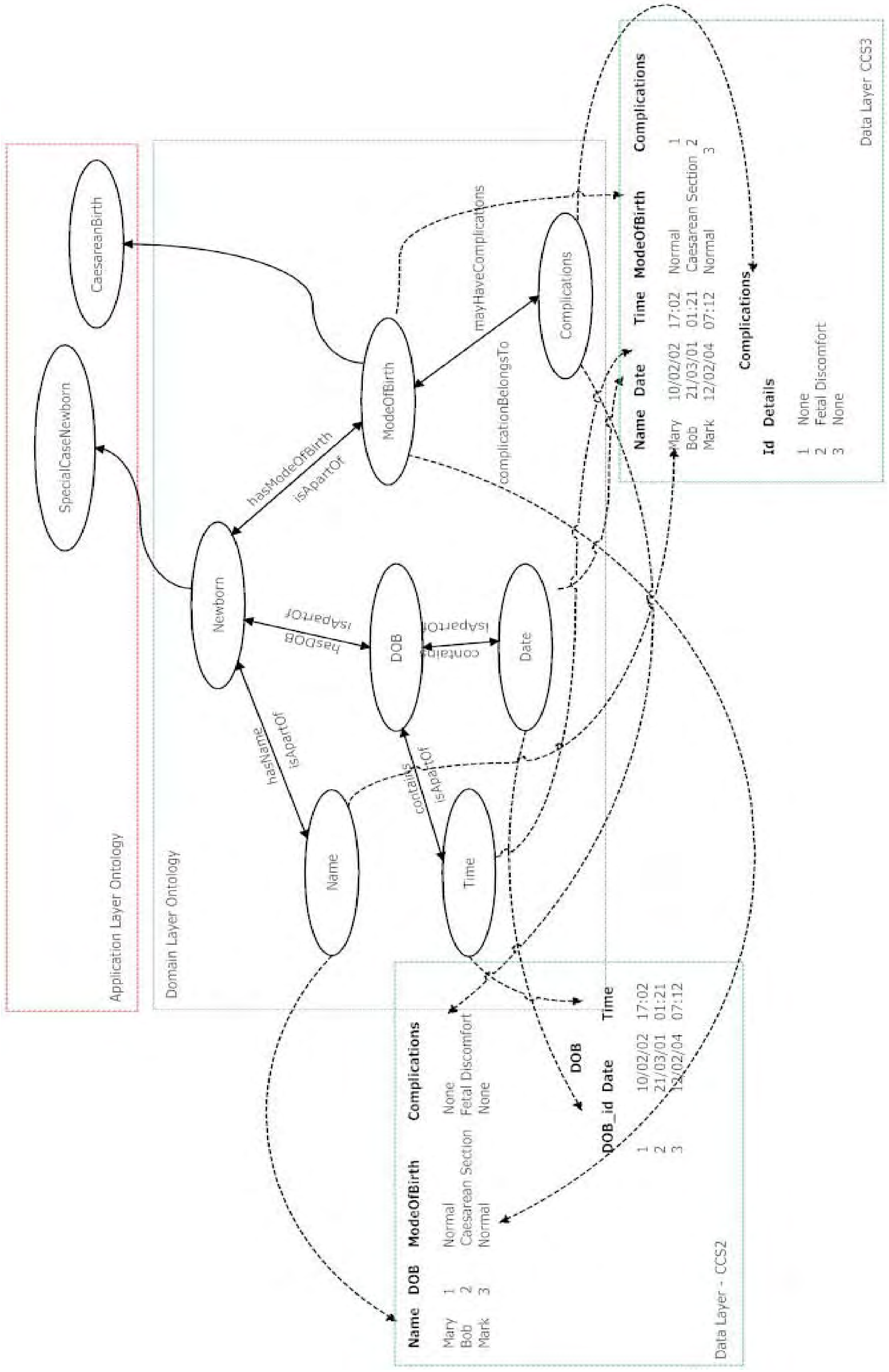


FIGURE 5-7: CONCEPTUAL ARCHITECTURE

The mapping between the data and ontology is illustrated in Figure 5-8 where the two data layers belong to CCS2 and CCS3 with the common domain layer ontology mapped to both. This is shown by the dotted arrows leading from the domain concept to the corresponding data stored in tables. The application layer ontology contains the new concepts built on top of the domain layer ontology by CCS1. The doctor requests for the desired data through an interface in CCS1 in the form of a search application. CCS1 would translate the inputs from the doctor into concepts in the application ontology. In this case, the concepts CaesareanBirth and SpecialCaseNewborn are defined in description logic statements (2) and (3) respectively in Section 5.2.2. CCS1 then sends out an agent A1 with instructions to request data for those concepts, that is, look for patient records of newborns that were born through a caesarean section due to fetal discomfort. Agent A1 speaks to Agent A2 giving the concepts CaesareanBirth and SpecialCaseNewborn. Agent A2 does not recognise these concepts due to the fact that CCS1's application ontology differs from CCS2. Agent A2 will refer to the domain ontology (Figure 5-5) which is shared between all the CCS and derive the meaning of both CaesareanBirth and SpecialCaseNewborn from the domain ontology. Now that Agent A2 understands the meaning of those concepts, it will retrieve the data mapped to the concepts used to create the application layer concepts CaesareanBirth and SpecialCaseNewborn. Similarly, Agent A1 repeats the same process with Agent A3. The data collected by Agent A1 is then formatted and displayed on CCS1's user interface.



Name	DOB	ModeOfBirth	Complications
Mary	1	Normal	None
Bob	2	Caesarean Section	Fetal Discomfort
Mark	3	Normal	None

DOB_id	Date	Time
1	10/02/02	17:02
2	21/03/01	01:21
3	12/02/04	07:12

Name	Date	Time	ModeOfBirth	Complications
Mary	10/02/02	17:02	Normal	1
Bob	21/03/01	01:21	Caesarean Section	2
Mark	12/02/04	07:12	Normal	3

Id	Details
1	None
2	Fetal Discomfort
3	None

FIGURE 5-8: MEDIATION ONTOLOGY

5.5 Evaluation and Discussion

In terms of satisfying the objectives that were initially set out, the following table summarises the evaluation of each (refer to section 3.1 for details of each objective):

Objective	Evaluation
1. Structured meta-model for reuse and sharing	Although a very simple domain ontology was used for this case study, we can still see potential reuse. The meta-model separates the application ontology from the domain ontology. This will allow the generic concepts such as "Name" or "DOB" to be reused.
2. Move business logic and domain knowledge from underlying agent code to higher level.	By allowing new and advanced concepts to be formulated in the ontology, new vocabulary is created for the agents. This will change their behaviour in terms of the understanding and retrieval of the information that is requested.
3. Facilitate the use of tools	Successfully demonstrated the use of tools such as the GT guided Tool, Protégé, Bean Generator etc.
4. Reuse existing ontology.	As with Chapter 4, both case studies were too small to make use of existing generic ontology, however, some concepts maybe borrowed from them.
5. Distinguish roles between domain expert and agent developer	There is a clear distinction between the roles of domain expert and agent developer. The two parts of MOMA separates these roles. However there is still a necessity for the agent to request information

	from the domain expert.
6. Enable Usability by domain expert without the agent developer	The ontology part of the methodology can easily be used by a domain expert in this case.

TABLE 5-1: CHAPTER 5 EVALUATION OF MOMA METHODOLOGY

5.6 Summary

This chapter looked at a minor case study in the e-Health domain which utilised the MOMA methodology presented in chapter 3. Scenarios in this case study were used as a proof of concept. The agent application solution in this case study solved the original interoperability problem. Similarly to the case study in Chapter 4, most of the original objectives set out for MOMA were met (although not fully). A discussion of the case study was presented in Section 5.5.

The next chapter conclude the thesis with further discussions, overall evaluation, limitations and consideration for future works.

Chapter 6. Conclusion

This chapter concludes the thesis by summarising what has been achieved in the previous chapters. This thesis started with an introduction into ontology-based multi-agent systems, their benefits in modelling the real world and ability to solve complex problems. The problem of existing AOSE methodologies lacking support for ontology was identified. MOMA was proposed as a solution and the design and research of the MOMA methodology was given.

Chapter 2 provided the background for the problem domains of ontology and multi agent systems, as well as their usefulness and limitations. A review of existing AOSE methodologies confirmed the original problem that we set out of solve.

In chapter 3, the MOMA methodology was proposed.

Chapter 4 presented a case study using MOMA to create an agent application for the simulation of the stock market. Another case study was presented in Chapter 5 in the domain of e-Health. This case study using MOMA was for creating an agent application to solve interoperability in information retrieval in the domain of e-Health.

The next section will present the overall evaluations of the MOMA methodology presented in Chapter 3. Section 6.1 will present evaluation of MOMA through the two case studies in Chapter 4 and 5. Section 6.2 will consider future work. Finally, Section 6.3 will finish with concluding remarks.

6.1 Evaluation

We have seen the MOMA methodology used through the two case studies in Chapters 4 and 5. A qualitative evaluation of the agent application in solving the original problem as well as how MOMA met its original objectives was given for each case study. The original objectives of MOMA are presented in Section 3.1.

In both case studies, a structure for reuse and sharing was established. However due to the nature of the case studies, the ontology were very simplistic and the full extent of reuse cannot be tested fully. In both cases, the generic ontology in each respective case study was simple and generic enough to be used for a similar application. Tools played an essential role in both case studies. They not only drove the development process, but were also a methodological way of completing tasks (e.g. concept identification). In both case studies, domain knowledge was moved from the agent code to ontology (at design time). However, some of the behaviour and business logic of the agents themselves could not be modelled in an ontology. This means that MOMA have not completely satisfied the objective of allowing the domain expert to specify the application at a higher level without having to have knowledge of lower level programming. The roles in each of domain expert and agent developer or software engineer were clearly separated by the two parts of development. However, we have found that communication between the domain expert and agent developer was still essential when implementing the application. Because the business logic of the agent application was not able to be modelled in ontology, the requirements must be passed onto the agent developer so that they can implement those directly into the agents themselves.

6.1.1 Discussion

The case study scenarios allowed the methodology to be tested in real world domains, demonstrating that the methodology is versatile enough to work in a wide and diverse range of domains and situations.

The size and extent of the case studies were quite small and therefore made it was difficult to determine the difference between required developments time and effort compared to traditional formal AOSE methodologies. However, for the same end result, fewer steps were required when using MOMA. In terms of implementation, MOMA makes use of many tools and techniques such as the Protégé IDE and Bean Generator to speed up the development process. Once again this was difficult to compare as the majority of the AOSE methodologies are used only for the design and analysis stages.

Collaboration required between the programmer and domain expert was considerably reduced in the implementation stages. This was because the ontology was generated into Java code and provided as stub files for the programmer to work with. However in terms of design and analysis, the programmer role was not required, but instead knowledge of agent modelling expert was. This possibly had not been originally considered.

In terms of facilitating development with the use of tools, MOMA uses tools throughout modelling, code generation, implementation and testing. These tools considerably helped speed the process of development.

Compared to the AOSE methodologies analysed, MOMA undoubtedly offers better support for the use of ontology. Very few of the existing AOSE methodologies support the use of ontology as none are ontology driven.

6.1.2 Limitations

There are several limitations of MOMA that were realised both before and after the case studies.

Section 3.1.1 details the limitations that were noted before the case studies were carried out. These are summarised below:

- MOMA does not address the design of agents and agent “societies”, nor the interaction and behaviour of agents with established agent theories.
- MOMA uses the Java Agent DEvelopment Framework (JADE) agent framework implemented in Java (refer to Section 3.4.2 for details). This limits MOMA to a single architecture.
- MOMA introduces support for adding logic in ontology through rules and axioms, but it does not contain a methodology to formulate these from the requirements.
- The ontology exists only at a conceptual level at the time of design.

After carrying out the case study, a few new problems and limitations were discovered. These are detailed in the evaluation sections of Chapter 4 and 5.

- The ontology development steps are not explicitly iterative.
- There is a missing sub-step for customising domain and task ontology for discovering intermediate concepts.

6.2 Future works

Continuation of research will focus mostly on the methodology itself, and the extension of the methodology to address the above mentioned limitations. Below is a list of the major areas that can be considered as future work.

1. Expanding case studies into various business sectors with more complex problems and real users. This will allow us to see if MOMA can be used other domains to solve different problems.
2. Testing usability and implementation performance. In this thesis, we have assumed the role of the domain expert in MOMA. Usability studies needs to be conducted for real users (i.e. experts in different fields). The resulting implementation also needs to be tested for its performance in solving the problem at hand.
3. Support for generation of axioms and rules into code will definitely be useful. By having a tool that does this, we can extract both business logic and domain knowledge from the agent implementation in the development process.
4. The GT Guided Tool implementation is for now a proof of concept. An improved user interface is required to render it more usable. Features like word count and text analysis would also be welcomed to make the tool easier to use and help the domain expert identify concepts from large amounts of text quickly.
5. The Agent Development part of MOMA was treated as a black box. Although the ontology were intended for consumption by agents, it would be interesting to see if MOMA would work with different emerging technologies other than agents. For example, semantic web services.

6.3 Concluding remarks

In summary, the research in this thesis proposed a design methodology for ontology-based multi-agent applications called MOMA. MOMA improves on existing AOSE methodologies in terms of the support for the use of ontology. MOMA was also intended to be used by domain experts and researchers without the agent development and software engineering expertise and knowledge. Through the use of tools and driven by ontology, MOMA was applied through two case studies in the domains of financial services and e-Health. It is hoped that future work will improve upon MOMA by integrating the agent development part with better tools. Ultimately, we envision MOMA as a useful methodology for the creation of ontology-based agent applications that can be used by domain experts.

References

1. Abrahamsson, P., Warsta, J. S. & Ronkainen, J. 2003, 'New Directions on Agile Methods: a Comparative analysis', *Proceedings of the International Conference on Software Engineering*, Portland, Oregon, USA.
2. Alonso, S., Bas, J., Bellido, S., Contreeas, J., Benjamins, R., Gomez, J. 2005 *WP10: Case study eBanking D 10.7 Financial Ontology DIP.*
3. Benjamins, R., de Barros, L. & Valente, A., 1996, Banff, Canada.
4. Bergenti, F. & Poggi, A. 2001, 'Agent-oriented Software Construction with UML', *Handbook of Software Engineering and Knowledge Engineering Vol 2*, ed. S.K. Chang, pp. 757-770, Singapore: World Scientific Publishing Co.
5. Bernon, C., Gleizes, M., Picard, G., & Glize, P. 2002, 'The ADELFE methodology for an intranet system design', *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems*, Toronto, Canada.
6. Bessel, TL., MacDonald, S., Silagy, C.A., Anderson, J.N., Hiller, J.E. & Sansom, L.N. 2002, 'Do internet interventions for consumer cause more harm than good?' A systematic review, *Health Expect* 2002; 5(1), pp. 28-37
7. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., & Perini, A. 2004, 'TROPOS: An Agent-Oriented Software Development Methodology', *Journal of Autonomous Agents and Multi-Agent Systems 8*, pp. 203-236
8. Burrafato, P., & Cossentino, M. 2002, 'Designing a multi-agent solution for a bookstore with the PASSI methodology', *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented INformation Systems*, Toronto, Canada.
9. Calvanese, D., de Giacomo, G., & Lenzerini, M. 2001 'A framework for ontology integration', *Proceedings of the 1st International Semantic Web Working Symposium*, Stanford, USA, pp. 303–317
10. Castro, J., Kolp, M., & Mylopoulos, J. 2001, 'A Requirements-Driven Development Methodology', *Proceedings of the 13th International Conference on Advanced Information Systems Engineering CAiSE01*, Interlaken, Switzerland
11. Castro, J., Kolp, M., & Mylopoulos, J. 2002, 'Towards Requirements-Driven information Systems Engineering: The Tropos Project', *Information Systems 27*, pp. 365-389

12. Chandrasekaran, B., Josephson, J.R., & Benjamins, V.R. 1999, ‘What are ontologies, and why do we need them?’, *IEEE Intelligent Agents 14(1)*, pp. 20-26

13. Collinot, A., & Drogoul, A. 1998, ‘Using the Cassiopeia Method to Design a Soccer Robot Team’, *Applied Artificial Intelligence Journal 12*, pp. 127-147

14. Collinot, A., Drogoul, A., & Benhamou, P. 1996, ‘Agent Oriented Design of a Soccer Robot Team’, *Proceedings of the 2nd International Conference on Multi-Agent Systems*, Kyoto, Japan, pp. 41-47

15. Cossentino, M., & Potts, M. 2002, ‘A CASE tool supported methodology for the design of multi-agent systems’, *Proceedings of the 2002 International Conference on Software Engineering Research and Practice*

16. Cristani, M., Cuel, R. 2005, *A Survey on Ontology Creation Methodologies 2005*

17. Davies, J., Fensel, D., & Van Harmelen, F. 2003, *Towards the semantic web:ontology-driven knowledge management*, Chichester, England: Wiley.

18. Davis, J., Studer, R. & Warren, P. (ed.) 2006, ‘Semantic Web Technologies: Trends and Research in Ontology-based Systems’, *John Wiley 2006*, ISBN: 978-0-470-02596-3

19. Decker, S., Erdmann, M., Fensel, D. & Studer, R. 1999, ‘Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information’, *Proceedings of the IFIP TC2/WG2.6 8th Working Conference on Database Semantics-Semantic Issues in Multimedia Systems*, New Zealand, pp. 351-369

20. Deloach, S. A., Wood, M. F. & Sparkman, C. H. 2001, ‘Multiagent Systems Engineering’, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No.3, pp. 231-258

21. DiLeo, J., Jacobs, T. & DeLoach, S. 2002, ‘Integrating Ontologies into Multiagent Systems Engineering’, Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems, Bologna, Italy

22. Ding, Y. 2001, ‘The role of ontology’, *Proceedings of the 4th International Conference of Asian Digital Libraries*, Bangalore, India

23. Ehrig, M., & Sure, Y. 2004, ‘Ontology Mapping an Integrated Approach’, *Proceedings of the 1st European Semantic Web Symposium*, Heraklion, Greece

24. Elammari, M. & Lalonde, W. 1999, ‘An Agent-Oriented Methodology: High Level and Intermediate Models’, *Proceedings of the 1st Bi-Conference Workshop on Agent-Oriented Information Systems,* Heidelberg, Germany

25. Eurescom 2001, *MESSAGE: Methodology for Engineering Systems of Software Agents - Final Guidelines for the Identification of Relevant Problem Areas where Agent Technology is Appropriate*, [Online], Available from: <http://www.eurescom.de/public/projectresults/P900-series/907d2.asp>

26. FIPA 2002 “FIPA Interaction Protocols, [Online], Available from: <http://www.fipa.org/repository/ips.php3> [March 2008]

27. Falasconi, S., Lanzola, G. & Stefanelli, M. 1996, ‘Using Ontologies in Multi-Agent Systems’, in *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW’96)*, Banff, Canada

28. Fan, X. 2000, ‘Towards a building methodology for software agents’, *Proceedings of the 6th International Conference on Object-Oriented Information Systems*,London, UK, pp. 45-53

29. Fensel, D. 2001, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, Berlin

30. Flores-Mendez, R. 1999, ‘Towards a Standardization of Multi-Agent System Frameworks’, *ACM Crossroads Student Magazine*, [Online], Available from: <http://www.acm.org/crossroads/xrds5-4/multiagent.html>

31. Gamper, J., Nejdl, W. & Wolpers, M. 1999, ‘Combining Ontologies and Terminologies in Information Systems’, in *Proceedings of the 5th International Congress on Terminology and Knowledge Engineering*, Innsbruck, Austria, 152-168.

32. Ganguly, P., Ray, P. & Parameswaran, N. 2005, ‘Semantic Interoperability in Telemedicine through Ontology-Driven Services’, *Telemedicine and e-Health Journal*, Vol. 11, No. 3

33. Girardi, R., & deFaria, C. 2004, ‘An Ontology-Based Technique for the Specification of Domain and User Models in Multi-Agent Domain Engineering’, *Clei Electronic Journal 7*

34. Glaser, N. 1997, ‘The CoMoMAS Approach: From Conceptual Models to Executable Code’, *Proceedings of the 8th European Workshop On Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering*, Ronneby, Sweden

35. Glaser, N. 1997, ‘The CoMoMAS Methodology and Environment for Multi-Agent System Development’, *Multi-Agent Systems - Methodologies and Applications*, pp. 1-16

36. Guarino, N. (ed.) 1997, ‘Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration’, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, Springer Verlag, Berlin, pp. 139-170

37. HMT. 2005, ‘$162 Billion in Annual Savings Possible’, *Health Management Technology*, Nov 2005; 26, 11; ABI/INFORM Global

38. Henderson-Sellers, B., Simons, A. & Younessi, H. 1998, *The OPEN Toolbox of Techniques*, Addison Wesley Longman Ltd, England

39. Hevner, A.R., S.T., March, J., Park, J., & Ram, S. 2004, ‘Design science in information systems research’, *MIS Quarterly*, 28(1), pp. 75-106

40. Honavar, V. 1999, *Intelligent Agent and Multi Agent Systems*, Tutorial at IEEE CEC

41. Horlait, E. 2003, *Mobile Agents for Telecommunication Applications*,Kogan Page Science, England

42. Hwang, C.H. 1999, ‘Incompletely and imprecisely speaking: Using dynamic ontologies for representing and retrieving information’, *Technical report*, Microelectronics and Computer Technology Corporation, Texas, USA

43. Iglesias, C., Garijo, M., Gonzalez, J. & J.R., V. 1996, ‘A Methodological Proposal for Multi-Agent Systems Development Extending CommonKADS’, *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada

44. Iglesias, C., Garijo, M., Gonzalez, J., & J.R., V. (1998). Analysis and Design of Multi-Agent Systems using MAS-CommonKADS. Intelligent Agents IV (LNAI Volume 1365), pp. 313-326

45. Jennings, N.W. 1995, ‘Applying Agent Technology’, *Applied Artificial Intelligence* 9, (4)351-359

46. Kalfoglou, Y., Schorlemmer, M. 2003, *The Knowledge Engineering Review* 18, Cambridge University Press, pp. 1-31

47. Kalfoglou, Y., Schorlemmer, M. 2003, ‘IF-Map: an ontology mapping method based on Information Flow theory’, *Journal on Data Semantics* 1(1), pp. 98-127

48. Katehakis, D.G., Sfakianakis, S., Tsiknakis, M. & Orphanoudakis, S.C. 2001, ‘An Infrastructure for integrated electronic health record services: The role of XML’, *J. Medical Internet Res.*, Vol. 3, No. 1, pp. e7

49. Kendall, E. 1999, ‘Role modelling for agent system analysis, design and implementation’, *Proceedings of the 1st International Symposium on Agent Systems and Applications*, Palm Strings, California, pp. 204-218

50. Kinny, D., Georgeff, M. & Rao, A. 1996, ‘A Methodology and Modelling Technique for Systems of BDI Agents’, *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherland, pp. 56-71

51. Knoblock, C., Arens, A. & Hsu, C. 1994, ‘Cooperating Agents for Information Retrieval’, *Proceedings of the 2nd International Conference on Cooperative Information Systems*, Toronto, Canada

52. Kuziemsky, C.E., Downing, M., Black, F.M., Lau, F. 2007, ‘A grounded theory guided approach to palliative care systems design’, *International Journal of Medical Informatics*, 76S 2007 S141-S148

53. Lenat, D.B., & Guha, R.V. 1990, *Building large knowledge-based systems: Representation and inference in the CYC project*, Addison-Wesley, USA

54. Lesser, V. 1996, ‘Cooperative Multi-agent systems: A Personal View of the State of the Art’, *IEEE Transactions on Knowledge and data engineering*, pp. 133-142

55. Lind, J. 2000, ‘Issues in Agent-Oriented Software Engineering’, *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, Limerick, Ireland, pp. 45-58

56. Lind, J. 2000, ‘The MASSIVE development method for Multiagent Systems’, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agents*, Manchester, UK

57. Madhavan, J., Bernstein, P.A., Domingos, P. & Halevy, A.Y. 2002, ‘Representing and reasoning about mappings between domain models’, *Proceedings of the 18th National Conference on Artificial Intelligence*, Alberta, Canada, pp. 80-86

58. Malucelli, A. & Oliveira, E. 2004, ‘Ontology-Services Agent to Help in the Structural and Semantic Heterogeneity’, *Proceedings of PRO-VE'04 - 5th IFIP Working Conference on Virtual Enterprises*, Toulouse, France

59. March, S. & Smith, G.F. 1995, ‘Design and natural science research on information technology’, *Decision Support Systems* 15, pp. 251-266

60. Mars, N.J.I., Ter Stal, W.G., de Jong, H., van der Vet, P.E. & Speel, P.H. 1994, ‘Semi-automatic Knowledge Acquisition in Plinius: An Engineering Approach’, *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, 4.1-4.15

61. Mountzia, M. 1996, 'An Intelligent-Agent based Framework for Distributed Systems Management', *Proceedings of the 3rd HP OVUA Workshop*, Toulouse, France

62. Mukherjee, R., Dutta, P. & Sen, S. 2000, 'Analysis of domain specific ontologies for agent-oriented information retrieval', *Working notes of the AAAI-2000 Workshop on Agent-Oriented Information Systems*

63. Noy, N.F. & McGuinness, D.L. 2008, *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford University, Stanford, [Online], Available from: <http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html> [June 2008]

64. Neuberg, L. & Bertels, K. 2003, 'Heterogeneous Trading Agents', *Complexity Journal*, May, pp. 28-35

65. Niles, I. & Pease, A. 2001, 'Towards a Standard Upper Ontology', *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Chris Welty and Barry Smith (ed.), Ogunquit, Maine, October 17-19

66. Nwana, H.S. 1996, 'Software Agents: An Overview', *Knowledge Engineering Review,* Cambridge University Press, pp. 1-40

67. Object Management Group. 2003, OMG Unified Modelling Language Specification, [Online], Available from: <http://www.omg.org/technology/documents/formal/uml.htm> [20 September 2008]

68. Omicini, A. 2000, 'SODA: Societies and Infrastructure in the Analysis and Design of Agent-Based Systems', *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, pp. 185-194

69. Orguna, B., Vub, J. 2005, 'HL7 ontology and mobile agents for interoperability in heterogeneous medical information systems', *Computers in Biology and Medicine*, 2005, Aug 30 [Epub ahead of print]

70. Padgham, L. & Winikoff, M. 2002, 'Prometheus: A methodology for developing intelligent agents', *Proceedings of 3rd International Workshop on Agent-Oriented Software Engineering*, Bologna, Italy

71. Padgham, L. & Winikoff, M. 2002, 'Prometheus: A pragmatic methodology for engineering intelligent agents', *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, Seattle, USA, pp. 97-108

72. Parent, C. & Spaccapietra, S. 1998, ‘Issues and approaches of database integration’, *Communications of the ACM* 41(5), pp. 166-178

73. Pavon, J. & Gomez-Sanz, J. 2003, ‘Agent Oriented Software Engineering with INGENIAS’, *Proceedings of the 3rd International Central and Easter European Conference on Multi-Agent Systems*, Prague, Czech Republic, pp. 394-403

74. Pazzaglia, J-C.R. & Embury, S.M. 1998, ‘Bottom-up Integration of Ontologies in a Database Context’, *Proceedings of the 5thInternational Workshop on Innovative Application Programming and Query Interfaces*, Seattle, USA, 7.1-7.7

75. Peng, Y., Finin, T., Labrou, Y., Chu, B., Long, J. & WJ. 1998, ‘A Multi-Agent System for Enterprise Integration’, *International Journal of Agile Manufacturing*

76. *RACER 2007 Renamed Abox and Concept Expression Reasoner* <http://www.sts.tu-harburg.de/~r.f.moeller/racer/> [November 2007]

77. Ray, P. 2003, *Integrated Management from E-Business Perspective - Concepts, Architectures and Methodologies*, Kluwer Academic/Plenum Publishers, New York

78. Rector, A.L., Nolan, W.A. & Kay, S. 1991, ‘Foundations for an electronic medical record’, *Methods Inform. Med.*, Vol. 30, pp. 179-186

79. *SFI Artificial Stock Market*, [Online], Available from: <http://www.santafe.edu/sfi/publications/Bulletins/bulletinFall99/news/stockMarket.html> [March 2004]

80. SWRL, [Online], Available from: <http://www.w3.org/Submission/SWRL/> [June 2008]

81. Schreiber, A., Wielinga, B., de Hoog, R., Akkermans, J. & Van de Velde, W. 1994, CommonKADS: A comprehensive methodology for KBS development, *IEEE Expert* 9, pp. 28-37

82. Searle, J.R. 1969, *Speech Acts*, Cambridge University Press

83. Shave, M. 1997, ‘Ontological Structures for Knowledge Sharing’, *New Review of Information Networking* 3, pp. 125-133

84. Smith, B. & Ceusters, W. 2006, ‘HL7 RIM: An Incoherent Standard’, *Studies in Health Technology and Informatics*, 124, pp. 133–138, Presented at Medical Informatics Europe, Maastricht, August 2006

85. Snoussi, H., Magnin, L. & Nie, J. 2003, ‘Toward an Ontology-based Web Data Extraction’, *The Fifteenth Canadian Conference on Artificial Intelligence AI 2002, BASeWEB Proceedings*

86. Sugumaran, V. & Storey, V. 2001, 'Creating and Managing Domain Ontologies for Database Design', *Proceedings of the 6th International Workshop on Applications of Natural Language to Information Systems*, Madrid, Spain, pp. 17-26

87. Sujanani, A., Ray, P., Bhar, R. & Paramesh, N., 2005, 'The Development of Ontology Driven Multi-Agent Systems: A Case Study in the Financial Services Domain', *Proceedings of the International Workshop on Business Services and Networks (BSN05)*, Hongkong, March 2005

88. Sycara, K. 1998, 'Multiagent Systems', *AI Magazine* 19, pp. 79-92

89. UMLS Unified Medical Language System, [Online], Available from: <http://www.nlm.nih.gov/research/umls/> [November 2007]

90. Uschold, M.A. 1996, 'Ontologies: principles, methods, and applications', *Knowledge Engineering Review*, 11(2), pp. 93-155

91. Valente, A. 1995, 'Legal knowledge engineering, a modeling approach', *Doctorial Dissertation*, IOS Press, Amsterdam

92. van Heijst, G., Schreiber, A. & Wielinga, B. 1997, 'Using Explicit Ontologies in KBS Development', *International Journal of Human Computer Studies* 46, pp. 183-292

93. Vergara, J.E., de López, V.A., Villagrá, J.I., Asensio, J., Berrocal 2003, 'Ontologies: Giving Semantics to Network Management Models', *IEEE Network, special issue on Network Management*, Vol. 17, No. 3, May/June

94. W3C. n.d., *OWL Web Ontology Language*, [Online], Available from: http://www.w3.org/TR/owl-ref/

95. Wimalasiri J. S., Ray p., and Wilson C.S., An Ontology Driven Multi-agent Approach to Electronic Prescriptions, Proceedings of the 24 th Conference of Health Information Management Association of Australia (HIMAA03), Sydney, Australia, July 2003

96. Winikoff, M. & Padgham, L. 2004, 'The Prometheus Methodology. Methodologies and Software Engineering for Agent Systems', *The Agent Oriented Software Engineering Handbook*, Chapter 11

97. Winston, M.E., Chaffin, R. & Herrmann, D. 1987, 'A taxonomy of part-whole relations', *Cognitive Science*, Volume 11, Issue 4, October-December, pp. 417-444

98. Wong, A., Ray, P., Parameswaran, N. & Strassner, J. 2003. 'Ontology Mapping for the Interoperability Problem in Network Management', *IEEE Journal of Special Applications*

99. Wooldridge, M. 1999, ‘Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence’, *Intelligent Agents,* London: The MIT Press, pp. 27-77

100. Wooldridge, M. 2002, *An Introduction to MultiAgent Systems*, John Wiley & Sons Ltd, ISBN 0-471-49691-X

101. Wooldridge, M.A. 2000, ‘Agent-Oriented Software Engineering: The State of the Art’, *1st International Workshop on Agent Oriented Software Engineering*, Limerick, Ireland, pp. 1-28

102. Wooldridge, M., Jennings, N., & Kinny, D. 2000, ‘The Gaia Methdology for Agent-Oriented Analysis and Design’, *Autonomous Agents and Multi-Agent Systems* 3, pp. 285-312

103. Ying W., Ray P., Paramesh N. 2006 “Ontology Engineering in eFinance”, workshop paper in the *1st Asian Semantic Web Conference,* Beijing 2006

104. Ying W., Sujanani A., Ray P., Paramesh N., Lee D. 2009 “The development of ontology driven multi-agent systems: A case study in the financial services domain” – accepted for publishing in the Computer and Informatics Journal Vol. 3 2009.

105. Zambonelli, F. 2000, ‘Organisational Abstractions for the Analysis and Design of Multi-Agent Systems’, *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, pp. 127-141

106. Zambonelli, F., Jennings, N. & Wooldridge, M. 2003, ‘Developing Multiagent Systems: The Gaia Methodology’, *ACM Transactions on Software Engineering and Methodology* 12, pp. 317-370

107. Zhang, Z., Zhang, C. 2003, ‘Agent-Based Hybrid Intelligent Systems’, *Design and application of hybrid intelligent systems*, ISBN:1-58603-394-8, pp. 799-808

108. Zhang, Z., Zhang, C. & Ong, S. 2000, ‘Building an Ontology for Financial Investment’, *Proceedings Of Intelligent Data Engineering and Automated Learning - IDEAL 2000: Data Mining*, Financial Engineering, and Intelligent Agents 19, LNCS 1983, 2000

List of Publications

1. Weir Ying, Pradeep Ray, Paramesh N. "Ontology Engineering in eFinance" – in the 1st Asian Semantic Web Conference Workshop proceedings, Beijing 2006.

2. Weir Ying, Anjalee Sujanani, Pradeep Ray, N. Paramesh, Damien Lee "The development of ontology driven multi-agent systems: A case study in the financial services domain" – accepted in the Computing and Informatics Journal Vol.3 to appear in April 2009.

3. Weir Ying, Jaminda S. Wimalasiri, Pradeep Ray, Subhagata Chattopadhyay and Concepción S. Wilson "An Ontology Driven Multi-agent Approach to Integrated e-Health Systems" - submitted for publication.

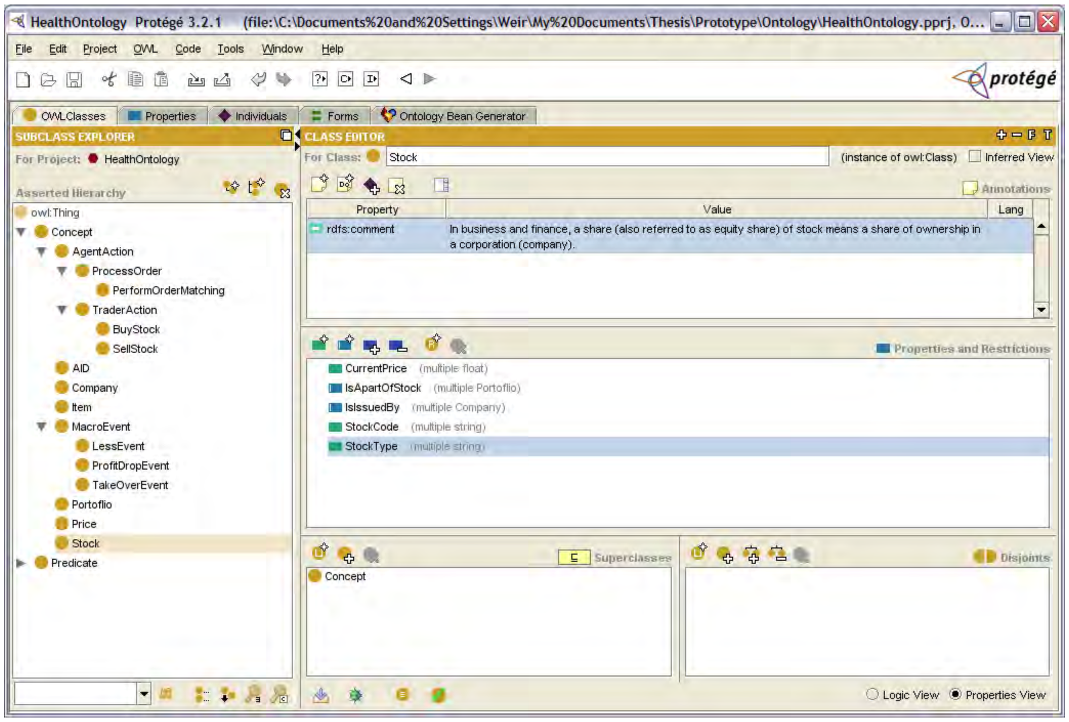
4. Weir Ying, Pradeep Ray "Methodology for development of ontology-based intelligent systems" - submitted for publication.

Appendix A: Chapter 4 Implementations

Appendix A1: Chapter 4 ontology implementation in OWL/RDF

File path /Chapter 4/ontology/

Development snapshot of the ontology being modelled in Protégé is shown below:



HealthOntology Protégé 3.2.1 (file:\C:\Documents%20and%20Settings\Weir\My%20Documents\Thesis\Prototype\Ontology\HealthOntology.pprj, O...

For Project: HealthOntology

```
owl:Thing
▼ Concept
    ▼ AgentAction
        ▼ ProcessOrder
            PerformOrderMatching
        ▼ TraderAction
            BuyStock
            SellStock
      AID
      Company
      Item
    ▼ MacroEvent
        LessEvent
        ProfitDropEvent
        TakeOverEvent
      Portoflio
      Price
      Stock
▶ Predicate
```

For Class: Stock (instance of owl:Class) Inferred View

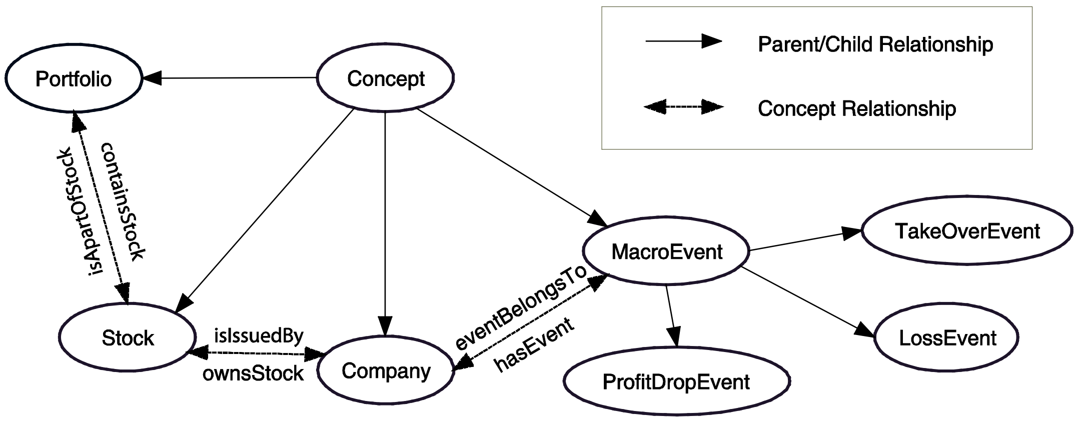
Property	Value	Lang
rdfs:comment	In business and finance, a share (also referred to as equity share) of stock means a share of ownership in a corporation (company).	

```
CurrentPrice   (multiple float)
IsApartOfStock (multiple Portoflio)
IsIssuedBy     (multiple Company)
StockCode      (multiple string)
StockType      (multiple string)
```

Superclasses: Concept

Appendix A2: Chapter 4 generated java code

The following code was generated by Bean Generator for the ontology illustrated in Figure 4-1 shown below.



Each of the Java files listed below represent the concepts. Relationships and attributes of each concept are contained in each concept class as methods.

Files:

OntoMarketSimOntology.java
Company.java
Stock.java
Portfolio.java
TakeOverEvent.java
DropProfitEvent.java
LossEvent.java

These files are in the folder /Chapter4/generated/

Appendix B: Chapter 5 Implementations

Appendix B1: Chapter 5 data source for GT Guided Tool

Source one: "Foundation for the Electronic Health Record: An Ontological Analysis of the HL7's Reference Information Model" by Lowell Vizenor, Barry Smith, Werner Ceusters http://ontology.buffalo.edu/medo/HL7_2004.pdf

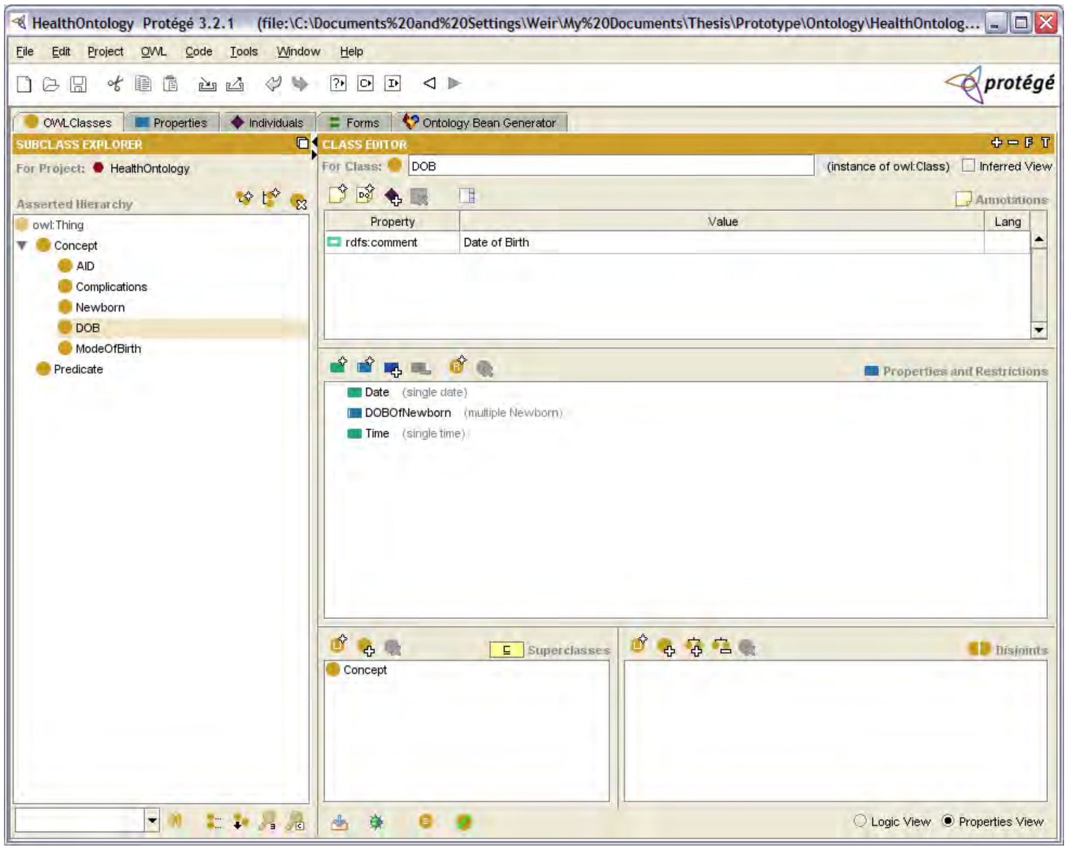
Source two: An experienced medical doctor.

Source three: UMLS. The Unified Medical Language System (UMLS) is a compendium of many controlled vocabularies in the biomedical sciences. It provides a mapping structure among these vocabularies and thus allows one to translate among the various terminology systems; it may also be viewed as a comprehensive thesaurus and ontology of biomedical concepts. We used UMLS as a source of reference for concept identification.

Appendix B2: Chapter 5 ontology implementation in OWL

File path /Chapter 5/ontology/

Development snapshot of the ontology being modelled in Protégé is shown below:


HealthOntology Protégé 3.2.1 (file:\C:\Documents%20and%20Settings\Weir\My%20Documents\Thesis\Prototype\Ontology\HealthOntolog...
File Edit Project OWL Code Tools Window Help
protégé
OWLClasses Properties Individuals Forms Ontology Bean Generator
SUBCLASS EXPLORER
For Project: HealthOntology
Asserted Hierarchy
owl:Thing
Concept
AID
Complications
Newborn
DOB
ModeOfBirth
Predicate
CLASS EDITOR
For Class: DOB
(instance of owl:Class)
Inferred View
Annotations
Properties and Restrictions
Superclasses
Disjoints
Logic View
Properties View

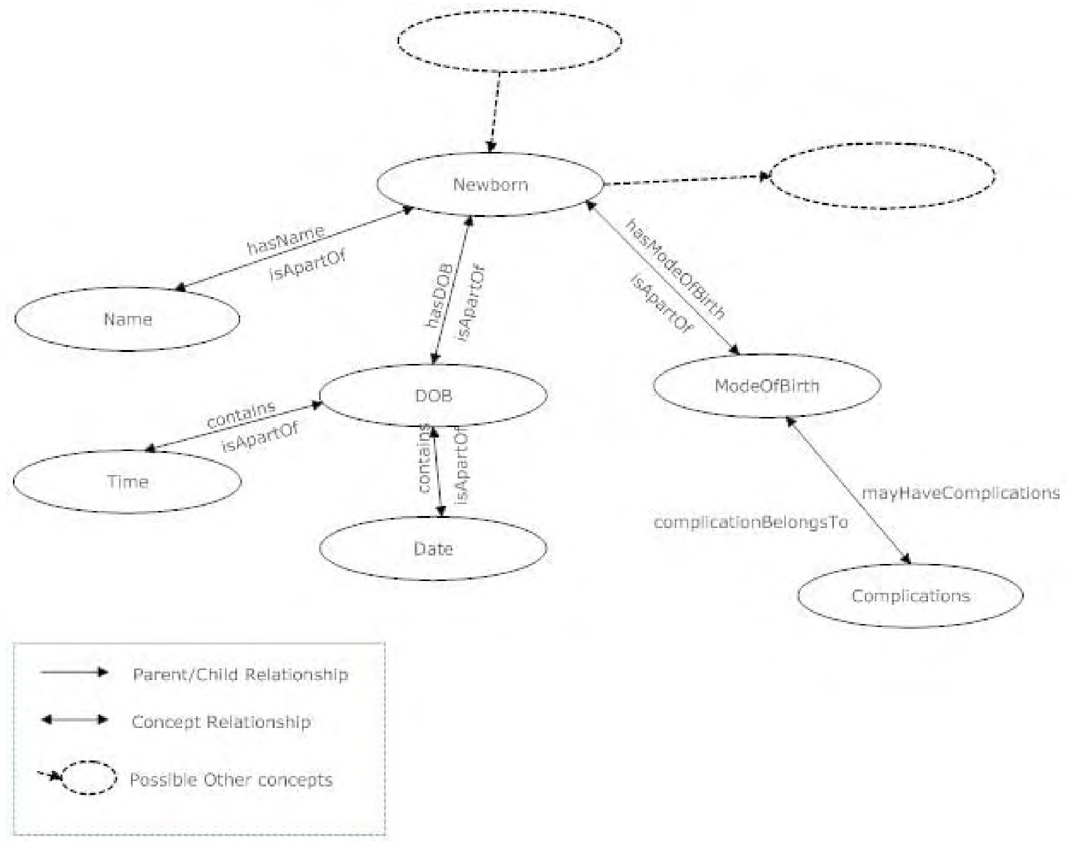
Property	Value	Lang
rdfs:comment	Date of Birth	

Date (single date)
DOBOfNewborn (multiple Newborn)
Time (single time)

Concept

Appendix B3: Chapter 5 generated java code

The following code was generated by Bean Generator for the ontology illustrated in Figure 5-4 shown below.



Each of the Java files listed below reprsent the concepts. Reslationships and attributes of each concept are contained in each concept class as methodods.

Files:

NewbornOntology.java:
Complications.java
DOB.java:
ModeOfBirth.java:
NewBorn.java

Appendix C: Resources and source code

The following appendixes represent the files that are located on the attached CD.

Appendix C1: GT Guided Tool implementation

The GT tool is implemented in Java using Java's SWING as the graphical user interface.

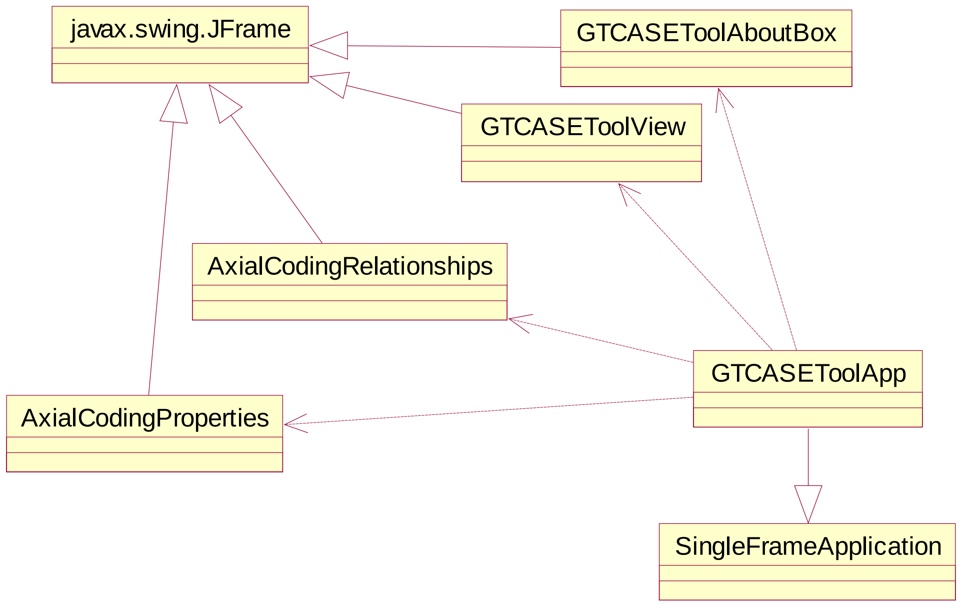


FIGURE 7-1: CLASS ASSOCIATION FOR THE MAINS FILES OF GT GUIDED TOOL IMPLMENATION

The GT Guided Tool is started by GTCASEToolApp.java and instantiates the rest of the SWING JFrame GUI.

The main files are:

AxialCodingProperties.java
AxialCodingRelationships.java
GTCASEToolAboutBox.java
GTCASEToolApp.java
GTCASEToolView.java
Output.java

The files are packaged in NetBeans IDE (http://www.netbeans.org) working directory form. This can be directed imported as a project in NetBeans IDE.

Appendix C2: Other tools

A list of tools and packages that is used to develop and run the agent application:

Protege 3.3.1

Protégé is a free, open source ontology editor and a knowledge acquisition system. Like Eclipse, Protégé is a framework for which various other projects suggest plugins. This application is written in Java and heavily uses Swing to create the rather complex user interface. Protege recently has over 100,000 registered users.

Protégé is being developed at Stanford University in collaboration with the University of Manchester.

Bean Generator

The ontology bean generator plugin is a Protégé Tab widget which generates java files representing an ontology that can be used with the JADE environment. With the beangenerator tool you can generate FIPA/JADE compliant ontologies from RDF(S), XML and Protégé projects.

JADE 3.3

Java Agent DEvelopment Framework, or JADE, is a software framework for multi-agent systems, in Java that has been in development since at least 2001. The JADE platform allows the coordination of multiple FIPA-compliant agents and the use of the standard FIPA-ACL communication language in both SL and XML.

NetBeans 6.5

NetBeans refers to both a platform for the development of applications for the network (using Java, JavaScript, PHP, Python, Ruby, Groovy, C, and C++), and an integrated development environment (IDE) developed using the NetBeans Platform.

Java SDK 6.5

The Java Development Kit (JDK) is a Sun Microsystems product aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java SDK.