MOBMAS - A methodology for ontology-based multi-agent systems development

Author:
Tran, Quynh Nhu

Publication Date:
2005

DOI:
https://doi.org/10.26190/unsworks/23631

License:
https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/24254 in https://unsworks.unsw.edu.au on 2024-04-27

MOBMAS

- A Methodology For Ontology-Based Multi-Agent Systems Development

by

Quynh Nhu Tran

B. Sc. (Hons), University of Newcastle, Australia

Submitted in total fulfilment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

July 2005

School of Information Systems, Technology and Management
The University of New South Wales
Australia

CERTIFICATE OF ORIGINALITY

I hereby declare that this submission is my own work and to the best of my knowledge it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at The University of New South Wales or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the resarch by others, with whom I have worked at The University of New South Wales or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Quynh Nhu Tran

ABSTRACT

"Agent-based systems are one of the most vibrant and important areas of research and development to have emerged in information technology in the 1990s" (Luck et al. 2003). The use of agents as a metaphor for designing and constructing software systems represents an innovative movement in the field of software engineering: "*Agent-Oriented Software Engineering* (AOSE)" (Lind 2000; Luck et al. 2003).

This research contributes to the evolution of AOSE by proposing a *comprehensive ontology-based methodology for the analysis and design of Multi-Agent Systems* (MAS). The methodology is named **MOBMAS**, which stands for **"*M*ethodology for *O*ntology-*B*ased *MAS*s"**. A major improvement of MOBMAS over the existing agent-oriented MAS development methodologies is its explicit and extensive **support for *ontology-based* MAS development**. Ontologies have been widely acknowledged for their significant benefits to interoperability, reusability, MAS development activities (such as system analysis and agent knowledge modelling) and MAS operation (such as agent communication and reasoning). Recognising these desirable ontology's benefits, MOBMAS endeavours to identify and implement the various ways in which ontologies can be used in the *MAS development process* and integrated into the *MAS model definitions*. In so doing, MOBMAS has exploited ontologies to enhance its *MAS development process* and *MAS development product* with various strengths. These strengths include those ontology's benefits listed above, and those additional benefits uncovered by MOBMAS, e.g. support for verification and validation, extendibility, maintainability and reliability. Compared to the numerous existing agent-oriented methodologies, MOBMAS is the first that explicitly and extensively investigates the diverse potential advantages of ontologies in MAS development, and which is able to implement these potential advantages via an ontology-based MAS development process and a set of ontology-based MAS model definitions.

Another major contribution of MOBMAS to the field of AOSE is its ability to address all key concerns of MAS development in one methodological framework. The methodology provides support for a comprehensive list of methodological requirements,

which are important to agent-oriented analysis and design, but which may not be well-supported by the current methodologies. These methodological requirements were identified and validated by this research from three sources: the existing agent-oriented methodologies, the existing evaluation frameworks for agent-oriented methodologies and conventional system development methodologies, and a survey of practitioners and researchers in the field of AOSE. MOBMAS supports the identified methodological requirements by combining the strengths of the existing agent-oriented methodologies (i.e. by reusing and enhancing the various strong techniques and model definitions of the existing methodologies where appropriate), and by proposing new techniques and model definitions where necessary.

The process of developing MOBMAS consisted of three sequential research activities. The first activity *identified and validated a list of methodological requirements for an Agent Oriented Software Engineering methodology* as mentioned above. The second research activity *developed MOBMAS* by specifying a development process, a set of techniques and a set of model definitions for supporting the identified methodological requirements. The final research activity *evaluated and refined MOBMAS* by collecting expert reviews on the methodology, using the methodology on an application and conducting a feature analysis of the methodology.

ACKNOWLEDGEMENTS

First and foremost, I wish to express my deepest gratitude to Prof. Graham Low, my supervisor of this PhD dissertation, for his valuable guidance and dedication to every stage of my research. It can be said that apart from myself, he is the one who has read my dissertation the most often. His meticulous comments on every page of my writing, and his enthusiastic attention to every step of my research, have resulted in significant corrections and improvements to my work. His devotion is sincerely appreciated.

I also wish to express special thanks to Prof. Mary-Anne Williams for her wholehearted support at various important stages of my research. I am thankful especially for her valuable help in finding the topic for my research, adverstising for my survey, reviewing MOBMAS and offering important suggestions for improvement.

I am very much indebted to Dr. Ghassan Beydoun for his generous devotion of time and effort to review my dissertation. His insightful advices have helped to notably improve the dissertation's coherence and completeness. I also sincerely thank Prof. Brian Henderson-Sellers and Dr. Cesar Gonzalez-Perez for their enthusiastic and dedicated involvement in the evaluation of MOBMAS. Their constructive criticisms were truly valuable to the methodology. It is with much gratitude that I thank Mr. (soon to be Dr.) Vincent Pang for the countless times when he offered his generous support and assistance to me during my research, including his help with the binding of this thesis.

My appreciation extends to the Faculty of Economics and Commerce at The University of New South Wales, who awarded me the "Faculty Postgraduate Research Scholarship". Without its financial support, this research would not have been possible.

Last but not least, I am forever thankful to my mother Tuyet Anh Thi Duong, my father Nam Duc Tran and my sister Dr. Giao Quynh Tran for their never-ending and immeasurable love, understanding and support. Above all, I am truly grateful to God for His amazing grace and everything that He has blessed me, including the blessing of the most wonderful mother and father that I have. This dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACL	Agent Communication Language
AOSE	Agent-Oriented Software Engineering
BDI	Belief-Desire-Intention agent architecture
BDIM	Methodology for BDI agents
HLIM	Methodology with High-Level and Intermediate levels
MAS	Multi-Agent System
MEI	Methodology for Enterprise Integration
OCL	Object Constraint Language
OO	Object Oriented
P2P	Peer to peer
PC	Program Committee
SDLC	System Development Lifecycle
UML	Unified Modelling Language
UCM	Use Case Map

CHAPTER 1
INTRODUCTION

> "*There is still much work to do and a long way to go before agent-oriented software engineering can evolve into its maturity*."
>
> (Fan 2000, p45)

1.1. INTRODUCTION

This chapter firstly provides some brief background on the Agent paradigm and Ontology, thereby revealing the motivations for an ontology-based Agent-Oriented Software Engineering (AOSE) methodology for Multi-Agent Systems (MAS) development (Section 1.2). Section 1.3 then specifies the objective of this PhD research, followed by Section 1.4 which highlights the significance of the research. The research's design is summarised in Section 1.5, while the dissertation's outline is presented in Section 1.6.

1.2. BACKGROUND AND MOTIVATIONS

Agent technology has become one of the most active and promising areas of research and development activity in computing in recent years (Wooldridge and Ciancarini 2000; Mountzia 1996). Agents are highly autonomous, situated, interactive software entities that have been hailed as "the next significant breakthrough in software development" (Sargent 1992, p28), "the new revolution in software" (Guilfoyle and Warner 1994, p1) and "the backbones for the next generation of mainstream software systems" (Fan 2000, p45). Originating from artificial intelligence, agent technology has progressively drawn on a diversity of computing areas, including software engineering, distributed computing, networking, mobile computing, collaborative computing, security and robotics (Sundsted 1998; Honavar 1999).

The greatest potential of agent technology is revealed through **MASs** (Wooldridge 1997; Huhns and Singh 1998; Zambonelli 2000). MASs are computational systems in

which two or more agents are interacting or working together to achieve a set of goals (Lesser 1996). 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 (Huhns and Singh 1998; Nwana and Wooldridge 1996). The powerfulness of MASs can be particularly realised in the engineering of open systems, distributed systems, heterogeneous systems, dynamic and adaptive systems.

It is widely accepted that appropriate **AOSE methodologies**, guiding developers, are required for agent technology to become a widespread commercial success (Flores-Mendez 1999; Jennings and Wooldridge 1995; Sycara 1998b; Zambonelli 2000). While for small development projects it may be acceptable to apply informal software engineering principles to the development of MASs, the absence of specialised AOSE methodologies for MAS construction will generally result in cumbersome, error prone, and hence expensive, 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). Indeed, a number of methodologies have been proposed to support the analysis and design of MASs. Nevertheless, an evaluation of prominent methodologies revealed that most are lacking in one or more of the following areas of MAS development: agent internal design (i.e. the design of agent mental constructs such as beliefs, goals, plans and actions), agent interaction design, and MAS organisation modelling (i.e. the design of acquaintances and authority relationships amongst agents/agents' roles). This research also conducted a survey of AOSE experts and practitioners, and a feature analysis of the existing AOSE methodologies, which together confirmed that no individual methodology offers support for developing all of the requirements of an MAS system.

In addition to the absence of a comprehensive methodology which addresses common concerns for any given system, it was noted that two concerns are largely ignored by all existing methodologies. These are: extending the functionality and lifetime of a system, through *interoperability* with other systems in heterogeneous environments and *reuse* of system design as requirements change. These are critical long-term concerns for any system, which will ultimately affect the take-up of the agent technology by the industry. In this thesis, a methodology which addresses those two concerns and combines all key

concerns of AOSE practitioners is synthesized. The methodology is called: ***M*ethodology for *O*ntology-*B*ased *MAS*s (MOBMAS).** The current research is driven by both the growing interest in agent technology and MASs, and the increasing recognition of **ontologies** in the computing community as a cornerstone towards interoperability and software reuse (Malucelli and Oliveira 2004; Uschold and Gruninger 1996; Richards 2000; Shave 1997).

In recent years, ontologies have been employed in many computing areas, including knowledge engineering, knowledge management, natural language processing, information retrieval and integration, and database design and integration (Gamper et al. 1999; Guarino 1998; Fensel 2001). In the realm of MAS, ontologies have been acknowledged for being beneficial to various MAS development activities, particularly system analysis and agent knowledge modelling (Uschold and Gruninger 1996; Falasconi et al. 1996; Weiss 1999; Shave 1997). 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 or between agents and non-agent software components) and the reasoning of agents. Reusability of system design through ontology has been recognised in single agent knowledge-based systems (Uschold and Gruninger 1996; Chandrasekaran et al. 1999; Mukherjee et al. 2000; Falasconi et al. 1996). Notwithstanding the benefits of ontology to MASs, most of the existing AOSE methodologies do *not* provide support for ontology-based MAS development. Specifically, they neither support the use of ontologies in the MAS development *process*, nor the inclusion of ontologies in the MAS development *model definitions*. Even though a few existing methodologies show some consideration for ontology, they do not comprehensively investigate the diverse ways in which ontology can be integrated into the MAS development process and MAS model definitions as MOBMAS endeavours. As a result, the development *processes* and *products* of the existing AOSE methodologies either do not provide, or provide to a lesser extent, the various important capabilities that an ontology-based development process and product can naturally provide, for example, support for interoperability and reusability.

1.3. RESEARCH OBJECTIVE

This research was conducted to

"*Contribute to the field of AOSE by proposing a **comprehensive ontology-based AOSE methodology** for the analysis and design of MASs. This methodology aims to provide support for ontology-based MAS development and various other AOSE methodological requirements which are important to an AOSE methodology but which may not be well-supported by the existing methodologies. The proposed AOSE methodology is named "**MOBMAS**", which stands for "**M**ethodology for **O**ntology-**B**ased **M**ulti-**A**gent **S**ystems".*

A MAS system is ontology-based when its design specification explicitly includes ontologies, and ontologies are used by agents at run-time to facilitate the operation of MAS (Yuan 1999; Guarino 1998).

The scope of MOBMAS does **not** include support for the actual process of developing ontologies. The methodology assumes that ontologies used by MAS and integrated in MAS model definitions are developed by a separate ontology engineering effort, which is conducted by domain experts, ontology engineers or the MAS developer himself. Numerous methodologies are currently available for this purpose, e.g. IDEF5 (Knowledge Based Systems Inc 1994), METHONTOLOGY (Fernandez et al. 1997) and Grüninger and Fox' methodology (1995). MOBMAS focuses instead on:

- the use of ontologies in the MAS analysis and design process; and
- the inclusion of ontologies in MAS model definitions.

The scope of MOBMAS is also limited to the **Analysis** and **Design** phases of the system development lifecycle (SDLC), which traditionally contains four phases, Requirements Engineering, Analysis, Design and Implementation (Eliason 1990; Dennis and Wixom 2003). MOBMAS process starts from a set of system functionality (which is identified by a separate Requirements Engineering effort) and ends with a design of a MAS system. Even though the Implementation phase is not covered, MOBMAS addresses various important implementation-related issues such as deployment configuration and selection of agent architectures.

1.4. SIGNIFICANCE OF THE RESEARCH

The research effort of this thesis, embodied in MOBMAS, contributes to state of the art of AOSE in three essential ways. Firstly, it provides developers with a framework to handle interoperability issues in a heterogeneous environment at design time. Secondly, it explicitly integrates the use of ontology for knowledge representation with its actual design and development, giving developers a solid framework for promoting reuse of software design. Thirdly, it combines all key concerns of AOSE practitioners into one methodological framework.

The first two contributions are inter-related. It is by the explicit and extensive support for *ontology-based* MAS development that MOBMAS accommodates interoperability concerns in heterogeneous environments. Systems designed with MOBMAS can be formed from loosely coupled components connected through ontological mappings. They are inherently flexible and their actual design and architecture are reusable across different areas of applications and in different settings. The explicit support of MOBMAS for ontology-based MAS development is as follows:

- *In the MAS development process*, just as ontology analysis has been employed to facilitate the process of constructing and validating knowledge-based systems (Chandrasekaran et al. 1999; Uschold and Gruninger 1996), MOBMAS makes use of ontology to facilitate the process of constructing and validating its MAS analysis and design models. Specifically, ontologies are used to help identify and validate the functional requirements of the target MAS, actions of agent classes and exchanged messages between agents. MOBMAS also shows how the MAS development process can, in return, assist in the development of ontologies. Specifically, the investigation of a system's functional requirements, agent goals, plans, reflexive rules, actions and exchanged messages helps to identify and validate the concepts to be included in ontologies; and

- *In MAS development model definitions*, MOBMAS dedicates one of its "model kinds"[1] for the representation of ontologies, namely "Ontology Model Kind". This model kind captures all of the ontologies that are necessary for agents in the target

[1] The term "model kind" is used to refer to a specific class of models (Standards Australia 2004). The models themselves will be built by the developer during the development process.

MAS to operate. Agents' knowledge is then modelled in terms of these ontologies. The modelling of agent behaviour and agent interactions is also based upon ontologies: concepts in the ontologies are used to formulate agents' goals, plans, reflexive rules, actions and content of communication messages. MOBMAS also models the conceptualisation of non-agent resources and the mappings between these conceptualisations and the domain ontologies shared amongst agents.

By using ontology in the MAS development process and including ontology in the MAS model definitions as described above, MOBMAS is able to enhance its *MAS development process* and *MAS design product* with many important ontology-related strengths. These strengths include those that have been widely acknowledged in the ontology literature (e.g. efficient system analysis, structured and reusable agent knowledge modelling, semantically-consistent agent communication and facilitated agent reasoning), and those that are newly uncovered by MOBMAS (e.g. support for verification and validation, maintainability, extendibility and reliability). These ontology-related strengths are either not provided, or provided to a lesser extent, by the existing AOSE methodologies due to their lack of support for ontology.

With respect to the second contribution of this thesis, MOBMAS offers support for many important methodological requirements of AOSE, which are suggested by practitioners and researchers in the field and the existing MAS development methodologies (e.g. support for agent internal design steps, agent interaction design steps, MAS organisation modelling steps, diverse agent-related properties and modelling concepts). The support provided by MOBMAS was based upon the reuse, enhancement and unification of the existing AOSE methodologies' strengths, as well as the proposal of new techniques and model definitions where the existing support is weak.

Ultimately, the proposal of a comprehensive, unified, ontology-based AOSE methodology for the analysis and design of MASs helps to foster the widespread deployment of agent-based systems by industry, hence contributing to the commercial success of agent technology.

1.4.1. Application Domains of MOBMAS

With its explicit and extensive support for ontology throughout the MAS analysis and design processes, MOBMAS is particularly suitable to the development of the following types of agent systems.

- *Heterogeneous systems*: These are systems that contain heterogeneous agents (in term of their internal knowledge structures) and/or heterogeneous non-agent resources that are wrapped around by the agents. An example of this type of application is an information gathering system, where each "Searcher" agent pertains to a different structure of beliefs; for instance, one "Searcher" agent may possess beliefs on medicine, while another on travel. An information gathering system normally encompasses heterogeneous knowledge sources such as relational databases, search engines and/or web pages, each of which has a different internal information structure. MOBMAS facilitates the design and run-time operation of these heterogeneous systems by explicitly conceptualising the knowledge of each system component (either agents and/or non-agent resources) by ontologies, thereafter enabling the interoperability of these components via the explicit specification of ontological mappings.
- *Systems that involve legacy components*: Legacy systems exist quite commonly in manufacturing and process control applications, where functionally-essential software components are technologically obsolete, but cannot readily be replaced or modified due to the costs and/or the time required (Wooldridge and Jennings, 1998). In an agent system, these legacy components can be used by being wrapped with an agent layer that enables them to interoperate with other components via a uniform communication interface (Jennings and Wooldridge, 1995). Accordingly, an agent system containing legacy systems is basically a heterogeneous MAS formed from loosely-coupled heterogeneous components. MOBMAS is thus particularly suitable to its development due to the reasons listed in the previous paragraph.
- *Open systems*: An open MAS is one which allows for dynamic addition and/or removal of system components at run-time (Sycara 1998b). Common applications where MASs need to reside in an open environment are information gathering applications (as "Searcher" agents can be frequently added or removed) and e-commerce applications, such as those mimicking a market place (as "Seller" and "Buyer" agents, for instance, can frequently enter or leave the system). MOBMAS

facilitates the design and run-time operation of these open systems in various ways. Firstly, by supporting heterogeneity via ontological mappings, MOBMAS removes the interoperability concerns that arise when adding new heterogeneous agents into an existing MAS. The methodology also offers an option to conceptualize the agent interaction protocols during the design time. This explicit conceptualization of the interaction protocols will allow any new agents to join the pre-existing conversations at run-time, and allow the interaction protocols to change over time during run-time.

While being particularly advantageous to the above types of applications, MOBMAS is also suitable to the development of any typical agent systems. In comparison with the existing popular AOSE methodologies, MOBMAS is capable of reducing more development costs for the analysis and design of MASs. This is because:

- MOBMAS makes it easy to reuse MAS design components. The core design models of MOBMAS are composed in terms of ontologies, for example, agent internal knowledge model, agent behaviour model and agent interaction model. As such, the developer can adapt the past MAS design models to a new application by simply changing the ontologies involved. In addition, MOBMAS implements the idea of using ontologies to decouple the modelling of agent's domain knowledge from agent's behavioural/problem-solving knowledge, thereby supporting the reuse of these two knowledge components across agents.
- MOBMAS provides extensive support for verification and validation during the MAS development processes, thus increasing the likelihood of a correct system. In particular, MOBMAS recommends the developer to exploit application ontologies to verify and validate the completeness and correctness of various core MAS analysis and design models. Since ontologies are often constructed by a separate development team (e.g. domain experts or knowledge engineers), they can serve as a reliable tool for verification and validation.
- MOBMAS facilitates the maintenance of a MAS system design. This is because the specification of the MAS' application domains, tasks and resources are formally documented in ontologies, and the core MAS design models such as agent internal knowledge model, agent behaviour model and agent interaction model are consistently defined in term of these ontologies.

- MOBMAS makes it easy to extend an existing MAS design. When the MAS needs to cover new domains, tasks or resources, the agents can easily extend their knowledge by adding new ontologies into their knowledge models.

1.5. RESEARCH DESIGN

To achieve the research objective, three core research activities were performed.

1. Research Activity 1 – Identify the methodological requirements of MOBMAS

This activity aimed to identify and validate the methodological requirements of MOBMAS in terms of the ***features*** that MOBMAS should support, ***steps*** that MOBMAS development process should include, and ***modelling concepts*** that MOBMAS model kinds should represent. Note that the desirable *steps* identified by this activity are *not* meant to be the "exact" steps that MOBMAS must specify. MOBMAS can define its steps differently from these desirable steps. However, the actual steps of MOBMAS must correspond to, or cover, these desirable steps.

Research Activity 1 was carried out in four research steps.

- Step 1 – Identify the "potential" methodological requirements of MOBMAS:
  The potential *features* were identified by investigating a number of evaluation frameworks for AOSE methodologies and conventional system development methodologies (including object-oriented (OO) methodologies). The potential *steps* and *modelling concepts* were discovered by examining the existing AOSE methodologies.
- Step 2 – Conduct a survey on practitioners and researchers in the field of AOSE to validate the identified potential features, steps and modelling concepts.
- Step 3 – Perform a detailed feature analysis on the existing AOSE methodologies to further validate the identified features, steps and modelling concepts, and arrive at the "actual" methodological requirements for MOBMAS.
- Step 4 – Identify "ontology-related steps" from amongst the required AOSE steps of MOBMAS, so as to enable MOBMAS to offer all of the widely-recognised benefits of ontology to MAS development and MAS operation as found in the literature review.

2. Research Activity 2 – Develop MOBMAS

This research activity specified the ***development process, techniques*** and ***model kinds*** for MOBMAS so as to support the required features, steps and modelling concepts identified in Research Activity 1. MOBMAS process, techniques and model kinds were developed by *reusing* and *enhancing* the techniques and model definitions offered by the existing AOSE methodologies where appropriate, and *developing new* techniques and model definitions where necessary.

3. Research Activity 3 – Evaluate and refine MOBMAS

MOBMAS was evaluated and progressively refined through three sequential research steps.

- Step 1 – Collect expert reviews on the preliminary version of MOBMAS.
- Step 2 – Use the refined methodology on a test application.
- Step 3 – Perform a feature analysis on the final version of MOBMAS.

The aim of expert reviews was to gather experts' evaluation of MOBMAS based on the experts' non-empirical investigation of the methodology. The use of MOBMAS on a test application then gathered external developers' evaluation of MOBMAS based on their empirical usage of the methodology. Lastly, the feature analysis was conducted to verify MOBMAS' ability to achieve its objective (which is, to provide support for ontology-based MAS development and the other important AOSE methodological requirements[2]; cf. Section 1.3), to compare MOBMAS with the existing AOSE methodologies, and to clarify MOBMAS' ontology-related capabilities.

1.6. ORGANISATION OF THE DISSERTATION

This dissertation is presented in eight chapters.

- **Chapter 1 –** "**Introduction**": provides an overview of the research's motivations, objective, significance and design.
- **Chapter 2 –** "**Background of Agents and Ontology**": presents background information on the two realms underlying the research, Agent Technology and Ontology. Definitions of concepts "Agent", "Multi-Agent System" and "Ontology"

[2] Through the justification of MOBMAS' support for its methodological requirements, this research was able to justify that MOBMAS' actual *steps* and *modelling concepts* in fact correspond to, or cover, the desirable steps and modelling concepts which were specified as part of the methodological requirements.

are provided, together with discussion on the potentials of the agent technology and MAS, as well as the benefits of ontology to MAS development and MAS operation.

- **Chapter 3 –** "**Review of Existing MAS Development Methodologies**": provides an account of the existing AOSE methodologies for MAS analysis and design, and discusses their general limitations.
- **Chapter 4 –** "**Research Design**": reiterates the research objective (from Section 1.3) and describes the details of the three research activities performed to achieve it.
- **Chapter 5 –** "**Methodological Requirements of MOBMAS**": documents the identification of MOBMAS' required features, steps and modelling concepts (i.e. Research Activity 1; cf. Section 1.5). The chapter also presents suggestions on how and where MOBMAS may obtain techniques and model definitions to support each of its methodological requirements.
- **Chapter 6 –** "**Documentation of MOBMAS**": presents the full documentation of MOBMAS. The chapter consists of seven sections.
  - Section 6.1 – "*Overview of MOBMAS*": presents an overall description of MOBMAS conceptual framework, development process and model kinds.
  - Sections 6.2 to 6.6: each documents each of the five activities of MOBMAS: "*Analysis*", "*MAS Organisation Design*", "*Agent Internal Design*", "*Agent Interaction Design*" and "*Architecture Design*".
  - Section 6.7: presents a summary of the chapter.
- **Chapter 7 –** "**Evaluation and Refinement of MOBMAS**": documents the refinement and evaluation of MOBMAS as a result of the expert reviews on MOBMAS, the use of MOBMAS on an application, and a feature analysis of MOBMAS (i.e. Research Activity 3; cf. Section 1.5).
- **Chapter 8 –** "**Conclusions**": concludes the dissertation with discussion of the research's contributions, limitations and suggestions for future research.

1.7. SUMMARY

This chapter has presented an overview of the research. It highlights the research's objective, motivations, significance and design. These issues will be elaborated further in Chapter 4. In the subsequent chapter, Chapter 2, background information about the Agent paradigm and Ontology is presented.

CHAPTER 2

BACKGROUND OF AGENTS AND ONTOLOGY

2.1. INTRODUCTION

With the research focus being "ontology-based MASs", this research spans two major realms: *Agent Technology* (particularly MAS) and *Ontology*. This chapter provides background information on each realm. Section 2.2 firstly defines "Agent" and "MAS", highlights the motivations for agents and MASs, and points out the limitations of the Agent paradigm. Section 2.3 subsequently defines "Ontology", discusses the benefits of ontology to MAS development and MAS operation, and provides an overview of the ontology's typology and representation languages.

2.2. AGENT TECHNOLOGY AND MAS

2.2.1. Definition of Agent

Generally defined, a "software agent" is an entity or a piece of software that acts on behalf of its user to accomplish a task (Mountzia 1996). Nevertheless, the exact nature of agency has attracted much discussion and controversy (Mountzia 1996; Wooldridge 1999; Eurescom 2001a). A variety of definitions have been proposed, each offering a varied opinion as to what constitutes an agent (Franklin and Graesser 1996; Wooldridge and Jennings 1998; Eurescom 2001a). As noted by Wooldridge (1999), a universal definition of "software agent" may be impossible, since attributes characterizing agency may vary across domains. Above all, such a prescriptive universal definition is not really important, because "the notion of an agent is meant to be a tool for analysing systems, not an absolute characterisation that divides the world into agents and non-agents" (Russell and Peter 1995, p33).

This research adopts the definition proposed by Wooldridge (1999, p29). This definition has received much recognition from researchers in the field.

> "*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives*."

The definition emphasizes two major attributes of agency: *interaction with the environment* and *autonomy*. Interaction with the environment refers to the ability to perceive the environment and act upon it, while autonomy can be understood as the ability to have complete control over one's state and behaviour. Being autonomous, an agent is capable of decoupling the process of receiving a request message from another agent from the process of executing actions upon receiving the message (Fisher et al. 1997).

The above definition of agent covers a wide spectrum of computational entities, from Microsoft Tip Wizards, software daemons and simple control systems (such as thermostats) to very large expert systems (Jennings and Wooldridge 1995; Wooldridge 1999). This research, however, is interested particularly in "*intelligent agents*", which are, as defined by Wooldridge (1999, p32)

> "... agents that are capable of flexible autonomous action, where flexibility means three things:
>
> - *reactivity*: intelligent agents are able to perceive their environment and respond in a timely fashion to changes that occur in it;
> - *proactiveness*: intelligent agents are able to exhibit goal-directed behaviour by taking the initiative; and
> - *social ability*: intelligent agents are capable of interacting with other agents (and possibly humans)."

Even though intelligent agents may assume other attributes such as mobility, adaptability and personality, the above attributes sensibly characterise 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 (Lesser 1996). Each agent is considered as a locus of problem-solving activity which operates asynchronously with respect to other agents (Lesser 1996).

A MAS typically exhibits the following major characteristics (Sycara 1998b).

- Each agent has incomplete information or capabilities or resources for achieving the global goal and thus has a limited viewpoint.
- There is no global control over the whole system.
- Data is decentralised.
- Computation is asynchronous.

2.2.3. Motivations for Agents and MASs

Agents are believed to represent the next advance in software engineering. They offer a notably more powerful and natural abstraction for modelling and developing systems than conventional abstractions such as procedural abstraction, abstract data types and objects (Wooldridge et al. 1999). The concept of agents as autonomous software components, capable of flexibly interacting with each other to satisfy their objectives, is very natural to software engineers (Wooldridge and Ciancarini 2000). For example, in an electronic commerce application, it is natural to model participants in a trade transaction as agents which buy and sell goods on behalf of human users (Wooldridge and Ciancarini 2000).

The powerfulness of agents and MASs is particularly realised in the engineering of *open systems* (Jennings and Wooldridge 1995; Sycara 1998b; Jennings et al. 1998). These systems are often dynamic in structure. Their system components are usually not known in advance, highly heterogeneous and capable of changing over time. Thus, the ability to engage in flexible and robust interactions among the system components is crucial. Agents exhibit this ability through negotiation and coordination capabilities. These capabilities are facilitated by the use of "agent communication languages" (ACL) such as KQML (UMBC Lab for Advanced Information Technology n.d.a) and FIPA-ACL (FIPA n.d.a). In addition, the core properties of agents – namely, autonomy,

proactiveness and reactivity – allow them to deal with dynamic and unpredictable environments. Agents can continually monitor their environment, revise their goals and proactively adopt new goals when opportunities arise (Jennings and Wooldridge 1995; Omicini 2000).

Another important contribution of agents and MASs is in the *engineering of distributed systems* (Jennings and Wooldridge 1995; Jennings et al. 1998; Eurescom 2001a; Zhou et al. 2000; Huhns and Stephens 1999; Wood and DeLoach 2000a). In such systems, it is difficult to specify a single locus of control because the systems are built out of distributed components, each of which possibly attempts to achieve conflicting individual goals (Wood and DeLoach 2000a; Eurescom 2000a). It is therefore natural to map the distributed entities onto autonomous problem-solving agents, which negotiate and coordinate autonomously and flexibly to resolve conflicts and achieve the global goals (Jennings and Wooldridge 1995; Wooldridge and Ciancarini 2000). In addition, the proactiveness of agents makes it possible to abstract away from the control issue, thereby dealing with the decentralisation of control (Omicini 2000). 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, hence limiting the need for communication across the network and reducing network traffic (Horlait 2003; Huhns and Stephens 1999).

In addition, agents offer a natural way to incorporate legacy systems into modern MASs, hence supporting *heterogeneity* and *interoperability* (Jennings and Wooldridge 1995; Jennings et al. 1998; Eurescom 2000a). Legacy systems exist quite commonly in manufacturing and process control applications. They are functionally essential software components that are technologically obsolete but cannot readily be replaced or modified due to cost or time (Wooldridge and Jennings 1998; Jennings and Wooldridge 1995). The agent paradigm solves this problem by "agentifying" the legacy components, wrapping these components with an agent layer that enables them to interconnect and interoperate with other system components via a uniform communication interface (Jennings and Wooldridge 1995; Jennings et al. 1998; Eurescom 2000a).

Agents also provide the benefits of the conventional OO paradigm, namely modularity, concurrent execution, reliability and reusability. When a problem is complex or

unpredictable, the most effective way to address it is to develop a number of modular agents, each of which specializes at solving a particular aspect of the problem (Jennings and Wooldridge 1995; Sycara 1998b). A MAS, however, represents more than a modular object-based system. As earlier discussed, agents can interact and coordinate in an autonomous, flexible and context-dependent manner so as to ensure that the tasks are properly managed (Jennings and Wooldridge 1995; Sycara 1998b). Concurrent execution is inherently provided by a MAS, since each agent is assumed to have at least one thread of control (Wooldridge 1999). Reliability is also encouraged, as agents can cooperate and dynamically take up the responsibilities of other agents that fail (Sycara 1998b). Finally, reusability is supported by reusing the design or coding of a similar agent in a past MAS development experience (Mountzia 1996).

Nowadays, with the availability of numerous agent architectures, agent-oriented programming languages and agent/MAS implementation platforms, the adoption of agent technology in the commercial environment has been greatly facilitated. Regarding *agent architectures*, a well-known architectural model is the Belief-Desire-Intention (BDI) architecture proposed by Rao and Georgeff (1991; 1995). A BDI agent is composed of three data structures: beliefs (i.e. the agent's knowledge of the world), desires (i.e. the agent's goals, objectives or allocated tasks) and intentions (i.e. the desires that the agent is committed to achieving at a certain point in time). This agent architecture has been adopted by many agent implementation platforms such as PRS (Myers 1997), JACK (Agent Oriented Software 2004) and dMARS (d'Inverno et al 1997), and many agent-oriented methodologies such as PROMETHEUS (Winikoff and Padgham 2004), Kinny and Georgeff's methodology (1996) and TROPOS (Castro et al. 2001). Regarding *agent-oriented programming languages*, various languages have been developed, including Agents Kernal Language (Franzén et al. 1992), Telescript (General Magic Inc. 1995), Agent Tcl (Gray, 1995), Obliq (Cardelli 1994) and Java. Regarding *agent implementation platforms*, a large number of platforms are currently available, for example, JACK (Agent Oriented Software 2004), JADE (Telecom Italia Lab 2004), AgentBuilder (Acronymics Inc. 2004), MADKIT (MADKIT 2002), ZEUS (British Telecommunications 2002) and Voyager (Glass 1998).

2.2.4. Limitations of Agents and MASs

Although the agent paradigm offers many exciting opportunities, it should not be oversold. For many applications, the added sophistication of agents is not needed (Wooldridge and Jennings 1998; Eurescom 2001a). 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.

Classes of problems for which intelligent agents and MASs are appropriate typically involve 1) distributed control, 2) complex communications, 3) autonomous behaviour, 4) high flexibility and adaptiveness, 5) interoperability, and 6) concurrent achievement of multiple, possibly conflicting, goals (Eurescom 2001a). A MAS solution may not be suitable to domains in which global constraints have to be maintained, deadlocks or livelocks must be avoided, globally optimal decisions have to be made, or the risk is too high to give agents absolute trust and delegation (Jennings and Wooldridge 1998).

2.3. ONTOLOGY

2.3.1. Definition of Ontology

Ontology is a very old concept that has generally been confined to the philosophical sphere in the past, since the time of Aristotle (Fensel 2001). 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 et al. 1999; Guarino 1998; Fensel 2001).

In the philosophical sense, "ontology" is defined as a systematic account of being or existence, from the Greek "ontos" (i.e. being) (Khan 2000; Gruber 1993a). It refers to a study of things that exist and attempts to answer the question of "what is being" (Chandrasekaran et al. 1999; Guarino and Giaretta 1995).

In the context of computing, ontology is confined to the specification of worldview with respect to a domain of interest (Yuan 1999). A prominent definition of ontology is given by Fensel (2001, p11): *"An ontology is a formal, explicit specification of a shared*

conceptualisation". "*Conceptualisation*" refers to an abstract model of some phenomenon in the world. It defines the relevant *concepts* or *entities* that exist in the universe of discourse and the *relations* that hold among them (Gruber 1993a). For example, the conceptualisation of a pile of blocks is (Genesereth and Nilsson 1987)

({a,b,c,d,e}, {on, above, clear, table})

where {a,b,c,d,e} is the universe of discourse (consisting of 5 blocks); and

{on, above, clear, table} is a set of relevant relations among these blocks.

Although not explicitly stated, this definition relies on the *intentional notion* of "conceptualisation" rather than extensional notion. The intentional notion means that the conceptualisation only defines the *meta-information* for describing the semantics of concepts and relations. It does not reflect particular states of affair as the extensional conceptualisation does. For instance, in the above example of block conceptualisation, the meaning of relation "on" (which specifies whether a block is on top of another block) should remain the same even if the blocks are arranged differently (i.e. when the state of affair changes) (Guariano and Giaretta 1995). As a result, it can be said that ontology only provides the *vocabulary* with which to represent the body of knowledge. The knowledge itself does not constitute ontology, but is a collection of factual situations represented using the vocabulary provided by ontology (van Heijst et al. 1997; Chandrasekaran et al. 1999).

The "*shared*" characteristic of an ontology implies that ontology should capture consensual knowledge, i.e. it is not restricted to some individual but accepted by a group. "*Explicit*" means that ontology should be explicitly defined. In the context of MAS, this means that ontologies used by agents need to be explicitly stated and not remain implicit within the agent codes (O'Brien and Nicol 1998). Finally, "*formal*" refers to the fact that an ontology should be machine-readable. Different degrees of formality are possible. Ontologies like WordNet provide a thesaurus for natural language terms explained in natural language. On the other end of the spectrum is CYC which provides formal axioms for knowledge (Fensel 2001).

2.3.2. Motivations for Ontologies in MAS

The literature is currently rich with discussion of ontologies' importance (Uschold and Gruninger 1996), such as in the areas of knowledge engineering (Shave 1997),

information retrieval (Ding 2001) and database design (Sugumaran and Storey 2001). This research focuses on the importance of ontologies in the context of MAS. Within this context, ontologies have been widely recognised for their significant benefits to ***interoperability, reusability, MAS development activities*** and ***MAS operation*** (Falasconi et al. 1996; Malucelli and Oliveira 2004; Yuan 1999; Knoblock 1994). These benefits are actually inter-related with each other, as will be mentioned throughout the discussion.

2.3.2.1. Benefits of ontologies to interoperability

Interoperability refers to the ability of heterogeneous components to interact and work with each other to achieve shared or individual goals (Finkelstein 1998). Interoperability involves not only communication between the heterogeneous components (c.f. Section 2.3.2.4), but also the ability of these components to use the exchanged information[3] (IEEE 1990). In MAS, interoperability issues may arise between heterogeneous *agents* or between heterogeneous *non-agent resources*[4] (such as knowledge sources and legacy application systems). Two prominent interoperability issues are (Wache et al. 2001; Sheth and Larson 1990; Tout 2001):

- *Semantic heterogeneity issue*: occurring when the knowledge base of each agent, or the information/application 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. concept "Employee" in one agent/resource means anyone currently on payroll but in another agent/resource means anyone currently receiving benefits, thus including retirees). Another example of semantic heterogeneity is the scaling conflict, where the same concept refers to the different scales or references of measurement (e.g. concept "Price" may be measured in dollar in one agent/resource but in euro in another); and
- *Structural heterogeneity issue*: occurring when the knowledge base of each agent, or the information/application of each resource, uses a different conceptual schema to represent its data. For example, concept "Customer-Name" is represented as an object in one agent/resource but as an attribute in another.

---

[3] Note that communication only results in the exchange of information between components.
[4] From here on, the term "resource" is used to mean non-agent software components that are incorporated into a MAS to provide agents with information and/or services (e.g. databases, web servers and legacy processing systems).

Both of these heterogeneity issues can be addressed by the use of ontologies (Malucelli and Oliveira 2004; Tout 2001; Shave 1997). Specifically, when the knowledge bases of heterogeneous agents and the information/applications of heterogeneous resources are explicitly conceptualised by ontologies, the structural and semantic interoperability between these agents and resources can be achieved by mapping between these ontologies. Such mechanism is known as "*ontological mapping*", i.e. specifying the semantic correspondences between the concepts of one ontology with those of another (Madhavan 2002). Some example semantic correspondences are "equivalent", "subsumes" and "intersects" (Parent and Spaccapietra 1998).

There are two major ways to map between ontologies: either to map the ontologies against each other (Figure 2.1a), or to map them against a common ontology (Figure 2.1b). The second approach is more efficient than the first because (Wache et al. 2001; DiLeo et al. 2002; Uschold and Gruninger 1996):

- it minimises the number of mappings between the ontologies. If there are *n* ontologies, the direct-mapping approach will require *(n-1)!* pair-wise mappings, while the use of a common ontology as an inter-lingua will result in only *n* mapping linkages[5];
- it minimises the maintenance required when an agent or resource changes its conceptualisation. With the direct-mapping approach, all ontological mappings between the changed ontology and all other ontologies need to be updated, while with the inter-lingua approach, only the mappings between the changed ontology and the common ontology need to be updated; and
- it facilitates the sharing of knowledge when each heterogeneous resource is wrapped by a different wrapper agent (Figure 2.2). In this case, the wrapper agents can easily share and interoperate their resources by, firstly, translating the resources' outputs from the resource-ontology's vocabulary into the common-ontology's vocabulary, thereafter communicating the outputs with each other using the common ontology's vocabulary.

[5] That is, pair-wise mapping linkages between the common ontology and each other ontology.

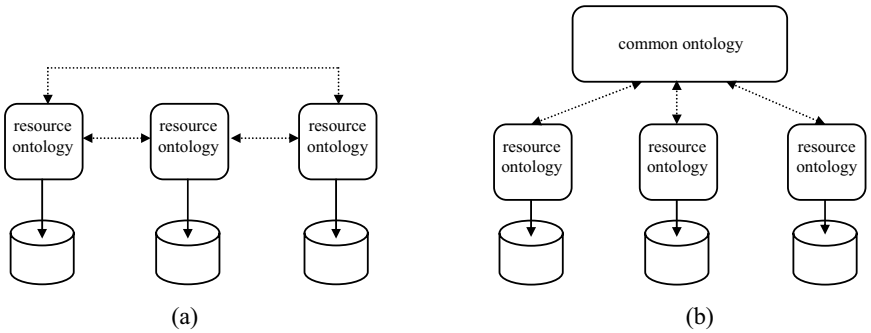


Figure 2.1 – Approaches for ontological mapping (Wache et al. 2001)

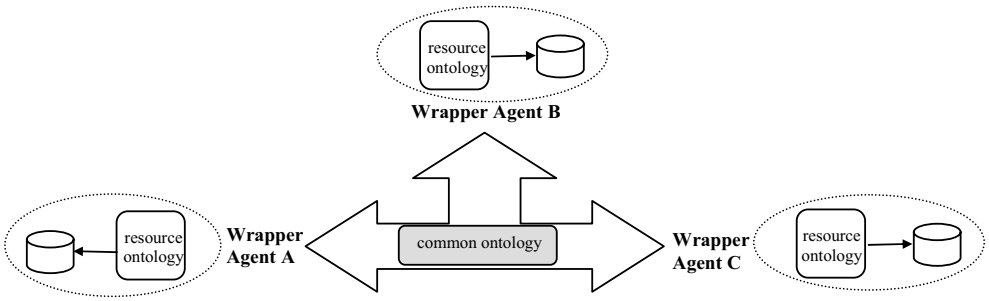


Figure 2.2 – Sharing of knowledge between wrapper agents

It should be noted that, by supporting interoperability between system components, ontologies are able to promote reusability (c.f. Section 2.3.2.2). In particular, legacy agents and/or resources can be reused and added to the current MAS without causing any interoperability problems with the existing agents and resources.

2.3.2.2. Benefits of ontologies to reusability

The capability of ontologies to enhance reuse has earlier been acknowledged and exploited by the Knowledge Engineering community in the development of knowledge-based systems (i.e. single-agent systems) (Gruber 1993b). An ontology was employed to capture domain knowledge of a system, while the system's problem solving knowledge, which specifies the domain-independent reasoning steps to solve the problem, was stored separately in a Problem Solving Method. Consequently, each knowledge-based system was designed as being composed of two components: a Problem-Solving Method and an ontology[6] (Benjamins 1995; Chandrasekaran et al. 1999; Fensel et al. 1997; Fensel 1997). This modularity in knowledge modelling, which was made possible by ontologies, enables the reuse of Problem Solving Methods across

[6] This ontology contains all the domain knowledge required by the Problem Solving Method.

different problem domains, and the reuse of domain knowledge across different problems (Uschold and Gruninger 1996; Mukherjee et al. 2000; Falasconi et al. 1996).

In the context of MAS development, the above ontology-based mechanism of reuse could still be applied, since each agent in a MAS is basically a knowledge-based system. As conjectured by this thesis, each agent can be modelled as being composed of two major knowledge components: the *behavioural knowledge* component, which captures the problem solving knowledge of an agent in the form of plans, reflexive rules and/or actions that guide the agent's behaviour in achieving its goals, and the *(local) domain knowledge*[7] component, which contains the ontologies defining the domain-related knowledge requirements of the agent's behaviour. Given this approach of agent knowledge modelling, an agent's behavioural/problem-solving knowledge can be reused across agents with similar behaviour/goals in different domains, and its domain-related knowledge can be reused across agents within the same domain area[8].

Another factor that enables ontologies to enhance reusability is its readability. Software reuse is typically promoted by the readability of software design and/or codes (Richards 2000). Ontologies enhance readability by offering a structured, explicit, human-readable mechanism for representing knowledge. They help the system developer to easily comprehend, inspect and reuse this knowledge for future applications.

In addition, when an existing MAS needs to be extended with heterogeneous add-in agents and/or resources, ontologies makes it easy for the current agents to interoperate with those newly added components (c.f. Section 2.3.2.1), thereby enabling the reuse of these components.

2.3.2.3. Benefits of ontologies to MAS development activities

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

---

[7] The term "local" is used to refer to the fact that the domain-related knowledge of each individual agent in a MAS is normally only a portion of the domain knowledge that MAS covers as a whole.
[8] In this case, the reused ontology may need to be adapted to fit the knowledge requirements of the individual behaviour of each agent.

- **System analysis** involves the formulation of the problem to be solved (e.g. elicitation of system goals) and/or the representation of the application's domain knowledge (e.g. Car domain, Education domain) (Girardi et al. 2004).
  - With regard to the *problem formulation*, 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. In fact, a weak ontological analysis often leads to an incomplete or inaccurate understanding of the application, thereby leading to an incoherent system (Shave 1997). This importance of ontologies has been realised and exploited by the Knowledge Engineering community in the engineering of knowledge-based systems (Shave 1997). The first step in developing an effective knowledge-based system has been recommended to be an effective ontological analysis (Chandrasekaran et al. 1999). Moreover, when the target application covers multiple domains, the mappings between domain ontologies will help the developer to grasp the associations amongst these domains. These associations are particularly important if the development project involves multiple developers working on different domains (Uschold and Gruninger 1996).
  - With regard to the *representation of the application's domain knowledge*, ontologies offer a structured, explicit, human-readable mechanism for representing domain knowledge. These characteristics promote the readability of an ontology, hence making it a *reuse*-enhancing representation mechanism, as previously mentioned in Section 2.3.2.2.

Given the above benefits of ontologies to system analysis, various methodological frameworks for developing MASs and knowledge-based systems have exploited ontologies to facilitate their problem-elicitation process (e.g. "GRAMO" – Girardi and de Faria 2004) and domain knowledge modelling (e.g. "GRAMO" – Girardi and de Faria 2004, and "CommonKADS" – Schreiber et al. 1994). In fact, a metamodel of MAS modelling concepts recently proposed by Beydoun et al. (2005) also advocates the use of ontologies to model application domain for a given MAS system.

- **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 application's domain knowledge, an ontology can be used as an effective representation mechanism for agents' local domain-related knowledge (which is typically a portion of the application's domain knowledge) (Mukherjee et al. 2000; Tamma and Bench-Capon 2001). Different (parts of) ontologies can be assigned to different agents to represent the agents' different views of the world (Tamma and Bench-Capon 2001; Falasconi et al. 1996). In addition, as previously discussed in Section 2.3.2.2, ontologies offer a mechanism for decoupling the modelling of agent domain-related knowledge from its problem-solving knowledge, hence promoting the *reuse* of agent knowledge modules. Various methodologies for developing single-agent knowledge-based systems have implemented this modelling mechanism, e.g. KAMET II (Cairo and Alvariz 2004) and CommonKADS (Schreiber et al. 1994). It should be noted that, since the local domain-related knowledge of each agent is extracted from the application's domain knowledge, the use of ontologies to represent the application's domain knowledge during *system analysis* would facilitate the use of ontologies to represent agents' local knowledge during agent knowledge modelling.

2.3.2.4. Benefits of ontologies to MAS operation

Ontologies are beneficial to two major aspects of MAS operation: communication and agent reasoning.

- **Communication** in a MAS may occur between agents, between agents and non-agent resources, and between agents and human users.
  - Regarding *inter-agent communication*, even though sharing a common ACL will allow agents to exchange messages (thanks to the common communication syntax), it does not ensure that the communicating agents will interpret the exchanged messages in a uniform and consistent manner, i.e. to share the same understanding of the semantics of the messages (Weiss 1999; Uschold and Gruninger 1996; Falasconi et al. 1996). Successful agent communication requires "*ontological commitment*" of the agents, i.e. an agreement between agents to share an ontology during communication (Gruber 1993a). 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 communicates with agent B using the following message (written in FIPA-ACL),

```
inform
:sender AgentA
:receiver AgentB
:language KIF
:ontology CarDomainOntology
:content (> (price car X) (price car Y))
```

then both agents need to commit to the Car Domain Ontology (stated in the field ":ontology") where concepts "price" and "car" 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 the ontology's role in agent communication at run-time and the modelling of agent knowledge at design-time (c.f. Section 2.3.2.3).

– Regarding *agent-resource communication*, non-agent resources are normally accessed by agents via "wrappers", i.e. specialised agents that provide interface to the resources (Jennings and Wooldridge 1995; FIPA 2001a). Client agents can relay ACL queries and commands to the wrapper agents, which in turn translate and invoke them onto the underlying resources (Figure 2.3).

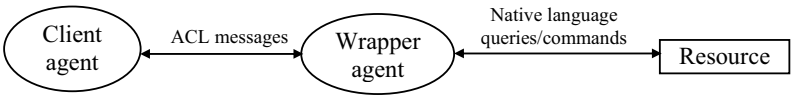


Figure 2.3 – Agent-resource communication

Ontologies can be used to conceptualise the resources' internal data and/or application, thereby allowing the wrapper agents to determine which vocabulary they should use to formulate input queries/commands to the resources and interpret outputs, without having to access the resource's internal structure (Gruber 1993a). For example, if the ontology of a Car Supplier database resource shows that a "Car" entity in the database has attributes "Car-Brand", "Price", "Transmission" and "Power-Steering", the wrapper agent can use these ontological concepts to compose queries to the database server, for instance,

Select * from **CarInfo** where **Car-Brand** = “Toyota”, **Price** < $50,000, **Transmission** = “auto”, **Power-Steering** = “yes”

– Regarding *human-agent communication*, ontologies can be used to facilitate the formulation of user queries and the representation of queries’ results. When a query/command needs to be formulated, the human user can consult the ontology committed by the agent receiving the query and use the vocabulary defined in that ontology as query terms (Figure 2.4) (Mahalingam and Huhns 1997; Yuan 1999). A query composed this way will be directly understood by the queried agent without any need for further query processing. When the results of the query are found, they can be represented using the same ontology as that previously used for query formulation. This allows the human user to receive a single representation scheme of the results, even if the results have been gathered from heterogeneous resources with different local representation schemes (Yuan 1999).

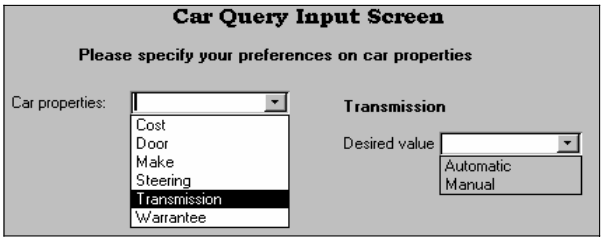


Figure 2.4 – User query formulation using concepts from ontology

Note: Concepts “Cost”, “Door”, “Make”, “Steering”, “Transmission” and “Warrantee” are defined in the Car Domain Ontology committed by the Car Seller agent. Concepts “Automatic” and “Manual” must have been defined as properties or sub-classes of concept “Transmission”.

- **Agent reasoning** at run-time operationalises the problem-solving knowledge of the agent, and uses the domain-related knowledge held by the agent as inputs (Benjamins et al. 1996). If the domain-related knowledge has been modelled as an ontology during agent knowledge modelling at design time, with all relevant domain concepts and relationships being explicitly defined (c.f. Section 2.3.2.3), the agent reasoning process can easily utilize this knowledge and make the most out of it. Followings are a few examples of how ontology-based knowledge can facilitate agent reasoning:
  – The taxonomy of concepts in an ontology can help agents to process a user query by decomposing it into sub-queries. For example, if the user query is “*Find the*

make of all cars", the taxonomy of concepts in the Car Domain Ontology (Figure 2.5) indicates that the Search agent can solve the query by firstly searching for the make of all Sport Cars, Family Cars and Four Wheel Drive, thereafter combining the results. In some cases, the agents can trace up the specialisation hierarchy to provide more generic or additional outputs to the user query if necessary (e.g. finding the make of all motor vehicles apart from cars).

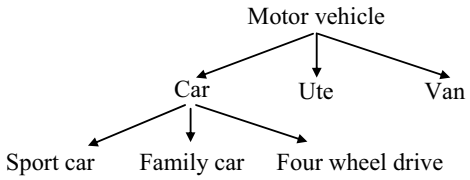


Figure 2.5 – Example fragment of Car Domain Ontology

- Mappings between ontologies may help agents to make useful inferences. For example, in Figure 2.6, given the mapping between the concept "Car audio system" in the Car Domain Ontology and the concept "Car audio" in the Entertainment System Ontology, the Car Seller agent can recommend the user consider buying various Car Audio products when a user submits a car purchase request. Semantic mappings between different ontologies also help agents to perform translation services (e.g. between Car terminology in French and Car terminology in English).

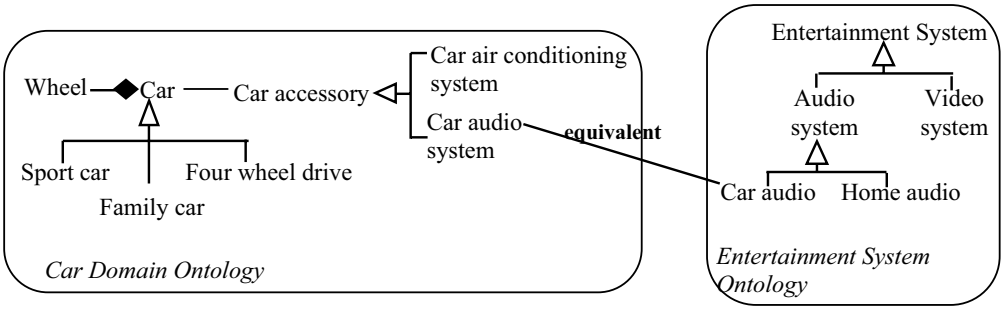


Figure 2.6 – Example ontological mappings between Car Domain Ontology and Entertainment System Ontology

- Mappings between ontologies of heterogeneous resources and a common ontology may help agents to determine the appropriate resources to use without having to access each resource's internal data (Knoblock et al. 1994; Singh 2000). For example, suppose a wrapper agent has access to several Car suppliers'

databases, and the agent is interested in finding out which suppliers offer a car guarantee upon sale of cars. Instead of examining the internal data of each database, the agent can find the answer by simply identifying the databases whose local ontology maintains a semantic mapping with the concept "warranty" in the common Car Domain Ontology.

– Axioms, rules and assertions that specify constraints on concepts and relations (if any) may help agents to reason. For example, a Car Seller agent should know that "Door" of a "Family Car" is never less than 3, and "Cost" of a car must never be lower than "Purchase-Price".

2.3.3. Typology of 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): Generic ontologies, Domain ontologies, Task ontologies and Application ontologies (Figure 2.7).

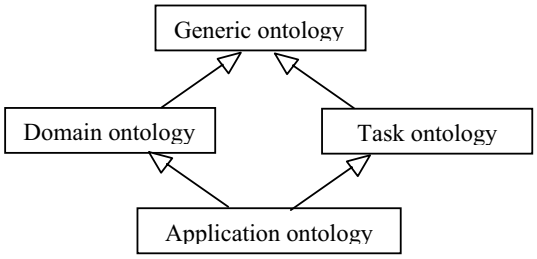


Figure 2.7 – Types of ontology (Guarino 1997)

- **Generic ontologies** define very general concepts about the world such as "Time", "Matter", "Object", "Event", "Action", "Process" and "Component". These concepts are independent of domains and tasks and thus can be reused across applications. For example, CYC (Lenat and Guha 1990) is a generic ontology that provides thousands of concepts and millions of axioms and rules for formalising commonsense knowledge for reasoning.

- **Domain ontologies** define concepts that are specific to particular domains. For example, a Car Domain Ontology defines concepts such as "Make", "Steering" and

"Transmission", while a Medicine Domain Ontology specifies concepts such as "Disease", "Symptom" and "Medication". Domain ontologies may be reused across applications that belong to the same domain. For example, the Unified Medical Language System ontology offers numerous biomedical and health-related concepts that can be reused across biomedical systems (Humphreys and Lindberg 1993). Domain ontologies can be developed by refining Generic ontologies.

- **Task ontologies** define 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 instance, a Negotiation Task Ontology may define concepts such as "Offer" and "Utility rating", while a Propose-and-Revise Task Ontology may capture concepts such as "Fix", "Constraints" and "Input variable" (Gennari et al. 1994; Studer et al. 1996). Task ontologies can be reused in similar tasks across different applications. Task ontologies can be also developed by refining Generic ontologies.

- **Application ontologies**: define concepts that are specific to an application. Since each application is typically characterised by both a particular domain(s) and a particular task(s), Application ontologies are basically a synthesis of Domain ontologies and Task ontologies that have been specialised to model the application's specific knowledge needs. For example, an Application ontology of a Car Selling MAS may define concept "Car-price-offer", which is the specialisation of concept "Car-price" from a Car Domain Ontology and concept "Offer" from a Negotiation Task Ontology. Application ontologies normally cannot be reused across applications, because each different application normally engages in a different combination of domains and tasks.

2.3.4. Ontology Representation Languages

To date, various *textual* and *graphical* modelling languages have been proposed for the representation of ontologies. Section 2.3.4.1 describes some well-known textual languages, while Section 2.3.4.2 reports on graphical languages.

2.3.4.1. Textual representation languages

Textual languages are those that specify ontologies using linear, logic-based expressions. Existing textual ontology languages adopt the following major schemes of knowledge representation.

- **First-order predicate logic**: Symbols of first-order predicate logic allow the representation of *constants* (i.e. specific concepts), *variables* (i.e. unspecified concepts), *predicates* and *functions* (i.e. relations between concepts) and *formula* (i.e. meaningful expressions combining concepts) (Lenat and Guha 1990). CycL (Lenat and Guha 1990) and KIF (Genesereth and Fikes 1992) are two well-known ontology languages which are based on first-order predicate logic. Below is an example fragment of ontology specified in CycL.

```
(genls Dog Mammal)
(#$thereExistAtMost 1 ?TAIL
   (#$and
        (#$anatomicalParts Dog ?TAIL)
        (#$isa ?TAIL #$Tail)))
(Dog is a Mammal. Tail is an anatomical Part of a Dog. Each Dog should have
at most one Tail).
```

- **Description logic**: Knowledge in Description Logic is represented in a hierarchical structure of concepts (Baader et al. 2003). Concepts can be defined by simply naming them and specifying where they fit in the hierarchy. The most important relationships between concepts are *subsumption relationship* (where one concept is the generalisation/specialisation of another) and *conjunction relationship* (where one concept is the joined specialisation of other concepts). KL-ONE (Brachman and Schomolze 1985) and CLASSIC (Borgida et al. 1989) are examples of ontology languages based on description logic. An illustration of KL-ONE ontology fragment is presented below.

```
Human ≤ Anything
Student  ≤ Human
Researcher ≤ Human
PhD-Student ≤ Student
PhD-Student ≤ Researcher
Male-student = (and Man Student)
```

- **Frame-based paradigm**: A frame is a single place in which properties and axioms of a *class* (i.e. an entity) are specified (Bechhofer et al. 2001). Relations between classes are expressed by stating dependencies or restrictions between classes. Two examples of frame-based ontology languages are Ontolingua (Gruber 1993b) and Frame-Logic (Kifer et al. 1995). An illustration of Ontolingua is presented below, where a class "author" is defined. Relations "author.name" and "author.documents" are specified as "slots" in the frame "author". The relation "value-cardinality" is used to express constraints on the slots.

```
(define-class AUTHOR (?author)
    :def (and (person ?author)
        (= (value-cardinality ?author AUTHOR.NAME) 1)
        (>= (value-cardinality ?author AUTHOR.DOCUMENTS) 1)))
```

- **Web-enabled languages**: In the late 1990s, the idea of a Semantic Web where information on the Web is presented in a machine-readable form (Berners-Lee et al. 2001) has called for the development of ontology languages that are compatible with current Web standards. Two examples of web ontology languages are XOL (Karp et al. 1999) and DAML+OIL (Horrocks and van Harmelen 2001). XOL is built upon frame-based approach and XML syntax, while DAML+OIL unifies description logic, frame-based language and RDF. The following example of XOL ontology fragment defines a class "person" with property "last-name" and "age".

```
<class>
        <name>person</name>
</class>
<slot>
        <name>last-name</name>
        <domain>person</domain>
        <value-type>string</value-type>
</slot>
<slot>
        <name>age</name>
        <domain>person</domain>
        <value-type>integer</value-type>
</slot>
```

2.3.4.2. Graphical representation languages

The use of graphical languages to represent ontology is compelling for many reasons.

- They are easier to use during the process of ontology engineering than structured textual language, because of the intuitiveness of the visual structures of the language (Knowledge Based Systems Inc 1994)
- They can easily be communicated with domain experts and users (Falbo et al. 2002; Cranefield et al. 2001).
- They provide a natural medium for representing relational structures, where concepts are modelled as nodes and relations between concepts as arcs (Kankaanpää 1999).

Some graphical languages for representing ontology are UML (Cranefield and Purvis 1999; Cranefield et al. 2001; Bergenti and Poggi 2001; Bergenti and Poggi 2002), IDEF5 Schematic Language (Knowledge Based Systems Inc 1994) and LINGO (Falbo et al. 1998; Falbo et al. 2002).

- **UML**: UML is a modelling language for OO analysis and design. However, it has been applied to the representation of ontologies. With UML, each ontology is modelled as a class diagram, where *classes* represent entities and *relationships* symbolize relations between entities (Figure 2.8). A class is characterised by its name and attributes, and each attribute is defined by its name and type. Operations/methods are not necessary for classes because ontologies only capture the conceptual structure of the entities (Bergenti and Poggi 2002). Relationships between entities can be *generalisation*, *aggregation* or *association*. The semantics and notation of each type of relationship are the same as in OO modelling. The ends of the association relationships may be labelled with "role names" of the relating classes. Associations that embrace attributes will be modelled by an "association class". Object Constraint Language (OCL) can be used to represent constraints on classes, attributes and relationships. These constraints are specified as *notes* in the UML class diagram.

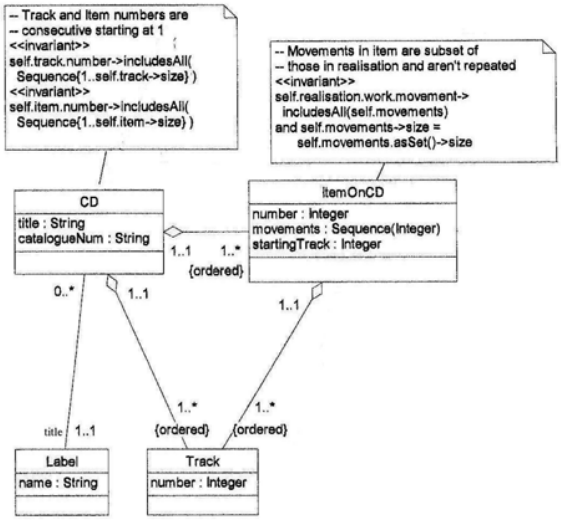


Figure 2.8 – Example of ontology representation in UML (Cranefield and Purvis 1999)

- **IDEF5 Schematic Language:** IDEF5 is a tool for creating and editing ontologies. It offers two languages for representing ontology: *IDEF5 Schematic Language* which provides graphical notation and *IDEF5 Elaboration Language* which provides first-order logic formalism.

  IDEF5 Schematic Language models ontological concepts as *kinds* (which is equivalent to classes in UML) and relationships between concepts as *relations* or *transitions*. Relations have the same semantics as in UML, while transitions refer to a special kind of relationship where the concept at one end of the relationship may be transformed into the concept at the other end. The process involved in each transition may be captured as a "*process*" entity attached to the transition (e.g. process "*Dry*" in Figure 2.9). Axioms and rules constraining the concepts, relations, transitions and processes can be recorded using IDEF5 Elaboration Language.

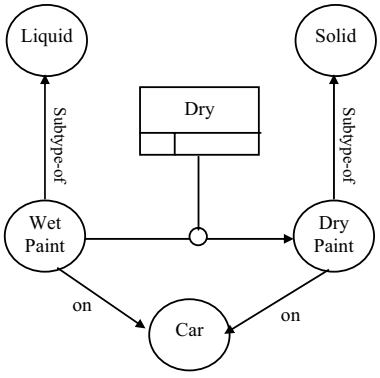


Figure 2.9 – Example of ontology representation in IDEF5 Schematic Language (Knowledge Based Systems Inc 1994)

- **LINGO:** The modelling primitives of LINGO are *concepts* and *relations*. Potential types of relations are *generalisation*, *composition* and *association* (with the same semantics as UML relationships). Axioms and rules about concepts and relations can be specified using first-order logic assertions accompanying the diagram.

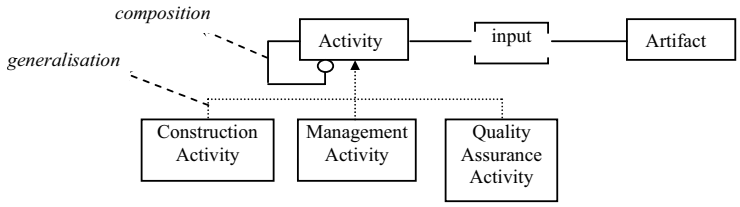


Figure 2.10 – Example of ontology representation in LINGO (Falbo et al. 1998)

2.4. 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.

In the next chapter, Chapter 3, a review of the existing AOSE methodologies for MAS development is documented. That chapter includes the identification of the limitations of these methodologies with regard to their support for MAS analysis and design, and their support for ontology-based MAS development. Limitations on the latter directly cause these methodologies to not being able to fully realise the benefits of ontology to MAS development and MAS operation which are listed in this chapter (Section 2.3.2.2).

CHAPTER 3

REVIEW OF EXISTING MAS DEVELOPMENT METHODOLOGIES

3.1. INTRODUCTION

This chapter reviews the AOSE methodologies that have been proposed in the literature for the analysis and design of MAS. It firstly describes each methodology in Section 3.2, thereafter identifying the general limitations of these methodologies in Section 3.3. The limitations include those relating to the generic MAS analysis and design activities (Section 3.3.1), and those relating particularly to the support for ontology-based MAS development (Section 3.3.2). A more detailed evaluation of these AOSE methodologies would be documented in Chapter 5.

From here on, the phrase "MAS development methodology" is used interchangeably with the phrase "AOSE methodology" to mean an AOSE methodology that covers the analysis and design activities of MAS development.

3.2. DESCRIPTION OF EXISTING MAS DEVELOPMENT METHODOLOGIES

Even though research in AOSE is still less developed than other conventional software engineering paradigms such as OO paradigm, work has increased in this area in recent years. A number of AOSE methodologies have been proposed to assist in the analysis and design of MASs. These methodologies vary significantly in their scope, approach, process steps, modelling concepts and modelling notation.

In total, sixteen AOSE methodologies are reviewed in this chapter. These methodologies were identified from an extensive search of the literature and selected for investigation based on the following criteria.

- The chosen methodology has been applied or tested on case studies or industrial projects.
- The chosen methodology has been referenced by other researchers in the field.
- The chosen methodology satisfies the definition of a "*software engineering methodology*". As defined by Henderson-Sellers et al. (1998), a software engineering methodology is one that provides the following key elements:
  - a software engineering *process* to conduct the development;
  - *techniques* to assist the process; and
  - *definition of work products*.

  Only AOSE methodological frameworks that provide all three elements were selected for study in the thesis.

In the following sections, a brief description of each selected methodology is presented.

3.2.1. MASE

MASE, **"**MultiAgent System Engineering" (Wood 2000; Wood and DeLoach 2000a; Wood and DeLoach 2000b; DeLoach 2005), takes an initial system specification and produces a set of formal design documents for a MAS. It is based upon the preceding research work in AOSE (such as Kendall and Zhao 1998 and Kinny *et al.* 1996) and conventional OO modelling techniques (such as OMT and UML). An overview of MASE is provided in Figure 3.1.

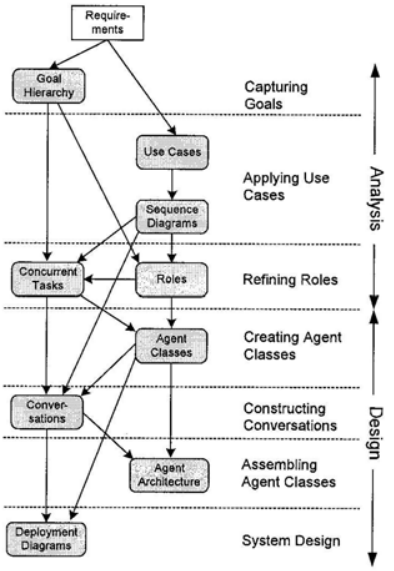


Figure 3.1 – Overview of MASE (Wood and DeLoach 2000a)

The development process 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 a Goal Hierarchy Diagram.
2. "*Applying use cases*" step produces Use Cases from the system requirements and elaborates them into Sequence Diagrams.
3. "*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 3.2). The developer may further elaborate the Role Model by defining tasks to be performed by each role and 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.

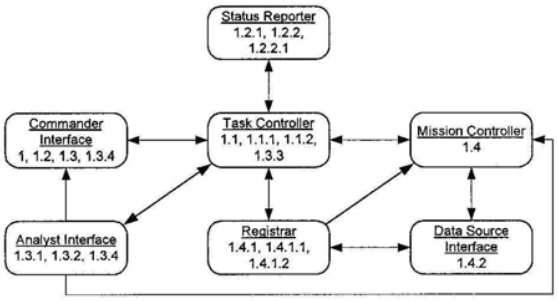


Figure 3.2 – MASE Role Model (Wood and DeLoach 2000a)

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.

1. "*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 (Figure 3.3).

2. "*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 (Figure 3.4).
3. "*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.
4. "*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 (Figure 3.5).

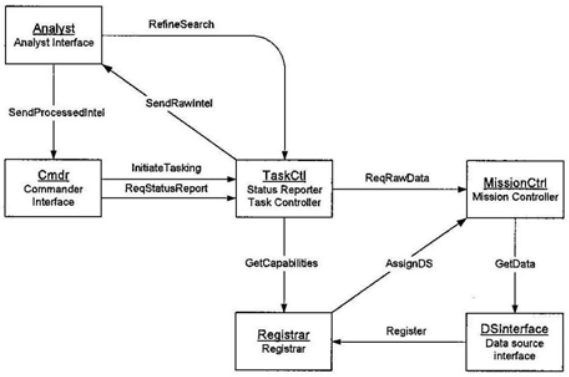


Figure 3.3 – MASE Agent Class Diagram (Wood and DeLoach 2000a)

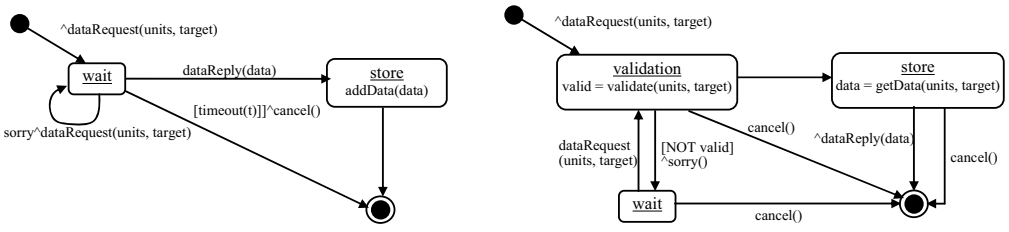


Figure 3.4 – MASE Communication Class Diagram for initiator (left) and responder (right) (Wood and DeLoach 2000a)

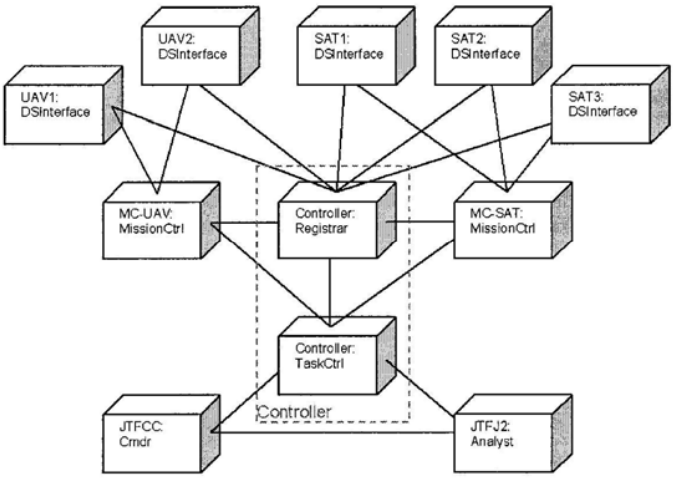


Figure 3.5 – MASE Deployment Diagram (Wood and DeLoach 2000a)

In a recent publication (DiLeo et al. 2002), MASE has been expanded to provide support for ontology-based MAS development. Ontology is introduced as a mechanism to model the application domain. An additional step – *"Building ontology*" – has accordingly been added to the Analysis phase (Figure 3.6). 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.

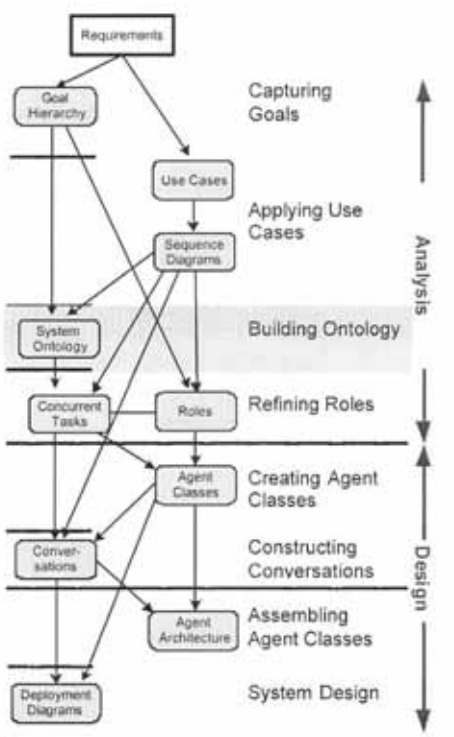


Figure 3.6 – Overview of extended version of MASE (DiLeo et al. 2002)

3.2.2. MASSIVE

MASSIVE (Lind 1999; Lind 2000a) follows an "iterative view engineering process" for MAS development, which is a product-centred development process that combines Round-trip engineering and Iterative Enhancement (Figure 3.7). In the first cycle of the development process, the developer firstly produces a preliminary version of the development product (1), which is composed of seven different "*views*" of the system. These views are then implemented (2) and refined if errors occur during implementation (3). The initial implementation is then tested and/or enhanced (4), which may result in enhancements to the views (5). 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 (6). After this step, the next cycle is executed until the entire system is fully implemented.

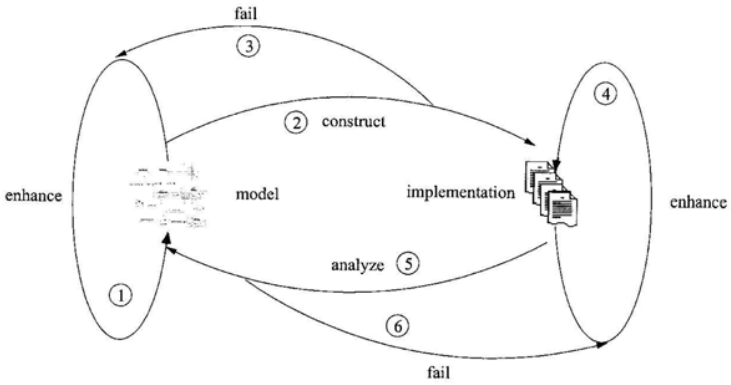


Figure 3.7 – MASSIVE Iterative View Engineering process (Lind 2000a)

Each of the seven views of MASSIVE describes a particular aspect of the target system which is conceptually linked to other views. These views are briefly discussed below.

- **Task View** specifies the tasks to be fulfilled by the target system. It is developed through iterative functional decomposition of the problem domain (Figure 3.8). Leaf nodes of the task hierarchy represent atomic activities which are to be used for roles identification.

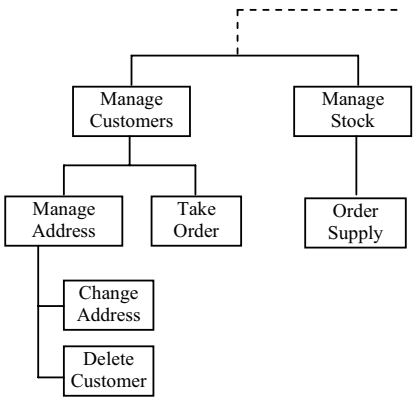


Figure 3.8 – MASSIVE Task View (Lind 1999)

- **Environment View** models MAS's environment from both the perspective of the developer and the perspective of the system. Regarding the developer's perspective, the environment should be characterised in term of its *organisational context* (i.e. accessible or inaccessible, deterministic or non-deterministic, episodic or non-episodic, static or dynamic) and *runtime context* (e.g. programming model, programming language and communication mode). Regarding the system's

perspective, the developer should determine the input/output mechanism used by the agents to interact with the environment (e.g. sensor/effector).

- **Role View** identifies roles for the target system and assigns roles to agents. To identify roles, leaf-node tasks in Task View are clustered in such a way as to maximize intra-cluster coherence while satisfying the physical resource constraints of the operational environment of the target system. MASSIVE does not provide any guidelines on how to assign roles to agents.

- **Interaction View** characterises the general nature of agent interactions in the target application, thereafter using this characterisation to choose an appropriate interaction scheme for the system (e.g. information exchange scheme, market-based scheme or blackboard interaction scheme). The developer should also identify any interaction protocols that are necessary for the system (e.g. Contract Net or Simulated Trading).

- **Society View** characterises the society of agents in MAS according to various dimensions, including type (i.e. open or closed), structure (i.e. flat or hierarchical), consistency (i.e. homogeneous or heterogeneous) and temporal context (i.e. static or dynamic). MASSIVE offers guidelines on how to design the social structure for the agent society given the characterisation.

- **Architectural View** specifies the system architecture and agent architecture. The modelling of system architecture should include the modelling of system entities (e.g. conventional objects besides agents), agent management tasks/facilities, database design and external components/devices. Regarding agent architecture, the developer is recommended to identify specific architectural requirements and select from the existing architectures before trying to develop a new one from scratch.

- **System View** deals with design issues that affect the MAS system as a whole (e.g. user interface design, exception handling and performance engineering).

3.2.3. SODA

SODA, "*S*ocieties in *O*pen and *D*istributed *A*gent 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.

The **Analysis Phase** is concerned with constructing three models.

- *Role Model* identifies and defines all roles in the target system. SODA derives roles from system tasks. Each task can either be assigned to a single role or a group of coordinating roles. In the latter case, the task is named a "social task" and each role in the group is called a "social" role. Each role and each role-group is defined in terms of their individual and/or social tasks, permissions to resources (which are identified in Resource Model) and interaction protocols and rules (which are defined in Interaction Model).

- *Resource Model* defines all abstract resources provided by the environment to the target MAS. Each resource is described in terms of its services, access modes (i.e. the different ways in which the services can be exploited by agents) and permissions granted to roles and role-groups.

- *Interaction Model* defines interaction protocols for roles and for resources, as well as interaction rules for role-groups. An interaction protocol specifies the information required/provided by a role to accomplish its tasks, or by a resource to invoke its services. An interaction rule for a role-group governs the interactions among social roles and resources so as to make the group accomplish its social task.

The **Design Phase** of SODA is concerned with transforming the preceding Analysis models into design abstractions that can be mapped one-to-one onto the actual components of the implemented MAS system. These design abstractions are captured in three related models.

- *Agent Model* defines agent classes in the system. Each agent class is composed of a set of roles (both individual and social) and accordingly characterised by its

individual/social tasks, permissions to resources and interaction protocols associated with its roles. Agent classes can be further described by cardinality, location and source (i.e. from inside or outside the system).

- *Society Model* describes agent societies in a MAS. Each society is formed by agents whose roles belong to a role-group. The developer must choose the most suitable "coordination model" for the target system, for example, one that provides abstractions expressive enough to model the society's interaction rules such as those surveyed in Papadopoulos and Arbab (1998). Interaction rules can be derived from the Interaction Model and embodied as coordination rules in the selected coordination model.

- *Environment Model* identifies "infrastructure classes" of the MAS environment by mapping from resources in the Resource Model. Each infrastructure class is given a location, owner and cardinality. A topological model of the MAS environment can be developed based on the developer's choice, such as the TuCSoN model (Cremonini et al. 1999).

3.2.4. GAIA

This widely referenced methodology aims to guide the developer from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly (Wooldridge et al. 1999; Wooldridge et al. 2000). GAIA has been recently extended to include new organisational abstractions that enable it to support the development of "open" MASs (Zambonelli et al. 2003).

The **Analysis Phase** of GAIA firstly investigates the potential existence of multiple sub-organisations within the target system. If multiple sub-organisations co-exist, they are analysed and designed as autonomous interacting MASs.

The *Environment Model* is then constructed to describe the MAS environment in terms of abstract computational resources (e.g. variables or tuples that the agents can read/access). Each resource is characterised by the types of actions that agents can perform on it.

A preliminary definition of roles is subsequently produced in the *Preliminary Role Model*. Roles may be identified from the system's real-world organisation (e.g. real-world offices or departments) or from the "basic skills" that are required by the organisation to achieve its goals. GAIA models each role in terms of responsibilities and permissions to resources (Figure 3.9). Responsibilities represent the role's functionality and are classified into two types: "safety" or "liveness". Safety responsibilities are typically predicates, specifying the acceptable state of affairs that should be maintained across all states of execution. Liveness responsibilities, on the other hand, specify the state of affairs that an agent must bring about (i.e. "something will be done"). Each liveness responsibility is defined as a set of activities and interaction protocols.

Role Schema: CUSTOMERHANDLER

Description:
Receives quote request from the customer and oversees process to ensure appropriate quote is returned.

Protocols and Activities:
AwaitCall, ProduceQuote, InformCustomer

Permissions:
reads supplied *customerDetails* // customer contact information
supplied *customerRequirements* // what customer wants
quote

Responsibilities
Liveness:
CUSTOMERHANDLER = (AwaitCall.GenerateQuote)ω
GENERATEQUOTE = (ProduceQuote.InformCustomer)
Safety:
- true

Figure 3.9 – GAIA Role Model (Zambonelli et al. 2003)

A *Preliminary Interaction Model* is also developed. In GAIA, a protocol is viewed as an institutionalised pattern of interaction. Each protocol definition only describes the interaction's purpose, initiator roles, responder roles, inputs, outputs and processing (Figure 3.10). It is abstracted away from any particular sequence of messages.

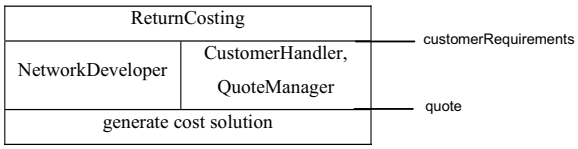


Figure 3.10 – GAIA Interaction Model (Wooldridge et al. 2000)
(Protocol *ReturnCosting* is initiated by role *NetworkDeveloper* and involves roles *CustomerHandler* and *QuoteManager*. The protocol takes as input *customerRequirements* and produces *quote*.)

The last step of the Analysis phase defines the *organisational rules* for the target system. "Liveness" organisational rules specify how the dynamics of the MAS organisation should evolve over time, while "safety" organisational rules define time-independent global invariants of the organisation.

The **Design Phase** of GAIA transforms the above Analysis models into sufficiently low-level abstractions, so that traditional design techniques (such as OO techniques) may be applied. The phase starts with the selection of an *organisational structure* for the target system. The developer should choose a structure that provides the most appropriate topology and authority relationship. Once the organisational structure is defined, the Preliminary Role and Interaction Models can be refined and elaborated into *Complete Role* and *Interaction Models*.

Other design models to be developed are Agent Model, Service Model and Acquaintance Model.

- *Agent Model* identifies agents from roles by applying a one-to-one mapping between roles and agent classes. A simple hierarchy (or hierarchies) can be used to model the agent class structure, where root nodes correspond to roles and other nodes correspond to agent classes. The Agent Model also shows the instantiation of each agent class (Figure 3.11).

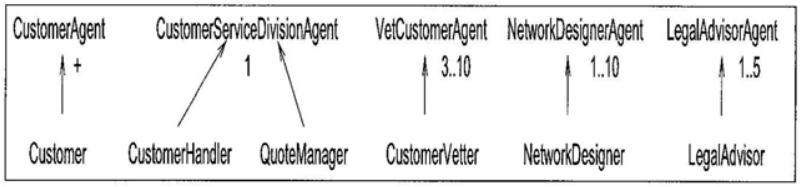


Figure 3.11 – GAIA Agent Model (Wooldridge et al. 2000)

- *Service Model* identifies services offered by each agent and properties of these services (i.e. inputs, outputs, pre-conditions and post-conditions). Services can be derived from roles' responsibilities, particularly liveliness responsibilities), and interaction protocols.

- *Acquaintance Model* specifies communication links between agent classes (Figure 3.12). The goal is not to specify what messages are sent and when, but to identify any potential communication bottlenecks between agents and to evaluate if the system is internally loosely coupled.

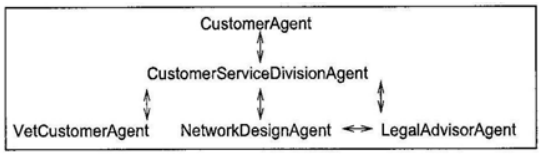


Figure 3.12 – GAIA Acquaintance Model (Wooldridge et al. 2000)

3.2.5. MESSAGE

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

The **Analysis Phase** is concerned with constructing five models.

- *Organisation Model* describes the structural and acquaintance relationships between the target system and its environments (Figure 3.13), and the acquaintance relationships between agents/roles and resources making the system (Figure 3.14).
- *Goal/Task Model* specifies the decomposition structure of goals of the target system. A Workflow diagram may be developed for each goal to specify what tasks are needed to achieve the goal and which roles are responsible for which tasks.
- *Agent/Role Model* describes the individual agents/roles in terms of their goals, resources and tasks.
- *Domain Model* shows domain-specific entities and relations that are relevant to the target application (Figure 3.15). This model is basically equivalent to a domain ontology.
- *Interaction Model* specifies, for each interaction between agents/roles, the initiator, collaborators, motivation and information supplied/achieved by each participant (Figure 3.16).

These five Analysis models are developed in a step-wise refinement manner, with the subsequent refinement cycles elaborating and expanding the models developed in the previous cycle. MESSAGE proposes three major approaches for this refinement, namely organisation-centred, agent-centred and goal/task refinement approaches.

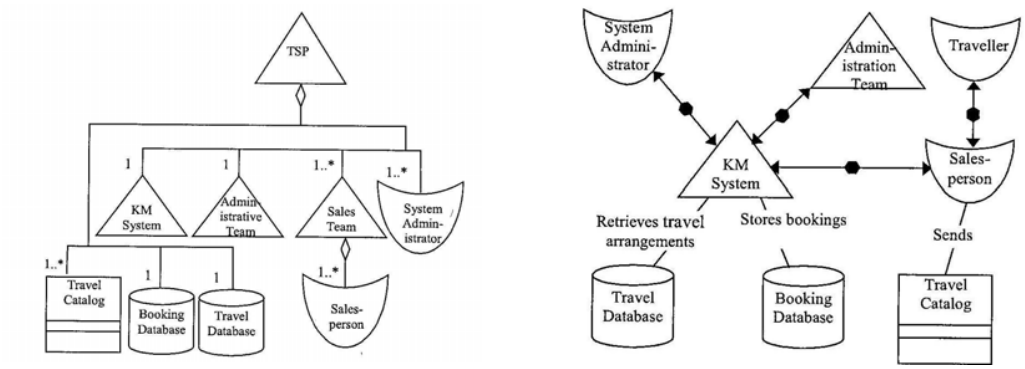


Figure 3.13 – MESSAGE Organisation Model – Structural Relationships (left) and Acquaintance Relationships (right) (Eurescom 2001b)

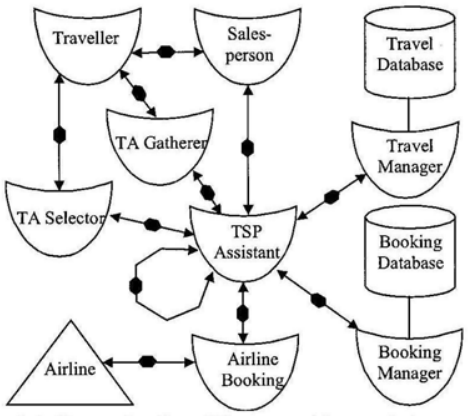


Figure 3.14 – MESSAGE Organisation Model - Agent/Role and Resources Acquaintance Relationships (Eurescom 2001b)

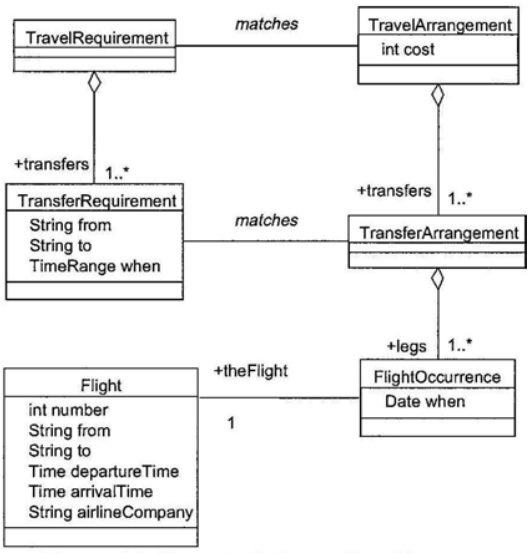


Figure 3.15 – MESSAGE Domain Model (Eurescom 2001b)

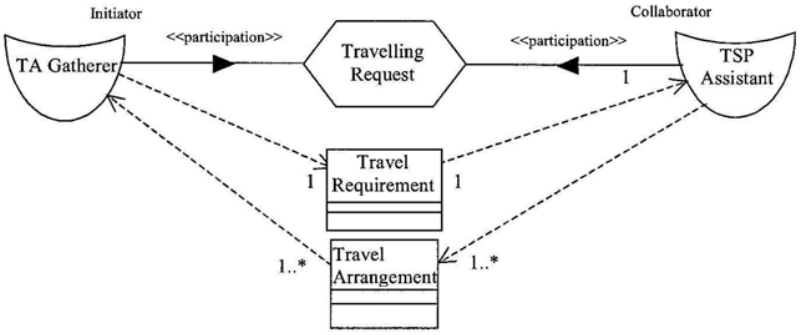


Figure 3.16 – MESSAGE Interaction Model (Eurescom 2001b)

The **Design Phase** of MESSAGE transforms the above Analysis models into computational entities that can be implemented on an agent platform. The phase is structured into *High-Level Design* and *Low-Level Design*.

During High-Level Design, the roles identified in the Analysis phase are assigned to agents. If agents have been identified during Analysis, they should be re-examined to check if they are indeed appropriate to be implemented as agents (some agents during Analysis may be implemented as simple classes or resources). Interactions identified in the Analysis phase should also be elaborated with interaction protocols and UML state-charts.

For Low-Level Design, MESSAGE proposes two approaches for mapping high-level design concepts to specific computational elements: Organisation Driven approach and Agent-Platform Driven approach. The former uses the Organisation Model to derive the MAS architecture, agent architecture, agent knowledge and resources. The latter is more platform-oriented and considers that each agent can be implemented as a simple class. A detailed description of each approach is provided in Eurescom (2001b).

3.2.6. Methodology for BDI Agents (BDIM)

As mentioned in Section 2.2.3, BDI is a prominent architectural model for agents. Each BDI agent is composed of *beliefs* (i.e. the agent's knowledge of the world), *desires* (i.e. the agent's motivations such as goals, objectives or allocated tasks) and *intentions* (i.e. the desires that the agent is committed to achieving at a certain point in time) (Rao and

Georgeff 1995). The BDIM methodology (Kinny and Georgeff 1996; Kinny et al. 1996) is specifically targeted at MASs that are based upon 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.

For the External models, a four-step process is proposed.

1. Identify major roles in the system and produce a draft *Agent Model*. This model captures the inheritance and aggregation relationships among abstract and concrete agent classes, as well as the instantiation of each concrete agent class (Figure 3.17). During this step, agent classes are expected to be quite abstract, not assuming any particular granularity of agency.
2. Identify responsibilities and associated services of each role. Each agent class should be accordingly decomposed to the service level.
3. Specify, for each service, interactions that may occur between roles/agents, thereby producing an *Interaction Model*. The model should describe the responsibilities and services of each agent class, the associated interactions and the control relationships between agent classes. BDIM however does not impose any modelling notation for its Interaction Model.
4. Refine the Agent Model to introduce any new abstract and/or concrete agent classes if necessary (for example, agent classes that offer some common services may be modelled as specialisations of an abstract agent class).

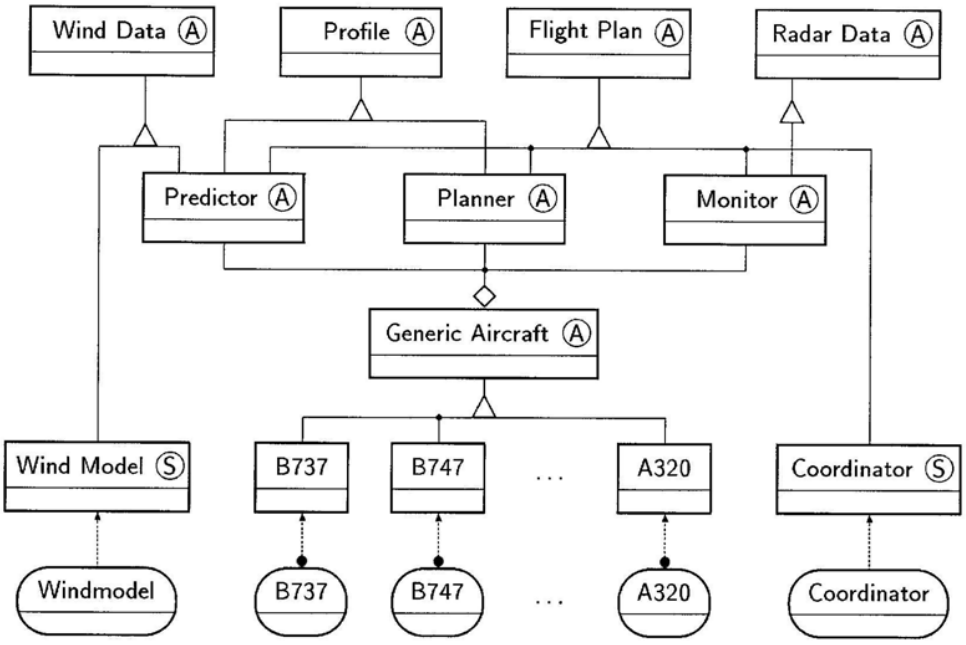


Figure 3.17 – BDIM Agent Model (Kinny et al. 1996)

The construction of Internal models for each agent class begins from the third step of the above process and involves two steps of its own.

1. Identify goals of each agent class and analyse the means for achieving these goals. This step generates a *Goal Model* and *Plan Model* for each agent class. Agent's goals can be derived from the services identified in step 2 of the External models' development process. The *Goal Model* consists of a Goal Set and one or more Goal States. Each goal is specified by goal formula signatures, e.g. achieve(!), verify(?) and test($). The *Plan Model* contains a Plan Set, which consists of a set of Plan Diagrams (Figure 3.18).
2. Model the agent's beliefs by analysing the contexts in which goals are achieved and the conditions that control the execution of plans' actions (including input and output data requirements). Agent's beliefs are captured in the *Belief Model*, which contains one Belief Set and one or more Belief States. The Belief Set (Figure 3.19) conceptualises the potential beliefs of the agent, while Belief States are particular instances of the Belief Set.

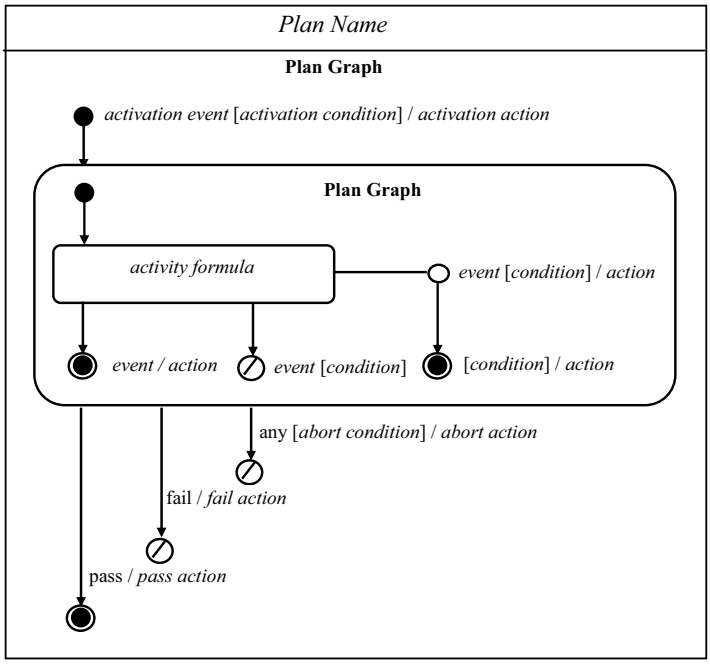


Figure 3.18 – BDIM Plan Diagram (Kinny et al. 1996)

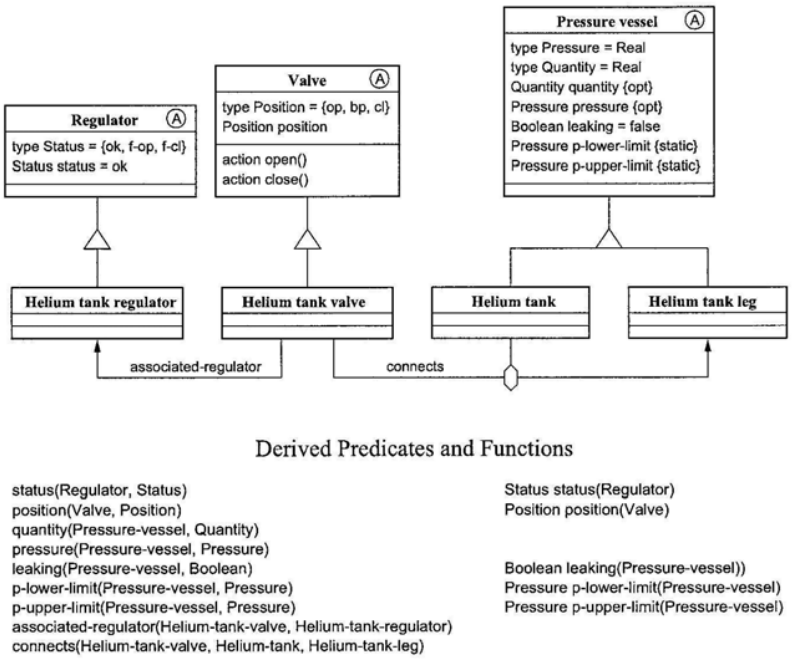


Figure 3.19 – BDIM Belief Set (Kinny and Georgeff 1996)

3.2.7. INGENIAS

INGENIAS (Pavon and Gomez-Sanz 2003; Pavon et al. 2005) is built upon MESSAGE/UML. 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.

The development process of INGENIAS adopts the Unified Software Development Process lifecycle. It contains around seventy steps that are distributed among the lifecycle phases and workflows. Figure 3.20 summarizes the outputs to be obtained in each phase and workflow of the INGENIAS development lifecycle.

		PHASES		
		Inception	**Elaboration**	**Construction**
WORKFLOWS	**Analysis**	o Generate use cases and identify actions of these use cases with interaction models. o Sketch a system architecture with an organization model. o Generate enviroment models to represent results from requirement gathering stage	o Refined use cases o Agent models that detail elements of the system architecture. o Workflows and tasks in organization models o Models of tasks and goals to highlight control constraints (main goals, goal decomposition) o Refinements of environment model to include new environment elements	o Refinements on existing models to cover use cases
	Design	o Generate prototypes perhaps with rapid application development tool such as ZEUS o Agent Tool.	o Refinements in workflows o Interaction models that show how tasks are executed. o Models of tasks and goals that reflect dependencies and needs identified in workflows and how system goals are achieved o Agent models to show required mental state patterns	o Generate new models o Social relationships that perfect organization behaviour.

Figure 3.20 – Outputs of each phase and workflow of INGENIAS development process (Pavon et al. 2005)

The Analysis and Design workflows of INGENIAS aim to incrementally construct five work products: Agent, Interaction, Goals/Tasks, Organisation and Environment Models. During the **Analysis Workflow**, the information to be included in each model is described below.

- *Agent Model* defines the prospective agents in terms of their roles, goals, tasks and requirements (e.g. intelligence or learnability).

- *Interaction Model* captures significant interaction paths within the system and the information to be passed between interacting parties.
- *Goal/Task Model* shows the initial goals of the target system, tasks for achieving these goals, decomposition of goals and tasks, success/failure conditions of goals and pre-/post-conditions of tasks.
- *Organisation Model* shows the structure of the target MAS organisation via system components such as groups, agents, roles, resources and applications (Figure 3.21). Tasks described in Goal/Task Model should also be included in the Organisation Model to show their executors (i.e. agents or roles), their inter-connections (i.e. workflows) and their required resources. Social dependencies among agents, roles and/or groups should also be defined (e.g. subordination or client-server dependencies).
- *Environment Model* specifies resources and applications that exist in the environment, and the perception mechanisms used by agents to perceive the outputs of these applications. Example perception mechanisms are sampling and notification.

The **Design Workflow** of INGENIAS refines and extends each of the above five models. The Agent Model is updated to include the detailed definition of each agent's mental states (i.e. beliefs, goals and plans), mental state manager and processor. The Interaction Model is elaborated to specify the exchanged elements (e.g. tuples, messages or remote procedure calls) and the order of their execution (e.g. iteration, concurrency and branching). The Goal/Task Model, Organisation Model and Environment Model are also incrementally refined from Analysis to Design.

INGENIAS also includes an **Implementation workflow** to generate code modules for the design specifications. The workflow involves incrementally generating prototypes for the specifications using the INGENIAS Development Kit and reusing templates and algorithms provided by the INGENIAS Development Kit.

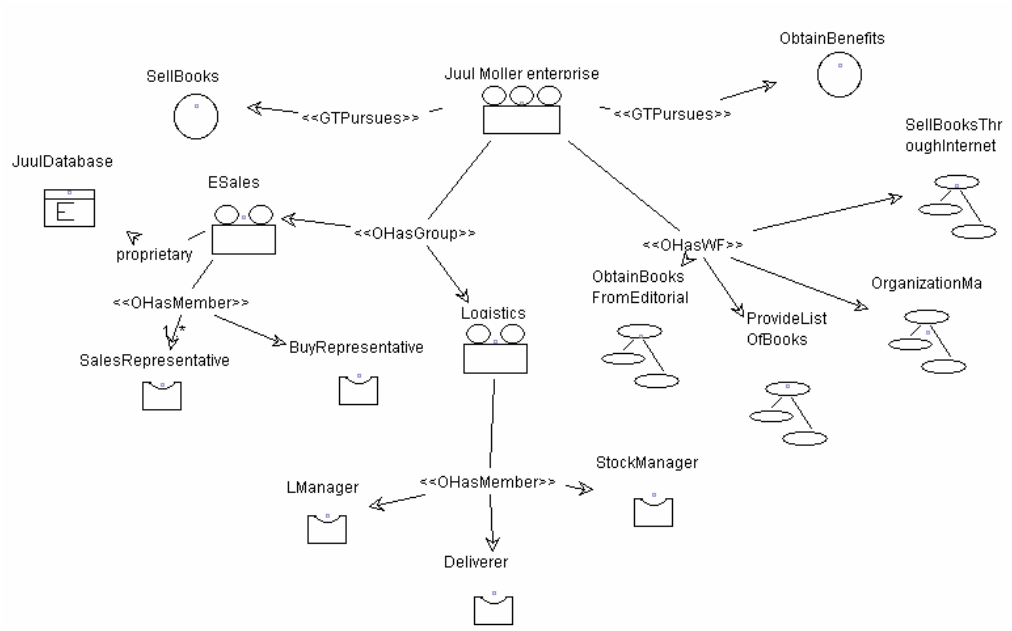


Figure 3.21 – INGENIAS Organisation Model (Pavon et al. 2005)

3.2.8. Methodology with High-Level and Intermediate Levels (HLIM)

HLIM (Elammari and 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.

In the **Discovery Phase**, a *High-level Model* is developed to capture the overall structure and behaviour of the system. The model is composed of a set of Use Case Maps (UCMs), each of which shows "paths" that trace a scenario from a start point to an end point, connecting the responsibilities of participating agents (Figure 3.22). The concept of "role" is used in UCMs to represent organisational places where agents may dynamically enter. Initial agents can be identified by examining the nouns of the application description. These nouns should be essential, autonomous and active in nature.

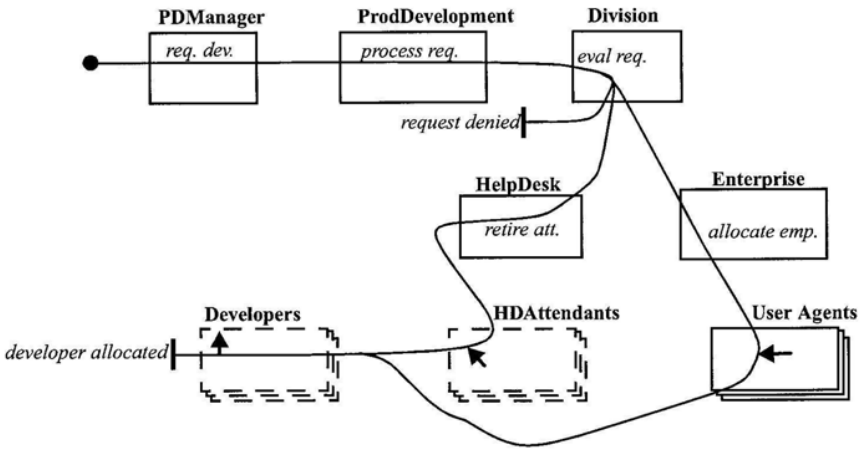


Figure 3.22 – HLIM Use Case Map (Elammari and Lalonde 1999)

The **Definition Phase** of HLIM then uses the High-level Model to produce four Intermediate Models.

- *Internal Agent Model* defines each agent in terms of goals, beliefs, tasks and plans (Figure 3.23). Agent's goals can be derived from the path segments traversing the agent in respective UCMs. Agent's beliefs correspond to the path's pre-conditions and post-conditions, while agent's tasks are derived from responsibilities along the path. A plan is represented by a combination of a particular goal, corresponding task and beliefs.

	Goal	Precondition	Postcondition	Task	Comment
1	Initiate call request	User off-hook	Request sent to answerer or call denied	Goal(process off-hook) Goal(originate call) Goal(notify caller)	Caller in main UCM
2	Process call request	There is an incoming call	Incoming call processed	Goal(terminate call) Goal(notify answerer)	Answerer in main UCM
3	Originate call	Number is collected	Request sent to answerer	send_request	Default plug-in for OC stub
4	Originate call	Number is collected	Request sent to answerer or call denied	check_list send_request notify_refuse	OCS plug-in for OC stub
5	Terminate call	There is an incoming call	Caller and/or answerer are notified	ring notify_caller	Default plug-in for TC stub
6	Terminate call	CF is on. There is an incoming call	Caller notified of a new destination	forward_req	CF plug-in for TC stub

Figure 3.23 – HLIM Internal Agent Model (Elammari and Lalonde 1999)

- *Agent Relationship Model* captures inter-agent relationships, which can be derived from path segments connecting two agents in UCMs. The model is composed of a *Dependency Diagram* and a *Jurisdictional Diagram.* The former captures goal

dependencies, task dependencies, resource dependencies and negotiation dependencies among agents, while the latter depicts the organisational structure of agents in terms of their authority status (Figure 3.24).

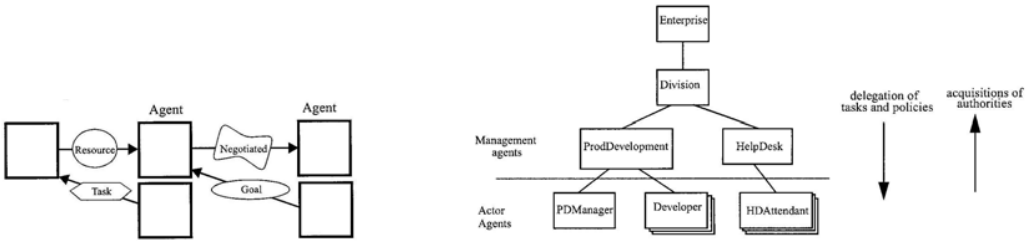


Figure 3.24 – HLIM Dependency Diagram (left) and Jurisdictional Diagram (Elammari and Lalonde 1999)

- *Conversational Model* uses tabular schemata to specify, for each agent, the messages it communicates with other agents (Figure 3.25). The model can be derived from Internal Agent Model and Agent Relationship Model.

	Receive	Send	Comment
1		Prop(:connectFrom a :connectTo b)	Originating
2		Prop(:connectFrom a :connectTo b)	OCS
3	Prop(:connectFrom a :connectTo b)	ACCEPT | REJECT	Terminating
4	Prop(:connectFrom a :connectTo b)	CProp(:connectFrom a :connectTo f)	CF
5	CProp(:connectFrom a :connectTo f)	Prop(:connectFrom a :connectTo f)	Originating

Figure 3.25 – HLIM Conversational Model (Elammari and Lalonde 1999)

- *Contract Model* specifies the contracts between different agents regarding the services they provide to each other. Each contract is defined in terms of the authorizations and obligations of the participating agents, and is represented using a textual schema.

3.2.9. Methodology for Enterprise Integration (MEI)

MEI (Kendall et al. 1995) 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.

In the development process of MEI, the developer is required to firstly describe the target application in terms of use cases and IDEF/CIMOSA models. The subsequent MAS development activities are not structured in any specific temporal order. MEI simply offers a set of mappings that can be applied on use cases and IDEF/CIMOSA models in order to derive the MAS system design. The mappings are summarized in Table 3.26.

Table 3.26 – Summary of mappings from Use Case Model and IDEF/CIMOSA Models to MAS design in MEI (Kendall et al. 1995)

IDEF/CIMOSA Models	Use case	MAS
Resource or mechanism	Active, proactive actor	Agent
Enterprise function with control output	Use case and use case extension	Agent's Goal and Plan
Functional input into enterprise function	- Input from actors - Input from domain objects via control objects - Domain object input	- Beliefs - Input from coexisting objects via sensor/agents
Functional output from enterprise function	Control object output targeted for actors or domain objects	Output to coexisting objects via effector /agents
Control input to enterprise function	Input from actors or entity objects. It determines which use case extension, if any, is followed	- Input from coexisting objects via sensor/agents - Can be represented as plan's invocation condition
Control output from enterprise function	Control object output targeted for actors or domain objects	- Goal/ subgoal - Can be transmitted to coexisting objects via effector/agents
- More than one resource per function - Information exchange between resources	- More than one actor per use case - Use case event trace - Use case abstraction and specialisation (inheritance)	- Agent collaboration - Coordination protocol
Information model	Domain objects	Beliefs and coexisting objects

As can be noted from the above table, MEI adopts a BDI-like model of agency. Each agent is composed of goals, plans, beliefs and intentions, and is connected to sensor and effector objects (Figure 3.27). The sensor watches the external environment while the effector brings about the changes desired by the agent. A passive object is used to hold the agent's beliefs.

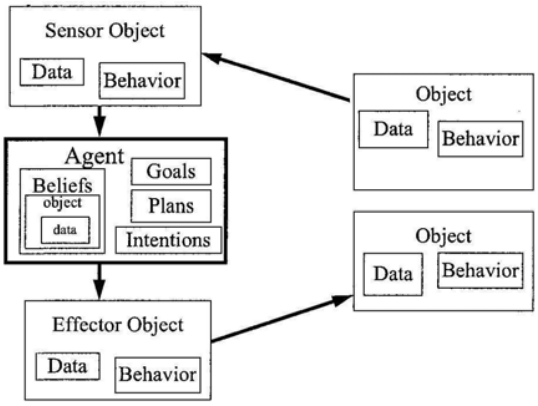


Figure 3.27 – MEI agent structure (Kendall et al. 1995)

For each agent, its goals and plans can be depicted as a tree structure, where goals are root nodes and plans are leaves. Each plan can be further defined by a state diagram. Coordination protocols can also be described using state diagrams. The specification of sensors and effectors for each agent is modelled using IDEF-like notation (Figure 3.28).

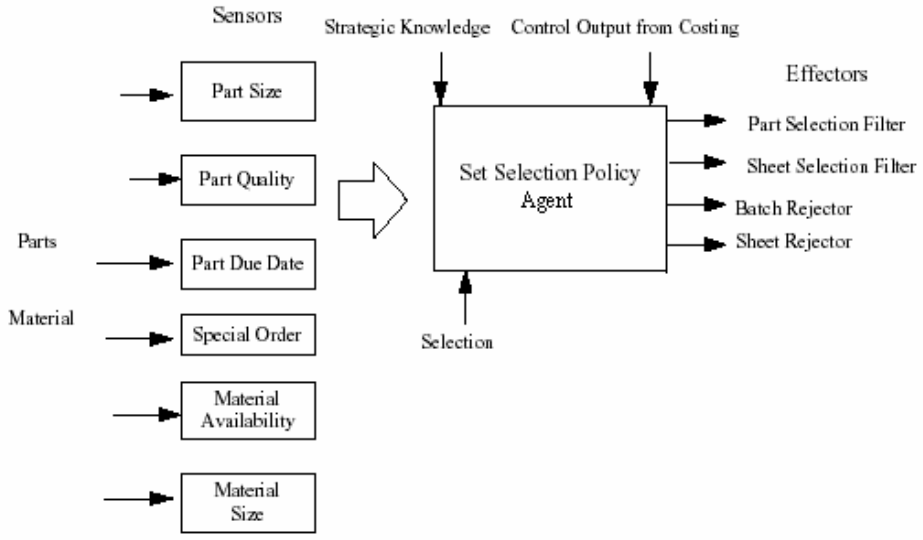


Figure 3.28 – MEI sensors and effectors specification (Kendall et al. 1995)

3.2.10. PROMETHEUS

PROMETHEUS (Padgham and Winikoff 2002a; Padgham and Winikoff 2002b; Winikoff and 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 (Figure 3.29).

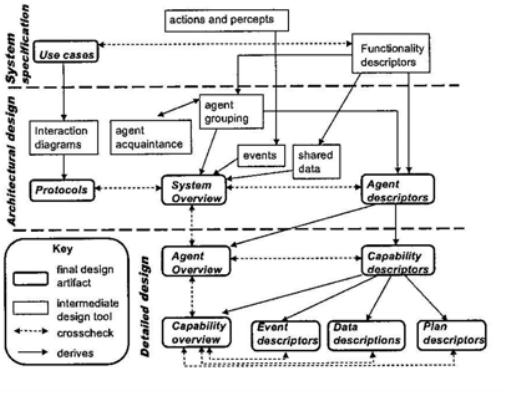


Figure 3.29 – Overview of PROMETHEUS (Padgham and Winikoff 2002a)

The **System Specification Phase** focuses on identifying the basic functionality, external interfaces and use case scenarios of the target MAS.

- Functionality describes what the system should do in a broad sense and is specified informally using textual *Functionality Descriptors*.
- External interfaces refer to incoming raw data from the environment and outgoing effects on the environment (i.e. percepts and actions respectively).
- Use case scenarios provide a holistic view of MAS functionality. Each activity in use cases should be annotated with the name of the associated functionality, thereby allowing the developer to perform consistency checking with the Functionality Descriptors.

The **Architectural Design Phase** uses outputs of the System Specification phase to identify agents, events, interactions among agents and shared data objects.

- Agent identification is carried out by assigning functionality to agents, in such a way as to promote strong intra-agent coherence and weak inter-agent coupling. High-level information about each agent (e.g. agent type, cardinality, incorporated functionality and communicating partners) should be captured in a textual *Agent Descriptor*.
- Events to be dealt with by each agent are identified from two sources: percepts from the environment and incoming messages from other agents.
- Interactions between agents are modelled using *Interaction Diagrams* and *Interaction Protocols* (Figure 3.30). Interaction Protocols are similar to Interaction Diagrams, except that they capture all potential interactions and elaborate the interactions in more detail.

- Data objects shared among agents need to be identified if they exist. A *System Overview Diagram* can be produced to tie together the identified agents, events and shared data objects (Figure 3.31).

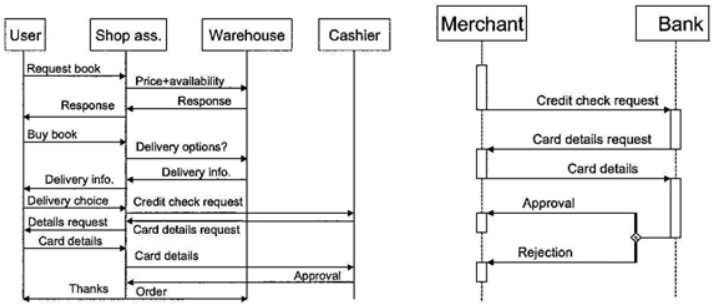


Figure 3.30 – PROMETHEUS Interaction Diagram (left) and Interaction Protocol (right) (Padgham and Winikoff 2002a)

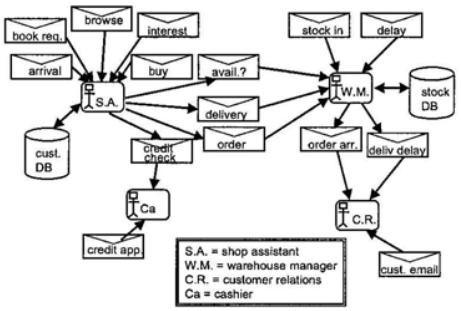


Figure 3.31 – PROMETHEUS System Overview Diagram (Padgham and Winikoff 2002a)

The **Detailed Design Phase** of PROMETHEUS is concerned with agent internal design, namely the design of agent capabilities, plans, events and data. Capabilities can be thought of as "modules" of functionalities handled by an agent. They may be derived from the functionalities identified in the System Specification phase. An *Agent Overview Diagram* can be produced to describe the top-level capabilities of an agent (Figure 3.32), while a *Capability Diagram* models each capability in terms of plans, events and data (Figure 3.33). Each plan can be described by a textual *Plan Descriptor*, which specifies the triggering event, plan steps and output events and messages of the plan. Each event is described by a textual *Event Descriptor* which documents the purpose of the event and any data carried by the event. Finally, a textual *Data Descriptor* is used to specify the fields and methods of any classes employed for data storage within the system.

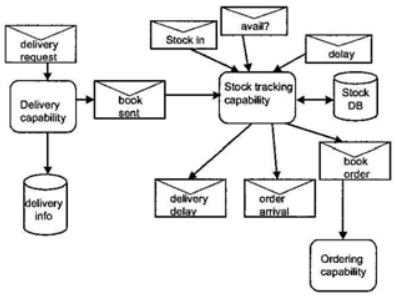


Figure 3.32 – PROMETHEUS Agent Overview Diagram (Padgham and Winikoff 2002a)

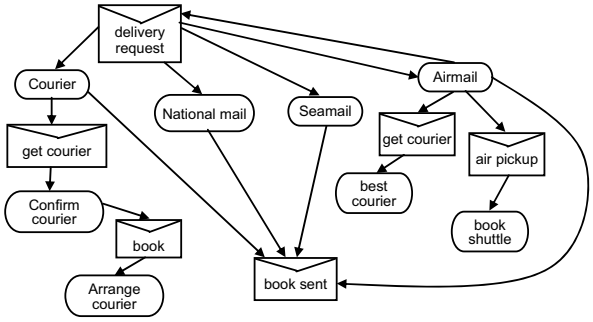


Figure 3.33 – PROMETHEUS Capability Diagram (Padgham and Winikoff 2002a)

3.2.11. PASSI

PASSI, "a Process for Agent Societies Specification and Implementation" (Burrafato and Cossentino 2002; Cossentino and Potts 2002; Cossentino 2002), offers a step-by-step requirement-to-code process for MAS development. It consists of twelve steps, grouped according to their outputs (Figure 3.34).

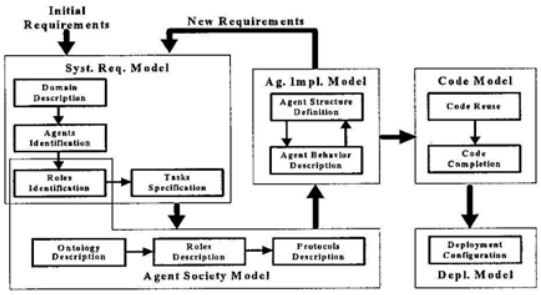


Figure 3.34 – Overview of PASSI (Burrafato and Cossentino 2002)

The first four steps produce a *System Requirement Model*. This model provides an anthropomorphic representation of the system requirements in terms of functionality and agency. It is constructed by:

- developing a hierarchical series of use case diagrams to describe the system functionality in the “Domain description” step;
- packaging these use cases into agent in the “Agent identification” step (Figure 3.35);
- exploring the roles of each agent by examining role-specific agent interaction scenarios in the “Role identification” step; and
- specifying tasks for each agent in the “Task specification” step. This step simply summarizes what an agent is capable of doing and ignores information about roles that the agent plays while carrying out the tasks.

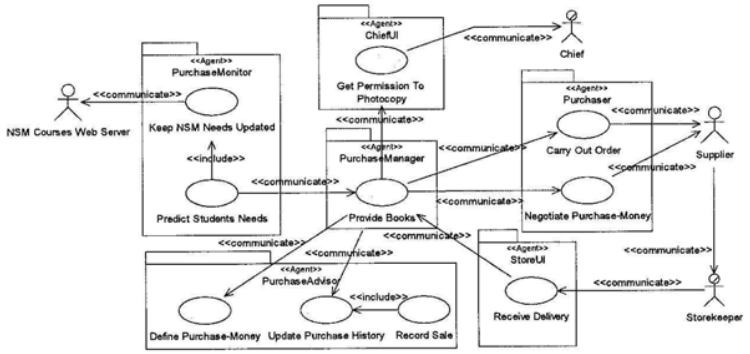


Figure 3.35 – PASSI Agent Identification Diagram (Burrafato and Cossentino 2002)

The subsequent three steps develop an *Agent Society Model* to specify the interactions and dependencies among agents.

- “Ontology description” step employs class diagrams and Object Constraint Language (OCL) to specify concepts and entities that define the domain’s knowledge (i.e. domain ontologies; Figure 3.36). It also determines which domain ontology governs each agent interaction protocol (Figure 3.37).
- “Role description” step describes the roles played by each agent, tasks performed by each role, changes between roles, and interactions and dependencies among roles (Figure 3.38).
- “Protocol description” step defines each agent interaction protocol. PASSI recommends standard FIPA protocols.

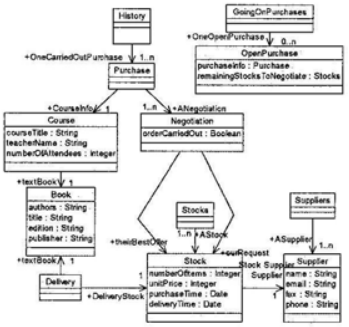


Figure 3.36 – PASSI Domain Ontology Diagram (Burrafato and Cossentino 2002)

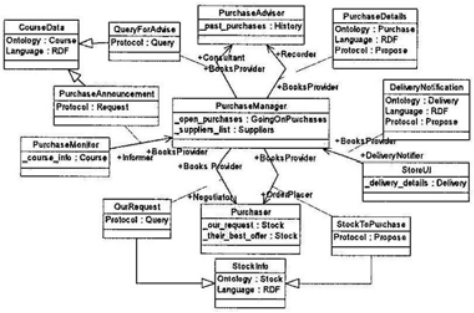


Figure 3.37 – PASSI Communication Ontology Diagram (Burrafato and Cossentino 2002)

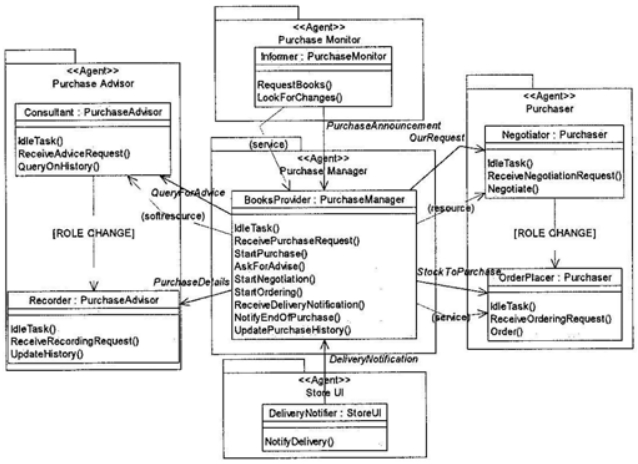


Figure 3.38 – PASSI Roles Description Diagram (Burrafato and Cossentino 2002)

The next two steps of PASSI produce an *Agent Implementation Model*. The model defines the target MAS in terms of architecture and behaviour.

- "Agent structure definition" step specifies the overall architecture of the system and the internal structure of each agent. The former shows the agents making up the system and their tasks (Figure 3.39), while the latter reveals the attributes and

methods of each agent, as well as the attributes and methods of each agent's task (Figure 3.40).

- "Agent behaviour definition" step specifies the flow of events between and within the agents as method invocations and message exchanges.

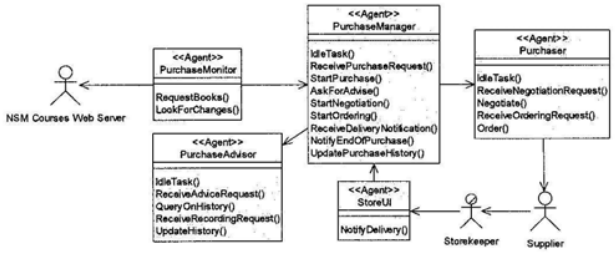


Figure 3.39 – PASSI MAS Structure Definition Diagram (Burrafato and Cossentino 2002)

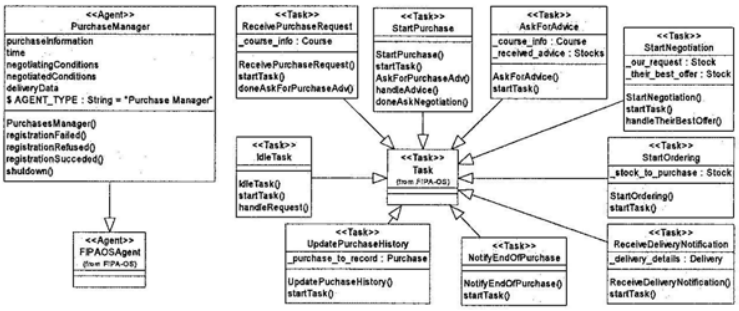


Figure 3.40 – PASSI Agent Structure Definition Diagram (Burrafato and Cossentino 2002)

The *Code Model* is subsequently constructed to specify the target MAS at the code level. It is developed by reusing the predefined patterns of agents and tasks (i.e. "Code reuse" step), and by generating the source code for the target system (i.e. "Code completion" step).

Lastly, the *Deployment Model* is built through step "Deployment configuration". It specifies the allocation of agents to processing units and any constraints on the migration and mobility of agents.

3.2.12. ADELFE

ADELFE (Bernon et al 2002a; Bernon et al 2002b; Institut de Recherche en Informatique de Toulouse n.d.) 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 known as "non cooperative situations".

The development process of ADELFE covers four phases. The **Requirement Phase** is concerned with:

- defining the target system through a set of keywords;
- clarifying the functionality of the system via use cases; and
- describing the system's environment in terms of actors (i.e. active and passive entities that interact with the system), system context (i.e. description of data flows between these active/passive entities and the system) and environment characteristics (i.e. whether the environment is dynamic, accessible, non-deterministic, and/or continuous).

Output of these steps is stored in Keyword Set Document, Requirement Set Document and Environment Definition Document respectively.

The **Analysis Phase** then identifies agents and applying the AMAS theory to the target application. It consists of five major steps.

1. "*Domain analysis and architecture study*" step analyses use cases in order to develop a Preliminary Class Diagram that shows entities composing the system (Figure 3.41).
2. "*Adequacy of AMAS theory*" step helps the developer to decide if the AMAS theory is indeed appropriate to the target system, since this kind of modelling is useless to certain applications.
3. "*Agent identification*" step determines which system entities are suitable to be implemented as agents, thereby producing a Refined Class Diagram (Figure 3.42). The consideration should take into account the entities' characteristics such as autonomy, proactiveness and negotiation capabilities.
4. "*Adequacy of the AMAS Theory at the local level*" step identifies which agents need to be adaptive. It then applies the AMAS theory to each of them by decomposing the agent into a system of sub-agents that interact flexibly with each other to provide the adaptive behaviour for the composing agent.

5. "*Study of interactions*" step develops a set of sequence diagrams and activity diagrams to describe the interactions among entities within the system.

Outputs of the above Analysis steps are stored in a Software Architecture Document.

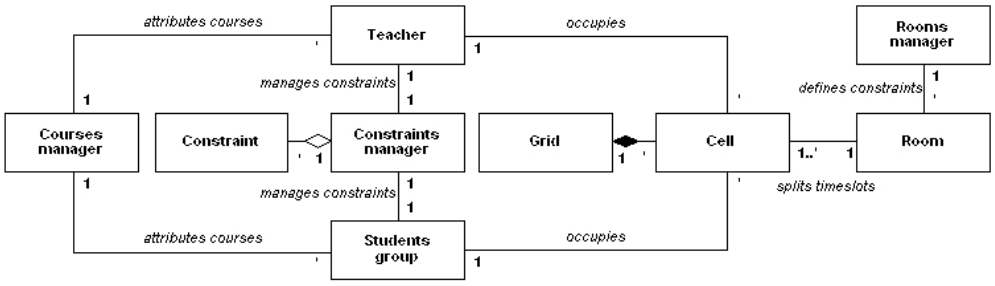


Figure 3.41 – ADELFE Preliminary Class Diagram (Institut de Recherche en Informatique de Toulouse n.d.)

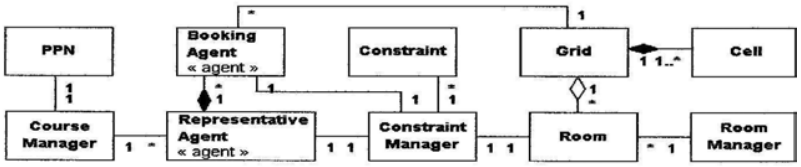


Figure 3.42 – ADELFE Refined Class Diagram (Institut de Recherche en Informatique de Toulouse n.d.)

The **Design Phase** of ADELFE deals with the detailed design of system architecture, agent internal structure and non-cooperative situations. Overall architecture of the system is modelled in terms of packages, classes (of agents and objects) and relationships between them. The architecture of each agent is designed as a composition of "representations" (i.e. the agent's beliefs about the environment and itself), "aptitudes" (i.e. the agent's capabilities on its knowledge), "skills" (i.e. capabilities that the agent brings to its collective), "interaction language" (i.e. protocols used by the agent) and "non-cooperative situations" (i.e. rules for dealing with unusual cooperative situations that the agent may face with) (Figure 3.43). ADEFLE describes each non-cooperative situation in terms of its name, conditions for its detection, the state in which the agent is when detecting the situation, and actions that the agent may perform to remove the situation (Figure 3.44). All outputs of the Detailed Design phase are stored in a Detailed Design Document.

The **Implementation phase** of ADELFE reuses activities from the conventional Rational Unified Process lifecycle.

BookingAgent	
Representations	
• constraints • bookState • partnershipState • brotherConstraints	• partners • recentlyMetAgents • RAFather • currentCell
Skills	
• moveInTheGrid • manageConstraints • manageBooking	• managePartnership • manageMessages
Aptitudes	
• bookARoom • cancelBooking • negotiateBooking • establishPartnership	• cancelPartnership • negotiatePartnership • SendMessage • InterpretMessage
Interaction Language	
• messageInteraction	• contactInteraction

Figure 3.43 – ADELFE Agent Internal Structure (Bernon et al. 2002a)

Booking Conflict
State
Any
Description
The BA is in a cell that is interesting to book but this cell is already booked
Conditions
The BA is in a cell **AND** this latter is already booked **AND** yet the cell would be suitable if not booked
Actions
IF the cost of the new booking is less than the older one **THEN** the BA books the cell **ELSE** the BA moves elsewhere

Figure 3.44 – ADELFE Non-Cooperative Situation (Bernon et al. 2002a)

3.2.13. COMOMAS

COMOMAS (Glaser 1996; Glaser 1997a; Glaser 1997b) 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.

The development process of COMOMAS consists of five steps (Figure 3.45).

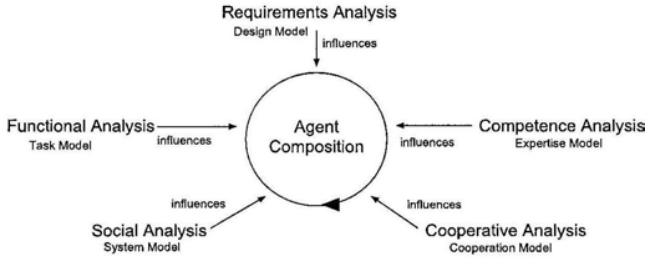


Figure 3.45 – COMOMAS steps and models (Glaser 1997a)

- "*Functional analysis*" step identifies the tasks that need to be solved by the target MAS. A task hierarchy, along with each task's details (i.e. input, output and control flow between tasks) is specified to form the Task Model.
- "*Requirement analysis*" step identifies non-functional design requirements of the system (e.g. efficiency and robustness), rankings of the requirements and interdependencies between the requirements. This information is captured in the Design Model.
- "*Competence analysis*" step identifies different types of knowledge that are required for agents to achieve the specified tasks. They include "task knowledge" (i.e. knowledge of previously accomplished tasks), "problem-solving knowledge" (e.g. strategies and methods for achieving particular tasks) and "reactive knowledge" (i.e. reactive responses to stimuli). Competence analysis produces the Expertise Model, which can be formalized using Conceptual Modelling Language (Figure 3.46).

```
EXPERTISE-MODEL Transport-application;
  domain-knowledge
  inference-knowledge
  TASK-KNOWLEDGE
    TASK plan-navigation-path ... END TASK [plan-navigation-path;]
    TASK localize-robot ... END TASK[localize-robot;] .
    TASK avoid-obstacles ... END TASK [avoid-obstacles;] .
    TASK place-robot ... END TASK [place-robot;] .
    TASK load-box ... END TASK [load-box;] .
    TASK unload-box ... END TASK [unload-box;] .
    TASK wander-around ... END TASK [wander-around]; .
    TASK move-towards ... END TASK [ move-towards;] .
  END TASK-KNOWLEDGE .
  psm-knowledge
  reactive-knowledge
END EXPERTISE-MODEL [Transport-application;] .
```

Figure 3.46 – COMOMAS Expertise Model (Glaser 1997a)

- "*Cooperative analysis*" step defines cooperation protocols, cooperation methods (e.g. data sharing or message passing), conflict situations and negotiation strategies for agents to resolve these conflicts. The results are captured in the Cooperation Model.
- "*Social analysis*" step identifies social competencies required by agents to act more smoothly during cooperation. In particular, it identifies agents' roles, agents' commitments to goals, and dependencies between agents in terms of goals and data. The results are stored in System Model.

Knowledge structures derived from the above five conceptual models are then used to compose each agent via an Agent Model (Figure 3.47). The developer can identify

agents by clustering the identified competencies while ensuring that the specified non-functional design requirements are satisfied.

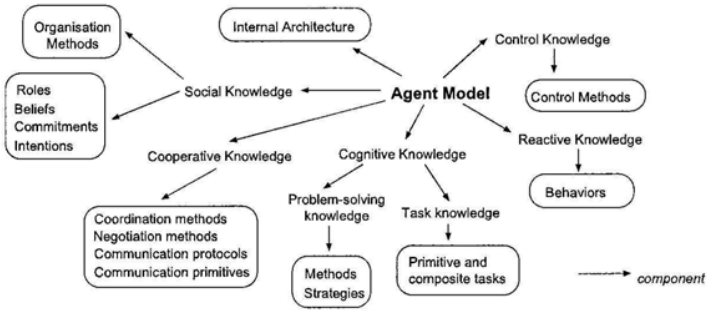


Figure 3.47 – COMOMAS Agent Model (Glaser 1997a)

3.2.14. MAS-CommonKADS

Like COMOMAS, MAS-CommonKADS (Iglesias et al. 1996; Iglesias et al. 1998) is also based on CommonKADS (Schreiber et al. 1994). However, the methodology also takes advantage of various OO techniques such as use case analysis and CRC cards.

The development process of MAS-CommonKADS covers the conceptualisation phase through to a detailed MAS design that can be directly implemented. The **Conceptualisation Phase** obtains a preliminary description of the problem domain via use cases and Message Sequence Charts.

The **Analysis Phase** then investigates the system requirements via five CommonKADS-based models. These models are developed in a cyclic risk-driven manner.

- *Agent Model* identifies agents from the analysis of use cases, problem statements, CRC cards and heuristics. Textual schemas can be used to describe each agent in terms of name, type, role, position, services, goals, skills, reasoning capabilities and permissions.
- *Task Model* specifies all the tasks that need to be fulfilled by the target system. It consists of a task hierarchy and a textual description of each task (e.g. name, inputs, outputs, task structure, frequency of application and required capabilities of performers).
- *Coordination Model* describes the dynamic relationships between agents. It is constructed via two activities: 1) defining the possible communication channels between agents by examining prototypical scenarios, and 2) analysing each inter-

agent conversation to determine its complexity and coordination protocols. Various OO diagrams can be used to represent this model, including Message Sequence Charts and Event Flow Diagrams for modelling communications between agents (Figure 3.48), High Level Message Sequence Charts for modelling coordination protocols, and State Transition Diagrams for modelling the processing of interactions (Figure 3.49).

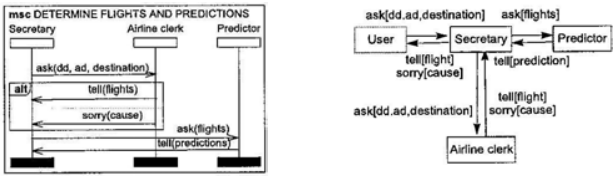


Figure 3.48 – MAS-CommonKADS Message Sequence Chart (left) and Event Flow Diagram (right) (Iglesias et al. 1998)

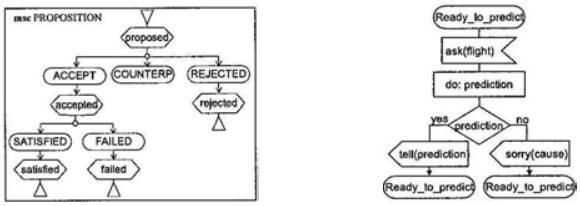


Figure 3.49 – MAS-CommonKADS High Level Message Sequence Chart (left) and State Transition Diagram (right) (Iglesias et al. 1998)

- *Expertise Model* defines the knowledge required by each agent to achieve its goals. This knowledge includes domain knowledge (i.e. domain ontology), inference knowledge (i.e. inferences to be made on domain knowledge), task knowledge (i.e. order or structure of inferences to achieve a task) and problem-solving knowledge (i.e. methods for carrying out each inference). These types of knowledge are captured respectively in Domain Knowledge Ontology (Figure 3.50), Inferences Diagrams (Figure 3.51), Task Knowledge Specification and Problem Solving Method Template.

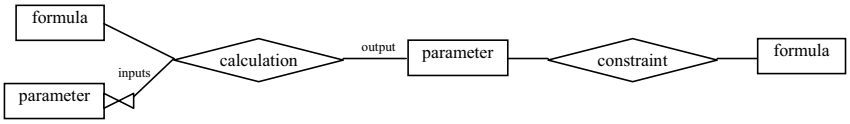


Figure 3.50 – MAS-CommonKADS Domain Knowledge Ontology (Schreiber et al. 1994)

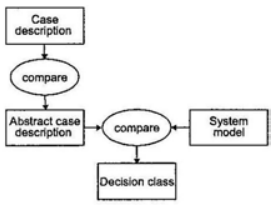


Figure 3.51 – MAS-CommonKADS Inferences Diagram (Iglesias et al. 1998)

- *Organisation Model* extends CommonKADS' Organisation Model to show static/structural relationships between agents (Figure 3.52).

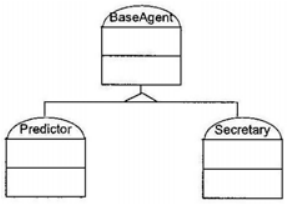


Figure 3.52 – MAS-CommonKADS Organisation Model (Iglesias et al. 1998)

The **Design Phase** of MAS-CommonKADS consists of three major activities.

- *Agent Design* determines the most suitable architecture for each agent. It subsequently maps the agent's logical modules onto the selected architecture.
- *Agent Network Design* defines the infrastructure of the target MAS, including network facilities (e.g. yellow-pages service), knowledge facilities (e.g. ontology servers) and coordination facilities (e.g. protocol servers).
- *Platform Design* selects the most suitable software and hardware for MAS implementation.

All design specifications are captured in a Design Model.

3.2.15. CASSIOPEIA

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

1. "*Definition of Domain-Dependent Roles*" step identifies all the roles that are required to achieve the collective task, by grouping together the elementary behaviour needed to fulfil the task. Agents are subsequently defined as sets of roles.

Each agent may assign a particular role to act as its "active" role at a given time while other roles are “idle”. For example, in the application of soccer playing robots, every “Player” agent can take on four roles "Shooter", "Placer", "Blocker" and "Defender", however only one of these roles can be active at a given time.

2. “*Definition of Relational Roles*” step specifies the organisational structure of MAS via relational roles. If an agent is dependent on another agent (due to dependencies between their domain-dependent roles), the former agent will play the relational role of an “influencing agent”, while the latter plays the relational role of an “influenced agent”. A Coupling Graph can be developed to reveal the dependencies among agents and their domain-dependent roles (Figure 3.53). This step also defines “influence signs” between agents (i.e. interaction messages) and “relational behaviour” performed by each agent to handle these influence signs.

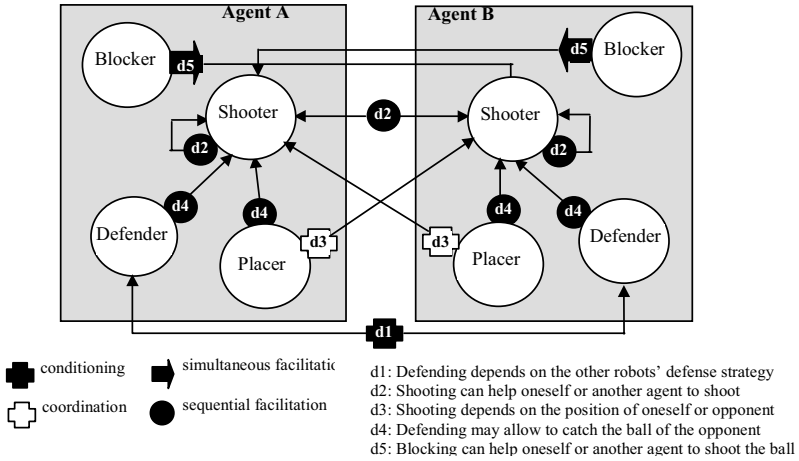


Figure 3.53 – CASSIOPEIA Coupling Graph (Collinot and Drogoul 1998)

3. “*Definition of Organisation Roles*” step addresses the dynamics of MAS organisation by assigning the organisational roles of “group initiator” and “group participant” to different agents. This step also specifies the “organisational behaviour” of each agent when playing its organisational role (i.e. group formation behaviour, commitment behaviour and group dissolution behaviour). The “influence signs” generated by this behaviour should also be defined (e.g. commitment signs and dissolution signs).

3.2.16. TROPOS

TROPOS (Castro et al. 2001; Castro et al. 2002; Bresciani et al. 2004) is based upon the *i** organisational modelling framework proposed by Yu (1995). It employs the concepts 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.

- The **Early Requirements Phase** investigates the intentions of system stakeholders via two models. *Strategic Dependency Model* shows the relevant stakeholders, represented as actors, and their inter-dependencies, including goal/soft-goal dependencies, task dependencies and resource dependencies (Figure 3.54). *Strategic Rationale Model* then elaborates how the stakeholders' dependencies can be fulfilled through means-end analysis (Figure 3.55).

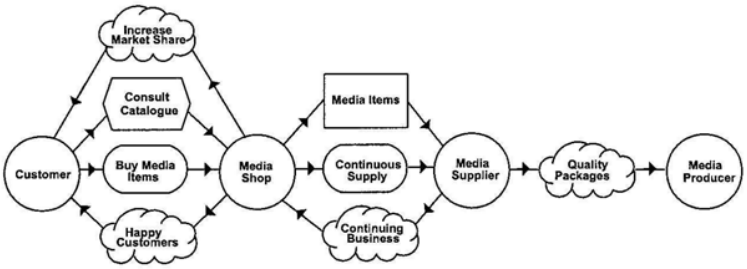


Figure 3.54 – TROPOS Strategic Dependency Model in Early Requirement phase (Castro et al. 2002)

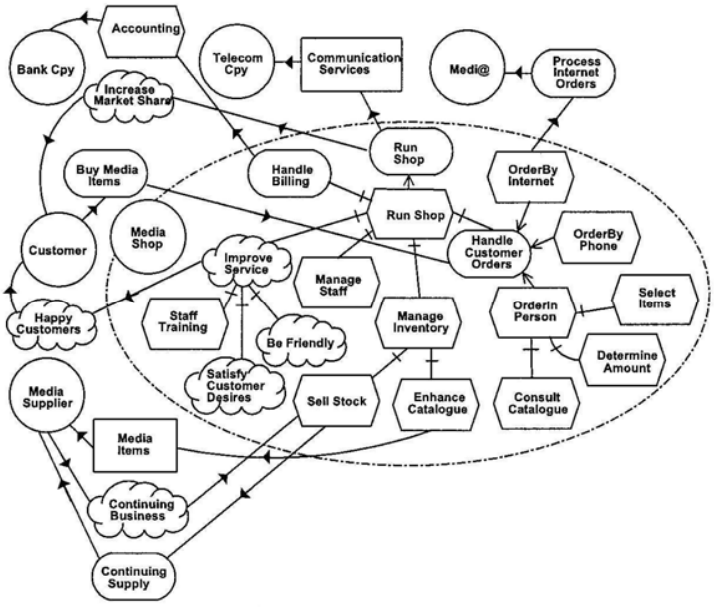


Figure 3.55 – TROPOS Strategic Rationale Model in Early Requirement phase (Castro et al. 2001)

- The **Late Requirements Phase** identifies functional and non-functional requirements of the target system by extending the Strategic Dependency Model and Strategic Rationale Model. Firstly, the target MAS is introduced as a new actor in the Strategic Dependency Model which contributes to the fulfilment of the stakeholders' goals (Figure 3.56). Means-end analysis is then performed on this system actor to produce a new Strategic Rationale Model (Figure 3.57). If necessary, the system actor can be decomposed into several sub-actors, resulting in a refined Strategic Dependency Model and Strategic Rationale Model.

- The **Architectural Design Phase** selects a suitable organisational structure for the target MAS by evaluating the quality attributes of the candidate organisational structures against the system's soft-goals. TROPOS offers a catalogue of organisational styles that can be selected and reused. The selected organisational pattern may result in changes in the Strategic Dependency Model and Strategic Rationale Model, with actors/sub-actors being added, removed or changed. The final set of system actors/sub-actors serves as candidates for agents.

- The **Detailed Design Phase** deals with agent interactions and agent internal design. Agent interactions are modelled using UML sequence diagrams and/or collaboration diagrams. Agent internal structure is defined in accordance with the BDI model. Specifically, "plans" are used as a mechanism for agents to achieve goals, perform tasks or respond to an event. Agent's beliefs are made up of resource entities owned by the agent. A *Class Diagram* and *Plan Diagrams* are developed for each agent to describe its internal structure and plans (Figure 3.58 and Figure 3.59 respectively).

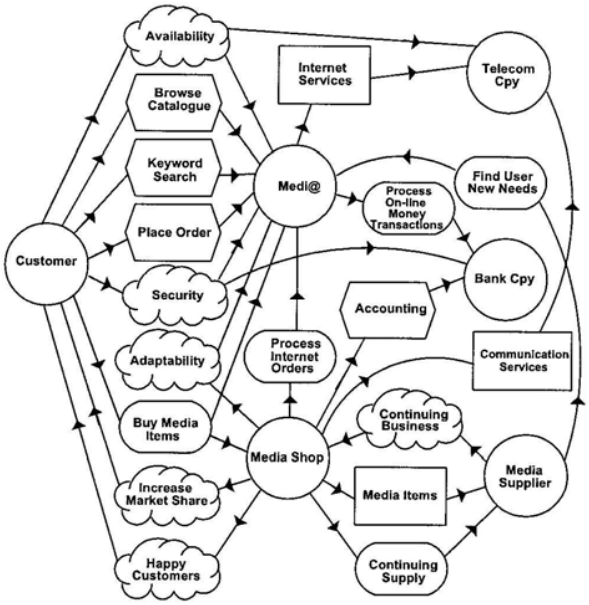


Figure 3.56 – TROPOS Strategic Dependency Model in Late Requirement phase (Castro et al. 2001)

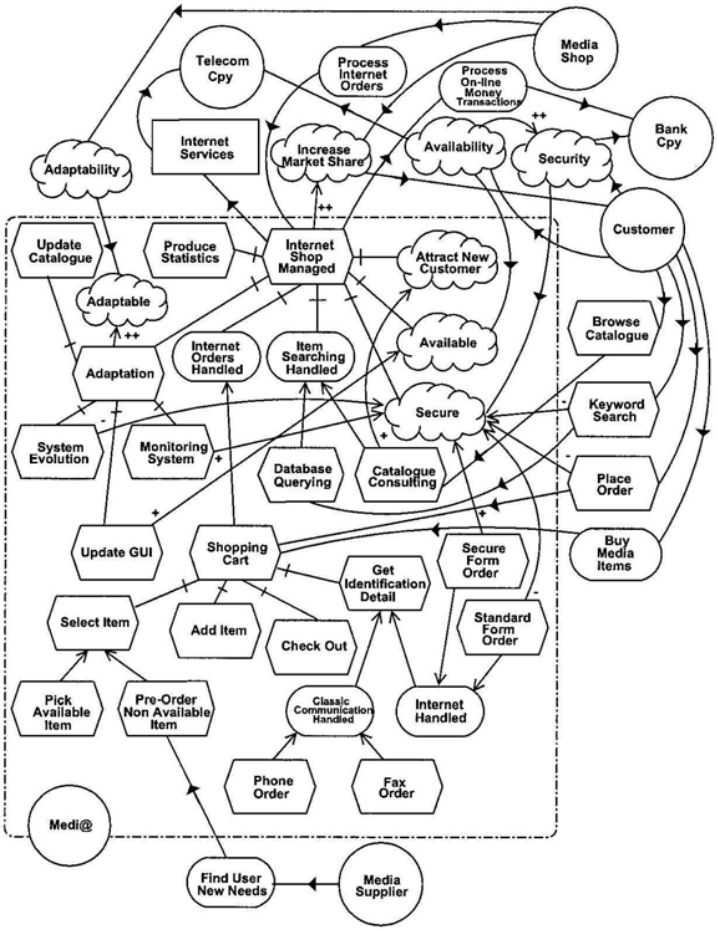


Figure 3.57 – TROPOS Strategic Rationale Model in Late Requirement phase (Castro et al. 2001)

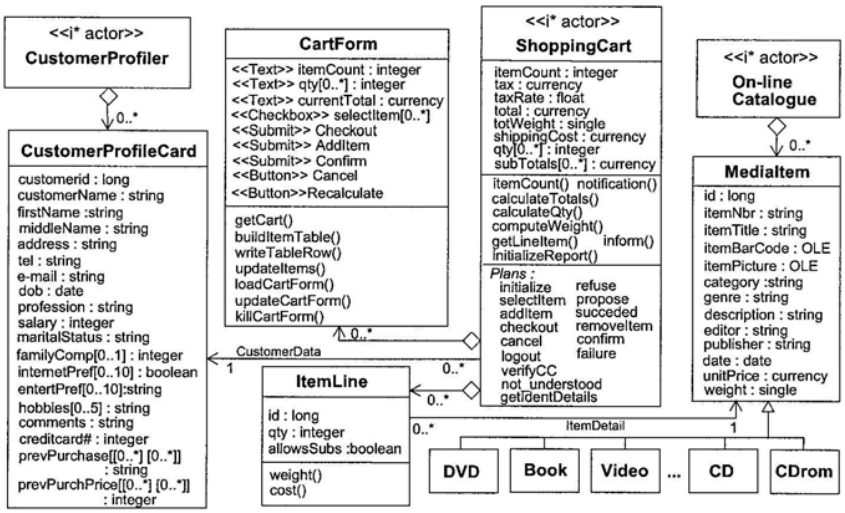


Figure 3.58 – TROPOS Agent Class Diagram (Castro et al. 2002)

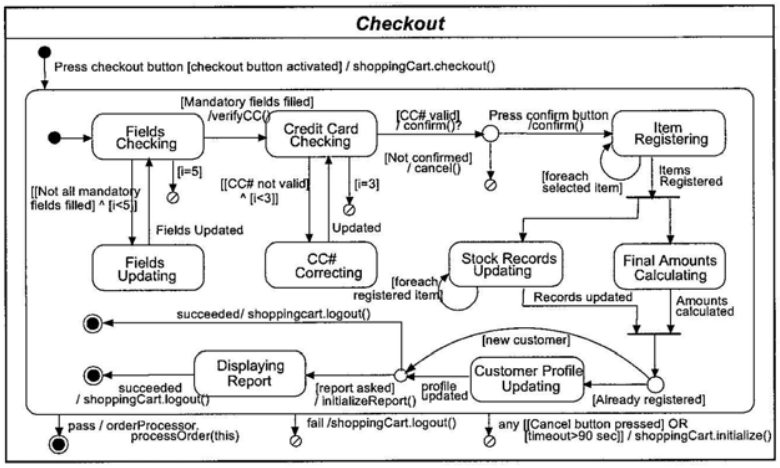


Figure 3.59 – TROPOS Plan Diagram (Castro et al. 2002)

3.3. GENERAL LIMITATIONS OF EXISTING MAS DEVELOPMENT METHODOLOGIES

Section 3.2 has described each of the sixteen AOSE methodologies for MAS development which were selected from the literature. Each methodology makes a valuable contribution to the area of AOSE, by offering a different set of steps, techniques and model definitions for the analysis and design of MAS. However, as will be revealed in this section, each existing AOSE methodology discloses a number of general limitations. Section 3.3.1 identifies the limitations relating to the general

analysis and design activities of MAS, while Section 3.3.2 exclusively discusses the limitations regarding the support for ontology-based MAS development. The latter directly causes the existing AOSE methodologies to not being able to fully realise the benefits of ontologies to interoperability, reusability, MAS development activities and MAS operation (cf. Section 2.3.2), as would be shown in Section 3.3.2. A more detailed evaluation of the existing AOSE methodologies was conducted at a later stage by the research and is presented in Chapter 5.

3.3.1. Limitations Regarding MAS Analysis and Design

MASE is weak in agent internal design. Although it provides guidelines on how an agent may be assembled[9], it does not address how the internal components used to assemble agents can be identified (if reused) or developed (if defined from scratch), such as goal, belief, plan or reflexive rule components. The methodology is also weak in MAS organisation design, since it does not investigate the authority relationships amongst agents or roles in the system.

MASSIVE improves on MASE by paying extensive attention to the design of MAS overall architecture and organisation. Nevertheless, it is very weak in agent internal design. Apart from the Role View which specifies roles played by each agent and the Architectural View which selects agent architecture, MASSIVE does not offer any steps, techniques and model definitions for the specification of agent's mental constructs such as beliefs, goals, capabilities, plans, reflexive rules and/or actions. In addition, MASSIVE does not provide any modelling notation for the representation of its model kinds except for Task View. The methodology merely presents guidelines on *what* to be modelled and not *how* these can be represented.

Like MASSIVE, **SODA** lacks support for the internal design of agents. It only addresses the specification of agents' high-level behaviour such as roles and tasks. The specification of agent internal architecture and mental constructs such as beliefs, goals

[9] That is, by either reusing a pre-defined agent architecture and internal components, or retrieving pre-defined internal components and assemble them into a user-defined architecture, or developing both internal components and agent architecture from scratch.

and plans is not covered. In addition, SODA does not present any notation for representing its model kinds.

Similar to MASSIVE and SODA, **GAIA** lacks support for agent internal design. Its Agent Model only specifies roles for each agent, without defining agent internal architecture and mental constructs (e.g. agent's beliefs, goals, plans and actions). GAIA's support for agent interaction design is also limited. The Acquaintance Model simply identifies the communication pathways between agents and the Interaction Model merely shows the "institutionalised patterns" of interactions. No detailed design of agent communication is given (e.g. the potential sequences in which messages are exchanged or the contents of exchanged messages).

MESSAGE is weak in the usability of its process steps, particularly in the Design phase where many steps are not supported by comprehensive techniques. For example, the identification of agents (from roles) is to be based merely on the developer's intuition and experience. The need for elaborating interaction protocols is mentioned, but no techniques are provided for the specification of message sequences and contents.

INGENIAS is also weak in usability due to the complexity of its model definitions and development process. The Organisation Model, for example, endeavours to capture a large number of concepts within its content, including "agent", "group", "workflow", "task", "role", "goal", "application" and "resource". Using an unfamiliar notation, the clarity and ease of understanding of the developed model is degraded even further. The development process of INGENIAS is not easy to follow, since the transition between the construction of different models within each workflow is not clear.

BDIM is weak in its support for agent interaction modelling. The methodology does not provide any techniques for the specification of interaction protocols. It also does not offer any modelling notation for the representation of agent interactions. The modelling of MAS organisation in terms of roles'/agents' acquaintances and authority relationships is also overlooked.

HLIM is weak in terms of its modelling capability. The modelling notation used by HLIM for many of its model kinds is found to be inefficient. For example, the use of

simple textual tables in Internal Agent Model and conversation to specify agent beliefs and interaction protocols is not adequately powerful. These tables cannot express information such as relationships between beliefs or alternative, concurrent or conditional exchanges of messages.

MEI focuses merely on the discovery of agents and agent internal design, without paying attention to the modelling of MAS organisation. The modelling capability of MEI is also weak, since no explicit, formal modelling notation is recommended for the representation of its model kinds (except for agent plans and coordination protocols which are suggested to be represented by state diagrams).

PROMETHEUS is limited in its support for agent internal design. It exclusively targets plan-based, BDI-like agents via the specification of plans, without addressing the internal design for other styles of agents such as purely reflexive agents or hybrid agents. The methodology is also weak in MAS organisation design, since it does not investigate the authority relationships amongst agents or roles in the system.

PASSI is weak in its support for agent internal behavioural design. The methodology suggests defining agent behaviour via event flows and method invocations, which is more suited to object behaviour than agent behaviour. Planning agents, for example, require the specification of plans, while reflexive agents require the modelling of reactive policies (Wooldridge and Jennings 1994; Vidal et al. 2001).

ADELFE offers exclusive support to the development of adaptive MASs. While this is a strength, it is also a weakness because if a MAS does not need to be adaptive, ADELFE may be inappropriate or inapplicable. For example, the internal model of an agent as designed by ADELFE[10] is not applicable to all types of agents, such as purely reflexive agents[11] or purely planning agents[12]. The methodology is also weak in agent interaction design, since even though it mentions this activity, there are no techniques to support the specification of interaction protocols.

---

[10] ADELFE models each agent as being made up of "representations", "aptitudes", "skills", "interaction languages" and "non-cooperative situations".
[11] Purely reactive agents do not need "representations" and "aptitudes".
[12] ADELFE agents do not have "plans" in their internal structure.

COMOMAS is weak in its support for agent interaction design. Although a Cooperation Model is developed, no detailed techniques are provided on how message contents are specified.

MAS-CommonKADS fails to offer adequate support for the development of Agent Model. It is unclear how the developer can determine various properties for each agent as required by the Agent Model, such as role, position, offered services, goals, skills, general capabilities norms, preferences and permissions. If these properties are to be derived from other model kinds of MAS-CommonKADS, the interconnections between the model kinds are not highlighted by the methodology.

CASSIOPEIA does not provide any support for agent internal design. The methodology also does not specify any explicit, formal set of model kinds, except for the Coupling Graph which captures agents' roles and agents' dependencies.

Finally, **TROPOS** lacks structured and detailed techniques for its Detailed Design phase. In particular, it is unclear how agent interaction protocols can be derived and how agent plans can be constructed.

3.3.2. Limitations Regarding Support for Ontology-Based MAS Development

As discussed in Section 2.3.2, ontologies are widely acknowledged in the literature for their significant benefits to:

- interoperability;
- reusability;
- MAS development activities, namely system analysis and agent knowledge modelling; and
- MAS operation, specifically communication and agent reasoning.

Nevertheless, 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 ontologies in their MAS development process, nor integrate ontologies into their MAS model definitions. Of the 16 investigated AOSE methodologies, only four were found to show some consideration for ontologies: 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 (c.f. Section 2.3.2.3). 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 exchanged messages to be formulated in term of shared ontological concepts (c.f. Section 2.3.2.4). 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 (which operationalises the agent reasoning at run-time; c.f. Section 2.3.2.4). 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 (c.f. Section 2.3.2.1). As a result, MAS-CommonKADS' support for *reusability* is also limited, since the methodology cannot show how legacy (heterogeneous) system components can be reused (c.f. Section 2.3.2.2).

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* (c.f. Section 2.3.2.3). However, unlike MAS-CommonKADS, 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 (c.f. Section 2.3.2.4). 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 (c.f. Section 2.3.2.4). 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 (c.f. Section 2.3.2.1). As a result, MESSAGE cannot illustrate the role of ontologies in promoting the *reuse* of legacy (heterogeneous) system components (c.f. Section 2.3.2.2).

The extended version of **MASE** (DiLeo et al. 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 (c.f. Section 2.3.2.3). 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 (c.f. Section 2.3.2.1). MASE' support for reusability is thus enhanced, since it allows the legacy (heterogeneous) system components to be reused (c.f. Section 2.3.2.2). 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 (c.f. Section 2.3.2.4).

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 (c.f. Section 2.3.2.3). 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 (c.f. Section 2.3.2.4). 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 (c.f. Section 2.3.2.4). PASSI also does not exploit ontologies to support *interoperability*, as it overlooks the existence of heterogeneous system components in a MAS and the need for ontological mappings between them (c.f. Section 2.3.2.1). As a result, PASSI's support for *reusability* is limited, because it cannot show how (heterogeneous) legacy components can be reused (c.f. Section 2.3.2.2).

In summary, even though the above four AOSE methodologies exercise the use of ontologies in their MAS development process and product, they do not comprehensively acknowledge and implement all of the diverse roles of ontologies in MASs, namely those identified in Section 2.3.2. More specifically, although all four methodologies exploit ontologies to facilitate their *system analysis* and *agent knowledge modelling* activities, none of them – by itself – can illustrate the use of ontologies to support *interoperability*, *reusability*, *agent communication* and *agent reasoning* altogether. This limitation prompts for the development of a methodology which acknowledges all of the significant benefits of ontologies to MAS, and which integrates the use of ontology into every applicable AOSE step and model definition to realise these benefits.

3.4. SUMMARY

This chapter has provided a review of the sixteen 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. The next chapter, Chapter 4, puts forward the objective of this PhD research in response to the limitations of the existing AOSE methodologies.

CHAPTER 4
RESEARCH DESIGN

4.1. INTRODUCTION

This chapter provides a detailed description of the design of this research. It firstly specifies the research's objective in Section 4.2, thereafter presenting an outline of the research activities in Section 4.3. Sections 4.4, 4.5 and 4.6 then describe each research activity in terms of its aims, associated steps and research methods.

4.2. RESEARCH OBJECTIVE

As seen in Chapter 3, a number of methodologies have been published for the analysis and design of MAS. Each methodology offers a valuable contribution to the field of AOSE via its unique development process, techniques and model definitions. Nevertheless, from a preliminary evaluation as shown in Section 3.3, each methodology exposes a number of general limitations. One particular limitation is the weak *support for ontology-based MAS development*. Despite of the important benefits that ontology can offer to interoperability, reusability, MAS development activities and MAS operation (cf. Section 2.3.2), most methodologies neither mention the use of ontologies in their MAS development process, nor integrate ontologies in their MAS model definitions. Although four methodologies show some consideration for ontology, they do not investigate all of the diverse potential uses of ontology and implement them (cf. Section 3.3.2). In addition, each methodology was also found to provide limited support for at least one of the following areas of MAS development: agent internal design, agent interaction design and MAS organisation design (cf. Section 3.3.1).

Acknowledging the above limitations of the existing AOSE methodologies, this research sets its objective as follows.

"*Contribute to the field of AOSE by proposing a **comprehensive ontology-based AOSE methodology** for the analysis and design of MASs. This methodology aims to provide support for ontology-based MAS development and various other AOSE methodological requirements which are important to an AOSE methodology but which may not be well-supported by the existing methodologies. The proposed AOSE methodology is named "**MOBMAS**", which stands for "**M**ethodology for **O**ntology-**B**ased **M**ulti-**A**gent **S**ystems".*

MOBMAS does *not* aim to support the process of ontology engineering itself. This process is assumed to be a separate analysis effort conducted by domain experts, ontology engineers or the MAS developer himself. Ontologies can be developed using specialised ontology-engineering methodologies such as IDEF5 (Knowledge Based Systems Inc 1994), Grüninger and Fox (1995) and METHONTOLOGY (Fernandez et al. 1997). MOBMAS focuses instead on:

- *the use of ontologies in the MAS analysis and design process*; and
- *the inclusion of ontologies in MAS model definitions*.

The MAS resulted from using MOBMAS are called *ontology-based MASs*, since their design specification explicitly includes ontologies, and ontologies can be used by agents at run-time to facilitate the operation of MAS (Yuan 1999; Guarino 1998).

The scope of MOBMAS is limited to the **Analysis** and **Design** phases of the system development lifecycle (SDLC), which traditionally contains four phases, Requirements Engineering, Analysis, Design and Implementation (Eliason 1990; Dennis and Wixom 2003). MOBMAS process starts from a set of system functionality requirements (which is identified by a separate Requirements Engineering effort *not* included as part of MOBMAS) and ends with a design of a MAS system. Even though the Implementation phase is not covered, MOBMAS addresses various important implementation-related issues such as deployment configuration and selection of agent architecture.

It should be noted that, acknowledging the strengths of the existing AOSE methodologies, this research did not develop an AOSE methodology totally from

scratch, but reused and enhanced the work of the existing methods where appropriate[13]. Nevertheless, the research did *not* aim to simply merge existing AOSE methodologies per-se into one comprehensive methodology. Rather, it arrived at a comprehensive AOSE methodology by:

- making a pioneering effort in identifying the methodological requirements of a "standard" AOSE development methodology, by consulting the existing MAS methodologies as well as the opinions of practitioners and researchers in the field; and
- developing a comprehensive AOSE methodology that supports the identified requirements, by combining the strengths of the existing methods, as well as proposing new techniques and model definitions where the existing support is weak.

4.3. RESEARCH ACTIVITIES

The work of this research can be classified as *design science* – one of the two core paradigms that characterise much of the research in the Information Systems discipline: "behavioural science" and "design science" (Hevner et al. 2004; March and Smith 1995). The behavioural science research paradigm seeks to develop and verify theories that explain or predict human/organizational behaviour surrounding the development and use of information systems. Meanwhile, the design science paradigm – where this research fits – seeks to create innovative artifacts through which the development and use of information can be effectively and efficiently accomplished. In general, the artifacts to be produced by a design science research can be of four types: *methods* (i.e. sets of steps, guidelines or algorithms), *models* (i.e. abstractions and representations), *constructs* (vocabularies and symbols) and *implementation* (i.e. prototype systems) (March and Smith 1995; Hevner et al. 2004). This PhD research aims to create two of these artifacts: method and models. The method to be developed is the MOBMAS methodology, while the generated models are the set of models accompanying the MOBMAS methodology (i.e. those produced by MOBMAS steps).

As identified by March and Smith (1995), a typical design science research should comprise of two basic processes: *build* and *evaluate*. Build refers to the construction of

[13] That is, where the existing techniques and/or model definitions are evaluated to be good, with respect to a particular methodological requirement.

the artifacts – in this case, the MOBMAS methodology and models. The evaluation process refers to the use of appropriate evaluation methods to assess the artifacts' performance. In compliance with this principle, this PhD research has been designed to include activities that fulfil these two required processes. Specifically, it consists of three research activities: the first two activities carry out the *build* process, while the third activity executes the *evaluation* process.

1. Research Activity 1 – Identify the methodological requirements of MOBMAS

As defined by Henderson-Sellers et al. (1998), a software engineering methodology is one that provides the following elements:

- a software engineering ***process*** to conduct the system development;
- ***techniques*** to assist the process; and
- ***definition of work products***.

The "process" element itself should contain ***activities*** and ***steps***[14] (Henderson-Sellers et al. 1998; Firesmith and Henderson-Sellers 2002). "Activities" are large-scale descriptions of what needs to be done, such as "requirements engineering" activity, "design" activity, "implementation" activity and "testing" activity. If the process is a waterfall process, these activities might be referred to as "phases". "Steps", on the other hand, are smaller-scale "jobs to be done" associated with each activity in the process. Steps are then linked with techniques, which provide the way to carry out the steps, i.e. the "how" (Figure 4.1).

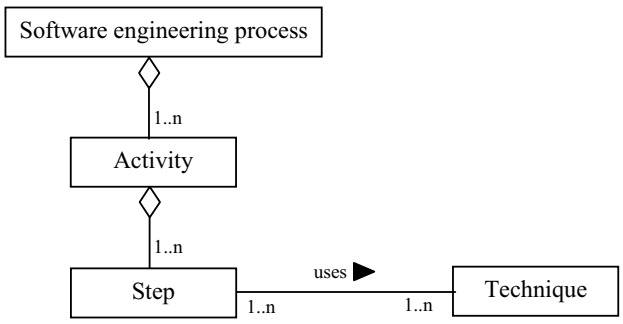


Figure 4.1 – Associations between "process", "activity", "step" and "technique" (represented in UML)

[14] Henderson-Sellers et al. (1998) and Firesmith and Henderson-Sellers (2002) use the term "task" instead of "step". However, since the term "task" will be used frequently in Chapter 6 – "*Documentation of MOBMAS*" to refer to software functionality, the term "step" is used as a substitute.

Regarding the definition of work products, since MOBMAS covers Analysis and Design activities, its work products should consist of ***models of MAS analysis and design***.

As a result, the required methodological elements of MOBMAS are (Figure 4.2):

- a software engineering ***process*** that contain ***activities*** and associated ***steps*** to conduct the system development;
- ***techniques*** to assist the process steps; and
- ***definition of model kinds***. Note that the term "model kind" is used rather than "models" because the methodology only provides a definition of the specific classes of models (Standards Australia 2004). The models themselves refer to actual deliverables produced by the developer during the development process.

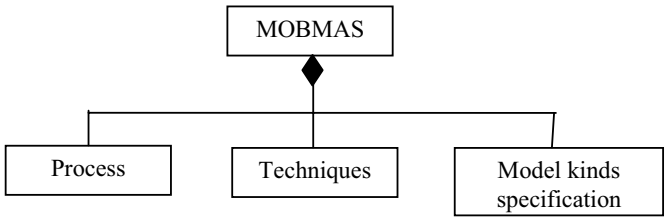


Figure 4.2 – Components of MOBMAS (represented in UML)

In order to define the above elements for MOBMAS, it is necessary to determine the **features**, **steps** and **modelling concepts** that are desirable to be supported by MOBMAS *process*, *techniques* and *model kinds*. These desirable features, steps and modelling concepts are referred to as "*methodological requirements*" of MOBMAS. Research Activity 1 was concerned with identifying these methodological requirements. It should be noted that, the *steps* that are specified as MOBMAS' methodological requirements are *not* meant to be the "exact" steps that MOBMAS must provide. MOBMAS can define its steps differently from these desirable steps. However, the actual MOBMAS' steps are required to correspond to, or cover, the desirable steps.

In Research Activity 1, apart from identifying the required features, steps and modelling concepts for MOBMAS, it was also necessary to identify the desirable "*ontology-related steps*" from amongst these required steps, so as to allow MOBMAS to support ontology-based MAS development. These ontology-related steps should enable

MOBMAS to offer all of the widely-recognised benefits of ontology to MAS development and MAS operation as listed in Section 2.3.2.

2. Research Activity 2 – Develop MOBMAS

This research activity defined the **development process**, **techniques** and **model kinds** for MOBMAS so as to support the desirable features, steps and modelling concepts that were identified by Research Activity 1. MOBMAS process, techniques and model kinds were developed by *reusing* and *enhancing* the techniques and model definitions offered by the existing AOSE methodologies where appropriate, and *developing new* techniques and model definitions for MOBMAS where necessary.

3. Research Activity 3 – Evaluate and refine MOBMAS

MOBMAS was evaluated and progressively refined by collecting *expert reviews*, having external developers *use the methodology on a test application*, and performing a *feature analysis* on the methodology. The expert reviews gathered professional evaluation of MOBMAS based on the experts' non-empirical investigation of the methodology. The use of MOBMAS on a test application then sought external developers' evaluation of MOBMAS based on their empirical usage of the methodology. Lastly, the feature analysis was conducted to verify MOBMAS' ability to achieve its objective (which is, to provide support for ontology-based MAS development and the other important AOSE methodological requirements that were identified in Research Activity 1[15]; cf. Section 4.2), to compare MOBMAS with the existing AOSE methodologies, and to clarify MOBMAS' ontology-related capabilities.

[15] Through the justification of MOBMAS' support for its methodological requirements, this research was able to justify that MOBMAS' actual *steps* and *modelling concepts* in fact correspond to, or cover, the desirable steps and modelling concepts which were specified as part of the methodological requirements.

4.4. RESEARCH ACTIVITY 1 – IDENTIFY METHODOLOGICAL REQUIREMENTS OF MOBMAS

This section and the subsequent two sections (Sections 4.5 and 4.6) elaborate on the design of each research activity listed in Section 4.3. Research Activity 1 – "*Identify the methodological requirements of MOBMAS*" – is described in this section.

Research Activity 1 was conducted in four steps.

4.4.1. Step 1 – Identify "Potential" Requirements of MOBMAS

The objective of this step was to determine a list of **features**, **steps** and **modelling concepts** that were *potentially* desirable to the system development process, techniques and model kinds of MOBMAS. These features, steps and modelling concepts were subsequently validated into "*actual*" requirements of MOBMAS during Steps 2 and 3 of Research Activity 1 (cf. Sections 4.4.2 and 4.4.3).

In order to identify the potentially desirable **features** for MOBMAS, this research investigated a number of *existing evaluation frameworks* for assessing:

- AOSE methodologies; and
- conventional system development methodologies, including OO methodologies.

The evaluation frameworks for AOSE methodologies contain evaluation criteria that relate to important *agent-oriented* and *MAS-specific* features, while the evaluation frameworks for conventional methodologies help to reveal important *generic system engineering features*, which may have been overlooked by AOSE evaluation frameworks.

The potentially desirable **steps** and **modelling concepts** of MOBMAS were identified by investigating the *existing AOSE methodologies* (which are described in Chapter 3).

Each existing methodology offers a different set of steps for the MAS development process and a different set of model kinds for a variety of AOSE modelling concepts.

4.4.2. Step 2 – Conduct a Survey on Practitioners and Researchers in the Field of AOSE

The survey was performed to achieve the following two objectives.

- **To validate the identified potential requirements of MOBMAS**: The survey asked the respondents to *rate* each feature, step and modelling concept identified in Step 1 in terms of how important the feature, step or concept is to a "standard" AOSE methodology (on a scale of "Very high", "High", "Medium", "Low" and "Very low"). The survey respondents were also asked to *rank order* these features, steps and modelling concepts in a decreasing order of importance. The "rating of importance" and "order rank"[16] of each feature, step or concept would later be combined with the outputs of Step 3 in order to determine the "*actual"* requirements for MOBMAS (Figure 4.3).

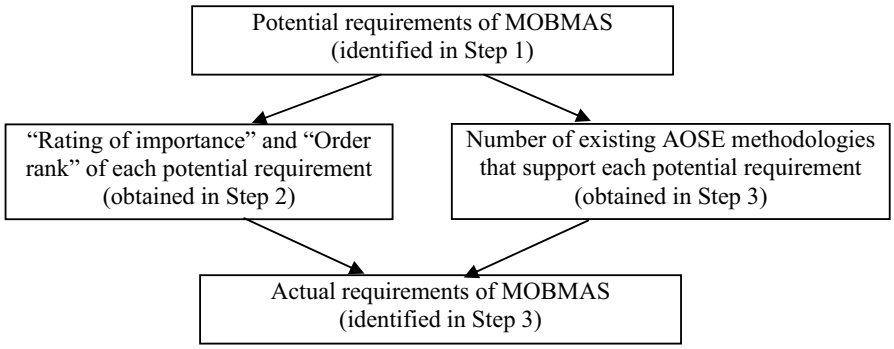


Figure 4.3 – Determination of "actual" requirements of MOBMAS

- **To obtain professional recommendations on various issues that are useful to the development of MOBMAS**: A segment of the survey collected professional suggestions on various issues that pertain to the construction of a "standard" AOSE methodology, such as suggestions on the desirable MAS development lifecycle,

[16] Note that both "rating of importance" and "order rank" were collected for the potential requirements because if only one of these statistics was collected, it would not reflect a comprehensive indication of the requirements' importance. For example, a set of steps may be given the same "rating of importance" but distinct order ranks (i.e. they are not truly equally important); or, a top-ranked step may have an overall "Low" rating of importance.

desirable agent identification approach and desirable level of commitment to an agent architecture by an AOSE methodology.

4.4.3. Step 3 – Perform a Feature Analysis on Existing AOSE Methodologies

This step was performed after the completion of the survey in order to achieve the following two objectives.

- **To further validate the identified potential requirements of MOBMAS and determine the "actual" requirements of MOBMAS**: The feature analysis investigated all sixteen existing AOSE methodologies (described in Chapter 3) to determine *how many* methodologies offer support for each feature, step and modelling concept identified in Step 1. This finding was then combined with the "ratings of importance" and "order ranks" obtained from the survey in Step 2 in order to determine the "*actual*" requirements of MOBMAS (Figure 4.3). Specifically, a potential requirement was qualified to be an actual requirement if:
  - it was supported by a majority of the existing AOSE methodologies (i.e. 9 or more out of 16); *OR*
  - it was given a High to Very High "rating of importance" in the survey; OR
  - it was given a Medium "rating of importance" in the survey *AND* its "order rank" is *not* the least important with respect to other requirements within the same category.

  All other potential requirements were excluded from list of actual requirements of MOBMAS.

  It should be noted that, all the *steps* that were specified as MOBMAS' "actual" requirements were *not* meant to be the "exact" steps that MOBMAS must provide. MOBMAS can define its steps differently from these desirable steps. However, the actual MOBMAS' steps were required to correspond to, or cover, the desirable steps.

- **To identify and evaluate the techniques and model definitions provided by each existing AOSE methodology**: This identification and evaluation helped the research to:
  - identify a pool of existing techniques and model definitions that may be *reused* or *enhanced* by MOBMAS to support its required features, steps and modelling concepts; and
  - identify which features, steps and modelling concepts of MOBMAS need to be supported by *new* techniques and model definitions (i.e. those that are currently *not* efficiently supported by the existing AOSE methodologies, either in terms of the small number of supporting methodologies, or the insufficiency of the available techniques and model definitions).

  This information was used as inputs to the development of MOBMAS in Research Activity 2.

4.4.4. Step 4 – Identify Ontology-Related Steps From Amongst the Required MOBMAS' Steps

After the methodological requirements of MOBMAS were determined in Step 3, Step 4 was performed to identify which of the required *steps* should be "*ontology-related*" (i.e. which steps should use ontologies in their techniques and/or integrate ontologies into their model definitions), so as to enable MOBMAS to realise all of the widely-acknowledged benefits of ontologies to MASs, namely those previously identified in Section 2.3.2:

- support for interoperability;
- enhancement of reusability;
- support for MAS development activities, namely system analysis and agent knowledge modelling; and
- support for MAS operation, specifically communication and agent reasoning.

Each of these benefits was investigated to identify the desirable ontology-related steps. In particular, if a benefit was found to be realised through the use of ontology in an AOSE step(s), this step(s) was flagged as a desirable ontology-related step. By doing so, this research was able to ensure that MOBMAS, with its support for these ontology-related steps, can realise all of the diverse benefits of ontology to MASs.

4.5. RESEARCH ACTIVITY 2 – DEVELOP MOBMAS

Given the methodological requirements of MOBMAS as identified by Research Activity 1 (cf. Section 4.4), Research Activity 2 was carried out to develop the MOBMAS methodology. This activity constructed MOBMAS by defining the system development **process**, **techniques** and **model kinds** to support the required features, steps and modelling concepts Note that MOBMAS' actual steps and modelling concepts were not required to be identical to those identified by Research Activity 1. However, the former was required to correspond to, or cover, the latter.

The process, techniques and model kinds of MOBMAS were developed by:

- *reusing* and *enhancing* the existing techniques and model definitions offered by the available AOSE methodologies where appropriate; and
- *developing new* techniques and model definitions where necessary.

MOBMAS considered ***reusing*** an existing technique or model definition if that technique or model definition was given a positive or high assessment[17] by the feature analysis in Step 3 of Research Activity 1 regarding its support for a particular requirement.

MOBMAS ***enhanced*** the existing work by refining, adapting, elaborating, extending and/or integrating various existing techniques and modelling notation to improve their usability and applicability. With regard to integration, the integration of techniques or model definitions may result in:

- a synthesised, internally consistent technique or model kind; or
- a set of separate techniques or model kinds, each of which best suits a different situation. In this case, MOBMAS provides guidelines on how to select the most appropriate technique or model kind to use in a particular situation.

[17] The type of assessment depends on whether the corresponding evaluation criterion is a yes/no question or a high/medium/low rating question.

Again, findings of the feature analysis in Step 3 of Research Activity 1 served as a useful input. Evaluation of the existing techniques and model definitions helped to identify those that could be enhanced. Another valuable resource was the outputs of the survey in Step 2 of Research Activity 1. Recommendations given by survey respondents on the various issues relating to AOSE methodology construction helped to provide ideas for enhancement.

The need for ***new*** techniques and model definitions for MOBMAS arose when there was a lack of existing techniques or model definitions for supporting a particular requirement, and/or when the existing techniques or model definitions were low in usability. New techniques and model definitions were developed for MOBMAS by consulting the work in the respective literature (e.g. literature on ontology, agent planning and agent coordination mechanisms). In addition, outputs of the survey in Step 2 of Research Activity 1 were also used. Ideas were obtained from the open-ended recommendations given by survey respondents on issues relating to AOSE methodology construction. The feature analysis in Step 3 of Research Activity 1 also helped to identify those features, steps and modelling concepts that needed to be better supported by new techniques and model definitions.

During Research Activity 2, particular attention was given to the "ontology-related steps" identified in Step 4 of Research Activity 1 (cf. Section 4.4.4). These steps required the use of ontologies in their techniques and/or the inclusion of ontologies in their model definitions. In addition, since the existing AOSE methodologies either do not provide support for ontology-based MAS development, or are insufficient in their support, MOBMAS needed to make a lot of enhancement to the existing techniques and model definitions, as well as develop many new techniques and model definitions, in order to support the ontology-related steps.

4.6. RESEARCH ACTIVITY 3 – EVALUATE AND REFINE MOBMAS

After MOBMAS was constructed by Research Activity 2 (cf. Section 4.5), it was evaluated and refined progressively by Research Activity 3 in three sequential steps.

4.6.1. Step 1 – Obtain Expert Reviews

A non-empirical review of MOBMAS was collected from two experts in the field of AOSE and ontology. The objective of the expert reviews was to:

- obtain experts' opinions on the *strengths* and *areas for improvement* of MOBMAS; and
- obtain experts' *suggestions* on how to improve these areas.

The two expert reviews were obtained in an independent and sequential manner. The review from the first expert was used to refine MOBMAS *before* the second expert was asked to review the refined version. All refinements made to MOBMAS as a result of each expert review were discussed with the relevant expert to ensure that he/she was satisfied with the changes made.

4.6.2. Step 2 – Use MOBMAS on a Test Application

After being non-empirically reviewed and refined, MOBMAS underwent empirical evaluation and refinement by being used on a specific application by two external developers. These developers were requested to provide, based on their usage of MOBMAS:

- opinions on the *strengths* and *areas for improvement* of MOBMAS;
- *suggestions* on how to improve these areas;
- *rating* of the "*ease of understanding*" and "*ease of following"* of each *step* of the MOBMAS development process (on a High-Medium-Low scale); and
- *rating* of the "*ease of understanding*" of each *model kind* of MOBMAS (on a High-Medium-Low scale).

The two developers applied and evaluated MOBMAS in an independent and sequential manner. The evaluation from the first developer was used to refine MOBMAS *before* the second developer was asked to apply and evaluate the refined version. All refinements made to MOBMAS as a result of each usage were discussed with the relevant developer in order to ensure that he was satisfied. In addition, the refinements made given the second developer's feedback were also discussed with the first developer in order to ensure that no conflicts of opinions occurred.

Apart from the evaluation of MOBMAS, the developers were also asked to produce a set of analysis and design models to demonstrate their use of MOBMAS on the test application.

4.6.3. Step 3 – Perform a Feature Analysis on MOBMAS

The feature analysis was performed on the final version of MOBMAS to:

- verify whether MOBMAS, as the final product, is able to achieve its objective, which is, to provide support for ontology-based MAS development and various other important AOSE methodological requirements which were identified in Research Activity 1 (cf. Section 4.2). It should be noted that, through the justification of MOBMAS' support for its methodological requirements, this research was able to justify that MOBMAS' actual *steps* and *modelling concepts* in fact correspond to, or cover, the desirable steps and modelling concepts which were specified as part of the methodological requirements;
- document the origin of MOBMAS techniques and model definitions (i.e. which techniques and model definitions have been reused and enhanced from the existing AOSE methodologies, and which have been newly developed); and
- compare MOBMAS with the existing AOSE methodologies in terms of various specific evaluation criteria. The comparison also highlighted the strengths of MOBMAS that resulted from its comprehensive support for ontology-based MAS development, and which are not provided (or provided to a lesser extent) by the existing methodologies due to their lack or low level of support for ontology.

4.7. SUMMARY

This chapter has stipulated the objective of this research and described the design of the three research activities that were performed to achieve this objective, namely:

- Research Activity 1: Identify the methodological requirements of MOBMAS – a "***M***ethodology for ***O***ntology-***B***ased ***M***ulti-***A***gent ***S***ystems";
- Research Activity 2: Develop MOBMAS; and
- Research Activity 3: Evaluate and refine MOBMAS.

In Chapters 5, 6 and 7, the performance and outcome of each activity are sequentially documented.

CHAPTER 5

METHODOLOGICAL REQUIREMENTS OF MOBMAS

5.1. INTRODUCTION

This chapter reports on the execution and outcome of the first research activity of the research's plan presented in Chapter 4 – "*Identify the methodological requirements of MOBMAS*" (cf. Section 4.3). The phrase "methodological requirements" refers to the **features**, **steps** and **modelling concepts** that are desirable to be supported by MOBMAS *process*, *techniques* and *model kinds*. Their identification was conducted systematically through four research steps (cf. Section 4.4):

- *Step 1 – Identify the "potential" requirements of MOBMAS*:
  This step aimed to determine a list of features, steps and modelling concepts that were *potentially* desirable to the system development process, techniques and model kinds of MOBMAS;
- *Step 2 – Conduct a survey on practitioners and researchers in the field of AOSE*:
  This step worked towards validating the potential requirements of MOBMAS, by gathering professional opinions on these requirements' rating and order ranking of importance. Step 2 also obtained professional recommendations on various issues that were useful to the development of MOBMAS;
- *Step 3 – Perform a feature analysis on the existing AOSE methodologies*:
  This step aimed to further validate the potential requirements of MOBMAS, by analysing the existing AOSE methodologies. This analysis was combined with the professional opinions obtained from Step 2 to determine the "actual" methodological requirements for MOBMAS. Step 3 also identified and evaluated the techniques and model definitions provided by each existing AOSE methodology for supporting each methodological requirement; and

- *Step 4 – Identify ontology-related steps from amongst the required MOBMAS' steps*: This step aimed to identify which of the required steps of MOBMAS should be related to ontology, so as to enable MOBMAS to offer all of the widely-recognised benefits of ontology to MAS development and MAS operation. These benefits have been listed in Section 2.3.2, and include those relating to the analysis of application domain, agent knowledge modelling, reusability, communication between MAS components, interoperability between heterogeneous components, and agent reasoning.

The execution and outcome of each research step are documented in Sections 5.2, 5.3, 5.4 and 5.5 respectively.

5.2. IDENTIFICATION OF POTENTIAL REQUIREMENTS OF MOBMAS

Step 1 of Research Activity 1 was concerned with identifying the *features*, *steps* and *modelling concepts* that were *potentially* desirable to MOBMAS process, techniques and model kinds.

5.2.1. Identification of Potential Features

Features potentially important to MOBMAS were identified by investigating the *existing evaluation frameworks*, namely:

- those for evaluating AOSE methodologies; and
- those for evaluating conventional system development methodologies, including OO methodologies.

The former contain evaluation criteria that relate to important *agent-oriented* and *MAS-specific* features, while the latter helped to identify important *generic system engineering features* which may have been overlooked by AOSE evaluation frameworks.

5.2.1.1. Evaluation frameworks for AOSE methodologies

A search of the literature revealed a limited number of evaluation frameworks for AOSE methodologies. This research investigated all of the identified frameworks.

- **Shehory and Sturm's Framework** (2001): Shehory and Sturm's evaluation criteria assess both generic software engineering features and specific agent-oriented features of an AOSE methodology. However, this research discarded a number of features that relate to system implementation issues because these are outside the scope of MOBMAS (cf. Section 4.2). Some other features were found desirable to MOBMAS, but they were not specified in the list of MOBMAS' potential features because they can be indirectly supported via other features or modelling concepts. For example, feature "*Modelling of communication richness*" evaluated by Shehory and Sturm (Table 5.1) is equivalent to the modelling of Agent Interaction concepts such as "*Agent acquaintance*", "*Interaction protocol*" and "*Content of exchanged messages*". Since these concepts would later be included in the "potential modelling concepts" of MOBMAS (cf. Section 5.2.3), this feature was not restated in this section to avoid redundancy in MOBMAS requirements.

  Table 5.1 displays the selection of evaluation features from Shehory and Sturm's framework and the reasons for discarding the others.

Table 5.1 – Selection of features from Shehory and Sturm's framework (2001)

Evaluation Criteria	Selected for the identification of potential features for MOBMAS?
Preciseness of models	✓
Accessibility of models	✓
Expressiveness of models	✓
Support for modularity	✓
Complexity Management	✓
Support for executability	× Outside the scope of research
Support for refinability	✓
Support for analysability	✓
Support for openness	✓
Modelling of autonomy	✓
Modelling of complexity	✓ Assessed via criteria "Expressiveness of models", "Support for modularity" and "Complexity management" of this framework
Modelling of adaptability	✓
Modelling of distribution	✓ Assessed via the modelling of "Agent instance deployment" concept (cf. Section 5.2.3.4)
Modelling of communication richness	✓ Assessed via the modelling of "Agent Interaction" concepts (cf. Section 5.2.3.3)

- **O'Malley and DeLoach's Framework** (2001): This framework evaluates both the technical features and management features of an AOSE methodology. Since the management issues are outside the scope of this research, criteria relating to them were excluded from the investigation (Table 5.2).

Table 5.2 – Selection of features from O'Malley and DeLoach's framework (2001)

Evaluation Criteria	Selected for the identification of potential features for MOBMAS?
Cost of acquiring the methodology	× Outside the scope of research
Cost of acquiring support tools	× Outside the scope of research
Effects on organisational business practices	× Outside the scope of research
Compliance with standards	× Outside the scope of research
Traceability of changes	✓
Legacy system integration	× Not applicable to all applications
Availability of reusable components	✓
Support for distribution	✓ Assessed via the modelling of "Agent instance deployment" concept (cf. Section 5.2.3.4)
Support for dynamic system structure	✓
Support for interaction	✓ Assessed via the modelling of "Agent Interaction" concepts (cf. Section 5.2.3.3)
Support for scalability	✓
Support for agility and robustness	✓

- **Cernuzzi and Rossi's Framework** (2002): Cernuzzi and Rossi proposed a step-by-step process for evaluating MAS development methodologies, supplemented by a set of evaluation criteria. All of these criteria were studied to identify the potential features for MOBMAS (Table 5.3).

Table 5.3 - Selection of features from Cernuzzi and Rossi's framework (2002)

Evaluation Criteria	Selected for the identification of potential features for MOBMAS?
Modelling of autonomy	✓
Modelling of reactivity	✓
Modelling of proactiveness	✓
Modelling of mental constructs (beliefs, goals)	✓ Assessed via the modelling of "Agent Interaction" concepts (cf. Section 5.2.3.3)
Modelling of agent interaction attributes	✓ Assessed via the modelling of "Agent Interaction" concepts and "Overall System Design" concepts (cf. Sections 5.2.3.3 and 5.2.3.4)
Support for modularity	✓
Support for abstraction	✓
Modelling of system view	✓
Communication support	✓

- **Sabas et al.'s Framework** (2002): Sabas et al. presented a framework called MUCCMAS for the comparative analysis of AOSE methodologies. MUCCMAS

offers a set of well-organised multi-dimensional evaluation criteria, many of which were selected for investigation (Table 5.4). The discarded criteria were those that focus on implementation-related issues (thus are outside the scope of MOBMAS), or those that do not relate to specific AOSE features, but merely aim to compare the different methodologies in terms of their applicability (e.g. target application, programming paradigm and agent types).

Table 5.4 - Selection of features from Sabas et al.'s framework (2002)

Evaluation Criteria	Selected for the identification of potential features for MOBMAS?
Specification of process phases	✓
Specification of development models	✓
Specification of development approach	✓
Degree of user implication	× Outside the scope of research
Support for models reuse	✓
Availability of software support	× Outside the scope of research
Support for system division	✓
Support for formalism	✓
Support for derivation	✓
Models quality	× Too generic, unclear what "quality" embraces
Supported agent nature	× Does not infer any feature but merely aims to compare methodologies in terms of their applicability
Supported agent type	× Does not infer any feature but merely aims to compare methodologies in terms of their applicability
Support for various agent attributes	✓
Modelling of organisation image	✓ Assessed via the modelling of "Role" concept and "Organisational structure" concept (cf. Sections 5.2.3.1 and 5.2.3.4)
Modelling of environment nature	✓ Assessed via the modelling of "Environment resource/facility" concept (cf. Section 5.2.3.4)
Supported types of communication	× Does not infer any feature but merely aims to compare methodologies in terms of their applicability
Supported communication mode	× Does not infer any feature but merely aims to compare methodologies in terms of their applicability
Supported communication language	× Does not infer any feature but merely aims to compare methodologies in terms of their applicability
Supported processing mode	× Outside the scope of research
Supported human-machine interface type	× Outside the scope of research
Supported programming paradigm	× Outside the scope of research
Environment of development	× Outside the scope of research
Supported application type	× Does not infer any feature but merely compares methodologies their applicability

5.2.1.2. Evaluation frameworks for conventional development methodologies

Considering the large number of evaluation frameworks for conventional system development methodologies, including OO methodologies, this research limited itself to only a number of well-known frameworks, namely Wood et al.'s framework (1988), NIMSAD (Jayaratna 1994), IFIP WG 8.1 (Olle et al. 1983) and The Object Agency's framework (The Object Agency Inc 1995).

- **Wood et al.'s Framework** (1988): This framework offers a large number of evaluation criteria, many of which are too application-specific or too technical (e.g. "Can stimulus/response relationships be represented in a time-dependent manner?" or "Does the methodology provide a representation that clearly draws a boundary around the system and separates it from its environment"). The relatively general criteria selected for investigation by this research are listed in Table 5.5[18].

Table 5.5 - Selection of features from Wood et al.'s framework (1988)

Selected evaluation criteria for the identification of potential features for MOBMAS
• Support for reuse • Completeness of representations • Consistency of representations • Complexity of representations • Ambiguity of representations • Abstraction of representations • Support for exception handling • Support for robustness

- **NIMSAD** (Jayaratna 1994): NIMSAD evaluates an Information Systems development methodology by determining whether, and how, the methodology supports different components of a proposed "standard" development framework. Three components of this "standard" framework are "*methodology context*", "*methodology user*" and "*problem-solving process*". Only evaluation questions pertaining to the "*problem-solving process*" component are relevant to this research, since they evaluate a methodology's development process and techniques, not the implementation context of the methodology. Among "*problem-solving process*"

[18] This figure, and figures 5.6, 5.7 and 5.8, do *not* document all of the evaluation criteria/questions provided by the corresponding frameworks because the number of criteria/questions provided by each framework is very large. In addition, the major reasons for discarding the unselected criteria have already been stated in the text and therefore are not repeated in the figures.

evaluation questions, many were discarded because they do not relate to any specific system engineering features, but instead assist in the in-depth understanding of a particular methodology, such as *"*What level of expressions does the methodology advocate*"* or *"*What criteria does the methodology offer for defining the problems*"*. Table 5.6 presents the criteria that were ultimately selected.

Table 5.6 - Selection of features from NIMSAD framework (1994)

Selected evaluation criteria for the identification of potential features for MOBMAS
• What modelling notions and techniques does the methodology offer for expressing situation characteristics? • Does the expression provide sufficient information to help gain a feel for the situation of concern? • What context information is captured or expressed? • What steps or techniques does the methodology offer in the formulation of solutions?

- **IFIP WG 8.1** (Olle et al. 1983): The IFIP Working Groups presented eight feature analysis studies of Information Systems development methodologies. Two of these frameworks were disregarded because they did not propose any specific desirable software engineering features, but merely aimed to compare the different methodologies in terms of their scope and applicability, namely Olive's study[19] (1983) and Falkenberg et al.'s study[20] (1983). Nissen's study (1983) was also discarded because it focuses on the implementation aspect of a methodology[21], which is not in this research's scope of interest. Of the remaining five frameworks, many criteria/questions were disregarded because they either pertain to implementation aspects (e.g. "Training" and "Methodology transferability" criteria in Bodart et al.'s study, 1983) or they are too application-specific (e.g. "What types of decisions – identification, functional, technico-economic, organisational, management – are considered in the methodology*"* in Bodart et al.'s study, 1983; or "Whether the design of databases in the methodology is data-oriented or processing-oriented" in Iivari and Kerola's study, 1983). Table 5.7 presents the selected evaluation criteria/questions from each investigated feature analysis study.

[19] Olive's study (1983) compares methodologies according to their supported abstraction levels (namely, external, conceptual, logical, architectural and physical levels) and the target types of information systems (e.g. database system or decision support information system).

[20] Falkenberg et al.'s study (1983) compares methodologies with respect to their coverage of the development lifecycle and the level of support for each lifecycle phase.

[21] Nissen's study (1983) evaluates a methodology in terms of how well the documentation produced by the methodology can be used to support different groups of interested people (e.g. designers, managers, computer-operator personnel and end-user).

Table 5.7 - Selection of features from IFIP WG 8.1 frameworks (1983)

Framework	Selected evaluation criteria for the identification of potential features for MOBMAS
Brandt's study (1983)	Development process (i.e. the phases or development steps proposed in the methodology) Model (including concepts, degree of formalism and abstraction) Representation means (i.e. graphical elements, use of formal languages, forms etc) Iteration and Tests (involving iterative routines and procedures for validation and verification)
Wasserman et al.'s study (1983)	Coverage of SDLC (i.e. What phases of the software development process are covered by the methodology) Support for top-down and bottom-up development Usability of the methodology (Is the methodology easy to use?) Support for validation and verification (what is the explicit means by which the completed system is validated against the original requirements; for each work product, what is the method used to assure the quality of the product). Support for problem analysis and understanding (i.e. problem-solving steps, problem-solving and modelling techniques) Support for communication among interested parties (i.e. modelling notation and concepts supported by the methodology's models)
Iivari & Kerola's study (1983)	Which are the main components of the conceptual structure Does the conceptual structure allow/support descriptions at different levels of abstraction, different levels of detail? Does the conceptual structure cover the interaction between the data system and its user? Are descriptions made using the specified languages unambiguous? Are descriptions made using the specified languages understandable?
Kung's study (1983)	Understandability of the conceptual model (i.e. readability, unambiguity, clarity and intuitivity) Expressiveness of the conceptual model (i.e. whether the modelling concepts and constructs are powerful enough to express everything that is needed to be specified, and have good resolution of detail) Consistency of the conceptual model
Bodart et al.'s study (1983)	Concepts (whether the concepts allow a complete modelling of all the organisation's aspects) Life cycle steps (including the set of models and formalisms involved) Step content

- **The Object Agency's Framework** (The Object Agency Inc 1995): This framework offers a well-organised set of evaluation questions, assessing an OO methodology in terms of diverse system engineering features. Evaluation criteria relating to OO concepts modelling were disregarded because they are not relevant to agent-oriented development. Several evaluation criteria on method marketability and pragmatics were also not considered because they focus on implementation aspects, thus lying outside the scope of the research. Most of the selected evaluation criteria pertain to modelling notation and system development process (Table 5.8).

Table 5.8 - Selection of features from the Object Agency's framework (The Object Agency Inc 1995)

Category of evaluation criteria	Selected evaluation criteria for the identification of potential features for MOBMAS
Notation	What are the components of the method's notation? What static concepts is the notation capable of expressing? What dynamic concepts is the notation capable of expressing? Are explicit rules presented for defining the notation symbols? Does there exist explicit logic for transforming models into other models, or partially creating a model from information present in another? Does the notation provide a partitioning mechanism?

Category of evaluation criteria	Selected evaluation criteria for the identification of potential features for MOBMAS
Process	What are the process steps for the development process within the methodology? What deliverables are generated from the development process? What aspects of the lifecycle are covered by the approach? Are the process steps well-defined? Are there heuristics available for the process steps? Does the process provide for verification? What development lifecycle best describes the methodology?
Pragmatics	What scope of effort is the method suited for? Is the method targeted at a specific type of software domain?
Support for Software Engineering	Reusability

5.2.1.3. Potential features of MOBMAS

Many evaluation criteria selected from the existing evaluation frameworks actually relate to the same or overlapping methodological features. Thus, the features extracted from these frameworks needed to be combined and synthesized into a coherent list. This list was then organised into four categories, each of which is described below.

One particular feature, namely "*Support for ontology-based MAS development*" (cf. Section 5.2.1.3.d), had not been considered in any existing evaluation frameworks. However it was included in the list of MOBMAS' potential features because this research is particularly interested in ontology-based MAS development.

5.2.1.3.a. Potential features for MOBMAS development process

This category contains six features that are potentially important to MOBMAS development process.

1. "*Specification of a system development lifecycle*": such as waterfall or iterative.
2. "*Support for verification and validation*": such as rules for verifying and validating the correctness of the developed models.
3. "*Specification of steps for the development process*".
4. "*Specification of model kinds and/or notational components*[22] *to be generated from each process step*".
5. "*Specification of techniques and heuristics for performing each process step and for producing each model kind*".

[22] Models are differentiated from notational components in that models are conceptual constructs that underlie the graphical or textual depictions, which are notational components (e.g. diagrams, tabular schemas).

6. "*Support for refinability"*: that is, whether the methodology provides a clear path for refining the models through gradual stages to reach an implementation or at least for clearly connecting the implementation level to the design specifications.

5.2.1.3.b. Potential features for MOBMAS model definitions

The following eight features are potentially important to MOBMAS model kinds and notational components.

1. "*High degree of completeness/expressiveness*": that is, the model kinds are capable of representing the system from different perspectives, capturing all necessary aspects such as static and dynamic aspects, system-level and agent-level aspects.
2. "*High degree of formalisation/preciseness*": that is, the syntax and semantics of the model kinds and notational components are clearly defined.
3. "*Provision of guidelines/logics for model derivation*": for transforming one model kind into other model kinds, or partially creating a model kind from information present in another model kind.
4. "*Guarantee of consistency*": between the levels of abstractions within each model kind and between different model kinds.
5. "*Support for modularity"*: that is, the model kinds are able to promote modularity in the design and representation of agents and the system.
6. "*Manageable number of concepts in each model kind and each notational component*".
7. "*Model kinds expressed at various levels of abstraction and detail*".
8. "*Support for reuse*".

5.2.1.3.c Potential agent properties to be captured/represented in MOBMAS model kinds

This category contains eight agent properties that are potentially important to be represented by MOBMAS model kinds.

1. "*Autonomy*": the ability to act without direct intervention of humans or others, and to control one's own state and behaviour.
2. "*Adaptability*": the ability to learn and improve with experience.
3. "*Cooperative behaviour*": the ability to work together with other agents to achieve a common goal.
4. "*Inferential capability*": the ability to reason and act on abstract task specifications.

5. "*Knowledge-level communication ability*": the ability to communicate with other agents with language more resembling human-like speech acts than typical symbol-level program-to-program protocols.
6. "*Personality*": the ability to manifest attributes of a "believable" human character.
7. "*Reactivity*": the ability to selectively sense and act in a timely manner.
8. "*Deliberative behaviour*": the ability to decide in a deliberation, i.e. proactiveness.

5.2.1.3.d. Potential features for MOBMAS as a whole

This category presents six high-level, supplementary features that are potentially important to MOBMAS as a whole.

1. "*Support for open systems*": which are systems that allow for dynamic addition/removal of agents.
2. "*Support for dynamic systems*": which are systems that allow for dynamic changes in agent behaviour and system structure.
3. "*Support for agility and robustness*": that is, the methodology captures normal processing and exception processing, provides techniques to analyse system performance for all configurations, and/or provides techniques to detect and recover from failures.
4. "*Support for heterogeneous systems*": that is, the methodology supports the use/incorporation of (heterogeneous) non-agent software components in the system.
5. "*Support for mobile agents*": for example, the methodology models which, when and how agents should be mobile.
6. "*Support for ontology-based MAS development*": that is, support for the use and inclusion of ontologies in MAS development process and MAS model definitions.

5.2.2. Identification of Potential Steps

Steps that are potentially desirable to MOBMAS development process were identified by investigating *the existing AOSE methodologies*, namely the sixteen methodologies documented in Chapter 3. Each methodology offers a different collection of steps for the MAS development process. Only Analysis and Design steps were investigated by the research. Implementation-related steps such as "Develop prototypes" of ADELFE or "Reuse code" of PASSI were not considered because they are outside the scope of MOBMAS (cf. Section 4.2). Steps that are too specific to a particular methodology were

also discarded, such as "Verify adequacy of AMAS theory" of ADELFE[23] or "Develop IDEF/CIMOSA models for the target system" of MEI[24].

After retrieving steps from the existing AOSE methodologies, these steps were synthesized and combined into a coherent superset. The synthesis process paid careful attention to the possibility of different methodologies using different terminology to refer to the same step. Table 5.9 presents the synthesized steps and their origins.

Table 5.9 – Identification of steps from the existing AOSE methodologies

Steps	MASE	MASSIVE	SODA	GAIA	MESSAGE	INGENIAS	BDIM	HLIM	MEI	PROMETHEUS	PASSI	ADELFE	CASSIOPEIA	COMOMAS	MAS-COMMONKADS	TROPOS
1. Identify system functionality	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
2. Specify use case scenarios	✓					✓		✓	✓	✓	✓	✓			✓	
3. Identify roles	✓	✓	✓	✓	✓	✓	✓	✓			✓		✓			
4. Identify agent classes	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5. Model domain conceptualisation	✓				✓						✓				✓	
6. Specify acquaintances between agent classes	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
7. Define interaction protocols	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
8. Define content of exchanged messages	✓				✓	✓	✓	✓		✓	✓	✓			✓	✓
9. Specify agent communication language					✓							✓				
10.Specify agent architecture	✓	✓			✓					✓	✓	✓			✓	
11.Define agent informational constructs (i.e. beliefs)	✓				✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
12.Define agent behaviour al constructs (e.g. goals, plans, actions, services)				✓			✓	✓		✓	✓	✓			✓	
13.Specify system architecture		✓			✓	✓				✓	✓	✓			✓	
14.Specify organisational structure/inter-agent authority relationships		✓		✓	✓	✓		✓					✓	✓	✓	✓
15.Model MAS environment		✓	✓	✓	✓	✓				✓		✓			✓	✓
16.Specify agent-environment interaction mechanism		✓				✓			✓	✓		✓				
17.Specify agent inheritance and aggregation				✓			✓					✓			✓	
18.Instantiate agent classes	✓			✓	✓		✓			✓					✓	
19.Specify agent instances deployment	✓										✓					

[23] This step is only applicable to ADELFE, which employs the theory of "AMAS" for the development of adaptive MAS systems (cf. section 4.2.12)

[24] This step is only applicable to enterprise integration applications (cf. section 4.2.9).

MOBMAS' potential steps were organised into four categories.

5.2.2.1. Potential Problem Domain Analysis steps

This category contains five steps that are potentially important to the understanding of the target application.

1. "*Identify system functionality*"
2. "*Specify use case scenarios*"
3. "*Identify roles* "
4. "*Identify agent classes*"
5. "*Model domain conceptualisation*"

5.2.2.2. Potential Agent Interaction Design steps

This category contains four steps that are potentially important to the design of agent interactions.

1. "*Specify acquaintances between agent classes*"
2. "*Define interaction protocols*"
3. "*Define content of exchanged messages*"
4. "*Specify agent communication language*"

5.2.2.3. Potential Agent Internal Design steps

This category presents three steps that are potentially important to the internal design of agents.

1. "*Specify agent architecture*"
2. "*Define agent informational constructs*" (i.e. beliefs)
3. "*Define agent behavioural constructs*" (e.g. goals, plans, actions, services)

5.2.2.4. Potential Overall System Design steps

The seven steps presented in this category are potentially important to the design of MAS overall structure and deployment.

1. "*Specify system architecture*" (i.e. overview of all system components and their connections)
2. "*Specify organisational structure/inter-agent authority relationships*"
3. "*Model MAS environment*" (e.g. resources, facilities)

4. "*Specify agent-environment interaction mechanism*" (e.g. sensors and effectors)
5. "*Specify agent inheritance and aggregation*"
6. "*Instantiate agent classes*"
7. "*Specify agent instances deployment*"

5.2.3. Identification of Potential Modelling Concepts

As for the potential steps, potentially desirable modelling concepts of MOBMAS were also identified from *the existing AOSE methodologies*, namely those documented in Chapter 3. Each methodology offers a set of model kinds and/or notational components (i.e. diagrams and textual schemas) that capture different AOSE modelling concepts. The concepts retrieved from the existing methodologies were then synthesized and combined into a coherent superset. This synthesis paid careful attention to the possibility of different methodologies using different terminology to refer to the same concept. Table 5.10 presents the identified modelling concepts and their origin.

Table 5.10 - Identification of modelling concepts from the existing AOSE methodologies

Concepts	**MASE**	**MASSIVE**	**SODA**	**GAIA**	**MESSAGE**	**INGENIAS**	**BDIM**	**HLIM**	**MEI**	**PROMETHEUS**	**PASSI**	**ADELFE**	**CASSIOPEIA**	**COMOMAS**	**MAS-CommonKADS**	**TROPOS**
System functionality	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
Use case scenario	✓					✓		✓	✓	✓	✓	✓			✓	
Role	✓	✓	✓	✓	✓	✓		✓			✓		✓			
Domain conceptualisation	✓				✓						✓				✓	
Agent-role assignment	✓	✓	✓	✓	✓	✓					✓		✓		✓	
Agent's goal/task						✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Agent's belief/knowledge						✓	✓	✓		✓	✓	✓		✓	✓	✓
Agent's plan/reasoning rule/problem solving method	✓					✓	✓	✓	✓	✓	✓				✓	✓
Agent's capability/service				✓				✓		✓	✓	✓			✓	
Agent's percept/ event		✓								✓					✓	
Agent architecture	✓	✓								✓	✓	✓			✓	✓
Agent acquaintance	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓
Interaction protocol	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
Content of exchanged messages	✓				✓	✓	✓	✓		✓	✓	✓			✓	✓
System architecture		✓			✓	✓				✓	✓	✓			✓	
Organisational structure/inter-agent authority relationships		✓		✓	✓	✓		✓						✓	✓	✓
Environment resources, facilities			✓	✓	✓	✓				✓		✓			✓	
Agent aggregation relationship				✓			✓					✓			✓	
Agent inheritance relationship							✓								✓	
Agent instantiation	✓		✓	✓			✓			✓					✓	
Agent instances deployment	✓		✓								✓					

MOBMAS' potential modelling concepts were organised into four categories.

5.2.3.1. Potential Problem Domain concepts

This category contains four modelling concepts that are potentially important to the description of the target problem domain.

1. "*System functionality*"
2. "*Use case scenario*"
3. "*Role*"
4. "*Domain conceptualisation*"

5.2.3.2. Potential Agent concepts

The seven concepts contained in this category are potentially important to the modelling of agents.

1. "*Agent-role assignment*"
2. "*Agent goal/task*"
3. "*Agent belief/knowledge*"
4. "*Agent plan/reasoning rule/problem solving method*"
5. "*Agent capability/service*"
6. "*Agent percept/event*" (i.e. event that triggers the agent's actions)
7. "*Agent architecture*"

5.2.3.3. Potential Agent Interaction concepts

Three concepts in this category are potentially important to the modelling of agent interactions.

1. "*Agent acquaintance*" (i.e. interaction pathways between agents)
2. "*Interaction protocol*"
3. "*Content of exchanged messages*"

5.2.3.4. Potential Overall System Design concepts

This category presents seven concepts that are potentially important to the modelling of MAS overall structure and deployment design.

1. "*System architecture*"
2. "*Organisational structure/inter-agent authority relationships*"

3. "*Environment resource/facility*"
4. "*Agent aggregation relationship*"
5. "*Agent inheritance relationship*"
6. "*Agent instantiation*"
7. "*Agent instance deployment*"

5.3. SURVEY

After the *potential* methodological requirements of MOBMAS were identified by Step 1 of Research Activity 1 as reported in Section 5.2, Step 2 – "*Conduct a survey on practitioners and researchers in the field of AOSE*" was conducted to help validate these potential requirements. The survey also obtained professional recommendations on various issues that were useful to the development of MOBMAS (cf. Section 4.4.2).

This section documents the procedure, questionnaire, testing process and results of this survey.

5.3.1. Survey Procedure

The survey consisted of a questionnaire, which was posted online due to the dispersed location of prospective respondents, the ease of disseminating the survey and the cost-effectiveness of survey execution. The questionnaire was also completed online so that automatic checking of the survey's responses could be performed (e.g. checking whether the compulsory questions were all answered and whether the responses were valid and consistent). Online completion also allowed the responses to be automatically recorded into an electronic database.

The survey website was hosted on a web server at the School of Information Systems, Technology and Management at The University of New South Wales. The survey questionnaire was designed using IBM Lotus Domino Developer software, while the survey results were stored in an IBM Lotus Notes database. To prevent public access, the site was password-protected.

The target population consisted of system analysts, system designers/developers, project managers and researchers/academia whose area of interest and practice is AOSE (in

general) and MAS development (in particular). This population was accessed via *UMBC AgentNews* newsletter (UMBC Lab for Advanced Information Technology n.d.b) and *UMBC Agents-Digest* mailing list (UMBC Lab for Advanced Information Technology n.d.c). These two media are the prominent and prestigious information resources and "meeting points" of the agent community.

The survey period was 1.5 months. During this period, advertisements were sent twice to *UMBC AgentNews* newsletter and *UMBC Agents-Digest* mailing list, with a 3-week interval between the first and second advertisements. The re-posting of advertisements helped to reach the prospective respondents who might have overlooked the first call for participation.

Information conveyed in the advertisements included (Appendix A):

- the research objective;
- activities involved in the survey, including the estimated time to complete the survey questionnaire;
- required expertise from the respondents (i.e. knowledge and/or experience in AOSE);
- benefits to the respondents, such as feedback of the survey's findings if desired;
- the facts that participation was completely voluntary and the respondents could remain anonymous if desired; and
- password to access the online survey questionnaire.

The respondents were not obliged to complete the survey questionnaire in one go. They could pause the survey at any point after saving their work. In such cases, they were given an ID Number which allowed them to return to their partially completed questionnaire as many times as needed until the survey was finished.

Near the end of the survey period, a reminder was sent to *UMBC AgentNews* newsletter and *UMBC Agents-Digest* mailing list to remind the respondents who had partially completed the survey questionnaire to finish it.

5.3.2. Survey Questionnaire

The questionnaire consists of five parts (Appendix B), taking approximately 30-40 minutes to complete.

Part 1 collected demographic and professional background of the respondents. Specific information gathered was:

- name, organisation and email address of each respondent (optional but required if the respondent wished to receive feedback on the survey's findings);
- field of work (e.g. system analyst, system developer/developer, project manager, programmer or researcher);
- level of theoretical knowledge and industrial experience with MAS in general and with MAS development in particular; and
- characteristics of MAS development projects in which the respondent had been involved (e.g. level of complexity, number of agents, application area and name of the adopted methodology, if any).

Part 2 gathered the respondents' opinions on a list of *features* with regard to how important these features are to a "standard" AOSE methodology. This list of features was obtained from Section 5.2 and contained (cf. Section 5.2.1):

1. features that are potentially desirable to an AOSE process;
2. features that are potentially desirable to AOSE model definitions;
3. agent properties that are potentially desirable to be captured/represented by AOSE model kinds; and
4. features that are potentially desirable to a MAS development methodology as a whole.

The respondents were requested to ***rate the importance*** of each feature on a scale of "Very high", "High", "Medium", "Low" and "Very low". They were also asked to ***order rank*** the features within each category in a decreasing order of importance, from rank "1" (the most important) to rank *n* (the least important) where *n* was the total number of features in the category. Some features might be ranked equally if they could not be differentiated.

The respondents were invited to provide suggestions on any features that they believed should be supported by an AOSE methodology but were not listed in this part of the survey.

Part 3 gathered the respondents' opinions on a list of *steps* with respect to how important these steps are to a "standard" AOSE process. These steps were obtained from Section 5.2 and categorized into (cf. Section 5.2.2):

1. Problem Domain Analysis steps;
2. Agent Interaction Design steps;
3. Agent Internal Design steps; and
4. Overall System Design steps.

Again, the respondents were asked to ***rate the importance*** of the specified steps on a scale of "Very high", "High", "Medium", "Low" and "Very low", and to ***order rank*** the steps in each category from "most important" (i.e. rank "1") to "least important" (i.e. rank *n*). Steps could be ranked equally if they could not be differentiated. The respondents were also invited to suggest any steps that they believed should be provided by a "standard" MAS development process but were not included in the survey.

Part 4 sought the respondents' opinions on a list of *modelling concepts* with respect to how important they are to be captured/represented by the model kinds of a "standard" AOSE methodology. This list of concepts was obtained from Section 5.2 and categorised into (cf. Section 5.2.3):

1. Problem Domain concepts;
2. Agent concepts;
3. Agent Interaction concepts; and
4. Overall System Design concepts.

As in the preceding two survey parts, the respondents were requested to ***rate the importance*** of each concept on a scale of "Very high", "High", "Medium", "Low" and "Very low", and to ***order rank*** the concepts in each category from "the most important" to "the least important". Some concepts might be ranked equally if they could not be differentiated. Suggestions were also collected on any modelling concepts that are important to the "standard" AOSE model kinds but were not included in the survey.

Part 5 of the survey obtained the respondents' recommendations on various issues that were relevant to the construction of a MAS development methodology, namely:

- the type of software development lifecycle that best suits an AOSE process;
- the importance of committing to a specific agent architecture (e.g. BDI) by an AOSE methodology; and
- the desirable approach for MAS development (e.g. role-oriented or non-role-oriented).

Each question in this section was accompanied by a request for the respondents' rationale for their answer.

5.3.3. Survey Testing

The survey was pilot-tested by three academic staffs, whose area of research is agent technology. The aim of this pilot-test was to evaluate and refine the content, layout and usability of the online survey questionnaire.

In the preliminary version, the survey questionnaire only requested the respondents to "rate the importance" of the specified features, steps and modelling concepts (i.e. without "order ranking"). However, the pilot-testers strongly recommended including "order ranking" as part of the survey requirements. Reason for their recommendation was because the listed features, steps and modelling concepts may all be given equal importance ratings, thus concealing their differentiation in terms of prioritization. The explicit ranking order of features, steps and concepts would allow the survey to accurately capture the respondents' prioritization of the features, steps and concepts.

With regard to the questionnaire's layout and usability, a number of suggestions for improvement were made, namely:

- allowing for the respondents to move back and forth between the different parts of the survey questionnaire;
- providing detailed explanation on some certain features, steps and modelling concepts in the form of popup windows;
- indicating the number of questionnaire parts yet to be completed;

- highlighting keywords in the instructions and questions to improve ease of understanding;
- checking the respondents' completion of the compulsory fields before allowing them to move to the subsequent questionnaire part; and
- checking and fixing potential errors in the respondents' "order ranking" of features, steps and modelling concepts. Two types of ranking errors that needed to be prevented are:
  - "*Internal coherence*" error: which occurs when multiple features are order-ranked equally but the next lower-ranked features do not have their ranks shifted accordingly. For example, if features A, B and C are order-ranked equally at rank "1", the next less important feature D should be ordered at rank "4"; and
  - "*External consistency*" error: which occurs when the "order ranks" of features are inconsistent with their "ratings of importance", or vice versa. For example, if feature A is given a "Very high" importance rating and feature B a "Medium" rating, the order rank of feature A should be correspondingly more important than the order rank of feature B.

  The detection of these errors was recommended to be performed when the respondents attempt to move from one part of the survey to the next, or when they save the (partially) completed survey. An "internal coherence" error should be automatically fixed by the online survey software. That is, the software should automatically adjust the order rankings in a way to preserve the intended ranking order but eliminate the incoherence problem. On the other hand, if an "external consistency" error is detected, the respondents should be warned of the error (by means of alert messages) and requested to fix the problem themselves.

All of the suggested improvements were implemented into the final version of the survey software.

5.3.4. Statistical Analysis and Results

In total, 41 respondents completed the survey. After processing and collation into a single data set, the survey data was input into a SPSS statistical package for analysis. Each of the following sections presents the statistical analysis of each part of the survey (cf. Section 5.3.2).

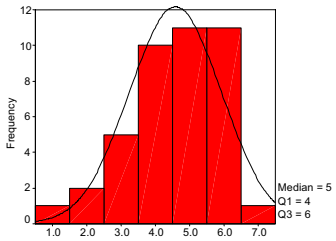
5.3.4.1. Part 1 – Demographic and professional characteristics of respondents

Information collected from this part of the survey produced eleven variables which pertain to the respondents' demographic and professional characteristics.

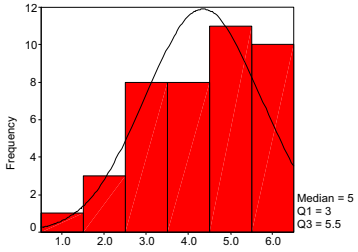
1. "*Field of work*"
2. "*Involvement in MAS development projects*" (Yes or No)
3. "*Size of past MAS projects*"
4. "*Level of complexity of past MAS projects*"
5. "*Application areas of past MAS projects*"
6. "*Adoption of AOSE methodologies in past MAS projects"*
7. "*Involvement in Ontology-Based MAS development projects*" (Yes or No)
8. "*Theoretical knowledge of MAS*" (e.g. knowledge about MAS characteristics)
9. "*Theoretical knowledge of MAS development*" (i.e. knowledge about MAS analysis and design)
10. "*Industrial experience with MAS*" (e.g. past use of MAS)
11. "*Industrial experience with MAS development*" (i.e. experience with MAS analysis and design)

Descriptive statistics of the first seven variables are presented in Appendix C. This section focuses on the last four variables, which jointly reflect the respondents' *expertise* on MAS in general and MAS development in particular. These four variables, referred to as "*expertise variables*", were given particular attention because they allowed the research to investigate the impact of respondents' expertise on the "rating of importance" and "order ranking" of features, steps and modelling concepts in Parts 2, 3 and 4 of the survey.

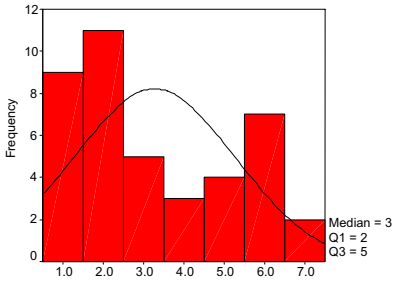
Response scores given to each expertise variable were obtained via a 7-point Likert scale, ranging from "1" (i.e. "Low") to "7" (i.e. "Extensive"). The distribution of the four variables showed that medians of both "*Theoretical knowledge of MAS*" and "*Theoretical knowledge of MAS development*" were "5", while the medians of both "*Industrial experience with MAS*" and "*Industrial experience with MAS development*" were "3" (Figure 5.11).



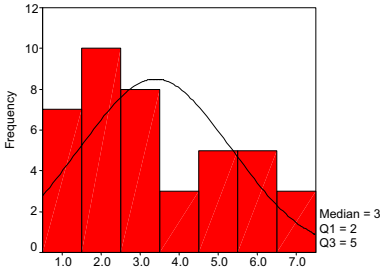
Theoretical knowledge of MAS



Theoretical knowledge of MAS development



Industrial experience with MAS



Industrial experience with MAS development

Figure 5.11 – Distribution of four expertise variables

Wilcoxin Signed-Ranked Tests[25] between pairs of these four variables further revealed that, at a significance level of 5%:

- there is no significant difference between
  - "*Theoretical knowledge of MAS*" and "*Theoretical knowledge of MAS development*" (p = 0.15); and
  - "*Industrial experience with MAS*" and "*Industrial experience with MAS development*" (p = 0.311); but
- there is a significant difference between
  - "*Theoretical knowledge of MAS*" and "*Industrial experience with MAS*" (p = 0.002); and
  - "*Theoretical knowledge of MAS development*" and "*Industrial experience with MAS development*" (p = 0.001).

[25] Wilcoxin Signed-Ranked Test was chosen because it is a well-known test for two-related-sample comparisons concerning continuous ordinal data (Leach 1979). The samples in this case were related because the response scores given to the different expertise variables were collected from the same set of respondents. The Wilcoxin Signed-Ranked Test assumed that each respondent's data was independent from the data of other respondents, which was reliably true for the survey data.

These findings were not surprising, considering the fact that a large proportion of the respondents worked in the field of research/academia (cf. Appendix C). They were thus expected to be more familiar with theoretical aspects than practical aspects of MAS and MAS development.

To analyse the impact of respondents' expertise on "rating of importance" and "order ranking" of features, steps and modelling concepts during Parts 2, 3 and 4 of the survey, the respondents were classified into two subject-groups, "*High Expertise*" and "*Low Expertise*", with respect to *each* expertise variable (Table 5.12). A respondent was viewed as having "*Low Expertise*" if his response score was less than 4 and "*High Expertise*" if it was in the range 4-7.

Table 5.12 – Number of respondents in each subject group

Expertise Variable	**Number of respondents**	
	"Low Expertise" response score <= 4	**"High Expertise"** response score > 4
"*Theoretical knowledge of MAS*"	18	23
"*Theoretical knowledge of MAS development*"	20	21
"*Industrial experience of MAS*"	13	28
"*Industrial experience of MAS development*"	13	28

5.3.4.2. Part 2 – Rating and order ranking of Features

Part 2 of the survey requested the respondents to ***rate the importance*** of each specified *feature* on a scale of "Very High", "High", "Medium", "Low" and "Very Low". The respondents were also asked to ***order rank*** the features in each category[26] in a decreasing order of importance, from rank "1" (the most important) to rank *n* (the least important) where *n* was the total number of features in the category. Equal ranks were allowed for features that could not be differentiated.

"Ratings of importance" and "order ranks" of all features are shown in Table 5.14. With regard to "**rating of importance**", each feature's rating was calculated as the *median* of the rating scores given by the 41 respondents. The *range* of rating scores is also presented in Table 5.14.

With regard to the "**order ranking**" of features in each category, the research firstly calculated the *mean rank* of each feature (which is the mean value of the ranking scores

[26] The features were organised into four categories (cf. Section 5.2.2).

given to the feature by the 41 respondents). The mean rank value was then used to sort the features in each category in a decreasing order of importance. The *smaller* the mean rank, the *more important* a feature is in relation with others in the category.

In order to determine if the order ranks of the different features in a particular category are indeed reliably different from each other, two tests were performed.

- A Friedman Test was firstly carried out between the multiple groups of responses, each of which contains the ranking scores given to each feature in the category by the 41 respondents[27]. The test helped to detect whether there is an *overall* significant difference between all the features in the category regarding their ranking scores.
- If the Friedman Test produced a significant result at a significant level of 5%, a pair-wise Sign Tests[28] was performed to compare between each pair of features. The aim was to identify which features were ranked reliably higher than which other features at a significance level of 5%.

After applying these two tests, the ranking order of the features was refined. While the preliminary ordering (using mean ranks) was preserved, features that were *not* ranked significantly different from each other were grouped inside a dashed-line box, as presented in Table 5.14. The arrow points from the most important feature to the least important.

Figure 5.13 illustrates how the ranking order results in Table 5.14 can be interpreted. In Figure 5.13a, feature A is ranked significantly more important than features B, C and D, which are equivalently ranked among themselves. Figure 5.13b means that feature A is ranked as the most important and D as the least important. Features B and C are ranked

---

[27] Friedman Test was chosen because it is well-known test for multiple-related-sample comparison (Leach 1979). The samples in this case were related because each sample contains ranking scores that were obtained from the same group of respondents.

[28] Sign Test was chosen because it is equivalent to the Friedman Test when only two samples are involved (Leach 1979). The procedure for pair-wise comparison was borrowed from Leach (1979): a Sign Test was first carried out between the feature with the *smallest mean rank* and the feature with the *largest mean rank*. If a significant difference was detected, the feature with the smallest mean rank would be compared with the feature with the *second* largest mean rank, and so on until a non-significant result was obtained. Next, the feature with the *second smallest mean rank* was compared with the feature with the *largest mean rank*, followed by the feature with the second largest mean rank, and so on until a non-significant result was obtained.

significantly less important than A but more important than D. B and C are equivalently ranked. Figure 5.13c presents a more complicated ranking order. Feature A is ranked significantly more important than D, with B and C being ranked somewhere in between. However, features B and C are not reliably ranked differently from either A or D.

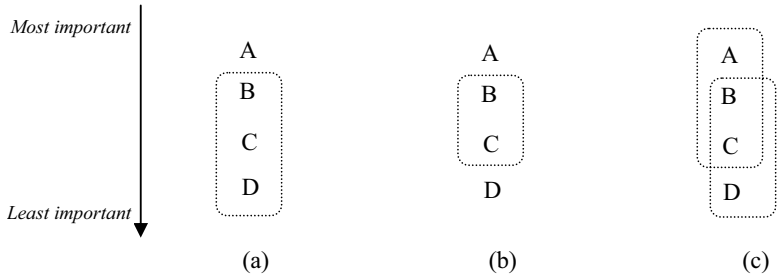


Figure 5.13 – Examples of ranking order results

Table 5.14 – "Rating of importance" and "order rank" of features

Features desirable to an AOSE process	**Median Rating of Importance***	**Range of Rating of Importance***	**Mean Rank**
1. Specification of model kinds and/or notational components	VH	[M; VH]	3.34
2. Specification of steps for the development process	VH	[M; VH]	3.46
3. Specification of techniques and heuristics for performing each process step and producing each model kind	VH	[L; VH]	3.78
4. Support for verification and validation	VH	[L; VH]	3.95
5. Support for refinability	VH	[VL; VH]	4.26
6. Specification of a system development lifecycle	VH	[VL; VH]	4.43
Features desirable to AOSE model definitions	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Guarantee of consistency	VH	[L; VH]	3.52
2. Model kinds expressed at various level of abstraction and detail	VH	[M; VH]	3.78
3. Support for reuse	VH	[L; VH]	3.80
4. High degree of completeness/expressiveness	VH	[M; VH]	3.81
5. Manageable number of concepts in each model kind and each notational component	VH	[VL; VH]	4.06
6. Support for modularity	VH	[L; VH]	4.23
7. High degree of formalisation/preciseness	VH	[VL; VH]	4.41
8. Provision of guidelines/logics for model derivation	VH	[VL; VH]	4.49

Agent properties desirable to be captured/represented in AOSE model kinds	Median Rating of Importance	Range of Rating of Importance	Mean Rank
1. Autonomy	VH	[M; VH]	2.54
2. Cooperative behaviour	VH	[M; VH]	3.46
3. Deliberative behaviour	VH	[L; VH]	3.92
4. Knowledge-level communication ability	VH	[L; VH]	4.14
5. Inferential capability	VH	[VL; VH]	4.34
6. Reactivity	VH	[M; VH]	4.75
7. Adaptability	VH	[L; VH]	5.48
8. Personality	M	[VL; VH]	7.98
Features desirable to a MAS development methodology as a whole	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Support for dynamic systems	VH	[VL; VH]	3.34
2. Support for open systems	VH	[L; VH]	3.78
3. Support for ontology-based MAS development	VH	[M; VH]	3.87
4. Support for heterogeneous systems	H	[VL; VH]	4.95
5. Support for agility and robustness	M	[VL; VH]	6.26
6. Support for mobile agents	M	[VL; VH]	6.71

* VH: Very High H: High M: Medium L: Low VL: Very Low

The research also investigated the impact of *expertise* on the respondents' "rating of importance" and "order ranking" of the features. As described in Section 5.3.4.1, respondents were classified into two subject-groups, "*High Expertise*" and "*Low Expertise*", with respect to each expertise variable ("*Theoretical knowledge of MAS*", "*Theoretical knowledge of MAS development*", "*Industrial experience with MAS*" and "*Industrial experience with MAS development*"). Mann-Whitney Tests were performed to compare between the two subject-groups in each expertise variable with regard to the "rating of importance" and "order ranking" of each feature.

Followings are features that were found affected by the respondents' expertise in their rating and/or order ranking (significance level = 5%).

- "**Specification of a system development lifecycle**": This feature was given a *higher* "rating of importance" and a *more important* "order rank" by the respondents who had "*Low Expertise*" in "*Theoretical knowledge of MAS*" compared to those who had "*High Expertise*" (one-tailed p = 0.02 for rating and p = 0.018 for order ranking).
- "**Support for reuse**": This feature was order ranked as *more important* by respondents with "*High Expertise*" in "*Industrial experience with MAS*" compared to those with "*Low Expertise*" (one-tailed p = 0.052).

- "**Knowledge-level communication ability**": This feature was order ranked as *more important* by respondents with "*High Expertise*" in "*Industrial experience with MAS*" and in "*Industrial experience with MAS development*" compared to those with "*Low Expertise*" in these two variables (one-tailed p = 0.028 and 0.006 respectively).
- "**Support for ontology-based MAS development**": This feature was order ranked as *more important* by respondents with "*High Expertise*" in *"Industrial experience with MAS*" compared to those with "*Low Expertise*" (one-tailed p = 0.014).

The respondents were also invited to provide suggestions on any other features that they believed should be supported by a "standard" MAS development methodology. However, no suggestions were made.

5.3.4.3. Part 3 – Rating and order ranking of Steps

Statistical analysis performed on the "ratings of importance" and "order rankings" of steps is the same as the analysis of features (cf. Section 5.3.4.2). The results are displayed in Table 5.15.

Table 5.15 – "Rating of importance" and "order rank" of steps

Problem Domain Analysis steps	**Median Rating of Importance ***	**Range of Rating of Importance***	**Mean Rank**
1. Identify system functionality	VH	[L; VH]	2.05
2. Identify agent classes	VH	[M; VH]	3.49
3. Model domain conceptualisation	VH	[L; VH]	3.61
4. Identify roles	H	[M; VH]	3.68
5. Specify use case scenarios	M	[VL; H]	3.72
Agent Interaction Design steps	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Define interaction protocols	VH	[M; VH]	1.02
2. Specify acquaintances between agent classes	VH	[L; VH]	1.54
3. Define content of exchanged messages	VH	[L; VH]	2.51
4. Specify agent communication language	M	[VL; VH]	3.75

Agent Internal Design steps	Median Rating of Importance	Range of Rating of Importance	Mean Rank
1. Define agent informational constructs	VH	[M; VH]	1.78
2. Define agent behavioural constructs	VH	[M; VH]	2.10
3. Specify agent architecture	VH	[L; VH]	2.34
Overall System Design steps	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Specify system architecture	VH	[M; VH]	1.90
2. Specify organisational structure/inter-agent authority relationships	VH	[L; VH]	2.05
3. Model MAS environment	VH	[M; VH]	2.90
4. Specify agent-environment interaction mechanism	H	[L; VH]	3.90
5. Instantiate agent classes	H	[VL; VH]	4.23
6. Specify agent instances deployment	H	[VL; VH]	4.89
7. Specify agent inheritance and aggregation	M	[VL; VH]	5.95

* VH: Very High H: High M: Medium L: Low VL: Very Low

Respondents' *expertise* was found to affect the "rating of importance" and "order rank" of two steps.

- "**Define content of exchanged messages**": This step was order ranked as *more important* by respondents with "*High Expertise*" in "*Theoretical knowledge of MAS*" compared to those with "*Low Expertise*" (one-tailed p = 0.04).
- "**Specify agent inheritance and aggregation**": This step was given a *higher* "rating of importance" by respondents with "*Low Expertise*" in "*Industrial experience with MAS development*" compared to those with "*High Expertise*" (one-tailed p = 0.028).

Although the respondents were invited to provide suggestions on any other steps that are desirable to a "standard" MAS development process, no suggestions were made.

5.3.4.4. Part 4 – Rating and order ranking of Modelling Concepts

The "rating of importance" and "order rank" of concepts were determined using the same statistical methods as those performed on features (cf. Section 5.3.4.2). The results are displayed in Table 5.16.

Table 5.16 – "Rating of importance" and "order rank" of modelling concepts

Problem Domain concepts	**Median Rating of Importance***	**Range of Rating of Importance***	**Mean Rank**
1. System functionality	VH	[L; VH]	1.82
2. Role	VH	[M; VH]	2.56
3. Domain conceptualisation	H	[L; VH]	2.80
4. Use case scenario	M	[VL; H]	3.73
Agent concepts	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Agent belief/knowledge	VH	[M; VH]	3.51
2. Agent goal/task	VH	[L; VH]	3.63
3. Agent-role assignment	VH	[M; VH]	3.90
4. Agent action/service	VH	[VL; VH]	4.05
5. Agent plan/reasoning rule/problem solving method	VH	[L; VH]	4.48
6. Agent architecture	H	[L; VH]	6.05
7. Agent percept/event	M	[L; VH]	6.52
Agent Interaction concepts	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. Interaction protocol	VH	[M; VH]	1.73
2. Content of exchanged messages	VH	[VL; VH]	2.22
3. Agent acquaintance	VH	[M; VH]	2.49
Overall System Design concepts	**Median Rating of Importance**	**Range of Rating of Importance**	**Mean Rank**
1. System architecture	VH	[M; VH]	1.75
2. Organisational structure/inter-agent authority relationships	VH	[M; VH]	1.90
3. Environment resource/facility	VH	[VL; VH]	2.90
4. Agent instance deployment	H	[VL; VH]	4.87
5. Agent instantiation	H	[VL; VH]	5.01
6. Agent aggregation relationship	M	[VL; VH]	6.49
7. Agent inheritance relationship	M	[VL; VH]	6.85

***** VH: Very High H: High M: Medium L: Low VL: Very Low

The respondents were invited to provide suggestions on any other modelling concepts that may be important to model definitions of a "standard" MAS development methodology. However, no suggestions were made.

5.3.4.5. Part 5 – Recommendations on AOSE methodological issues

Part 5 of the survey obtained the respondents' suggestions and comments on various issues that pertained to the construction of a MAS development methodology. The collected recommendations are presented and discussed below.

Issue 1: MAS development SDLC

When asked the open-ended question "*If an AOSE methodology must incorporate a SDLC, which SDLC do you think it should be?*", a majority of the respondents suggested a SDLC model that is iterative and incremental (Figure 5.17). A few other SDLC models were identified from the open-ended answers of the respondents. Four respondents suggested more than one SDLC model.

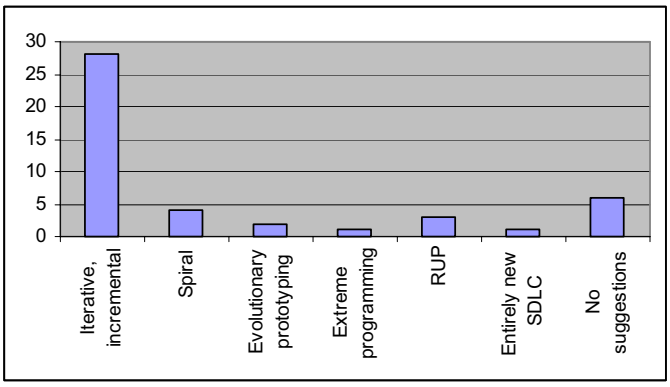


Figure 5.17 – Survey respondents' suggestions on MAS development SDLC

The reasons cited for the suitability of the "*Iterative and incremental SDLC*" to the development of MAS were synthesized as follows.

- Iteration is crucial to the development of non-trivial systems. Such systems cannot be built at one shot, but part by part, step by step, and functionality by functionality.
- Iterative and incremental cycle is the best way to prevent risks and facilitate maintenance.
- "Iteration" allows refinements to be made in an organised, predictable way. "Increments" allow for short delivery cycles and enhance project visibility.
- New agents may appear while others are made obsolete as the system continuously evolves. System functionality also needs to be refined or enhanced.

- MASs are generally more evolvable and dynamic than most other system models. It is therefore not desirable (or feasible) to define up-front everything the system is meant to do.

The reasons collected for the other SDLC models were as follows.

- *Spiral:*
  - Artificial intelligence is an empirical domain of science, and spiral SDLC is well suited to hypothesis verification.
  - MAS will typically be used for multiple generations of a product. This is essentially the learning model that is fostered by the spiral development model.
- *Evolutionary prototyping:*
  - Agent technology allows dynamic/evolving systems and naturally these systems should be developed similarly.
  - MAS development should deploy the power of distributed development and gradual system expansion instead of centralised heavyweight design effort.
  - MAS development should be open-ended with scope for dealing with unanticipated goals and discoveries.
- *Extreme programming:*
  - The development of agent systems requires iterative, frequent tests.
- *Rational Unified Process:*
  - The Rational Unified Process (i.e. iteration with shifting emphasis) is fairly generic and realistic.
  - It is well supported, well documented and well known.

Issue 2: Commitment to agent architecture

When asked "*Please indicate the importance of an AOSE methodology to commit to a particular agent architecture (e.g. BDI architecture)*", a large proportion of the respondents (17 out of 41) rated the importance as "Medium" (Figure 5.18).

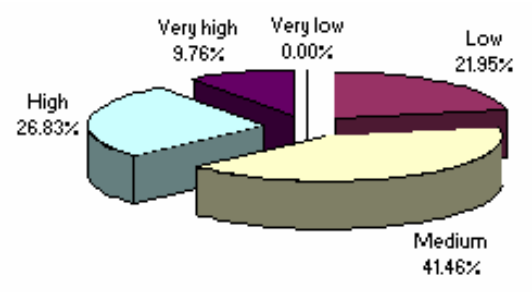


Figure 5.18 – Survey respondents' suggestions on the importance of a MAS development methodology to commit to an agent architecture

Qualitative analysis of the respondents' comments on this question revealed various reasons for, and against, the need to commit to an agent architecture by a MAS development methodology. The respondents' comments also included various suggestions on the matter.

- "*For*":
  - It is not the goal of a MAS development methodology to be universal.
  - It is necessary for a MAS development methodology to aim for a particular implementation platform (or at least "style") to provide useful guidelines in relation to implementation.
  - Many different architectures/implementation models are called "agent". Thus a clear commitment to a (set of) agent architectural model is needed.

- "*Against*":
  - Any kind of agent development toolkit and architecture should be appropriate to be used for implementing the produced design models.
  - The selection of target agent architecture should be a strategic decision made outside the development cycle of any specific MAS development project.
  - MASs should be able to integrate and coordinate agents of many kinds.
  - Any extension of functionality of an evolving MAS could involve a new agent architecture.
  - Flexibility is a very important factor of a system development methodology. Diverse architectural models would make a methodology rich.

- "*Suggestion*":
  - Analysis and architectural design of MAS should be architecture independent. However the detailed, internal design of each agent should use modular agent-oriented components/features that are specific to a particular agent architecture.
  - A MAS development methodology can be "componentised". That is, the choice of agent architecture only affects parts of the methodology. Changing from one architecture style/model to another would only require adapting a part of the methodology.
  - A MAS development methodology may provide ready-made architectural styles that can be reused by its users.

Issue 3: Approach for agent identification

The survey presented two major approaches for MAS development:

- *Role-oriented approach*: where "role" is employed as a major modelling concept and is used, for example, for the identification of agents; and
- *Non-role-oriented approach*: where "role" is not used anywhere in the MAS development process. Agents, for example, can be identified from other constructs such as use case scenarios, task specifications and workflow models.

Most respondents selected the first approach as the desirable method for MAS development (30 out of 41; Figure 5.19).

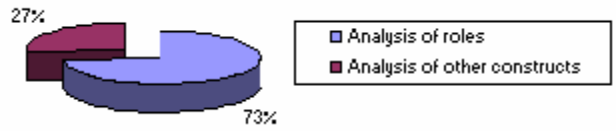


Figure 5.19 – Survey respondents' suggestions on the approaches to agent identification

The provided reasons for the role-oriented approach are listed below.

- Role provides an easy, natural way to map system aspects such as tasks, responsibilities and organisational positions onto agents.
- Agents are autonomous entities. Modelling agents as players/implementers of roles promote this autonomy.
- Using roles allows for modularity and extendibility in agent design.

- Role provides flexibility in design, since each agent may take on multiple roles, move from one role to another, or take on new roles.

Respondents who advocated the non-role-oriented approach presented the following reasons for their response.

- Other conventional constructs such as use case scenarios are more familiar to most developers. They thus help OO developers to adapt and familiarize to agent-oriented development.
- Agent-oriented development share many similarities with the conventional development paradigms such as OO. It should thus make use of (or be built upon) the conventional analysis and design constructs such as use case scenarios and workflow models.

5.4. FEATURE ANALYSIS OF EXISTING MAS DEVELOPMENT METHODOLOGIES

Following the survey on practitioners and researchers (Section 5.3), Step 3 of Research Activity 1 – "*Perform a feature analysis on the existing AOSE methodologies*" – was performed. Its aim was to further validate the potential methodological requirements of MOBMAS, and to identify and evaluate the techniques and model definitions provided by the existing AOSE methodologies for supporting these requirements (cf. Section 4.4.3).

This section firstly presents the evaluation framework of the feature analysis (Section 5.4.1) and the feature analysis' findings (Section 5.4.2). Based on these findings, the research then determined the "actual" methodological requirements of MOBMAS (Section 5.4.3) and identified a pool of techniques and model definitions that may be *reused* or *enhanced* by MOBMAS, as well as the methodological requirements that need to be supported by *new* techniques and/or model definitions (Section 5.4.4).

5.4.1. Evaluation Framework

A feature analysis requires an "evaluation framework" which defines a set of *evaluation criteria* to serve as yardsticks for assessing a methodology from different aspects (Siau and Rossi 1998). This research's evaluation framework was built directly upon the list of *potential requirements of MOBMAS* (i.e. the list of features, steps and modelling concepts identified in Section 5.2). Each criterion assesses *whether* an existing AOSE methodology provides support for a particular feature, step or modelling concept, and/or *how* the support is provided (i.e. the techniques and model definitions used by the existing methodology to support the feature, step or modelling concept).

Apart from these criteria, a small number of other criteria were included into the evaluation framework to assess the *ease of understanding* and *usability* of the development process, techniques and model definitions of the existing AOSE methodologies. One new criterion was also defined to explore the *approach towards MAS development* of the existing AOSE methodologies (namely, role-oriented approach or non-role-oriented approach). The final structure of the evaluation framework is shown in Figure 5.20.

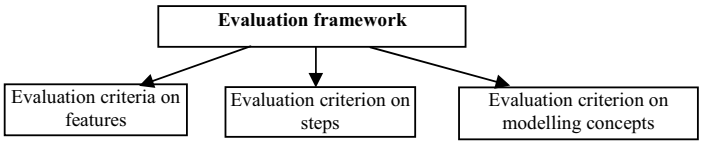


Figure 5.20 – Evaluation framework

- **Evaluation criteria on features** (Table 5.21): include 36 criteria that evaluate the support of an AOSE methodology for:
  - features relating to AOSE process;
  - features relating to AOSE model definitions;
  - agent properties; and
  - features relating to the methodology as a whole.

  These features and agent properties are obtained from Section 5.2.1.
- **Evaluation criterion on steps** (Table 5.22): includes one criterion that examines whether an AOSE methodology provides support for:
  - particular Problem Domain Analysis steps;

- particular Agent Interaction Design steps;
- particular Agent Internal Design steps; and
- particular Overall System Design steps.

These steps are obtained from Section 5.2.2.

- **Evaluation criterion on modelling concepts** (Table 5.23): includes one criterion that determines whether an AOSE methodology provides support for:
  - particular Problem Domain concepts;
  - particular Agent concepts;
  - particular Agent Interaction concepts; and
  - particular Overall System Design concepts.

  These modelling concepts are obtained from Section 5.2.3.

Each criterion is accompanied by an evaluation question, as presented in column "*Evaluation Questions*" of Tables 5.24, 5.25 and 5.26. Criteria marked with asterisk (*) are those that do not correspond directly to any potential requirements of MOBMAS, but were included to assess the usability of the methodology or to investigate the methodology's approaches towards MAS development as mentioned previously.

Even though developed particularly for this research, the above evaluation framework is applicable to the evaluation of any AOSE methodology. It has been published in Tran et al. (2003) and applied to the comparative analysis of various AOSE methodologies (Tran et al. 2004; Tran and Low 2005).

Table 5.21 – Evaluation criteria on features

Evaluation Criteria	***Evaluation Questions***
Evaluation criteria on features relating to AOSE process	
1. Specification of a system development lifecycle	What development lifecycle best describes the methodology (e.g. waterfall or iterative)?
2. Support for verification and validation	Does the development process of the methodology contain rules to allow for the verification and validation of the correctness of the developed models?
3. Specification of steps for the development process	Does the development process of the methodology define specific steps for MAS development?
4. Specification of model kinds and/or notational components	What model kinds (and/or notational components) are generated from each step?
5. Definition of inputs and outputs for steps*	Are inputs and outputs to each process step defined?
6. Specification of techniques and heuristics	a. What are the techniques used to perform each process step? b. What are the techniques used to produce each model kind or notational component (i.e. modelling techniques)?

7. Ease of understanding of techniques*	Are the techniques easy to understand?
8. Usability of techniques*	Are the techniques easy to follow
9. Provision of examples for techniques*	Are examples of the techniques provided?
10. Ease of understanding of the development process*	Do the steps result in a development process that is easy to understand?
11. Usability of the development process*	Do the steps result in a development process that is easy to follow?
12. Support for refinability	Do the process steps provide a clear path for refining models through gradual stages to reach an implementation, or at least for clearly connecting the implementation level to the design specification?
13. Approach for MAS development*	Does the methodology employ the abstraction of "role" in MAS analysis and design?
Evaluation criteria on feature relating to AOSE model definitions	
1. Completeness/ expressiveness	Are the model kinds of the methodology capable of representing the system from different perspectives, capturing all necessary aspects such as static and dynamic aspects, system-level and agent-level aspects?
2. Formalisation/preciseness	a. Are syntax and semantics of the model kinds and notational components clearly defined? b. Are examples of the model kinds and notational components presented?
3. Provision of guidelines/logics for model derivation	Do explicit process and guidelines exist for transforming model kinds into other model kinds or for partially creating a model kind from information present in another model kind?
4. Guarantee of consistency	a. Are there rules and guidelines to ensure consistency between the levels of abstractions within each model kind (i.e. internal consistency), and between different model kinds? b. Are model kinds represented in a manner that allows for consistency checking between them?
5. Support for modularity	Do the methodology and its model kinds promote modularity in the design of agents and the system?
6. Management of complexity	Are there a manageable number of concepts expressed in each model kind/notational component?
7. Levels of abstraction	Does the methodology allow for producing models at various levels of detail and/or abstraction?
8. Support for reuse	Does the methodology provide, or make it possible to use, a library of reusable models?
9. Ease of understanding of model definitions*	Are the model kinds and notational components clear and easy to understand?
Evaluation criteria on agent properties	
1. Autonomy	Can the model kinds support and represent the autonomous feature of agents (i.e. the ability to act without direct intervention of humans or others, and to control their own states and behaviour)?
2. Adaptability	Can the model kinds support and represent the adaptability feature of agents (i.e. the ability to learn and improve with experience)?
3. Cooperative behaviour	Can the model kinds support and represent the cooperative behaviour of agents (i.e. the ability to work together with other agents to achieve a common goal)?
4. Inferential capability	Can the model kinds support and represent the inferential capability feature of agents (i.e. the ability to reason and act on abstract task specifications)?
5. Knowledge-level communication ability	Can the model kinds support and represent a "knowledge-level" communication ability (i.e. the ability to communicate with other agents with language resembling human-like speech acts)?
6. Personality	Can the model kinds support and represent the personality of agents (i.e. the ability to manifest attributes of a "believable" human character)?

7. Reactivity	Can the model kinds support and represent the reactivity of agents? (i.e. the ability to selectively sense and act in a timely manner)
8. Deliberative behaviour	Can the model kinds support and represent the deliberative behaviour of agents (i.e. the ability to decide in a deliberation, or proactiveness)?
Evaluation criteria on features relating to methodology as a whole	
1. Support for open systems	Does the methodology provide support for open systems (open systems are those that allow for dynamic addition/removal of agents)?
2. Support for dynamic systems	Does the methodology provide support for dynamic structure (i.e. the methodology allows for dynamic reconfiguration of the system, e.g. change of roles of agents or change or organisational structure of MAS)?
3. Support for agility and robustness	Does the methodology provide support for agility and robustness (e.g. the methodology captures normal processing and exception processing, provides techniques to analyse system performance for all configurations, or provides techniques to detect and recover from failures)?
4. Support for heterogeneous systems	Does the methodology provide support for the use/incorporation of (heterogeneous) non-agent software components in the system?
5. Support for mobile agents	Does the methodology provide support for the use/integration of mobile agents in a MAS (e.g. the methodology models which/when/how agent should be mobile)?
6. Support for ontology-based MAS development	Does the methodology provide support for the use and specification of ontology in a MAS (i.e. Ontology-Based MAS)?

Table 5.22 – Evaluation criterion on steps

Evaluation Criterion	*Evaluation Question*
Support for steps	Which of the following steps are supported by the development process of the methodology? **Problem Domain Analysis steps** 1. Identify system functionality 2. Specify use case scenarios 3. Identify roles 4. Identify agent classes 5. Model domain conceptualisation **Agent Interaction Design steps** 1. Specify acquaintances between agent classes 2. Define interaction protocols 3. Define content of exchanged messages 4. Specify agent communication language **Agent Internal Design steps** 1. Specify agent architecture 2. Define agent informational constructs (i.e. beliefs) 3. Define agent behavioural constructs (e.g. goals, plans, actions, services) **Overall System Design steps** 1. Specify system architecture (i.e. overview of all system components and their connections) 2. Specify organisational structure/inter-agent authority relationships 3. Model MAS environment (e.g. resources, facilities) 4. Specify agent-environment interaction mechanism 5. Specify agent inheritance and aggregation 6. Instantiate agent classes 7. Specify agent instances deployment

Table 5.23 – Evaluation criterion on modelling concepts

Evaluation Criterion	*Evaluation Question*
Support for modelling concepts	Which of the following concepts are captured/represented by the model kinds of the methodology? **Problem Domain concepts** 1. System functionality 2. Use case scenario 3. Role 4. Domain conceptualisation **Agent concepts** 1. Agent-role assignment 2. Agent goal/task 3. Agent belief/knowledge 4. Agent plan/reasoning rule/problem solving method 5. Agent capability/service 6. Agent percept/event 7. Agent architecture **Agent Interaction concepts** 1. Agent acquaintance 2. Interaction protocol 3. Content of exchanged messages **Overall System Design concepts** 1. System architecture 2. Organisational structure/inter-agent authority relationships 3. Environment resource/facility 4. Agent aggregation relationship 5. Agent inheritance relationship 6. Agent instantiation 7. Agent instance deployment

5.4.2. Feature Analysis of Existing MAS Development Methodologies

In this section, the sixteen AOSE methodologies described in Chapter 3 are evaluated using the evaluation framework presented in Section 5.4.1. The analysis of ten of these methodologies has been published in Tran et al. (2004) and Tran and Low (2005), namely MASE, GAIA, MESSAGE, INGENIAS, BDIM, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS and TROPOS.

5.4.2.1. Evaluation of support for Features

❖ **Evaluation of support for features relating to AOSE process**

Of the fourteen evaluation criteria in this category (cf. Table 5.21), the following six criteria are discussed in Section 5.4.2.2 alongside criterion "*Support for steps*", because these criteria needed to use the list of steps in Table 5.22 as yardsticks:

"*Specification of model kinds and/or notational components*", "*Definition of inputs and outputs for steps*", "*Specification of techniques and heuristics*", "*Ease of understanding of techniques*", "*Usability of techniques*" and "*Provision of examples for techniques*". Evaluation of the remaining eight criteria is presented in Table 5.24.

Most of the evaluation results are self-explanatory, except for the following three criteria whose assessment is further clarified below.

- "*Usability of the development process*": This research rated a methodology as "Medium" or "Low" if the methodology disregards many steps in the construction of MAS[29] and/or fails to provide sufficient techniques to guide the performance of its steps and/or the construction of its model kinds.
- "*Approach for MAS development*": This research classified an AOSE methodology as Non-Role-Oriented ("NRO") if the methodology does not involve the use of abstraction "role" anywhere in its MAS development process. A Role-Oriented methodology ("RO"), on the other hand, employs "role" as a major modelling concept.

❖ **Evaluation of support for features relating to AOSE model definitions**

Nine evaluation criteria were used to conduct this evaluation (cf. Table 5.21). Again, the evaluation results are self-explanatory (Table 5.25). It should be noted that:

- regarding criterion "*Completeness/expressiveness*", a methodology was rated "High" if it offers a comprehensive set of model kinds and notational components to represent the target system from both static and dynamic aspects, and to capture a large variety of concepts[30];
- regarding criterion "*Support for modularity*", a methodology was evaluated "Yes" if it models agents as an encapsulation of either roles, goals, tasks/responsibilities, knowledge modules and/or capabilities; and
- regarding criterion "*Support for reuse*", a methodology was rated "Yes" if it *explicitly* provides a library of reusable modelling components (such as role patterns, protocol templates, knowledge modules and/or behavioural patterns), or at least discusses the possibility of reusing certain modelling components. A

[29] That is, the steps listed in Table 5.22.
[30] That is, the concepts listed in Table 5.23.

methodology was rated "Possibly" if it does not explicitly address the issue of reuse, nevertheless allows the developer to reuse modelling components.

❖ **Evaluation of support for agent properties**

Nine agent properties were investigated in total (cf. Table 5.21). The assessment of methodological support for these properties is presented in Table 5.26. Some notable findings are presented below.

- All sixteen methodologies were found to support "*autonomy*" via the modelling of agents as entities with purpose (represented as roles, goals, tasks and/or capabilities) and/or entities with internal control (represented as knowledge, plans, inference rules and/or problem solving methods).
- Two methodologies were found to touch on the issue of agent adaptability: MESSAGE recommends selecting a cognitive agent architecture if the agent needs to learn, and INGENIAS mentions the need to specify "learning" as a characteristic of an agent if applicable. It should be noted that while ADELFE supports adaptability at the system level, it does not address the issue of adaptability at the agent level.
- A majority of the existing methodologies (11 out of 16) were found to support "*inferential capability*" via the specification of agent beliefs/knowledge, plans, aptitudes, methods, agent control process and/or agent behavioural knowledge/expertise.
- A majority of the methodologies (11 out of 16) support "*reactivity*" through the explicit modelling of "events" that incur during agent interactions and/or agent internal processing. These methodologies also explicitly model reactive behaviour for agents. Five other methodologies were found to "possibly*"* support reactivity because, even though they do not explicitly discuss the modelling of events and agent reactive behaviour, these elements may have been embedded in the specification of agent interaction protocols, agent dependencies and agent responsibilities/plans/competence.
- Most of the methodologies (15) were found supportive of "*deliberative behaviour*" via the modelling of agents as entities with purposes (represented as agent goals, tasks and/or capabilities). Eleven methodologies also specify how agents fulfil these purposes, either via agent plans (BDIM, HLIM, MEI,

TROPOS and PROMETHEUS), methods/capabilities (PASSI and ADELFE), control procedures/rules (INGENIAS) or knowledge/expertise (COMOMAS, MAS-CommonKADS and MESSAGE).

❖ **Evaluation of support for features relating to the methodology as a whole**

There are six high-level, supplementary features that pertain to the MAS development methodology as a whole (cf. Table 5.21). Evaluation of these features' support is presented in Table 5.27. Notable findings are discussed below.

- "*Open systems*" were supported by only three methodologies, SODA, GAIA and ADELFE, via the modelling of resources and services offered by MAS environment (SODA and GAIA), specification of organisational rules to govern agent interactions and behaviour (SODA and GAIA) and modelling of potential "non-cooperative situations" between agents (ADELFE). MASSIVE "possibly" supports open systems since it mentions the characterisation of openness of the target agent society.
- "*Dynamic systems*" were supported by only four methodologies: MASSIVE, HLIM and PASSI model the dynamic assignment of roles to agents; CASSIOPEIA defines the behaviour of agents in dynamically forming, joining and dissolving agent groups. MASE "possibly" support dynamic systems because it acknowledges that agents can change roles dynamically, although it does not deal with this issue in any detail.
- "*Agility and robustness*" were supported by only five methodologies. ADELFE and MASSIVE identify potential failure situations of the system and specify the mechanisms to deal with them. PROMETHEUS, MASE and MAS-CommonKADS identify exceptional situations in interaction protocols and use cases; however they do not specify any exception handling mechanisms.
- "*Support for heterogeneous systems*" was provided by five methodologies. Four of them (INGENIAS, PROMETHEUS, GAIA and MASSIVE) mention the existence of non-agent objects and application systems in MAS, but do not discuss how the heterogeneous components of MAS can be supported. MASE does not consider non-agent system components, but addresses the interoperability between heterogeneous agents.

- "*Mobile agents*" is only supported by PASSI which models agent movement in its Deployment Configuration Diagram.
- "*Support for ontology-based MAS development*" was provided by only four methodologies (MASE, MESSAGE, PASSI and MAS-CommonKADS). Detailed discussion of their ontology support has been presented in Section 3.3.2.

Table 5.24 – Evaluation of support for features relating to AOSE process

	Specification of a system development (dev.) lifecycle	Support for verification & validation	Specification of steps for the dev. process	Ease of understanding of the dev. Process	Usability of the dev. process	Support for refinability	Approach for MAS dev.
MASE	Iterative across all phases	Yes	Yes	High	High	Yes	RO
MASSIVE	Iterative View Engineering process	Yes	Yes	High	Medium	Yes	RO
SODA	Not specified	No	Yes	High	Low	No	RO
GAIA	Iterative within each phase but sequential between phases	No	Yes	High	Medium	Yes	RO
MESSAGE	Rational Unified Process	Mentioned as future enhancement	Yes	High	Medium	Yes	RO
INGENIAS	Unified software development process	Yes	Yes	High	High	Yes	RO
BDIM	Not specified	No	Yes	High	Low	Yes	RO
HLIM	Iterative within and across the phases	No	Yes	High	High	Yes	RO
MEI	Not specified	No	Yes	High	High	Yes	NRO
PROMETHEUS	Iterative across all phases	Yes	Yes	High	High	Yes	NRO
PASSI	Iterative across and within all phases (except for coding and deployment)	Yes	Yes	High	High	Yes	RO
ADELFE	Rational Unified Process	Yes	Yes	High	High	Yes	NRO
COMOMAS	Not specified	No	Yes	High	Medium	Yes	NRO
MAS-CommonKADS	Cyclic risk-driven process	Mentioned but no clear guidelines	Yes	High	Medium	Yes	NRO
CASSIOPEIA	Not specified	No	Yes	High	Medium	Yes	RO
TROPOS	Iterative and incremental	Yes	Yes	High	Medium	Yes	NRO

Table 5.25 – Evaluation of support for features relating to AOSE model definitions

	Completeness/ expressiveness	Formalization/ preciseness	Provision of guidelines/ logics for model derivation	Guarantee of consistency	Support for modularity	Management of complexity	Levels of abstraction	Support for reuse	Ease of understanding of model definitions
MASE	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Yes	High
MASSIVE	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Yes	High
SODA	Medium	a. Medium b. Yes	Yes	a. Yes b. Yes	Yes	Yes	No	Possibly	High
GAIA	Medium	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Yes	High
MESSAGE	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Possibly	High
INGENIAS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	No	Yes	Possibly	Medium
BDIM	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Yes	High
HLIM	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
MEI	Medium	a. Low b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Possibly	Medium
PROMETHEUS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
PASSI	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	No	Yes	High
ADELFE	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
COMOMAS	High	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	No	Possibly	High
MAS-CommonKADS	High	a. Medium b. Yes	No	a. No b. Yes	Yes	Yes	Yes	Yes	Medium
CASSIOPEIA	Medium	a. High b. Yes	NA	a. No b. NA	Yes	Yes	No	Possibly	High
TROPOS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High

Table 5.26 – Evaluation of support for agent properties

	Autonomy	Adaptability	Cooperative behaviour	Inferential capability	Knowledge-level communication ability	Personality	Reactivity	Deliberative behaviour
MASE	Yes	No	Yes	Possibly	Yes	No	Yes	Yes
MASSIVE	Yes	Yes	Yes	No	No	No	Possibly	Yes
SODA	Yes	No	Yes	No	No	No	Possibly	Yes
GAIA	Yes	No	Yes	No	No	No	Possibly	Yes
MESSAGE	Yes	No	Yes	Yes	Yes	No	Yes	Yes
INGENIAS	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
BDIM	Yes	No	Yes	Yes	Yes	No	Yes	Yes
HLIM	Yes	No	Yes	Yes	Yes	No	Yes	Yes
MEI	Yes	No	Yes	Yes	No	No	Possibly	Yes
PROMETHEUS	Yes	No	Yes	Yes	Yes	No	Yes	Yes
PASSI	Yes	No	Yes	Yes	Yes	No	Yes	Yes
ADELFE	Yes	No	Yes	Yes	Yes	No	Yes	Yes
COMOMAS	Yes	No	Yes	Yes	No	No	Yes	Yes
MAS-CommonKADS	Yes	No	Yes	Yes	Yes	No	Yes	Yes
CASSIOPEIA	Yes	No	Yes	No	No	No	Possibly	No
TROPOS	Yes	No	Yes	Yes	Yes	No	Yes	Yes

Table 5.27 – Evaluation of support for features relating to the methodology as a whole

	Support for open systems	Support for dynamic systems	Support for agility & robustness	Support for heterogeneous systems	Support for mobile agents	Support for ontology-based MAS development
MASE	No	Possibly	Yes	Yes	No	Yes
MASSIVE	Possibly	Yes	Yes	Yes	No	No
SODA	Yes	No	No	No	No	No
GAIA	Yes	No	No	Yes	No	No
MESSAGE	No	No	No	No	No	Yes
INGENIAS	No	No	No	Yes	No	No
BDIM	No	No	No	No	No	No
HLIM	No	Yes	No	No	No	No
MEI	No	No	No	No	No	No
PROMETHEUS	No	No	Yes	Yes	No	No
PASSI	No	Yes	No	No	Yes	Yes
ADELFE	Yes	No	Yes	No	No	No
COMOMAS	No	No	No	No	No	No
MAS-CommonKADS	No	No	Yes	No	No	Yes
CASSIOPEIA	No	Yes	No	No	No	No
TROPOS	No	No	No	No	No	No

5.4.2.2. Evaluation of support for Steps

In this section, the sixteen methodologies are evaluated according to criterion "*Support for steps"* (cf. Table 5.22), together with six other criteria that pertain to the AOSE process and had not been discussed in section 5.4.2.1, namely:

- "*Specification of model kinds and/or notational components*";
- "*Definition of inputs and outputs of steps*";
- "*Specification of techniques and heuristics*";
- "*Ease of understanding of techniques*";
- "*Usability of techniques*"; and
- "*Provision of examples for techniques*".

All of these criteria used the list of steps in Table 5.22 as yardsticks. Evaluation findings of each methodology are presented in Appendix D. Table 5.28 provides a bird-eye view of the methodologies' support for steps by showing only the assessment of criterion "*Usability of techniques"*. Usability was evaluated as either high ("H"), medium ("M") or Low ("L").

Table 5.28 – Evaluation of "*Usability of techniques*"

	1. Identify system functionality	2. Specify use case scenarios	3. Identify roles	4. Identify agent classes	5. Model domain conceptualisation	6. Specify acquaintances between agent classes	7. Define interaction protocols	8. Define content of exchanged messages	9. Specify agent communication language	10. Specify agent architecture	11. Define agent informational constructs	12. Define agent behavioural constructs	13. Specify system architecture	14. Specify organisational structure/inter-agent authority relationships	15. Model MAS environment	16. Specify agent-environment interaction mechanism	17. Specify agent inheritance & aggregation	18. Instantiate agent classes	19. Specify agent instances deployment
MASE	H	H	H	H	H	H	H	H		M	M							H	H
MASSIVE	H		H	L		L	H			H			M	H	H	M			
SODA	L		M	M			H								M				
GAIA	M		H	H		M	H					H		H	M		H	H	
MESSAGE	H		M	M	M	M	H	L	M	H	H		H	H	L			L	
INGENIAS	H	H	M	H		H	H	M			H		H	H	H	H			
BDIM			L	H		L		L			H	M					H	M	
HLIM	H	H	M	M		M	H	M			H	L		H					
MEI	H	H		H		H	H				M					H			
PROME-THEUS	H	H		H		H	H	L		H	H	H	H		M	H		L	
PASSI	H	H	H	M	M	H	H	H		H	M	H	H						L
ADELFE	H	H		H		M	M	M	M	H	H	M	H		H	L	M		
COMOMAS	M			M			L				M			L					
MAS-COMMONA KDS	H	H		M	M	H	H	H		L	M	L	M	L	M		M	L	
CASSIO-PEIA	H		M	M		H	L							M					
TROPOS	H			H		M	H	M			M			H	H				

5.4.2.3. Evaluation of support for Modelling Concepts

In this section, the sixteen AOSE methodologies are evaluated according to criterion "*Support for modelling concepts*" (cf. Table 5.23). This criterion used the list of modelling concepts in Table 5.23 as yardsticks. Evaluation results are presented in Tables 8.29a and b. If a methodology was found to support a particular modelling concept, the name of the model kind or notational component capturing the concept is displayed. If a concept appears in many different model kinds or notational components of the methodology, only the model kind/notational component that represents the concept as the principal modelling element is shown.

Table 5.29a – Evaluation of support for modelling concepts (part a)

	ADELFE	COMO-MAS	MAS-CommonKADS	CASSIO-PEIA	TROPOS
1. System functionality	Use case model	Task model	Task model		Actor diagram, Rationale diagram
2. Use case scenario	Use case model		Use cases		
3. Role				Coupling graph	
4. Domain conceptualisation			Expertise model		
5. Agent-role assignment			Agent model	Coupling graph	
6. Agent goal/task	Detailed architecture document	Agent model	Agent model		Actor diagram
7. Agent belief/knowledge	Detailed architecture document	Expertise model, Agent model	Expertise model		Agent class diagram
8. Agent plan/reasoning rule/problem solving method			Expertise model		Plan diagram
9. Agent capability/service	Detailed architecture document		Agent model, Organisation model		
10. Agent percept/event			State transition diagram of Coordination model		
11. Agent architecture	Detailed architecture document		Design model		BDI architecture
12. Agent acquaintance	Software architecture document		Coordination model	Coupling graph	Sequence diagram /Collaboration diagram
13. Interaction protocol	Interaction language document	Cooperative model	Coordination model		Sequence diagram
14. Content of exchanged messages	Interaction language document		Coordination model		Sequence diagram /Collaboration diagram
15. System architecture	Detailed architecture document		Organisation model		
16. Organisational structure/inter-agent authority relationships		System model	Organisation model		Non-functional requirements framework
17. Environment resource/facility	Environment definition document		Organisation model, Design model		
18. Agent aggregation relationship	Detailed architecture document		Organisation model		
19. Agent inheritance relationship			Organisation model		
20. Agent instantiation			Organisation model		
21. Agent instance deployment					

Table 5.29b – Evaluation of support for modelling concepts (part b)

	MASE	MASSIVE	SODA	GAIA	MESSAGE	INGENIAS	BDIM	HLIM	MEI	PROMETHEUS	PASSI
1. System functionality	Extended role diagram	Task view	Role model	Role model	Goal/Task model	Goal & Task model		High level model	IDEF/CIMOSA Function Model	Functionality descriptor	System requirement model
2. Use case scenario	Use case diagram					Use case diagram		High level model	Use case model	Use case descriptor	System requirement model
3. Role	Role diagram	Role view	Role model	Role model	Agent/Role model	Agent model, Organisation model, Interaction model		High level model			System requirement model, Agent society model
4. Domain conceptualisation	Ontology				Domain view						Agent society model
5. Agent-role assignment	Agent class diagram	Role view	Agent model	Agent model	Agent/Role view	Agent model					Agent society model
6. Agent goal/task						Agent model	Goal model	Internal agent model	Goal-Plan diagram	Agent class descriptor	System requirement model
7. Agent belief/knowledge						Can be recorded in mental states, but not mentioned in methodology	Belief model	Internal agent model		Data descriptor	Agent implementation model
8. Agent plan/reasoning rule/problem solving method	Task state diagram					Agent model	Plan model	Internal agent model	Plan state diagram	Plan descriptor	Agent implementation model
9. Agent capability/service				Service model				Contract model		Capability diagram	Agent society model
10. Agent percept/event		Environment view								Percept descriptor	
11. Agent architecture	Agent class architecture diagram	Architectural view								Agent overview diagram	Agent implementation model
12. Agent acquaintance	Agent class diagram	Interaction view	Interaction model	Acquaintance model	Organisation model	Interaction model	Interaction model	Conversation model		Interaction diagram	System requirement model
13. Interaction protocol	Communication class diagram	Interaction view	Interaction model	Interaction model	Interaction model	Interaction model		Conversation model	Coordination protocol script	Interaction protocol	Agent society model
14. Content of exchanged messages	Communication class diagram				Interaction model	Interaction model	Interaction model	Conversation model		Interaction diagram & protocol	Agent implementation model
15. System architecture		Architectural view			System architecture diagram	Organisation model				System overview diagram	Agent implementation model
16. Organisational structure/inter-agent authority relationships		Society view		Organisational structure model	Organisation model	Organisation model		Agent relationship model			
17. Environment resource/facility			Resource model	Environmental model	Organisation model	Environment model				System overview diagram	
18. Agent aggregation relationship				Agent model			Agent model				
19. Agent inheritance relationship							Agent model				
20. Agent instantiation	Deployment diagram		Agent model	Agent model			Agent model			Agent class descriptor	
21. Agent instance deployment	Deployment diagram		Agent model								Deployment model

5.4.3. Actual Requirements of MOBMAS

In this section, the feature analysis findings were combined with the results of the survey in Section 5.3 to determine the "*actual*" requirements of MOBMAS (cf. Figure 4.3). Specifically, a potential requirement was determined to be an actual requirement of MOBMAS if:

- it was supported by a majority of the existing AOSE methodologies (i.e. 9 or more out of 16): A methodology is considered supportive of a feature, step or modelling concept if it was evaluated "*Yes"* or "*High*" for the respective evaluation criterion (depending on whether the criterion is a yes/no question or a high/medium/low rating question); OR
- it was given a High to Very High "rating of importance" in the survey; OR
- it was given a Medium "rating of importance" in the survey *AND* its "order rank" is *not* the least important with respect to other requirements within the same category.

All potential requirements that did not match these criteria were excluded from the list of actual requirements of MOBMAS.

Tables 5.33, 5.34 and 5.35 below extend Tables 5.14, 5.15 and 5.16 in Section 5.3 to show the "*Number of existing methodologies that support* [each] *feature*/*step*/*modelling concept*" and the selection of MOBMAS actual requirements from the list of potential requirements. Actual requirements are displayed in normal font while discarded potential requirements are displayed in *italic*.

Table 5.30 – Selection of MOBMAS' "actual" features

Features desirable to MOBMAS development process	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the feature** (out of 16)
1. Specification of model kinds and/or notational components	Very high	3.34	16
2. Specification of steps for the development process	Very high	3.46	16
3. Specification of techniques and heuristics for performing each process step and producing each model kind	Very high	3.78	16
4. Support for verification and validation	Very high	3.95	7
5. Support for refinability	Very high	4.26	15
6. Specification of a system development lifecycle	Very high	4.43	11
Features desirable to MOBMAS model definitions	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the feature** (out of 16)
1. Guarantee of consistency	Very high	3.52	9
2. Model kinds expressed at various level of abstraction and detail	Very high	3.78	12
3. Support for reuse	Very high	3.80	6
4. High degree of completeness/expressiveness	Very high	3.81	9
5. Manageable number of concepts in each model kind and each notational component	Very high	4.06	15
6. Support for modularity	Very high	4.23	16
7. High degree of formalisation/preciseness	Very high	4.41	13
8. Provision of guidelines/logics for model derivation	Very high	4.49	14
Agent properties desirable to be captured/represented by MOBMAS model kinds	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the feature** (out of 16)
1. Autonomy	Very high	2.54	16
2. Cooperative behaviour	Very high	3.46	16
3. Deliberative behaviour	Very high	3.92	15
4. Knowledge-level communication ability	Very high	4.14	10
5. Inferential capability	Very high	4.34	11
6. Reactivity	Very high	4.75	10
7. Adaptability	Very high	5.48	2
8. *Personality*	Medium	7.98	0
Features desirable to MOBMAS as a whole	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the feature** (out of 16)
1. Support for dynamic systems	Very high	3.34	5
2. Support for open systems	Very high	3.78	3
3. Support for ontology-based MAS development	Very high	3.87	4
4. Support for heterogeneous systems	High	4.95	5
5. *Support for agility and robustness*	Medium	6.26	6
6. *Support for mobile agents*	Medium	6.71	1

Table 5.31 – Selection of MOBMAS' "actual" steps

Problem Domain Analysis steps	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the step** (out of 16)
1. Identify system functionality	Very high	2.05	15
2. Identify agent classes	Very high	3.49	16
3. Model domain conceptualisation	Very high	3.61	4
4. Identify roles	High	3.68	10
5. *Specify use case scenarios*	Medium	3.72	7
Agent Interaction Design steps	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the step** (out of 16)
1. Define interaction protocols	Very high	1.02	15
2. Specify acquaintances between agent classes	Very high	1.54	16
3. Define content of exchanged messages	Very high	2.51	10
4. *Specify agent communication language*	Medium	3.75	3
Agent Internal Design steps	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the step** (out of 16)
1. Define agent informational constructs	Very high	1.78	12
2. Define agent behavioural constructs	Very high	2.10	6
3. Specify agent architecture	Very high	2.34	11
Overall System Design steps	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the step** (out of 16)
1. Specify system architecture	Very high	1.90	8
2. Specify organisational structure/inter-agent authority relationships	Very high	2.05	8
3. Model MAS environment	Very high	2.90	9
4. Specify agent-environment interaction mechanism	High	3.90	6
5. Instantiate agent classes	High	4.23	7
6. Specify agent instances deployment	High	4.89	3
7. *Specify agent inheritance and aggregation*	Medium	5.95	4

Table 5.32 – Selection of MOBMAS' "actual" modelling concepts

Problem Domain concepts	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the concept** (out of 16)
1. System functionality	Very high	1.82	14
2. Role	Very high	2.56	9
3. Domain conceptualisation	High	2.80	4
4. *Use case scenario*	Very high	3.73	7
Agent concepts	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the concept** (out of 16)
1. Agent belief/knowledge	Very high	3.51	8
2. Agent goal/task	Very high	3.63	10
3. Agent-role assignment	Very high	3.90	9
4. Agent plan/reasoning rule/problem solving method	Very high	4.05	9
5. Agent architecture	High	4.48	8
6. *Agent capability/service*	Medium	6.05	6
7. *Agent percept/event*	Medium	6.52	3
Agent Interaction concepts	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the concept** (out of 16)
1. Interaction protocol	Very high	1.73	14
2. Content of exchanged messages	Very high	2.22	10
3. Agent acquaintance	Very high	2.49	14
Overall System Design concepts	**Median Rating of Importance**	**Mean Rank**	**Number of existing methodologies that support the concept** (out of 16)
1. System architecture	Very high	1.75	7
2. Organisational structure/inter-agent authority relationships	Very high	1.90	7
3. Environment resource/facility	Very high	2.90	7
4. Agent instance deployment	High	4.87	3
5. Agent instantiation	High	5.01	6
6. *Agent aggregation relationship*	Medium	6.49	4
7. *Agent inheritance relationship*	Medium	6.85	2

5.4.4. Potential Sources of Techniques and Model Definitions for Supporting MOBMAS' Actual Requirements

The feature analysis enabled the research to identify and evaluate the techniques and model definitions provided by the existing methodologies to support each particular feature, step or modelling concept. This identification and evaluation helped the research to:

- *Identify a pool of existing techniques and model definitions that may be **reused** or **enhanced** by MOBMAS*: In Tables 5.33, 5.34 and 5.35, the potential AOSE methodologies from which MOBMAS may acquire techniques and/or model definitions are presented.
  - The listed methodologies for *features* (Table 5.33) are those that were evaluated "*Yes"* or "*High*" for the corresponding evaluation criterion.
  - The listed methodologies for *steps* (Table 5.34) are those that received a "*High*" rating for criterion "*Usability of techniques*".
  - The listed methodologies for *modelling concepts* (Table 5.35) are those that provide at least one model kind or notational component to capture the concept.

  It should be noted that if a methodology listed for a step or modelling concept was rated "No", "Medium" or "Low" in any of the following criteria, the techniques or model definitions selected from that methodology would likely need to be enhanced:
  - "Ease of understanding of the development process";
  - "Usability of the development process";
  - "Ease of understanding of techniques";
  - "Provision of examples for techniques"; and
  - "Ease of understanding of model definitions".

- *Identify MOBMAS requirements that need to be supported by **new** techniques and/or model definitions.* These are the features, steps and modelling concepts that are currently given limited support by the existing methodologies. In Tables 5.33, 5.34

and 5.35, these requirements are indicated by the phrase "*New support required*". Justification for the need for new support is presented in the parentheses.

Apart from the existing AOSE methodologies, MOBMAS also defined its techniques and model kinds by consulting the professional recommendations given by survey respondents in Section 5.3.4.5. In Tables 8.15, 8.16 and 8.17, the requirements marked with (S) are those that can be supported by examining these recommendations.

Table 5.33 – MOBMAS' required features and sources of potential techniques and/or model definitions for supporting these features

Required features of MOBMAS development process	**Potential sources of techniques and model definitions**
1. Specification of a system development lifecycle (S)	MASE, MASSIVE, GAIA, MESSAGE, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS, TROPOS
2. Support for verification and validation	MASE, MASSIVE, INGENIAS, PROMETHEUS, PASSI, ADELFE, TROPOS
3. Specification of steps for the development process	See Table 5.34
4. Specification of model kinds and/or notational components	See Table 5.35
5. Specification of techniques and heuristics for performing each process step and producing each model kind	See Table 5.34
6. Support for refinability	All methodologies except for SODA
Required features of MOBMAS model definitions	**Potential sources of techniques and model definitions**
1. High degree of completeness/expressiveness	MASE, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS, TROPOS
2. High degree of formalisation/preciseness	All methodologies except SODA, MEI and MAS-CommonKADS
3. Provision of guidelines/logics for model derivation	All methodologies except CASSIOPEIA and MAS-CommonKADS
4. Guarantee of consistency	MASE, SODA, GAIA, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE, TROPOS
5. Support for modularity	All methodologies
6. Manageable number of concepts in each model kind and each notational component	All methodologies except SODA and INGENIAS
7. Model kinds expressed at various level of abstraction and detail	All methodologies except SODA, PASSI, CASSIOPEIA and COMOMAS
8. Support for reuse	MASE, MASSIVE, GAIA, BDIM, PASSI, MAS-CommonKADS

Agent properties required to be captured/represented by MOBMAS model kinds	**Potential sources of techniques and model definitions**
1. Autonomy	All methodologies
2. Adaptability	MASSIVE, INGENIAS
3. Cooperative behaviour	All methodologies
4. Inferential capability	MESSAGE, INGENIAS, BDIM, HLIM, MEI, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS, TROPOS
5. Knowledge-level communication ability	MASE, MESSAGE, INGENIAS, BDLIM, HLIM, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS, TROPOS + **New support required** (existing methodologies do not integrate ontologies in the modelling and verification of exchanged messages)
6. Reactivity	PROMETHEUS , MASE, MESSAGE, INGENIAS, BDIM, HLIM, PASSI, ADELFE, TROPOS, COMOMAS, MAS-CommonKADS
7. Deliberative behaviour	All methodologies except CASSIOPEIA
Required features of MOBMAS as a whole	**Potential sources of techniques and model definitions**
1. Support for open systems	SODA, GAIA, ADELFE
2. Support for dynamic systems	MASSIVE, HLIM, PASSI, CASSIOPEIA
3. Support for heterogeneous systems	INGENIAS, PROMETHEUS, GAIA and MASSIVE **+ New support required** (existing methodologies do not conceptualise the content/knowledge of non-agent software components, thus failing to explicitly support the interoperability between agents and these components, or between these components themselves)
4. Support for ontology-based MAS development	MASE, MESSAGE, PASSI, MAS-CommonKADS **+ New support required** (cf. Section 3.3.2)

Table 5.34 – MOBMAS' required steps and sources of potential techniques for supporting these steps

Problem Domain Analysis steps	**Potential sources of techniques for performing steps**
1. Identify system functionality	All methodologies except SODA, GAIA, BDIM, COMOMAS
2. Identify roles	MASE, MASSIVE, GAIA, PASSI
3. Identify agent classes (S)	MASE, GAIA, INGENIAS, BDIM, MEI, PROMETHEUS, ADELFE, TROPOS
4. Model domain conceptualisation	MASE, PASSI, MESSAGE, MAS-CommonKADS **+ New support required** (existing methodologies lack detailed discussion of this step)
Agent Interaction Design steps	**Potential sources of techniques for performing steps**
1. Specify acquaintances between agent classes	MASE, INGENIAS, MEI, PROMETHEUS, PASSI, CASIOPEIA, MAS-CommonKADS
2. Define interaction protocols	All methodologies except BDIM, ADELFE, CASSIOPEIA, COMOMAS
3. Define content of exchanged messages	MASE, PASSI, MAS-CommonKADS **+ New support required** (existing methodologies do not integrate ontologies in the modelling and verification of exchanged messages)

Agent Internal Design steps	**Potential sources of techniques for performing steps**
1. Specify agent architecture	MASSIVE, MESSAGE, PROMETHEUS, PASSI, ADELFE
2. Define agent informational constructs	MESSAGE, INGENIAS, BDIM, HLIM, PROMETHEUS, ADELFE + **New support required** (existing methodologies do not integrate ontologies in the modelling and verification of agent beliefs)
3. Define agent behavioural constructs	PROMETHEUS, BDIM
Overall System Design steps	**Potential sources of techniques for performing steps**
1. Specify system architecture (i.e. overview of all system components & their connections)	MESSAGE, INGENIAS, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS
2. Specify organisational structure/inter-agent authority relationships	MASSIVE, GAIA, MESSAGE, INGENIAS, HLIM, TROPOS
3. Model MAS environment	MASSIVE, INGENIAS, ADELFE, TROPOS
4. Specify agent-environment interaction mechanism	INGENIAS, MEI, PROMETHEUS
5. Instantiate agent classes	MASE, GAIA, BDIM
6. Specify agent instances deployment	MASE

Table 5.35 – MOBMAS' required modelling concepts and sources of potential techniques and/or model definitions for supporting these concepts

Problem Domain concepts	**Potential sources of modelling techniques and model definitions**
1. System functionality	All methodologies except BDIM and CASSIOPEIA
2. Role	PASSI, MASE, SODA, GAIA, MESSAGE, INGENIAS, HLIM
3. Domain conceptualisation	PASSI, MESSAGE, MAS-CommonKADS
Agent concepts	**Potential sources of modelling techniques and model definitions**
1. Agent-role assignment	MASE, MASSIVE, GAIA, MESSAGE, INGENIAS, PASSI, CASSIOPEIA, MAS-CommonKADS
2. Agent goal/task	INGENIAS, BDIM, HLIM, MEI, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS, TROPOS
3. Agent belief/knowledge	BDIM, HLIM, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS **+ New support required** (existing methodologies do not integrate ontologies in the modelling and verification of agent beliefs)
4. Agent plan/reasoning rule/problem solving method	MASE, INGENIAS, BDIM, HLIM, MEI, PROMETHEUS, PASSI, MAS-CommonKADS, TROPOS
5. Agent architecture	MASE, MASSIVE, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS

Agent Interaction concepts	**Potential sources of modelling techniques and model definitions**
1. Agent acquaintance	All methodologies except MEI and COMOMAS
2. Interaction protocol	All methodologies except CASSIOPEIA and BDIM
3. Content of exchanged messages	MASE, MESSAGE, INGENIAS, BDIM, HLIM, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS, TROPOS + **New support required** (existing methodologies do not integrate ontologies in the modelling and verification of exchanged messages)
Overall System Design concepts	**Potential sources of modelling techniques and model definitions**
1. System architecture	MASSIVE, INGENIAS, PROMETHEUS, PASSI, ADELFE, MAS-CommonKADS
2. Organisational structure/inter-agent authority relationships	MASSIVE, GAIA, MESSAGE, INGENIAS, HLIM, COMOMAS, TROPOS
3. Environment resource/facility	GAIA, MESSAGE, INGENIAS, PROMETHEUS
4. Agent instantiation	MASE, GAIA, BDIM, PROMETHEUS, MAS-CommonKADS
5. Agent instance deployment	MASE, PASSI

5.5. IDENTIFICATION OF ONTOLOGY-RELATED STEPS

Step 3 of Research Activity 1 has determined the "actual" methodological requirements for MOBMAS (Section 5.4; Tables 5.33, 5.34 and 5.35). In this section, the execution and outputs of Step 4 – "*Identify ontology-related steps from amongst the required MOBMAS' steps* [Table 5.34]" – are documented. These ontology-related steps had to be identified in such a way as to allow MOBMAS to realise all of the widely-acknowledged benefits of ontologies to MASs, which are previously identified in Section 2.3.2. Consequently, each benefit of ontologies needed to be investigated closely.

- **Ontology's benefits to interoperability** (c.f. Section 2.3.2.1)

  Heterogeneous system components can be interoperated through the mappings of the ontologies conceptualising their respective knowledge/application. With respect to *agents*, the local knowledge of each agent should first be explicitly conceptualised by ontologies. Likewise, with respect to non-agent *resources*, ontologies should first be used to explicitly conceptualise the information/application of each resource. The semantic mappings between agents' and/or resources' local ontologies should then

be defined to allow for interoperability between them. These requirements can be implemented by the following AOSE steps.

- "*Model domain conceptualisation*": This step should define all the ontological mappings between the application's domain ontologies (where necessary). The domain ontologies may be mapped against each other, or against a common ontology (c.f. Section 2.3.2.1). Normally, when there are more than two ontologies to be mapped amongst themselves, the second approach should be favoured over the first, given the reasons listed in Section 2.3.2.1. The common ontology to be used in the second approach may be one of the existing application domain ontologies itself, or built from scratch as an inter-lingua of the existing ontologies.

- "*Define agent informational constructs*": This step should conceptualise each agent's local domain-related knowledge with ontologies. These local ontologies may be extracted from, or built upon, the application domain ontologies developed by step "*Model domain conceptualisation*". It should be noted that in this step, the developer normally does not need to define the semantic mappings between the agents' local ontologies. This is because these mappings (if necessary) should have been represented as either the relationships between concepts within a particular application domain ontology[31], or as mappings between different application domain ontologies[32].

- "*Model MAS environment*": This step should include the specification of the ontologies that conceptualise each resource's information/application. The semantic mappings between these resources' local ontologies should be defined. If each heterogeneous resource is wrapped by a different agent, each resource's local ontology would need to be mapped against the corresponding wrapper agent's local ontology. The different wrapper agents can then communicate with each other as would be described later in this section. If otherwise the

[31] This case applies if the local ontology of each agent comes from a different portion of the same application domain ontology.
[32] This case applies if the agents' local ontologies have been derived from distinct, but mapped, application domain ontologies.

heterogeneous resources are wrapped by the same agent, it is most efficient for each resource's ontology to be mapped against the agent's local ontology, which acts as the common inter-lingua.

❖ **Ontology's benefits to reusability**

As discussed in Section 2.3.2.2, ontologies promote reusability because:

- they offer high readability. This capability of ontologies can be exploited by steps "*Model domain conceptualisation*" and "*Define agent informational constructs*". By using ontology to model application domains and agents' local domain-related knowledge, these steps can improve the readability and comprehensibility of the resulting domain model and agent domain knowledge model;

- ontologies facilitate the interoperability between heterogeneous agents and resources. This benefit has been discussed earlier in Section 5.5; and

- ontologies make it easy to decouple the modelling of agents' behavioural knowledge from the domain-related knowledge, hence promoting the reuse of these two knowledge modules. This mechanism of reuse can be implemented by steps "*Define agent informational constructs*" and "*Define agent behavioural constructs*". The former can focus on defining the ontologies which conceptualise the domain-related knowledge of each agent, while the latter can focus on specifying the plans, reflexive rules and/or actions that guide the agent's behaviour. The latter should make reference to the ontology-based domain-related knowledge whenever necessary, e.g. to set the context for the agent behaviour or to serve as knowledge inputs.

❖ **Ontology's benefits to MAS development activities** (c.f. Section 2.3.2.3)

- With respect to *system analysis*, the elicitation of problem/system goals can be facilitated and validated by the ontological analysis effort of knowledge engineers or domain experts. Ontology can also be used as an effective representation mechanism for modelling application domains.

To realise these benefits, the following AOSE steps should integrate the use of ontologies into their techniques and/or generated products.

- "*Identify system functionality*": The identified system functionality identified should be verified/validated against the domain ontologies developed by knowledge engineers.
- "*Model domain conceptualisation*": This step should employ ontology as the representation mechanism for modelling the target application domains.

- With respect to *agent knowledge modelling*, ontologies provide an effective representation mechanism for modelling agents' local domain-related knowledge. As such, the AOSE step "*Define agent informational constructs*" should conceptualise the local domain-related beliefs of each agent through ontologies. The agents' local ontologies can be extracted from, or built upon, the application domain ontologies developed by step "*Model domain conceptualisation*".

❖ **Ontology's benefits to MAS operation** (c.f. Section 2.3.2.4)

- With respect to *communication*, by sharing an ontology and explicitly defining the semantics of the exchanged messages in accordance with this shared ontology, the interacting components in a MAS can communicate in a semantically consistent manner. This role of ontologies in communication can be implemented in step "*Define content of exchanged messages*". This step should require the developer to formulate the exchanged messages in terms of the concepts defined in an ontology shared between the communicating agents. Being "shared" means that the ontology needs to be included in the local knowledge of *both* communicating agents. If the communicating agents do not yet share a common ontology, such an ontology should be built and added to each agent' local knowledge. It should contain concepts that serve as the inter-lingua between the agents' local (heterogeneous) ontological concepts. This ontology may be derived from the application domain ontologies, since the agents' local ontologies are themselves extracted from there initially.

- With respect to *agent reasoning*, in order for agent reasoning at run-time to utilize ontology-based knowledge (as exemplified in Section 2.3.2.4), the agents' behavioural knowledge should be specified in such a way as to make reference to the domain-related knowledge modelled in ontologies whenever necessary. Accordingly, step "*Define agent behavioural constructs*" should use ontological concepts whenever appropriate to formulate agents' plans, reflexive rules and actions. For example, concepts in an agent's local ontology may be used to define the context of the agent's plans, or to specify the knowledge requirements of the agent's actions.

In summary, amongst the AOSE steps that are required to be supported by MOBMAS (cf. Table 5.34), the following steps should be ontology-related:

1. "*Identify system functionality*";
2. "*Model domain conceptualisation*";
3. "*Define content of exchanged messages*";
4. "*Define agent information constructs*"
5. "*Define agent behavioural constructs*"; and
6. "*Model MAS environment*"

5.6. SUMMARY

This chapter has reported on the performance and outcome of the four research steps of Research Activity 1 – "*Identify the methodological requirements of MOBMAS*". The aggregate outputs of this research activity are:

- a list of MOBMAS' methodological requirements, i.e. the *features*, *steps* and *modelling concepts* that are desirable to be supported by MOBMAS process, techniques and model definitions;
- recommendations of practitioners and researchers on the various issues that were useful to the development of MOBMAS;
- identification of a pool of techniques and model definitions that can be *reused* or *enhanced* by MOBMAS to support its methodological requirements, and identification of the methodological requirements that need to be supported by *new* techniques and/or model definitions; and
- a list of desirable ontology-related steps that MOBMAS should support.

All of these outputs were used as inputs into Research Activity 2 – "*Develop MOBMAS*" (cf. Section 4.3). Section 4.5 has explained how MOBMAS was developed using these inputs. In the next chapter, Chapter 6, the full MOBMAS methodology is documented. It should be noted that, the MOBMAS methodology presented in Chapter 6 is in its final version, after various evaluation and refinements have been made by Research Activity 3 – "*Evaluate and Refine MOBMAS*". These evaluation and refinements are reported in Chapter 7.

CHAPTER 6
DOCUMENTATION OF MOBMAS

This chapter presents the full documentation of MOBMAS. It is organised into seven sections.

- Section 6.1 presents an overview of MOBMAS, particularly MOBMAS' conceptual framework, development process and model kinds. This section also describes the application problems that were used to illustrate MOBMAS throughout its documentation.
- Sections 6.2 to 6.6 describe the five core activities in the development process of MOBMAS, namely "*Analysis*", "*MAS Organisation Design*", "*Agent Internal Design*", "*Agent Interaction Design*" and "*Architecture Design*". Each section specifies each activity's associated steps, techniques and model kinds.
- Section 6.7 presents a summary of the chapter.

The MOBMAS methodology documented in this chapter has undergone various evaluation and refinements that were made by Research Activity 3 – "*Evaluate and Refine MOBMAS*" (cf. Section 4.3). These evaluation and refinements are recorded in Chapter 7.

6.1. OVERVIEW OF MOBMAS

MOBMAS stands for "***M***ethodology for ***O***ntology-***B***ased ***M***ulti-***A***gent ***S***ystems". As stipulated in the research's objective (cf. Section 4.2), MOBMAS aims to provide comprehensive support for ontology-based MAS development and various other important AOSE methodological requirements which are documented in Chapter 5 (cf. Section 5.4.3).

Conforming to the definition of a software engineering methodology (Henderson-Sellers et al. 1998), MOBMAS is composed of (cf. Figure 4.2):

- a software engineering ***process*** that contains ***activities*** and associated ***steps*** to conduct the system development;

- ***techniques*** to assist the process steps; and
- ***definition of model kinds***[33].

An overview of MOBMAS *process* and *model kinds* is presented in Sections 6.1.2 and 6.1.3 respectively. MOBMAS techniques are presented later in the documentation of the methodology from Sections 6.2 to 6.6. But firstly, the conceptual framework of MOBMAS is documented in Section 6.1.1. This conceptual framework defines the essential abstractions that underlie MOBMAS development process and model kinds. Section 6.1.4 finally describes the application problems that were used to illustrate MOBMAS throughout Sections 6.2 to 6.6.

6.1.1. MOBMAS Conceptual Framework

MOBMAS borrows many abstractions from TAO ("Taming Agents and Objects") – a meta-model that extends UML to accommodate the development of large-scale MASs (Silva and Lucena 2004; Silva et al. 2003). TAO offers a variety of OO and agent-oriented abstractions, but MOBMAS chose to reuse and refine only those agent-oriented abstractions that are directly relevant to its process and model kinds. MOBMAS also introduces some other abstractions that are not included in TAO.

The definitions of MOBMAS' essential abstractions are presented below. TAO-based abstractions are marked with TAO reference.

1. **Agent class** (Silva et al. 2003): a template descriptor for a set of agents with similar characterisation. Each agent class is associated with a set of roles, agent goals, events, application ontologies, plan templates, reflexive rules and interaction pathways with other agent classes (referred to as "inter-agent acquaintances"). The term "**agent**" is used to refer to an instance of an agent class.

2. **Organisation** (Silva et al. 2003): A group of agents which play roles. A MAS is therefore viewed as an organisation. In MOBMAS, the organisational structure of MAS is modelled via roles, interaction pathways between roles (referred to as "inter-

[33] The term "model kind" is used to refer to the definition of a specific *class* of models (Standards Australia 2004). It is different from "model" in that models are actual deliverables produced by the developer for each model kind during the development process.

role acquaintances") and authority relationships amongst roles (e.g. peer-to-peer or superior-subordinate relationship). The organisational structure between agent classes at design time or between agents at run-time can be derived from this role-based organisational structure, given the role(s) that each agent class or agent plays.

3. **Resource**: a non-agent software system that provides application-specific information and/or services to agents in MAS, e.g. an information source or a legacy system.

4. **Environment** (Silva et al. 2003): the habitat of agents. From the perspective of a particular agent, its environment contains other agents in the system, resources and infrastructure facilities (which provide system-specific services such as naming service, agent directory service or message transport service).

5. **Role** (Silva et al. 2003): a definition of a position in the MAS organisation (Ferber and Gutknecht 1998; Demazeau and Costa 1996). In MOBMAS, each role is characterised by **Role-Tasks**, which are duties that the role is responsible for fulfilling. The role(s) played by each agent class defines the agent class' expected behaviour (because the agent class needs to behave in such a way as to fulfil its assigned role's role-tasks) and the agent class' position in the MAS organisation (because the position of an agent class is derived from the corresponding role's position in the inter-role organisational structure). At run-time, an agent may dynamically activate, suspend or switch amongst its assigned roles, thereby exhibiting dynamic behaviour and occupying dynamic positions in the MAS organisation.

6. **Agent-goal** (Silva et al. 2003): a state of the world that an agent class would like to achieve or satisfy. Agent goals signify the purpose of existence of an agent class. In MOBMAS, agent-goals are derived directly from role-tasks. An agent-goal may be decomposed into **sub-agent-goals** via AND- or OR-decomposition. AND-decomposition indicates that an agent-goal is achieved when *all* of the states specified in *all* of its sub-agent-goals are achieved, while OR-decomposition applies when an agent-goal can be achieved when *any* of the states specified in its sub-agent-goals is achieved.

7. **Event** (Silva et al. 2003): a significant occurrence in the environment to which an agent may react. This reaction may be the activation[34] of an agent-goal or a change in the agent's course of actions to satisfy an active agent-goal.

8. **Agent plan template**: a specification of various pieces of information that are useful to the formulation of plans to accomplish a particular agent-goal. Each agent plan template specifies, for each agent-goal, a set of *sub-agent-goals* and/or *actions* that may be executed by an agent to achieve the agent-goal, and *events* that may affect the agent's course of actions in achieving the agent-goal[35]. At run-time, built-in planners[36] of the agent architecture or implementation platform will formulate the specific plans for the agent to achieve the agent-goal, by selecting the appropriate sub-agent-goals and actions to execute from the agent plan template, taking into account the current state of the environment and the events that happen during the planning process.

9. **Reflexive rule**: a (sequence of) "if-then" rule that couples a stimulus[37] and/or a state of the environment with actions to be executed by an agent to satisfy a particular agent-goal. Each reflexive rule may specify a whole complete course of actions to achieve an agent-goal, or specify a partial course of actions that contributes towards the achievement of the agent-goal.

10. **Action** (Silva et al. 2003): an atomic unit of work that an agent performs.

11. **Belief state**: knowledge that an agent holds about a particular state of the world (Shoham 1993). Specifically, it captures run-time facts about the state of entities that exist in the agent's application (i.e. domains and tasks) and the environment (i.e. resources and other agents).

---

[34] An agent-goal is activated when the agent starts carrying out some processing to achieve/satisfy the agent-goal. Accordingly, an active agent-goal is one that is being actively pursued or satisfied.
[35] These are the major elements of an agent plan template. Other minor elements to be specified include: identity of the event that activates the target agent-goal (if any), conflict resolution strategy (if required) and the commitment strategy adopted by the agent during the planning process.
[36] "Planner" refers to a module/layer/subsystem in the agent architecture or implementation platform that can reason to generate plans on the fly for the agent.
[37] A stimulus may be an event or an internal processing trigger generated within the agent.

12. **Belief conceptualisation**: knowledge that an agent holds about the conceptualisation of the world, particularly the conceptualisation of the entities referred to in Belief State.

13. **Application ontology**: a conceptualisation of an application. A detailed definition of "application ontology" has been provided in Section 2.3.3. In MOBMAS, two sub-types of application ontology are defined.
    - **MAS application ontology**: a conceptualisation of the application provided by *the target MAS*. In particular, it defines the concepts and relations that the agents need to know, and share, about the MAS application domains and tasks.
    - **Resource application ontology**: a conceptualisation of the application provided by a *resource* of the MAS system. In particular:
      - if the resource is a *processing application system* (e.g. a legacy system), the corresponding Resource Application Ontology defines the concepts and relations that conceptualise the application domains and tasks/services of the resource; and
      - if the resource is an *information source* (e.g. a database), the corresponding Resource Application Ontology defines the concepts and relations that conceptualise the information stored inside the resource. It may be derived from the information source's conceptual schema (Hwang 1999; Guarino 1997).

    In MOBMAS, the specification of an agent's Belief Conceptualisation essentially comes down to the determination of which (part of[38]) MAS Application Ontologies and/or Resource Application Ontologies the agent should commit.

14. **System-task**: anything that the target system should or will do. System-tasks represent the required functionality of the MAS system. A particular system-task may be decomposed into sub-system-tasks via AND- or OR-decomposition. AND-decomposition indicates that the accomplishment of a system-task requires the execution of *all* of its sub-system-tasks, while OR-decomposition applies when the system-task can be accomplished by executing *any* of its sub-system-tasks.

---

[38] In many cases, the agent only needs to commit to a fragment of a particular MAS application ontology or Resource application ontology to do its work.

The relationships between MOBMAS abstractions are shown in Figure 6.1.

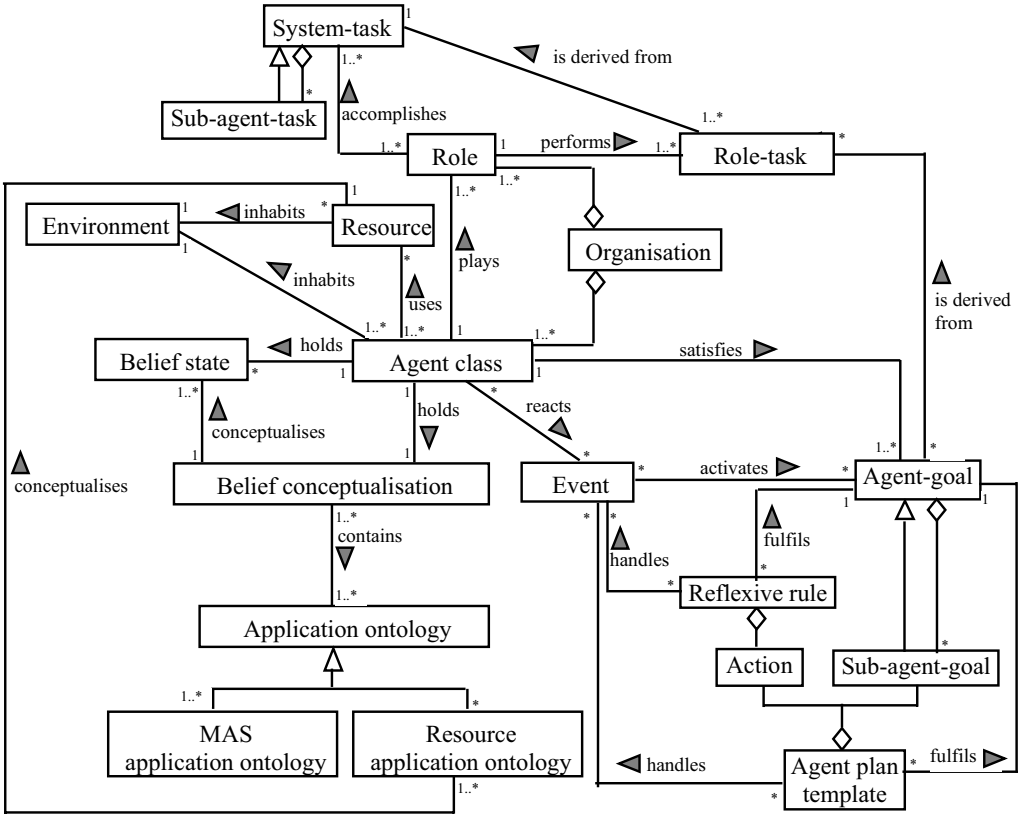


Figure 6.1 – MOBMAS abstractions and their relationships (represented in UML)

6.1.2. MOBMAS Development Process

The development process of MOBMAS consists of five **activities**, each of which focuses on a significant area of MAS development: analysis, agent internal design, agent interaction design, MAS organisation modelling and architecture specification. Each activity is composed of a number of **steps**.

1. **Analysis Activity:** This activity is concerned with developing a conception for the future MAS, namely a first-cut identification of the roles that compose the MAS organisation. The activity also involves capturing "MAS application ontologies" that conceptualise the application of the target MAS.

2. **MAS Organisation Design Activity:** This activity specifies the organisational structure for the target MAS and defines a set of agent classes that compose the system. If the MAS is a heterogeneous system that incorporates non-agent resources, these resources need to be identified. These resources' applications also need to also be conceptualised (i.e. "Resource application ontologies").

3. **Agent Internal Design:** This activity deals with the specification of each agent class' belief conceptualisation, agent-goals, events, agent plan templates and reflexive rules.

4. **Agent Interaction Design:** This activity designs the interactions between agent classes by, firstly, selecting a suitable interaction mechanism for the target MAS (e.g. direct interaction via ACL messages or indirect interaction via tuplespace/tuple-centre), thereafter defining the patterns of data exchanges amongst agent classes depending on the chosen interaction mechanism.

5. **Architecture Design Activity:** This activity deals with various architecture-related issues, namely the identification of agent-environment interface requirements, the selection of agent architecture, the identification of required infrastructure facilities, the instantiation of agent classes and the deployment configuration of agents.

Figure 6.1 lists the specific **steps** in each of the five activities of MOBMAS. It should be noted that MOBMAS' steps cover the *desirable AOSE steps* that are previously listed in Table 5.34, even though the former are named differently from the latter[39], and some MOBMAS' steps are defined as a combination or decomposition of the desirable AOSE steps[40] so as to form a coherent methodology. The correspondence between MOBMAS' steps and the desirable AOSE steps would be confirmed in the feature analysis of MOBMAS in Chapter 7 (particularly Table 7.5).

---

[39] For example, MOBMAS' step "*Develop System Task Model*" is equivalent to the desirable AOSE step "*Identify system functionality*" in Table 5.34.

[40] For example, MOBMAS' step "*Develop Agent Interaction Model*" encapsulates three desirable AOSE steps in Table 5.34: "*Specify acquaintances between agent classes*", "*Define interaction protocols*" and "*Define content of exchanged messages*". Meanwhile, the desirable AOSE step "*Define agent behavioural constructs*" in Table 5.34 is decomposed into three MOBMAS' steps: "*Specify agent goals*", "*Specify events*" and "*Develop Agent Behaviour Model*".

Each MOBMAS' step is associated with a **model kind** as each step allows the developer to produce or update models of a particular kind. The solid arrows indicate the flow of steps within and across activities, while the dotted arrows indicate the potential iterative cycles of steps. Step iteration is particularly necessary if the information collected in one step results in the refinement/extension of models previously produced by another step. Note that the arrows only serve as recommendations. In practice, the developer should be able to trace backward to any preceding step to refine or extend the corresponding model (e.g. when new requirements arise). Thus, the development process of MOBMAS is highly **iterative** and **incremental**, either within or across all activities.

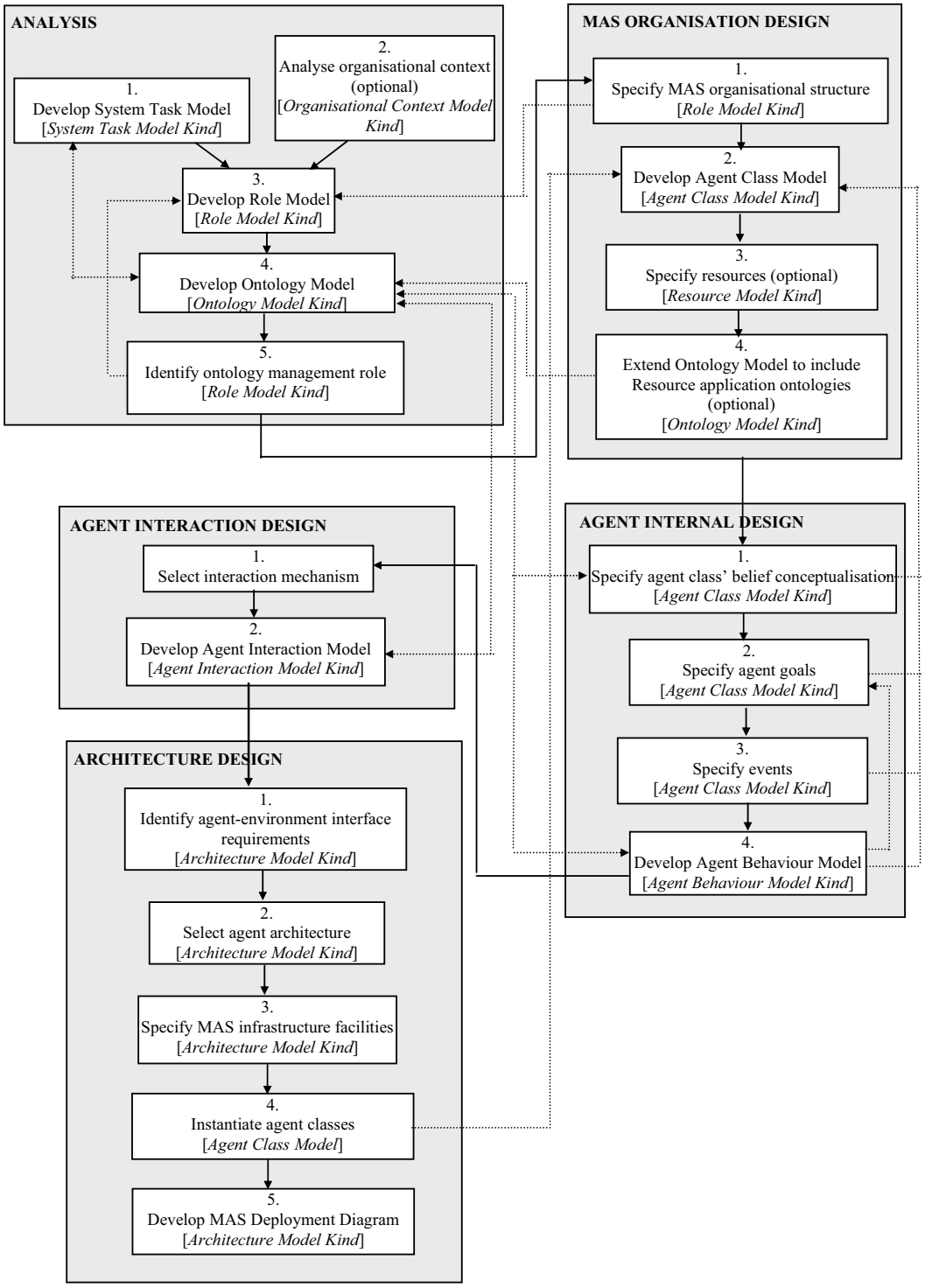


Figure 6.2 – MOBMAS development process

6.1.3. MOBMAS Model Kinds

MOBMAS defines nine model kinds for capturing the outputs of its process steps (Figure 6.3). During the development time, the developer is required to produce one model for each model kind. Every model kind is represented by one or more *notational components*, which are either graphical diagrams or textual schemas. Some model kinds and notational components are optional, since the steps generating them are optional (cf. Figure 6.2). Figure 6.3 shows the dependency and cross-check relationships between the model kinds.

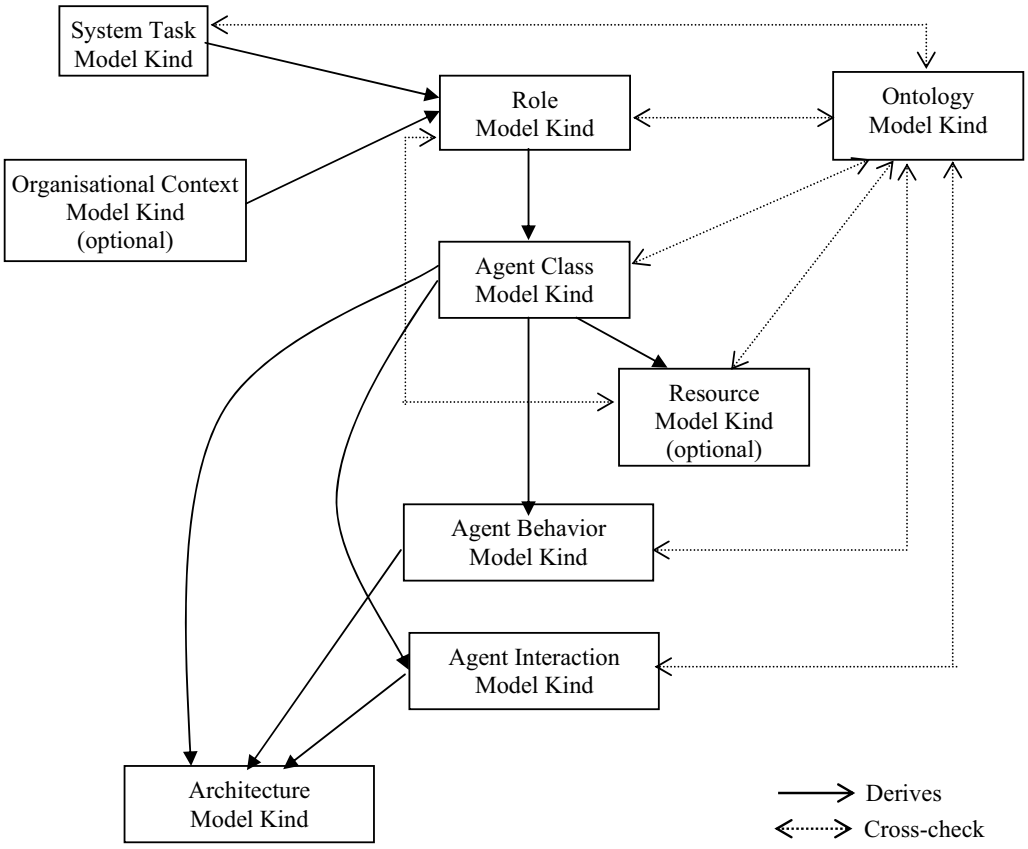


Figure 6.3 – MOBMAS Model Kinds

1. **System Task Model Kind:** This model kind captures the specification of system tasks, their hierarchical decomposition and conflicts (if any). This model kind is depicted by a ***System Task Diagram***.

2. **Organisational Context Model Kind** (optional): This model kind captures the pre-existing structure of the organisation which MAS supports, automates or monitors.

This structure is defined via organisational units and relationships between units (namely, acquaintance[41] relationships and membership relationships). The notational component of this model kind is an ***Organisational Context Chart***.

3. **Role Model Kind:** This model kind defines each role in the MAS organisation (i.e. role name and role-tasks), acquaintances between roles and authority relationships that govern inter-role acquaintances (e.g. peer-to-peer relationship or superior-subordinate relationship). Role Model Kind is depicted by a ***Role Diagram***.

4. **Ontology Model Kind**: This model kind captures the specification of all MAS application ontologies and Resource application ontologies needed for the target system. MOBMAS does not impose a specific modelling language for this model kind. However, for illustrative purpose, MOBMAS uses UML class diagrams to depict ontologies. These UML class diagrams are referred to as ***Ontology Diagrams***.

5. **Agent Class Model Kind**: This model kind captures the definitions of agent classes composing the target MAS. It is depicted by two notational components.
   - ***Agent Class Diagram***: which shows the specification of *each* agent class, namely the agent class' name, instantiation cardinality, roles, belief conceptualisation, agent-goals and events. A MAS typically requires multiple Agent Class Diagrams, one for each agent class.
   - ***Agent Relationship Diagram***: which shows all agent classes in the target MAS and the acquaintances between them. Various descriptive information about each inter-agent acquaintance is also shown (e.g. interaction protocol and application ontology that govern the interactions between the acquainted agent classes). If the target MAS incorporates resources, the relationships between "wrapper" agent classes and their wrapped resources are also displayed.

6. **Resource Model Kind** (optional): This model kind captures the specification of resources in the MAS, including the resources' name, type and corresponding Resource application ontology. The model kind also specifies the identity of agent

[41] Acquaintance refers to interaction pathway.

classes that wrap around the resources. The notational component of this model kind is a ***Resource Diagram***.

7. **Agent Behaviour Model Kind**: This model kind specifies the behaviour of *each* agent class. It is represented by the following notational components.
   - ***Agent Goal Diagram*** (optional): displays, for a particular agent class, the decomposition structure of its agent-goals and/or the conflicts amongst these agent-goals. This diagram is only necessary if the agent class is found to pursue multiple agent-goals, and these agent-goals are involved in decomposition relationships or are in conflict with each other.
   - ***Agent Plan Template***: documents various pieces of information that are needed to formulate plans for agents at run-time, including the identity of the agent-goal that the plan aims to fulfil, the potential sub-agent-goals and/or actions that may be executed to satisfy the agent-goal, events that activate the agent-goal or affect the agent's course of actions to satisfy the agent-goal, conflict resolution strategies (if required) and commitment strategy adopted by the planning process. If there exists a *tentative* course of sub-agent-goals/actions for achieving the agent-goal, this sequence can be depicted in an ***Agent Plan Diagram***.
   - ***Reflexive Rule Specification***: documents a particular reflexive rule of an agent class. It specifies the agent-goal that the reflexive rule aims to satisfy, a sequence of actions to (partially) fulfil the agent-goal, and the events, internal processing triggers and/or conditions that initiate an action or make an action applicable.

8. **Agent Interaction Model Kind**: This model kind defines the patterns of interactions amongst agent classes depending on the adopted interaction mechanism. If the mechanism is direct interaction via ACL messages, the model kind captures the definitions of interaction protocols between agent classes. These definitions are depicted by ***Interaction Protocol Diagrams***. If otherwise the adopted interaction mechanism is indirect interaction via tuplespace/tuple-centres[42], the Agent Interaction Model Kind specifies the interaction patterns between agent classes and

[42] MOBMAS identifies other types of indirect mechanisms, namely stigmergy and spatially founded mechanisms. However, since these mechanisms are very limited in their applicability, MOBMAS focuses on the indirect mechanism based on tuplespace/tuple-centre (cf. Section 6.5.1).

the tuplespace/tuple-centre. ***Agent-TC[43] Interaction Diagrams*** are used as the notational component for the Agent Interaction Model Kind in this case. Moreover, if the tuple-centre is used instead of tuplespace, the model kind also captures the definition of the tuple-centre's behaviour via ***Tuple-Centre Behaviour Diagram***.

9. **Architecture Model Kind**: This model kind captures various architecture-related specifications. It is represented by four notational components.
   - ***Agent-Environment Interface Requirements Specification***: documents any special requirements of the agents' sensor, effector and communication modules, so as to support the agents' perception, effects and communication needs at run-time.
   - ***Agent Architecture Diagram***: provides a schematic view of the architecture adopted by the agent classes in the target MAS. If different agent classes require different architectures, one Agent Architecture Diagram is required for each architecture.
   - ***Infrastructure Facilities Specification***: documents the specifications of core infrastructure facilities that are necessary to support the target MAS' operation (e.g. naming service, message transport service or agent directory service).
   - ***MAS Deployment Diagram***: shows the deployment configuration of the target MAS, including the allocation of agents to nodes and the connections between nodes.

The notation of each notational component of the nine model kinds is presented in Appendix E.

6.1.4. Illustrative Applications

Throughout the documentation of MOBMAS in Sections 6.2 to 6.6, two applications were used for illustration purposes:

- Product Search application; and
- Conference Program Management application.

[43] TC stands for "tuple-centre".

The Product Search application was used as the primary illustrative example, while the Conference Program Management application was used only when the former is not suitable for the demonstration of a particular MOBMAS step, technique or notational component. The following sections briefly describe each illustrative application.

6.1.4.1. Product search application

This application investigates the use of MAS in searching for product information – a major activity in e-business. The objective of the system is to assist users in searching and retrieving information on products from heterogeneous resources, including information sources provided by the potential suppliers (such as suppliers' databases and web servers) and various online search engines. The target domain is limited to Car Products for illustration purposes.

The user interacts with the system by submitting his search query. Upon receiving a query, the system extracts keywords from it, searching through the resources to gather information for the query, and displaying the final answer to the user.

The system also accepts and processes feedback from the user, which may help improving its future performance.

6.1.4.2. Conference program management application

This application has been used in various past research work in AOSE as case study (Ciancarini et al. 1998; Zambonelli et al. 2001a; Zambonelli et al. 2003; Ciancarini et al. 1999). Setting up a conference program is a multi-phase process, including submission, reviewing and final publication phases. For illustrative purposes, this research focuses on the review phase only. In this phase, the "program committee chair" has to work with "committee members" to distribute the submitted papers among the members. The members are assumed to have the authority to choose for themselves the papers they want to review. The chair does not impose the papers on them. Having collecting the papers, each committee member is in charge of finding an external referee for each paper and contacting these reviewers to send them papers. Eventually, the reviews come back to the respective committee members who determine the acceptance or rejection of the papers. The authors are then notified of these decisions by the committee chair.

6.2. ANALYSIS ACTIVITY

The Analysis activity of MOBMAS takes as inputs a set of system-tasks and develops a conception for the future MAS, namely a first-cut identification of the roles that compose the future MAS system. The activity consists of 5 steps, as shown on Figure 6.4 (which is a copy of Figure 6.2 but with the Analysis activity highlighted).

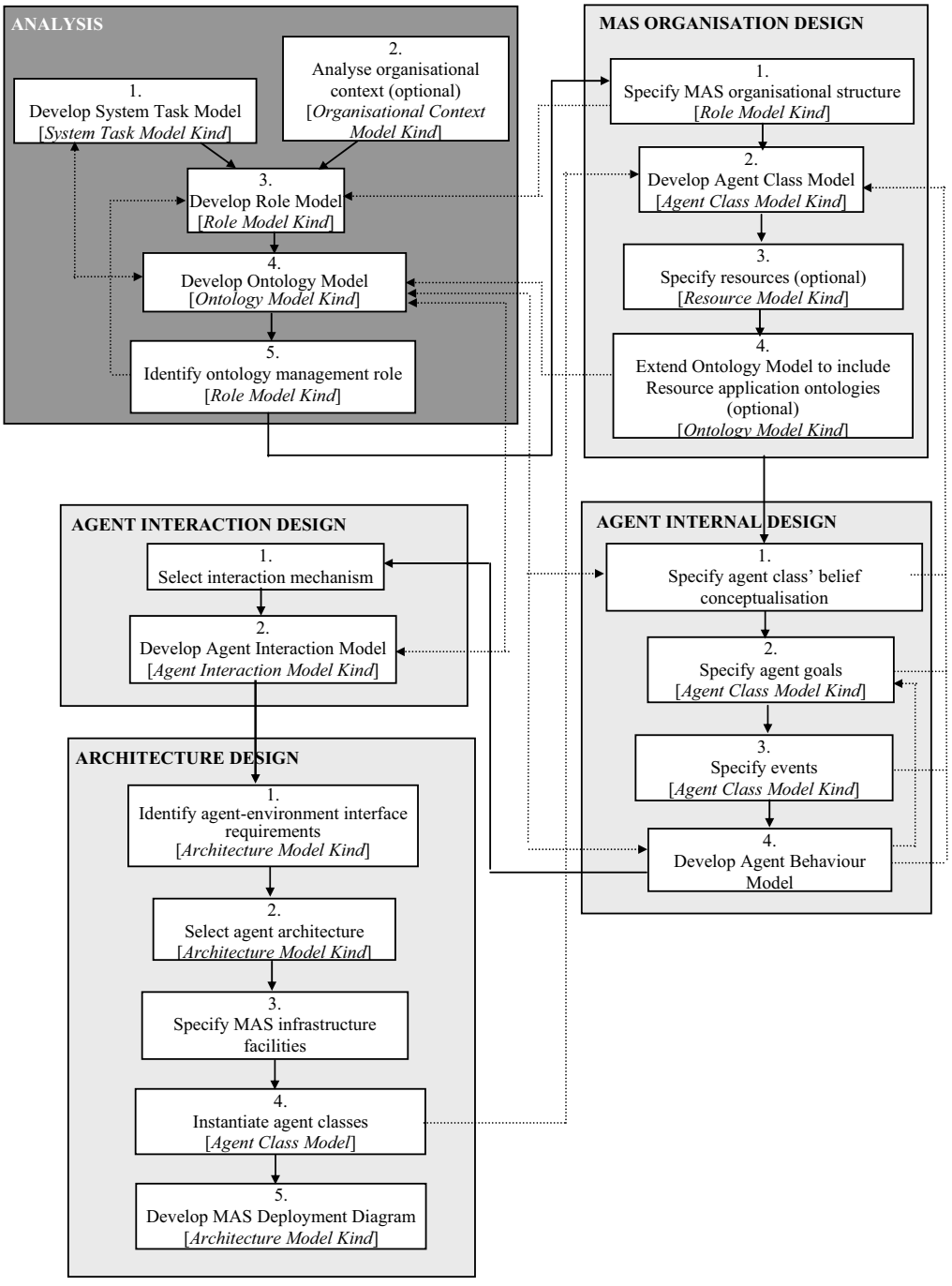


Figure 6.4 – MOBMAS development process

6.2.1. Step 1 – Develop System Task Model

The term "*system-task*" is used to mean anything that the target system should or will do. It represents the required functionality of the system. For example, system-tasks of the illustrative Product Search application (cf. Section 6.1.4.1) are "Satisfy user query" and "Process user feedback" (Figure 6.5).

The identification of system-tasks is not part of MOBMAS. It is presumed to be conducted by a separate Requirements Engineering effort. Hence, MOBMAS refers the developer to the vast amount of existing work on Requirements Engineering for more techniques on system-tasks elicitation, e.g. Kotonya and Sommerville (1998), Macaulay (1996), Haumer et al. (1998), Duursma (1993), Dardenne et al (1993), Yourdon (1989), DeMarco (1978), Potts (1999) and Wiegers 2003. The development process of MOBMAS starts from the identified set of system-tasks to produce a **System Task Model Kind**. This model kind aims to capture the following information.

- Identity of system-tasks.
- Conflicts amongst system-tasks (if any): Conflicts exist when different system-tasks cannot be accomplished together without being compromised (Dardenne et al. 1993). In the illustrative Product Search application, no conflicting system-tasks are found. However in many other applications such as MASs for library service management, system-tasks such as "Maintain long borrowing period" and "Maintain regular availability" are in conflict with each other (Dardenne et al. 1993).
- Functional decomposition of system-tasks (if required): A particular system-task may be decomposed into smaller-scale, constituent system-tasks which are referred to as "*sub-system-tasks*". Each decomposition may either be:
  - **AND-decomposition**: that is, when the accomplishment of a system-task requires the execution of *all* of its sub-system-tasks. For example, system-task "Find answer to user query" can be AND-decomposed into sub-system-tasks "Extract keywords from user query*"* and "Gather information from resources" (Figure 6.5); or
  - **OR-decomposition**: that is, when the accomplishment of a system-task involves the execution of *any* of its sub-system-tasks. For example, system-task "Identify appropriate resources*"* can be OR-decomposed into sub-system-tasks "Identify appropriate databases*"* and "Identify appropriate web-servers*"* (Figure 6.5).

For each decomposition, MOBMAS recommends the developer to specify whether the decomposition is full or partial.

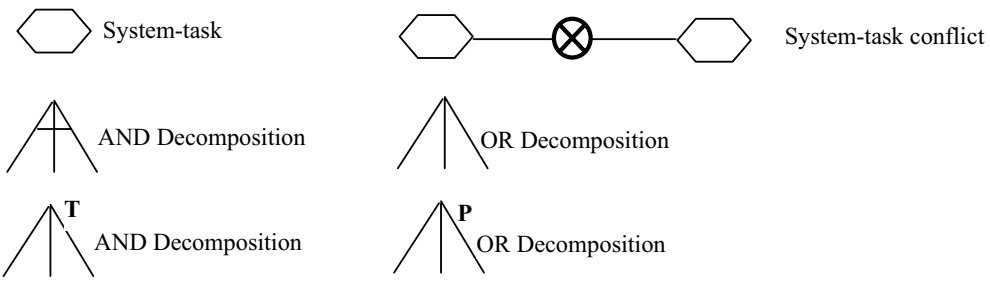
- **Full decomposition** applies when the accomplishment of a system-task is *totally equivalent* to the execution of its sub-system-tasks (either all or any of the sub-system-tasks, depending on whether the decomposition is an AND or OR decomposition). For example, system-task "Satisfy user query" is fully decomposed into sub-system-tasks "Accept user query", "Find answer to user query" and "Display result for query" (Figure 6.5) because the successful execution of these three sub-system-tasks automatically results in the accomplishment of the system-task "Satisfy user query".
- **Partial decomposition** applies when the accomplishment of a system-task is *not totally equivalent* to the execution of its sub-system-tasks. In other words, there are certain actions that need to be performed by the system-task but which are not accounted for by its sub-system-tasks. For example, system-task "Process user feedback" is partially decomposed into sub-system-tasks "Receive user feedback" and "Display acknowledge" (Figure 6.5) because, apart from receiving feedback and displaying acknowledgement, system-task "Process user feedback" also needs to compute feedback rating and refine search algorithms to improve the system's future performance.

The identification of full versus partial decomposition will assist the developer in identifying roles and roles' tasks later on.

After system-tasks are identified and a (preliminary) System Task Model is constructed, the developer is strongly recommended to validate the model against the Ontology Model, whose development is discussed in Step 4 of the Analysis activity (c.f. Section 6.2.4). The MAS Application ontologies specified in the Ontology Model may help to reveal new system-tasks that have not yet been uncovered, thus assisting in the refinement of the System Task Model. More discussion on this validation is presented in Section 6.2.4.1.c.

6.2.1.1. Notation of System Task Diagram

System Task Model Kind of MOBMAS is depicted by one or more **System Task Diagrams**. The notation for the System Task Diagram is as follows.



A System-Task Diagram for the illustrative Product Search MAS is presented in Figure 6.5. It should be noted that the functional decomposition of system-tasks does not always have a simple tree structure as in Figure 6.5. Two or more system-tasks may share the same sub-system-task(s).

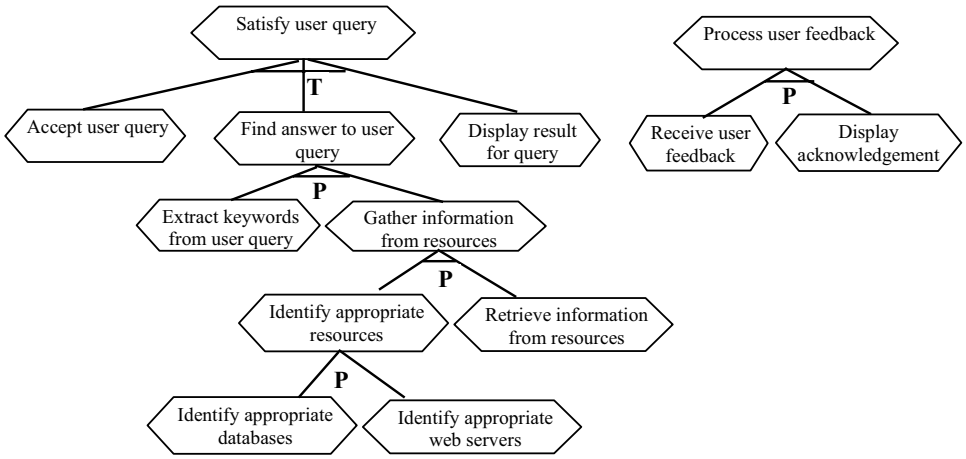


Figure 6.5 – System Task Diagram for Product Search MAS

6.2.2. Step 2 – Analyse Organisational Context (Optional)

Even though the analysis of system-tasks generally provide adequate inputs to the identification of roles later on, an investigation of the *existing structure of the MAS' organisational context* (i.e. the structure of the organisation which MAS supports, automates or monitors) can further assist in the process of role identification. This is so

because the organisational structure of MAS may be directly derived from the existing structure of the MAS' organisational context (e.g. consider enterprise information systems and workflow management systems) (Zambonelli et al. 2003).

MOBMAS suggests investigating the structure of the MAS' organisational context if this structure satisfies **all** of the following conditions.

- It is known and clearly defined.
- It is well-established, not likely to change, and has proven or been accepted to be an effective way to function. Accordingly, it is desirable for the future MAS to mimic this existing structure.

For example, consider the illustrative Conference Program Management application (cf. Section 6.1.4.2). The existing organisational structure of the human conference committee is composed of a "Program Committee (PC) Chair", several "PC Members" and many "Reviewers" (Ciancarini et al. 1998; Zambonelli et al. 2001a; Zambonelli et al. 2003; Ciancarini et al. 1999). Assuming that this structure has always been adopted by the human conference organisers, and that the organisers do not wish to change the way their conferences are managed, the developer should investigate this structure for the development of a software Conference Program Management MAS.

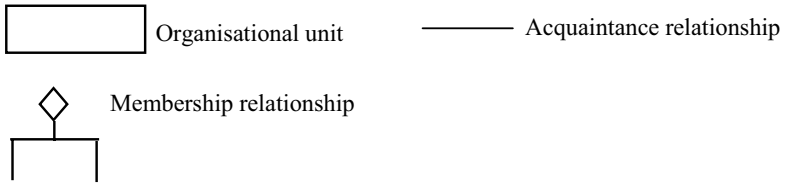
6.2.2.1. Develop Organisational Context Model Kind

The existing structure of the organisational context is captured in an **Organisational Context Model Kind**, which contains one notational component, **Organisational Context Chart**. Since this MOBMAS step is only recommended to applications where the structure of the organisational context is known, clear and well-established, it is generally a straightforward task to develop this model kind. The developer needs to specify:

- the *organisational units*: i.e. the positions or individuals or departments that exist in the organisational context; and
- the *relationships* between these units, namely "*acquaintance*" relationships (where one organisational unit interacts with another) and "*membership*" relationships (where one organisational unit is part of another).

The developer may identify these elements from various sources, including the business organisational chart, business process documentation, interviews, questionnaires, investigation of employee manuals, orientation pamphlets, memos and annual company reports (Awad 1985).

MOBMAS borrows UML notation for the Organisational Context Chart.



An Organisational Context Chart for the illustrative Conference Program Management MAS is presented in Figure 6.6.

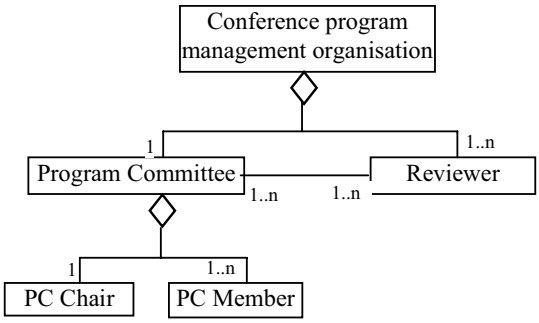


Figure 6.6 – Organisation Context Chart for the Conference Program Management MAS

6.2.3. Step 3 – Develop Role Model

The notion of "role" in agent-oriented development is analogous to the notion of "role" in a play or members in a typical company (Wood 2000; Kendall 1999). It refers to the position of an entity in an organisation and defines what the entity is expected to do in the organisation (Ferber and Gutknecht 1998; Demazeau and Costa 1996). In MOBMAS, roles serve as the building blocks for defining agent classes. Each agent class is associated with one or more roles, which establish the agent class' expected behaviour and position in the MAS organisational structure.

MOBMAS specifies all roles in the **Role Model Kind**, which consists of one notational component, **Role Diagram**. Sections 6.2.3.1 and 6.2.3.2 discuss the identification of

roles and role-tasks respectively, while Section 6.2.3.3 presents the notation for Role Diagram.

6.2.3.1. Identify roles

MOBMAS identifies roles from system-tasks and, if Step 2 – "*Analyse Organisational Context*" has been carried out, from the existing structure of the MAS' organisational context. Section 6.2.3.1.a presents techniques for identifying roles from system-tasks, while Section 6.2.3.1.b recommends how roles can be identified with the analysis of MAS's organisational context.

6.2.3.1.a. Identify roles from system tasks

Generally, each system-task specified in the System Task Model should be assigned to one role. However, each role can be delegated multiple system-tasks for the sake of efficiency.

The grouping of multiple system-tasks into one role should be guided by the principle of *strong internal coherence and loose coupling* in term of functionality. Each role should represent a functionally coherent cluster of system-tasks that is sufficiently different from other clusters (Lind 1999; Padgham and Winikoff 2002). This principle helps to promote modularity in system design.

Some other heuristics that may indicate the need to delegate multiple system-tasks into one role are:

- when the system-tasks are likely to interact significantly with each other (e.g. for inputs/outputs exchanges). Grouping these system-tasks into one role will help to reduce the amount of interactions amongst roles and ultimately amongst agents; or
- when the system-tasks require the same data (e.g. input information, domain knowledge). Assigning these system-tasks to one role means that only this one role needs to acquire, or be equipped with, the required data; or
- when the system-tasks need to access to the same resource (e.g. information sources or legacy systems). Delegating these system-tasks to one role means that only this one role needs to implement an interface with the resource.

On the other hand, some system-tasks may *not* be appropriate to be grouped into one single role, particularly when:

- the system-tasks are executed at different locations at the same point in time. For example, system-task "Accept user query" and "Retrieve information from resources" are executed at the user's site and the remote resources' site respectively, thus they should not be performed by the same role; or
- the system-tasks need to satisfy certain security and privacy requirements. For example, data associated with one system-task should not be available to another.

In the case when a system-task has been *fully decomposed* into sub-system-tasks (cf. Section 6.2.1), it does not need to be assigned to any role, because it is presumed to be accomplished via the execution of the sub-system-tasks. Accordingly, only the sub-system-tasks need to be assigned to roles. However if a system-task has been *partially decomposed* into sub-system-tasks, it must be assigned to a particular role because its accomplishment is not equivalent to the accomplishment of the sub-system-tasks.

In some cases, a single system-task needs to be assigned to multiple roles. This occurs when the system-task requires the *collective* effort of different roles. This type of task is referred to as a "**joint task**" (or "social task" in Omicini 2000 and Ciancarini et al. 2000). For example, in the illustrative Conference Program Management application, one system-task is to "Distribute papers among members" in such a way that each paper is allocated to a required number of PC members, and each member is assigned a required number of papers. This system-task should be modelled as a joint task because it is the shared responsibility of both "PC Chair" and "PC Member" roles (cf. Section 6.1.4.2; Figure 6.9).

Nevertheless, just because a system-task requires inputs from multiple roles does not mean that it should be modelled as a joint task. Such a system-task can be assigned to one single role, which acts as the primary accountable and controlling party for the task execution. This role will then interact with other roles when necessary for input/output information. For example, system-task "Find answer to user query" can be assigned to one role "Searcher", which interacts with "InfoSource Wrapper" role for information to fulfil the system-task.

Therefore, as a generic guideline, MOBMAS suggests classifying a system-task as a joint task (thus assigning it to multiple roles) only if the *control* of the system-task needs to be *equally* spread among the participating roles, or equivalently, if all the participating roles are *equally* accountable for the accomplishment of the system-task, such as in the case of system-task "Distribute papers among members" (note that for this system-task, "PC Members" are given the authority to choose for themselves the papers they want to review. "PC Chair" does not impose the papers on them[44]).

Figure 6.7 shows the identification of roles for the illustrative Product Search MAS.

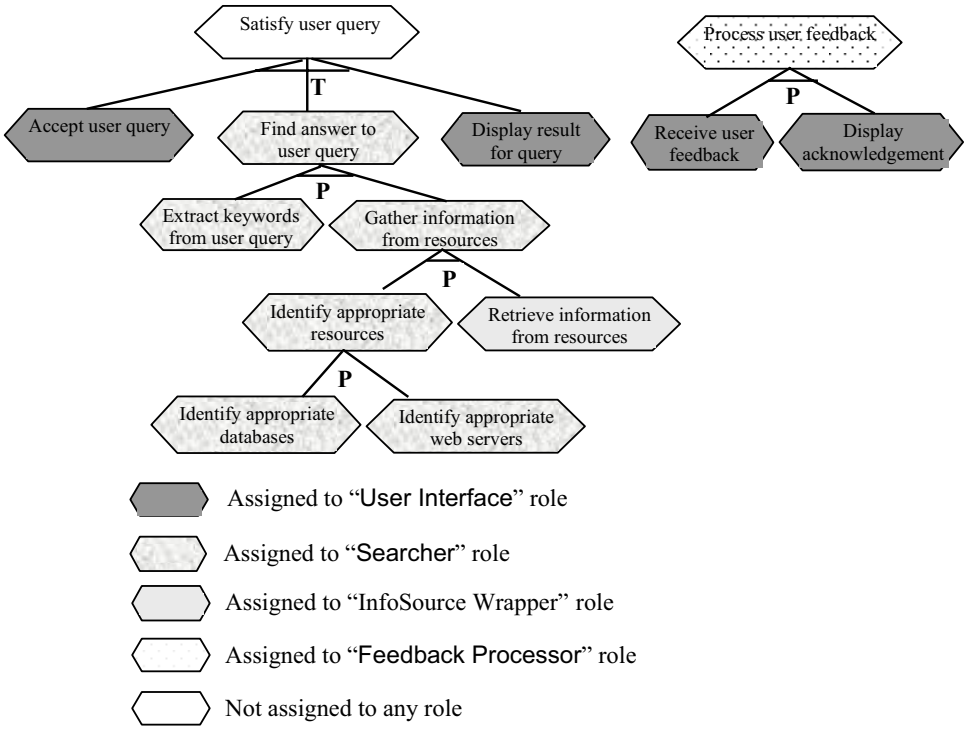


Figure 6.7 – Final roles for Product Search MAS

6.2.3.1.b. Identify roles from the structure of MAS' organisational context (optional)

If Step 2 – "*Analyse Organisational Context*" has been performed, the investigation of the existing structure of MAS' organisational context can greatly assist in the process of role identification. Specifically, some preliminary roles can be identified from the

[44] Even though this assumption does not always hold in real-life conferences, it is made in this thesis for the purpose of illustration (cf. Section 6.1.3.2).

organisational context's structure, thereafter being verified with the analysis of the system-tasks as discussed in Section 6.2.3.1.a.

In general, each *organisational unit* in the MAS' organisational context can be mapped onto one role, since the functionality of each organisational unit is often internally coherent and loosely coupled from the other units (Wooldridge et al. 2000). In cases when the MAS' organisational context exhibits a hierarchical structure (i.e. when there are "membership" relations among the units), the developer can either:

- map each *leaf-node* organisational unit to a role; or
- map the whole *upper-level unit* to a role.

The former is recommended if the leaf-node organisational units are loosely coupled in term of functionality, while the latter is appropriate if the leaf-node units are strongly coupled. For example, units "PC Chair", "PC Member" and "Reviewer" in Figure 6.6 can each be mapped to a different role since their functionality is loosely coupled from each other.

It should be noted that the direct mapping between existing organisational units and the software roles does not necessarily result in an efficient MAS design. This is so because, firstly, the reasons that may have driven the existing organisation to adopt a particular structure may not necessarily apply to the MAS organisation, and secondly, the mere presence of MAS may introduce changes to the existing organisation (Zambonelli et al. 2003). The developer therefore should *always* analyse the system-tasks to validate the set of identified roles (cf. Section 6.2.3.1.a). The existing structure of the MAS' organisational context should only serve as an additional resource for the identification of software roles.

6.2.3.2. Specify role-tasks

"Role-tasks" are tasks that a particular role is responsible for fulfilling. In MOBMAS, role-tasks can be *directly* mapped from the *system-tasks* that the developer delegates to roles during the process of role identification (cf. Section 6.2.3.1.a). This mapping is generally one-to-one. For example, role-tasks of "User Interface" role are "Accept user

query", "Display result for query", "Receive user feedback" and "Display acknowledgement" (cf. Figure 6.7).

Given the close inter-connection between roles, role-tasks and system-tasks, the development of *Role Model* and *System Task Model* should be performed in a highly iterative and spiral manner. In practice, the modelling of a particular role may discover some role-tasks that have not been identified in the System Task Model, leading to the revision of the System Task Model.

6.2.3.3. Notation of Role Diagram

The Role Model Kind of MOBMAS is depicted by a Role Diagram, which shows the *specification of each role*, *acquaintances* between roles and *authority relationships* that govern inter-role acquaintances.

The specification of each role involves the specification of role name and role-tasks, both of which have been defined during the process of role identification (cf. Section 6.2.3.1.a).

Acquaintances between roles represent inter-role interaction pathways. Preliminary role acquaintances can be identified from:

- *the relationships between system-tasks that the roles are responsible for*: If two roles are responsible for a system-task and a sub-system-task respectively, or if the two roles are responsible for sibling system-tasks[45], they are likely to interact with each other; or
- *the acquaintance relationships between the existing organisational units of the MAS' organisational context*, if Step 2 – "*Analyse Organisational Context*" has been performed: If two roles are responsible for two acquainted organisational units, they are likely to interact with each other.

The specification of roles and inter-role acquaintances will be validated/refined in later steps of MOBMAS development process. The specification of authority relationships between roles is discussed in the "*MAS Organisation Design*" activity (Section 6.3).

---

[45] That is, sub-system-tasks that share the same parent.

MOBMAS notation for the Role Diagram is as follows.

- Each role is depicted as a rectangle, which contains two compartments.
  - The top compartment, marked with keyword **role**, specifies the role name.
  - The bottom compartment, marked with keyword **role-tasks**, lists the role-tasks.
- Each inter-role acquaintance is depicted as an undirected line between roles.

For roles that share a common *joint task* (cf. Section 6.2.3.1.a), the task should be labelled with an **adornment *(J)*** to distinguish itself from other role-tasks. This highlight will help the developer to identify the existence of joint tasks in the target MAS during the subsequent steps of MOBMAS.

Notation for inter-role authority relationships is presented in the "*MAS Organisation Design*" activity.

Figure 6.8 shows the Role Diagram for the illustrative Product Search MAS, while Figure 6.9 illustrates a Role Diagram for the Conference Program Management MAS.

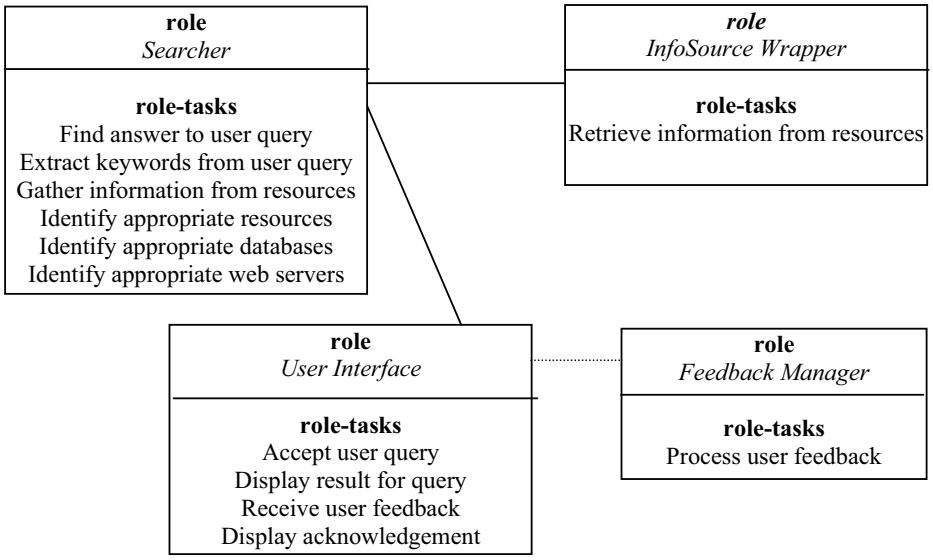


Figure 6.8 – Role Diagram for Product Search MAS (cf. Figure 6.7)

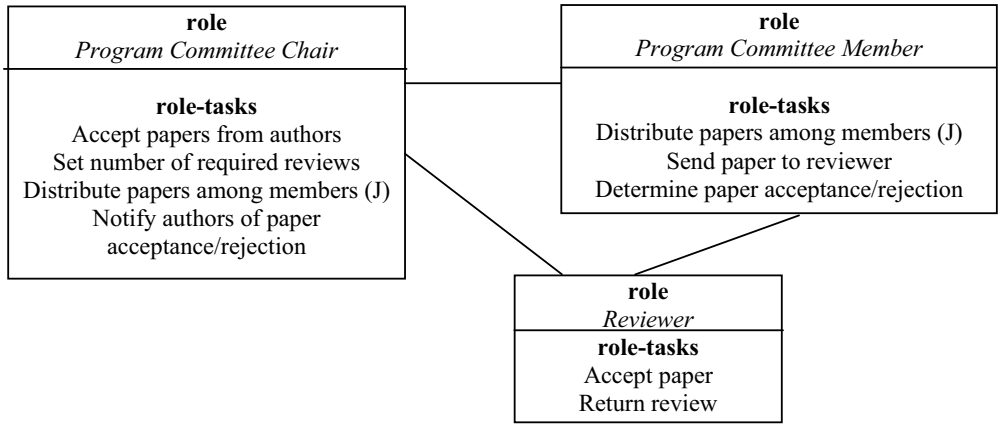


Figure 6.9 – Role Diagram for Conference Program Management MAS

6.2.4. Step 4 – Develop Ontology Model

MOBMAS captures all ontologies required by the target MAS in an **Ontology Model Kind**. This model kind is subject to ongoing refinement and verification throughout the development process of MOBMAS.

Previously in Section 2.3.3, a taxonomy of ontology has been presented. This taxonomy defines four types of ontology based on their level of generality: Top-level ontologies, Domain ontologies, Task ontologies and Application ontologies. The former three types are generic across applications since they are independent of the tasks at hand[46] and/or the domains at hand[47]. Meanwhile, Application ontologies are specific to a particular application because concepts in these ontologies are related to both domains and tasks at hand (Guarino 1998; van Heijst et al. 1997; Gennari et al. 1994).

In the context of software system development, only ***Application ontologies*** need to be defined and used. This is so because a software system only needs to concern with those concepts that are specific to the application at hand (Guarino 1998).

[46] Domain ontologies such as Medicine Domain Ontology, Car Domain Ontology and Automobile Domain Ontology are independent of the tasks at hand.
[47] Task ontologies such as Negotiation Task Ontology, Diagnosis Task Ontology, "Ranking by Weight" Task Ontology and Propose-and-Revise Task Ontology are independent of the domains at hand.

In the development of MAS in particular, MOBMAS recommends classifying Application ontologies into two categories: *MAS Application ontologies* and *Resource Application ontologies*.

- **MAS Application ontologies:** conceptualise the application provided by *the target MAS*. In particular, they define all the concepts and relations that the agents need to know and share about the MAS application (e.g. about the target domains and tasks). By committing to and sharing these MAS Application ontologies, agents in the system are equipped with the (same) conceptual knowledge of their application, thereby being able to operate and communicate with each other.

- **Resource Application ontologies**: conceptualise the application provided by a *resource* in the MAS system. In particular:
  - if the resource is a *processing application system* (e.g. a legacy system), the corresponding Resource Application ontology captures all the concepts and relations that conceptualise the domains and tasks/services of the resource; and
  - if the resource is an *information source* (e.g. a database), the corresponding Resource Application Ontology captures all the concepts and relations that conceptualise the information stored inside the resource. It may be derived from the information source's conceptual schema (Hwang 1999; Guarino 1997).

Resource Application ontologies are only necessary for heterogeneous MASs that contain non-agent software resources apart from agents. In these systems, only agents that directly interface with the resources will need to hold knowledge of the Resource Application ontologies, since only these agents are required to know about the conceptualisation of the resources' applications.

Figure 6.10 shows the difference between MAS Application ontologies and Resource Application ontologies in their scope of usage.

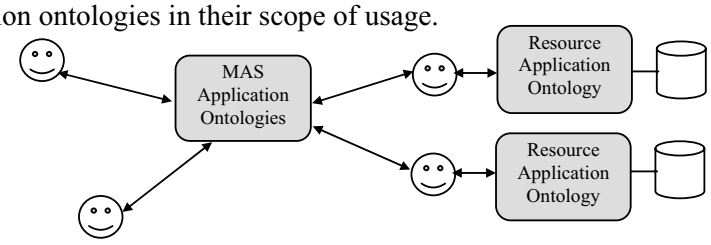


Figure 6.10 – MAS Application ontologies and Resource Application ontologies

The development of MAS Application ontologies is discussed in Section 6.2.4.1, while Resource Application ontologies are examined during the "*MAS Organisation Design*" activity (Section 6.3).

6.2.4.1. Develop MAS Application Ontologies

MAS Application ontologies can be built by selecting concepts from either or both of the relevant Domain ontologies and Task ontologies, thereafter specialising these concepts to suit the MAS application (van Heijst et al. 1997) (Figure 6.11). As recommended by Guarino (1998), a concept in the MAS Application ontology can be a specialisation (or an instantiation) of a concept in a relevant Domain ontology and/or a concept in a Task ontology. For example, the MAS Application ontology of a Car E-business MAS may define concept "Car-price-offer", which is the specialisation of concept "Price" from a Car Domain Ontology and concept "Offer" from a Negotiation Task Ontology.

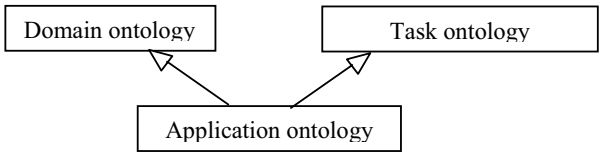


Figure 6.11 – Application ontology as a specialization of Domain ontology and Task ontology, represented in UML (Guarino 1998)

MOBMAS does *not* provide support for the process of ontology development. It assumes that all required MAS Application ontologies for the target system are developed by a separate ontology engineering effort, conducted by domain experts, ontology engineers or the MAS developer himself. MOBMAS however provides techniques for the identification of relevant input Domain ontologies and Task ontologies for the construction of MAS Application ontologies.

6.2.4.1.a. Identify input Domain ontologies and Task ontologies for the construction of MAS Application ontologies

MAS Application Ontologies should obtain and/or specialise concepts from those Domain ontologies that conceptualise the target domains of the MAS. For example, in the illustrative Product Search MAS, since the system deals with search queries on *cars*, its MAS Application ontology should specialise concepts defined in a Car Domain

Ontology, such as "Make", "Model", "Steering" and "Cost". These car-related concepts allow the agents to "understand" and process user queries, as well as communicate the car-related information with each other. Similarly, for the Conference Program Management MAS (Figure 6.9), the developer should reuse and/or specialise the concepts from a Conference Domain Ontology, such as "Paper", "Paper author" and "Reviewer".

In order to identify the target domains of MAS, the developer can investigate the *System Task Model* and *Role Model*. The identified system-tasks and role-tasks provide an overview of the MAS' purpose, scope and behaviour, thereby revealing whether the MAS is related to any specific domains. For example, role-tasks "Accept user query", "Extract keywords from user query" and "Display result for query" indicate the need to know about the Information Retrieval domain, which involves concepts such as "User Query", "Keyword" and "Hit".

MAS Application Ontologies may also need to reuse and/or specialise concepts from Task ontologies. This need arises when the knowledge required to fulfil MAS' tasks do not match the domain knowledge provided by Domain ontologies (Gennari et al. 1994). This mismatch may be a *semantic* gap (i.e. when the knowledge needed for the tasks is missing from the Domain ontology), or a *syntactic* mismatch (i.e. when the knowledge required for the tasks can be satisfied by the domain knowledge, but the domain knowledge needs to be rearranged or at least renamed) (Gennari et al. 1994). If the mismatch is syntactic, the MAS Application ontology can be derived solely from the Domain ontologies. However if the mismatch is semantic, the developer should build the MAS Application ontology by augmenting the Domain ontologies with those concepts obtained from Task ontologies.

For example, consider a Car E-business MAS which involves a "car price negotiation" task. Apart from the need to know about the price of cars (which is a domain-specific concept), the task also needs to deal with negotiation-specific concepts such as offer of car price, utility rating of offer and price settlement. Thus, the system should specialise concepts "Offer" and "Settlement" of the Negotiation Task Ontology into "Car-price-offer" and "Car-price-settlement" for the MAS Application ontology. These specialised

concepts represent the roles that the domain-specific concept "Car-price" plays in the negotiation task (Guarino 1996).

Note that all concepts defined in the MAS Application ontologies are no longer solely domain-specific or task-specific. They are essentially application-specific, that is, related to both domains and tasks at hand.

The developer should investigate the **Role Model**, particularly role-tasks, to identify the need for concepts from Task ontologies. As stated earlier, only role-tasks that require knowledge *semantically* mismatched from domain knowledge will need to be investigated. For example, in the illustrative Product Search MAS, role-task "Find answer to user query" may employ a "ranking by weight" problem-solving method to rank the results found for user query. Thus, it may require task-specific concepts such as "Hypothesis", "Weight" and "Rank". The developer should therefore consult the Ranking –by-Weight Task Ontology to obtain and specialise these concepts. On the other hand, role-tasks "Accept user query" and "Extract keywords from user query" deal directly with the domain concepts "User query" and "Keywords", thus do not require concepts from any Task ontology.

Note that the knowledge requirements of MAS' tasks may not be apparent until the "*Agent Internal Design*" activity (Section 6.4), thus indicating the need for iterative refinement of MAS Application ontologies.

Generally, the developer should consult the existing, reusable Domain ontologies and Task ontologies when building MAS Application ontologies for the target system. However, if no reusable ontologies can be found for the target domains and/or tasks, the developer can either:

- develop the Domain ontologies and Task ontologies from scratch, thereupon building the MAS Application ontologies by specialising these ontologies; or
- directly develop the MAS Application ontologies from scratch, without consulting any Domain ontologies or Task ontologies.

The former approach is time-consuming but facilitates the use of MAS Application ontologies by future applications, while the latter approach is time saving, but not capable of supporting reuse.

The development of MAS Application Ontologies can be supported by numerous specialised ontology-engineering methodologies and guidelines, e.g. IDEF5 (Knowledge Based Systems Inc 1994), METHONTOLOGY (Fernandez et al. 1997), Grüninger and Fox (1995), Uschold and King (1995), Noy and McGuinness (2001), Boicu et al. (1999), Gruber (1993a) and Wache et al. (2001).

6.2.4.1.b. Specify ontological mappings between MAS Application ontologies

When modelling MAS Application ontologies, the developer should pay particular attention to the specification of "*ontological mappings*" between these ontologies where necessary. As defined in Section 2.3.2.1, an ontological mapping is a semantic correspondence between two concepts of two different ontologies (Madhavan 2002). Research in linguistics, logics and psychology has proposed many potential semantic correspondences between concepts (Winston et al. 1987). Chaffin and Herrmann (1988), for example, provides a list of 31 possible semantic correspondences. Winston et al. (1987) presents a taxonomy of semantic correspondences that pertain to part-whole relationships. Storey (1993) suggested seven major semantic correspondences between concepts: "inclusion", "possession", "attribution", "attachment", "synonym", "homonym" and "case". The developer is therefore allowed to adopt whichever semantic correspondences that suit the mapping of the target MAS Application ontologies. However, MOBMAS suggests the developer consider the following three "basic" semantic correspondences, which cover most (if not all) of the possible semantic associations between concepts (Parent and Spaccapietra 1998):

- **equivalent**: i.e. where two concepts are semantically equivalent. For example, concept "Car audio system" in the Car MAS Application Ontology is equivalent to concept "Car audio" in the Entertainment Systems Domain Ontology (Figure 6.14);
- **subsumes**: i.e. where one concept semantically includes another concept (either in term of whole-part, specialisation or instantiation). For example, concept "Type" in

the CarInfo Resource Ontology subsumes concept "Sport car" in the Car MAS Application Ontology (Figure 6.32); and

- **intersects**: i.e. where one concept overlaps partially in semantics with another concept. For example, concept "Car air conditioning system" in the Car MAS Application Ontology intersects with concept "Heater" in the Electronic Product Domain Ontology (Figure 6.32).

The related MAS Application ontologies can either be mapped against each other, or against a common ontology (c.f. Section 2.3.2.1). Normally, when there are more than two ontologies to be mapped amongst themselves, the second approach should be favoured over the first, given the reasons listed in Section 2.3.2.1. The common ontology to be used in the second approach may be one of the existing MAS Application ontologies itself, or built from scratch as an inter-lingua of the existing MAS Application ontologies.

Generating ontological mappings is a labour-intensive and error prone task. The developer is referred to other research work for more information on this activity, e.g. Ehrig and Sure (2004), Kalfoglou and Schorlemmer (2003), Stumme and Maedche (2001), Calvanese et al. (2001) and Madhavan et al. (2002).

6.2.4.1.c. Validate System Task Model and Role Model against Ontology Model

As mentioned previously, ontologies used by MAS are constructed by a separate stream of development effort, conducted by either domain experts, ontology engineers or the MAS developer himself. This ontology development effort, while modelling the target system from the ontological point of view, is closely related and supplementary to the MAS development effort, since both involve a detailed investigation of the target application's requirements.

Accordingly, while the System Task Model and Role Model assist in the development of MAS Application ontologies, the contents of the developed MAS Application ontologies can be used to verify and validate the completeness of the two MAS analysis models. In particular, the concepts defined in the MAS Application ontologies may

correspond to, or indicate, some system-tasks or roles. For example, if concept "Keyword" has been defined in the MAS Application ontology of the Product Search MAS, the developer may uncover system-task "Extract keywords from user query" and add it to the System Task Model if not already included. Similarly, concept "Review" in the MAS Application ontology of the Conference Program Management MAS will clearly indicates a role "Reviewer" for the Role Model. Thus, the examination of ontological concepts may help to identify new system-tasks or roles and thereby help to refine the System Task Model and Role Model.

6.2.4.2. Language for Ontology Model Kind

Since there are currently many representation languages for ontologies (cf. Section 2.3.4), MOBMAS does not restrict itself to any particular modelling language. However, MOBMAS recommends using a graphical language for ontology modelling to facilitate the communication with users during the analysis and design of MAS (e.g. UML, IDEF5 Schematic Language and LINGO). Nevertheless, if a graphical language is not powerful enough in term of power of expression for the ontologies at hand, textual languages can be used (e.g. CycL, KIF, KL-ONE and DAML+OIL).

For the purpose of illustration, MOBMAS adopts *UML* and *OCL* notation for ontology modelling. With this notation, the Ontology Model Kind of MOBMAS consists of multiple **Ontology Diagrams**, each of which represents an ontology as an *UML class diagram*. Ontological concepts are represented as UML ***classes*** or ***attributes*** of classes, while relations between concepts are represented as ***relationships*** between classes (Cranefield and Purvis 1999; Cranefield et al. 2001). Operations/methods of classes are not modelled because ontologies only capture the structure of concepts, not their behaviour (Bergenti and Poggi 2002).

UML allows for the representation of the following types of relationships between concepts (Object Management Group 2003).

- **Specialisation**: that is, when concept A is a type of concept B. For example, "Sport car" is a type of "Car" (Figure 6.14).
- **Aggregation**: that is, when concept A is a part of concept B. For example, "Keyword" is a part of "User query" (Figure 6.15). **Composition** can be used to

model a stronger type of aggregation, when concept A belongs to only one whole concept B and lives and dies with the whole (e.g. concept "Wheel" and "Car" in Figure 6.14) (Object Management Group 2003).

- **Association**: that is, 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 and Poggi 2002). For example, "Car" is related to "Car accessory" via an "accessory" association (Figure 6.14). If an association relationship embraces attributes, the association can be modelled as an "association class" (Figure 6.12).

Each relationship between concepts should be annotated with "cardinality indicators", which indicate the number of potential instances of each concept that can be involved in the relationship.

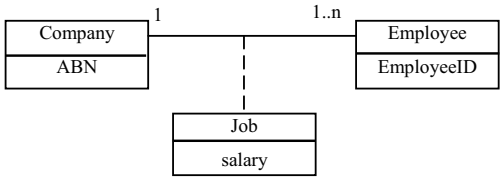


Figure 6.12 – Association Class in an ontology

With regard to the representation of *ontological mappings*, MOBMAS suggests extending the "dependency relationship" of UML (Figure 6.13). Each mapping relationship should be marked with a keyword stating the semantic correspondence, e.g. **equivalent**, **subsumes** or **intersects** (Parent and Spaccapietra 1998). If the semantic relationship is bi-directional (e.g. equivalent or synonym), the arrow can be double-headed.

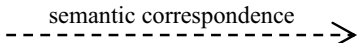


Figure 6.13 – Notation for ontology mapping

If there are axioms, rules or other assertions that specify constraints on ontological classes, attributes and relationships, these can be modelled by OCL. OCL constraints are represented as *notes* in Ontology Diagrams.

Figures 6.14 and 6.15 illustrate the Car MAS Application Ontology and Query MAS Application Ontology of the Product Search MAS.

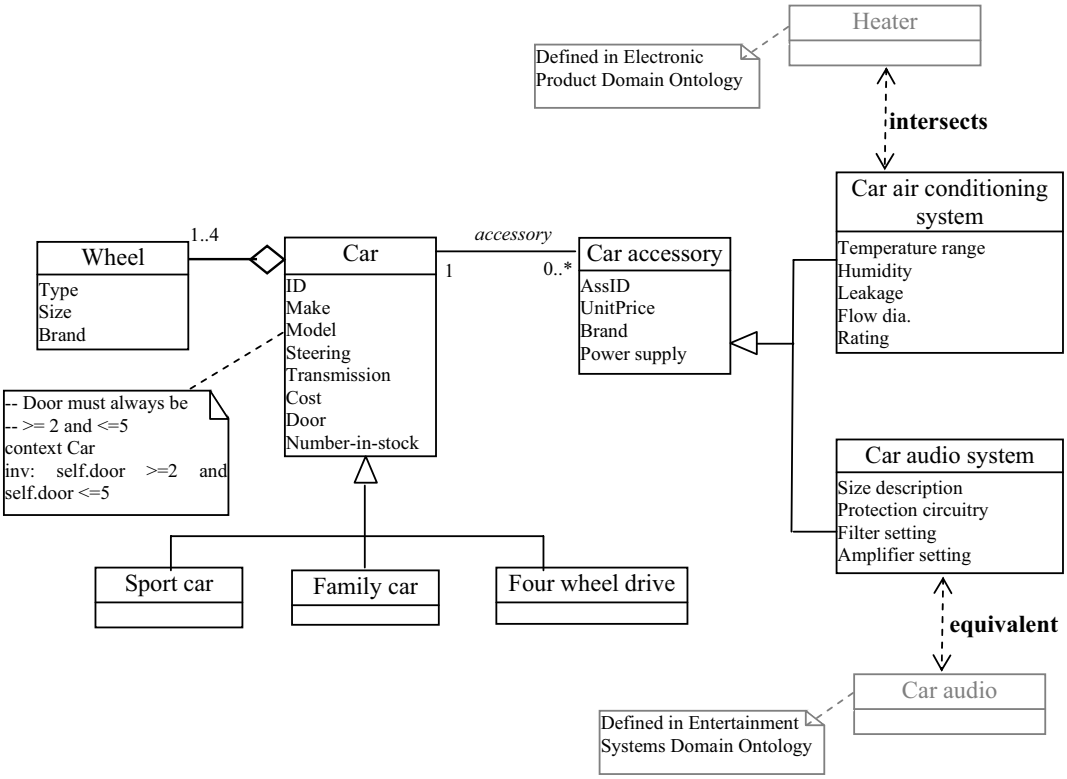


Figure 6.14 – Car MAS Application Ontology

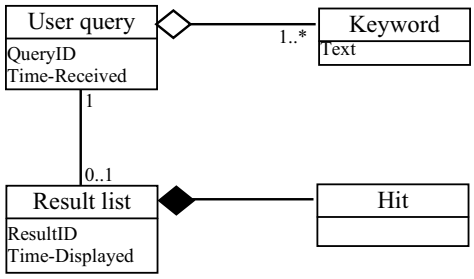


Figure 6.15 – Query MAS Application Ontology

6.2.5. Step 5 – Identify Ontology-Management Role

As recommended by FIPA (2001b), a MAS may store ontologies at an ontology server(s), which is exclusively controlled by an "Ontology Manager" agent[48]. Other agents in the system which wish to obtain, access or update ontologies have to communicate with the Ontology Manager (Figure 6.16).

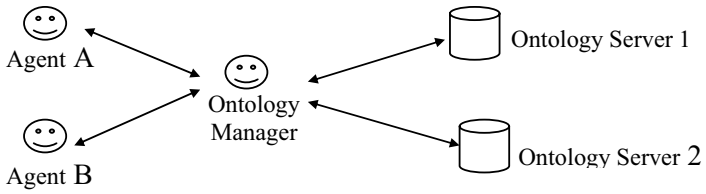


Figure 6.16 – Ontology Manager role

[48] The agent is referred to as Ontology Agent in FIPA (2001b).

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 ontologies to authorised agents;
- to control the update of ontologies (e.g. when suggestions of updates are sent by other agents); and/or
- to inform other agents (which are holding copies of ontologies) of changes in the ontologies.

The use of a specialised "Ontology Manager" agent is useful in that it helps to relieve the workload from 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 or to update an ontology. Note that low-level, simple reasoning and/or maintenance activities of ontologies can be provided by the underlying ontology servers (e.g. Ontolingua Server; Farquhar 1996). What an Ontology Manager agent offers is a higher-level layer of services that may be performed on the ontologies (e.g. authorisation or complex ontology-related query processing).

Note that "Ontology Manager" agent is an application-independent component that is generally provided by the implementation framework (e.g. FIPA-based platforms such as JACK, JADE, FIPA-OS and ZEUS). The developer therefore does not have to design one from scratch, but can fine-tune the provided specification of the "Ontology Manager" agent to suit the application at hand. However, the *Role Model* of the target MAS needs to be updated to add an "Ontology Manager" role.

Nevertheless, the developer can let the agents to have direct access to the ontologies, without using any specialised "Ontology Manager" agent (Cheikes 1995; Figure 6.17). The advantage of this design is its simplicity. However, its drawbacks are that any agent can access or change the ontologies (unless the ontologies are set to "read only" mode) and the agents have to perform all the reasoning and mapping activities on the ontologies to satisfy their ontology-related queries.

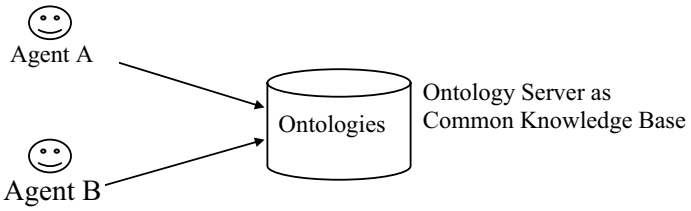


Figure 6.17 – Ontology servers without Ontology Manager role

Figure 6.18 presents the updated Role Diagram for the Product Search MAS (cf. Figure 6.8).

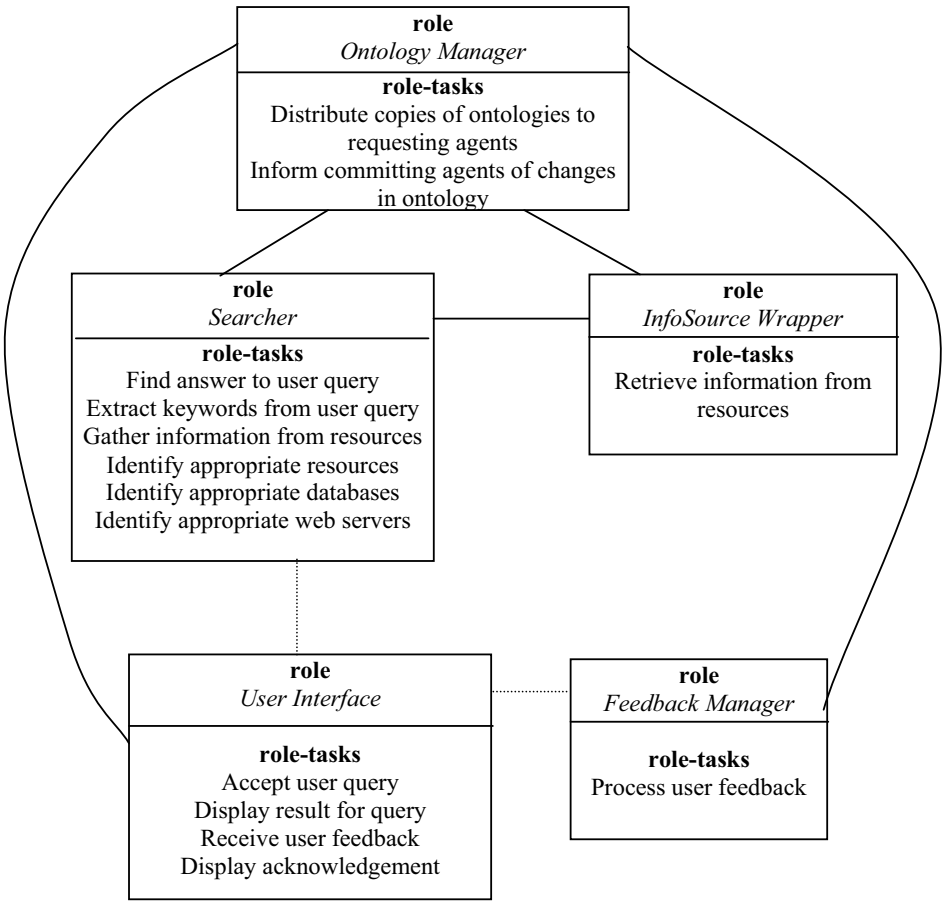


Figure 6.18 – Updated Role Diagram for Product Search MAS

6.3. MAS ORGANISATION DESIGN ACTIVITY

This activity of MOBMAS is concerned with specifying the organisational structure for the target MAS and a set of agent classes composing the MAS. If the MAS is a heterogeneous system that contains non-agent resources, these resources need to be identified and their applications conceptualised.

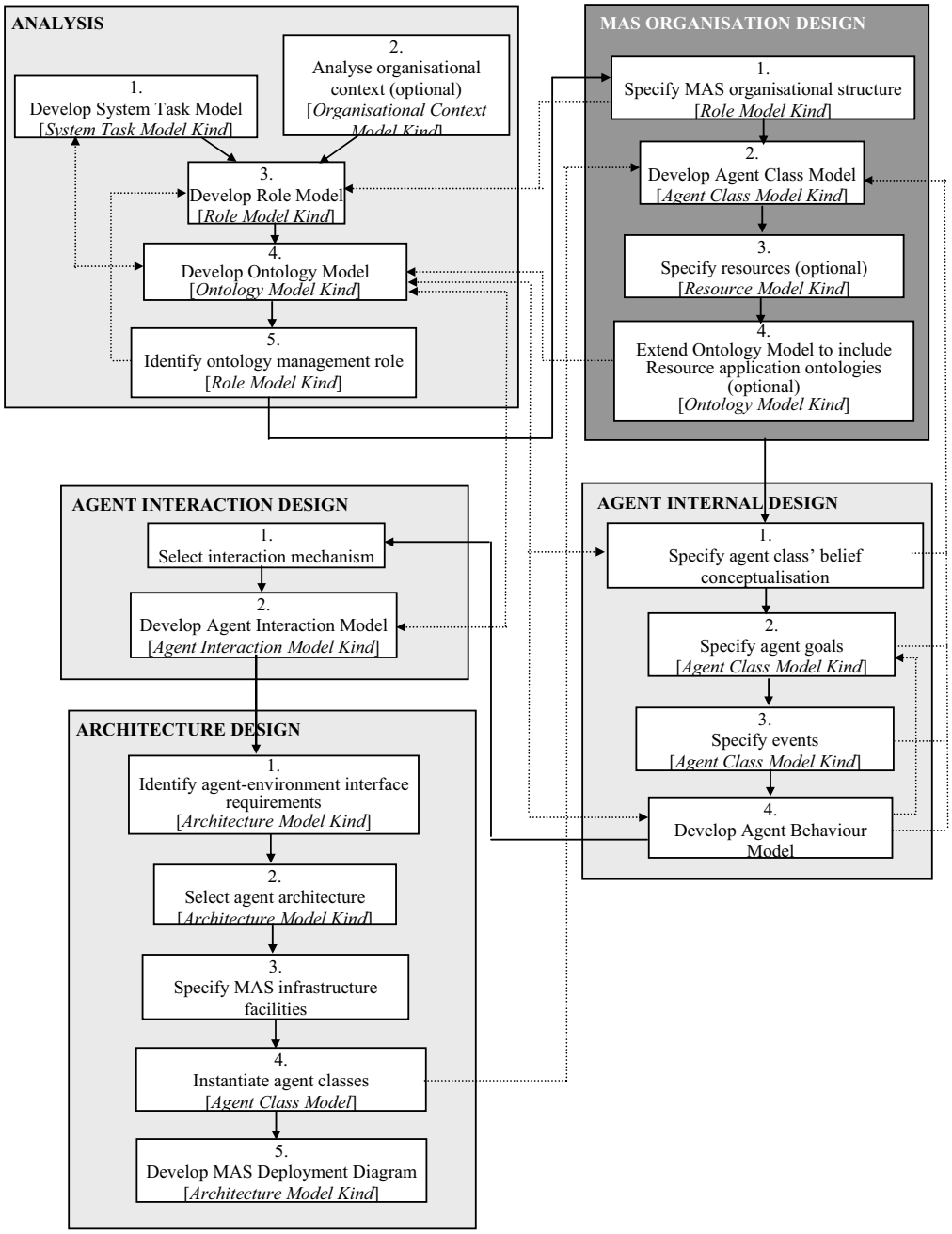


Figure 6.19 – MOBMAS development process

6.3.1. Step 1 – Specify MAS Organisational Structure

In MOBMAS, MAS organisational structure is modelled via *roles*, *acquaintances* between roles and *authority relationships* that govern these acquaintances (e.g. peer-to-peer or superior-subordinate relationship). This inter-role organisational structure will determine the run-time organisational structure between agents, since each agent will play a particular role(s) in the MAS organisation. It should be noted that, while the inter-role organisational structure is static and can be defined at design time, the inter-agent organisational structure may be dynamic, since each agent may change from one role to another at run-time.

In MOBMAS, a preliminary organisational structure of MAS has been defined via the identification of roles and acquaintances between roles in the "*Analysis*" activity (cf. Section 6.2.3). This current step further investigates and confirms the organisational structure of MAS, by examining the organisational structure style, specifying the authority relationships between roles, and if necessary, revising the acquaintances amongst roles. Even if the developer has performed Step 2 of the "*Analysis*" activity – "*Analyse Organisational Context*" (cf. Section 6.2.2), this step should still be performed, since the software organisational structure of MAS should *not* always mimic the structure of the MAS' organisational context. The result of this step is an updated Role Model, which is initially developed in the "*Analysis*" activity (cf. Section 6.2.3).

6.3.1.1. Determine MAS organisational structure

The organisational structure of MAS can be based upon any of the following four common, basic organisational styles (Fox 1981; Lind 1999; Shen and Norrie 1999; Parrott et al. 2003):

- **Peer-to-peer**: In this structure, all roles in MAS work together as peers, with each role assuming an equal authority status compared to other roles (Figure 6.20a). Coordination between roles is based upon mutually agreed decisions. The topology of interactions is a fully connected one, where each role is allowed to interact with any other roles in the system without having to go through a mediator party.

- **Hierarchical structure:** This organisational structure organises roles into a *hierarchy of layers*, where roles in the higher layer assume a "superior" status over those in the lower layer, which assume a "subordinate" status (Figure 6.20b). A superior role exercises its authority over the subordinate roles by delegating work to the latter and coordinating the latter's efforts. A subordinate role is obliged to perform the delegated tasks and should not reject a request from its superiors (thus, the autonomy of the subordinate roles is restrained by the superior roles). Interaction pathways across layers are limited, since roles within each layer are endorsed to only interact with its immediate superiors or subordinates.

- **Federation structure**: This structure organises roles into *peer-to-peer groups*, where roles in each group are mediated by a superior role within the group (Figure 6.20c). Roles within each group can directly interact with each other, but need to interact with roles in other groups via the superior role.

- **Hybrid structure**: This structure is one that integrates any of the above styles. For example, some roles may directly coordinate as peers with each other during the fulfilment of some particular tasks, but need to be controlled by a superior role during some other tasks (Figure 6.20d).

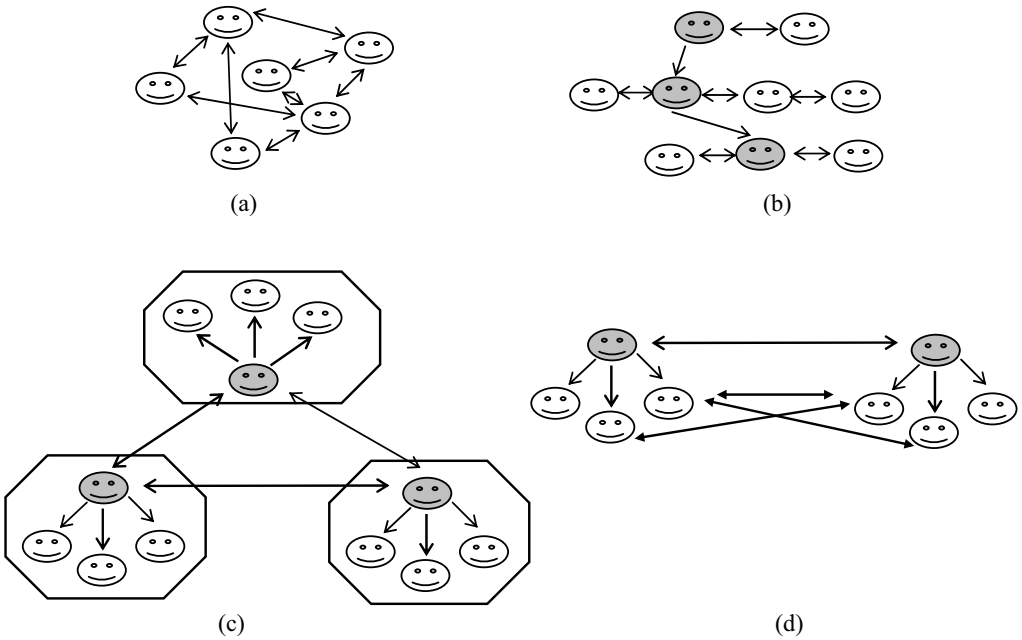


Figure 6.20 – Styles of organisational structure

The determination of which organisational style to adopt for the target MAS should take into account the following factors.

- *The existing structure of the MAS' organisational context* (if Step 2 of the "*Analysis*" activity – "*Analyse Organisational Context*" – has been performed; cf. Section 6.2.2): Generally, there is a natural tendency to mimic/reflect the existing structure of the MAS' organisation context into the software MAS system. For example, MASs that support e-business are likely to mimic the organisational structure of the human commercial transactions. This imitation is not only desirable due to the sake of conceptual simplicity, but in some cases may also come as a requirement (Zambonelli et al. 2003). Nevertheless, the developer should determine whether the existing structure is indeed efficient for the target MAS. For example, a strictly hierarchical structure found in a human organisation may be better replaced by a hybrid software organisation, where certain roles are allowed to autonomously interact as peers with each other to fulfil some tasks. This refined organisational structure promotes the autonomous capability of software roles/agents.

- *Modularity*: The developer should consider organising roles into layers or groups to promote modularity for the target system. Generally, this layering or grouping is applicable if there exist different sets of roles that are concerned with disjoint sets of responsibilities, interact loosely with each other, and/or require competencies that are not required by other sets of roles (Zambonelli et al. 2003).

- *Support for non-functional requirements*: If the target MAS has specific non-functional requirements such as security, scalability or flexibility, the adopted organisational structure should allow the MAS to meet these requirements. For example, if some information available to a group of roles should not be available to another group of roles, the developer should consider organising MAS into layers or groups. Similarly, if scalability needs to be supported (e.g. when new roles are added), federation and hierarchical structures may be more efficient than a peer-to-peer structure, since only the superior role in the respective group or layer needs to know about the existence of the new roles.

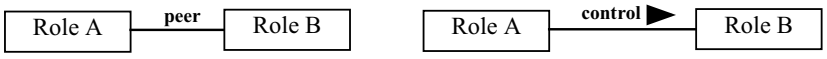
- *The number of roles in the system*: If the number of roles is small, the interaction overheads between roles may be sufficiently low for a peer-to-peer structure to be efficient. However when the number of roles is large, a hierarchical or federation structure may be more appropriate, given their ability to limit the interaction pathways amongst roles.

To date, there have been a number of attempts to catalogue the different organisational structures for software systems (based on the above four basic styles), e.g. Kolp et al. (2001), Tahara et al. (1999), Kendall (1999) and Kendall (2000). The developer can reuse and/or adapt these catalogued structures to the target MAS.

6.3.1.2. Update Role Model

Once the organisational structure of MAS has been determined, the Role Model previously developed in the "*Analysis*" activity should be revised to:

- include any *new roles* that have not been identified. For example, a "Mediator", "Coordinator" or "Broker" role may be uncovered if a hierarchical or federation structure is adopted;
- show *new acquaintances* between roles (if any); and
- show the *authority relationship* governing each acquaintance.
  - Keyword **peer** should be used to represent a *peer-to-peer* relationship where two roles have equal status (Figure 6.21a).
  - Keyword **control** should be used to represent a *superior-subordinate* relationship where one role has authority over another (Figure 6.21b). The keyword should be adorned with an arrow pointing from the superior role to the subordinate role.



(a) (b)

Figure 6.21 – Notation for authority relationships between roles in Role Diagram

Figures 6.22 and 6.23 show the updated Role Diagrams for the Product Search MAS and Conference Program Management MAS respectively. The former adopts a primarily peer-to-peer structure, although it contains a superior-subordinate relationship between "Searcher" and "InfoSource Wrapper" roles. The Conference Program Management MAS adopts a hierarchical structure.

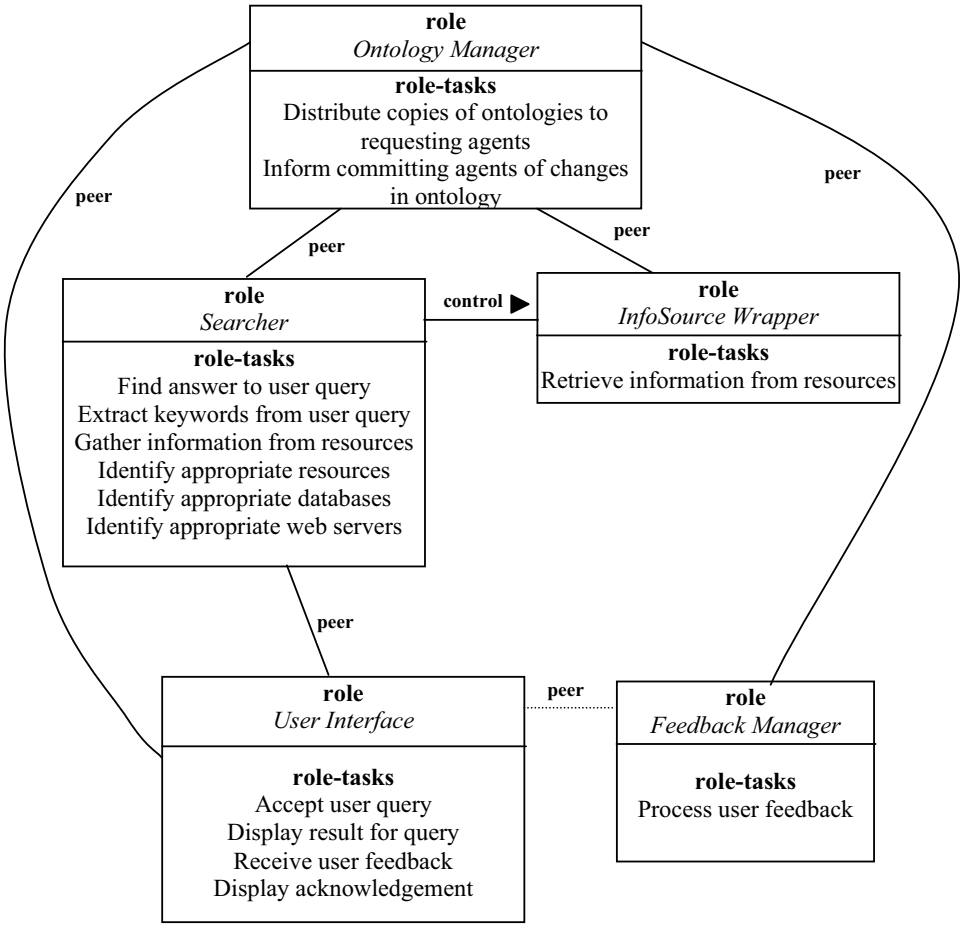


Figure 6.22 – Updated Role Model for Product Search MAS (cf. Figure 6.18)

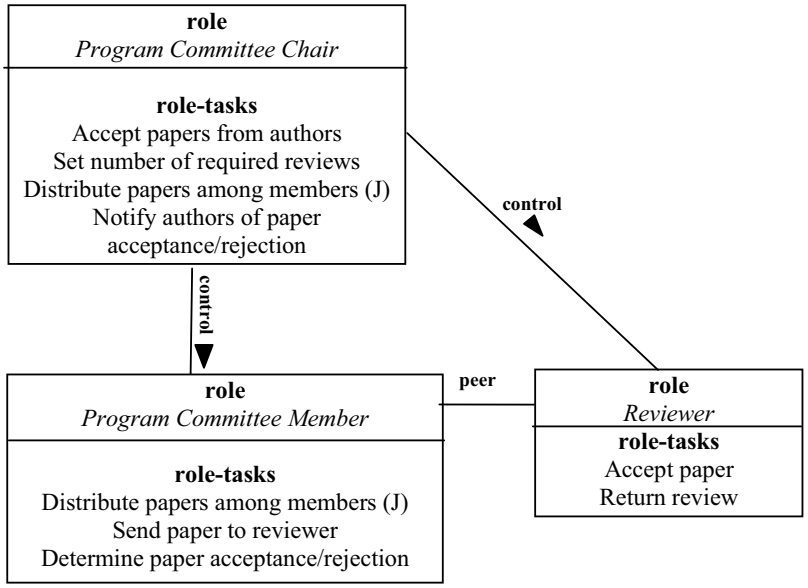


Figure 6.23 – Updated Role Model for Conference Program Management MAS (cf. Figure 6.9)

6.3.2. Step 2 – Develop Agent Class Model

The notion of "agent class" is analogous to the notion of "class" in OO modelling (Wood 2000). Each agent class is a template descriptor of a set of agents with similar characterisation. MOBMAS captures all agent classes in the **Agent Class Model Kind**. Techniques for identifying agent classes are presented in Section 6.3.2.1, while the notation for the Agent Class Model Kind is documented in Section 6.3.2.2.

6.3.2.1. Identify agent classes

In MOBMAS, agent classes are built upon roles, with each agent class being assigned one or more roles. At run-time, an agent from an agent class may dynamically change amongst its assigned roles, thereby exhibiting dynamic behaviour and occupying dynamic positions in the MAS organisation.

Generally, roles can be associated to agent classes via *one-to-one mappings*. This one-to-one correspondence from roles to agent classes can be justified by the modularity or functional coherence of roles – a characteristic resulted from the way roles are identified[49].

Nevertheless, multiple roles may be assigned to one single agent class for the purpose of convenience. The decision of whether, and how, to map multiple roles to one agent class should be driven by the following factors.

- *Modularity*: The grouping of roles must *not* compromise the modularity of the MAS design. In other words, each agent class should represent a *coherent* software entity that does not have disparate functionality, and the overall functionality of the agent class should be easy to understand. The following simple heuristic can be applied to evaluate the functional coherence of an agent class: to find a suitable name for the agent class that encompasses all of its functionality. A coherent agent class should be easily described by a single name without any conjunctions. For example, consider an Online Shop application. An agent class that plays both roles "Client

[49] Recall that in Step 3 of the "*Analysis*" activity – "*Develop Role Model*" (cf. Section 6.2.3.1.a), roles are identified from the system-tasks in such a way that promotes strong internal coherence and loose coupling in term of functionality.

welcomer" and "Seller" can still have a simple descriptive name "Sale assistant" (Padgham and Winikoff, 2002a). Although this heuristic is not always applicable, it offers a useful and quick way for assessing modularity in MAS design.

- *Efficiency considerations*: Associating multiple roles to one single agent class may result in various efficiency improvements. For example, having one agent class playing multiple roles will mean a smaller number of agents in the MAS system than if each agent class plays a single role. Likewise, if some roles interact intensively with each other, the assignment of these roles to one single agent class will mean less interaction between agent classes. Nevertheless, the grouping of roles may also result in lower system efficiency. For instance, mapping both roles "Searcher" and "InfoSource Wrapper" in the Product Search MAS to a single agent class, say "Information Gatherer", will mean that this "Information Gatherer" agent needs to *sequentially* interact with multiple information sources to find answers to a user query. However, if each role "Searcher" and "InfoSource Wrapper" is assigned to a separate agent class "Searcher" and "Wrapper" respectively, the "Searcher" agent can *simultaneously* dispatches the query to all relevant "Wrapper" agents, which then *simultaneously* access the information sources for answers. The response time is therefore greatly improved.

- *Other non-functional requirements considerations*: The binding of multiple roles to one agent class should *not* compromise any fault-tolerant, security or privacy requirements.

6.3.2.1.a. Characterise agent class' dynamics

If a particular agent class has been assigned multiple roles, the developer should characterise its dynamics, that is, whether the agent class is *static* or *dynamic* regarding its role-playing behaviour.

- A *static* agent class is one whose instances are required to play all of the assigned roles throughout their lifetime[50]. For example, a "Shop assistant" agent is active in both of its assigned roles, "Client welcomer" and "Seller", during its lifetime.

[50] MOBMAS' definition of static versus dynamic property of agent classes is based upon the definition of "dynamic activation" proposed by Odell et al. (2003b).

- A *dynamic* agent class, on the other hand, is one whose instances may change their active roles from one time to another. This dynamic change occurs when (Odell at al. 2003b):
  - an agent is initially active in one role but becomes also active in some other roles. For example, in a Human Resource Management MAS, a "Staff" agent constantly plays the role "Employee", but may also become active in role "Manager" as a result of a promotion (thus taking on additional managerial responsibilities apart from normal employee responsibilities); or
  - the agent has been active in one role but becomes inactive in that role. For instance, the "Staff" agent in the above illustration may become inactive in the role "Manager" after a demotion, thus retaining only one role "Employee"; or
  - the agent switches from one active role to another. For example, a "Soccer Player" agent often switches between role "Striker" and role "Defender".

The determination of each agent class' dynamics in term of its role-playing behaviour helps to predict the dynamics of the MAS system at run-time, since the roles that each agent plays at a point in time determine its behaviour, its interactions with other agents and the overall inter-agent organisational structure.

6.3.2.2. Notation of Agent Class Model Kind

The Agent Class Model Kind of MOBMAS is depicted by two notational components.

- **Agent Class Diagram** captures the specification of each agent class, including the listing of its roles, belief conceptualisation, goals and events.
- **Agent Relationship Diagram** shows the interaction pathways between agent classes, interaction pathways between agent classes and their wrapped resources (if exist), and instantiation cardinality of each agent class.

These two diagrams need to be developed in an ongoing manner throughout the MOBMAS development process. MOBMAS notation for Agent Class Diagram and Agent Relationship Diagram are presented in Figures 6.24 and 6.25 respectively.

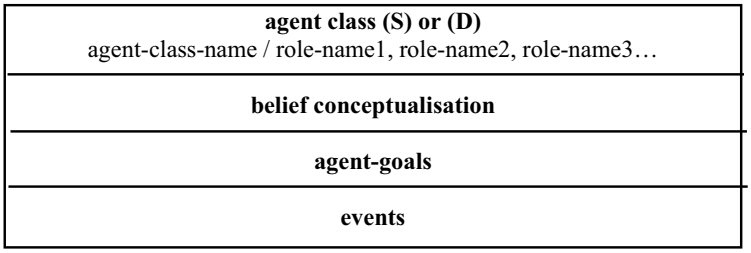


Figure 6.24 – Agent Class Diagram

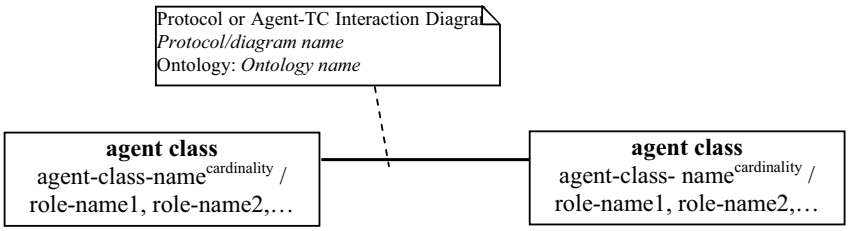


Figure 6.25 – Agent Relationship Diagram

In the **Agent Class Diagram** (Figure 6.24), each agent class is depicted as a rectangle with several compartments.

- The top compartment - marked with keyword **agent class** – specifies the agent class' name, roles and dynamics characteristic. The dynamics characteristic should be annotated as **(S)** for static agent classes, or **(D)** for dynamic agent classes. Note that if an agent class is assigned one single role, it is presumed to be static.
- The remaining compartments specify the agent class' belief conceptualisation, agent-goals and events. These constructs will be defined during the "*Agent Internal Design*" activity. At this stage, these compartments can be left empty.

In the **Agent Relationship Diagram**,

- each agent class is depicted as a rectangle marked with keyword **agent class**. The agent class' name and its roles should be restated. For simplicity, this diagram does not show the dynamics characteristic of each agent class. This information has been captured in the Agent Class Diagram;
- each acquaintance between agent classes is depicted as an undirected line connecting the agent classes. Inter-agent acquaintances can be derived from the acquaintances amongst roles in the Role Model; and
- descriptive information about each acquaintance between agent classes (e.g. ontology and interaction protocol governing the acquaintance) is attached to the

acquaintance as an UML note. This descriptive information will be determined during the "*Agent Interaction Design*" activity.

If the target MAS is a heterogeneous system which contains non-agent resources, the Agent Relationship Diagram should also show these resources and their connections with the wrapper agent classes. This issue will be discussed in Step 3 of this activity – "*Specify resources*" (Section 6.3.3).

Figures 6.26 and 6.27 present the preliminary Agent Class Diagram and Agent Relationship Diagram for the illustrative Product Search MAS (cf. Figure 6.22). Both diagrams will be refined in later steps of MOBMAS.

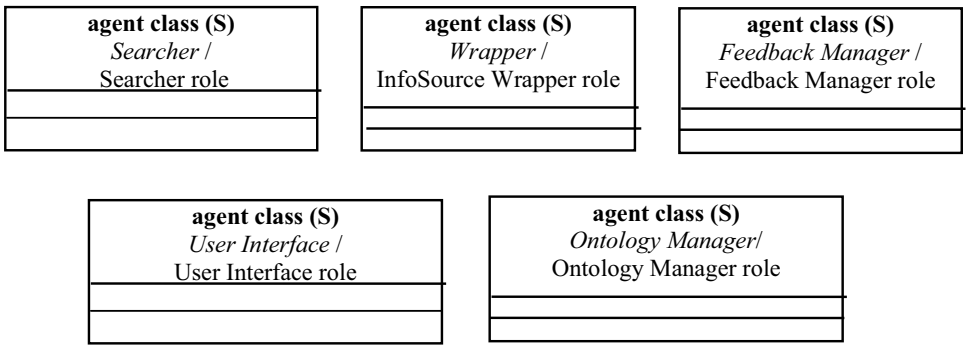


Figure 6.26 – Preliminary Agent Class Diagram for Product Search MAS

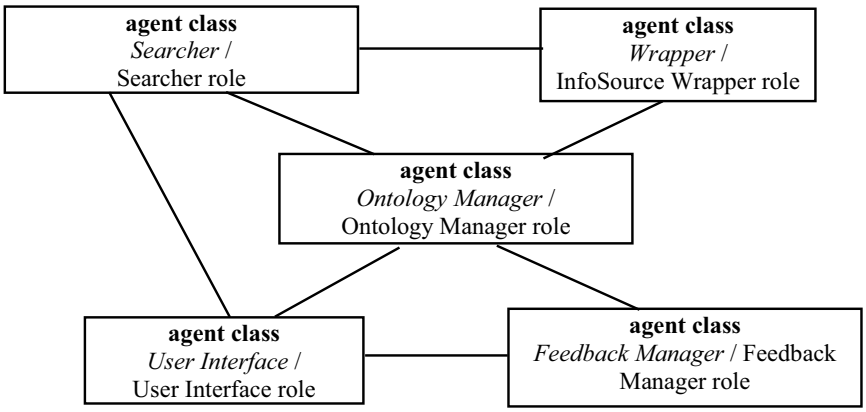


Figure 6.27 – Preliminary Agent Relationship Diagram for Product Search MAS

6.3.3. Step 3 – Specify Resources (Optional)

This step is only needed if the target MAS is a heterogeneous system which, apart from agents, contains non-agent resources that provide information and/or services to the agents. All of these resources are specified in the **Resource Diagram** of MOBMAS **Resource Model Kind**. Section 6.3.3.1 discusses the identification of resources for the target MAS, while Section 6.3.3.2 presents notation for the Resource Diagram.

6.3.3.1. Identify resources

A resource is a non-agent software system that provides application-specific[51] information and/or services to the agents in MAS. Resources in a MAS may include (FIPA 2001a; Jennings and Wooldridge 1995):

- *information sources*, e.g. databases or web servers; and
- *processing application systems*, e.g. legacy systems, language translation programs or web services programs.

Note that the resources do not need to belong *internally* to the system and owned/used exclusively by the system (e.g. legacy system) (Figure 6.28a). They may exist externally and are available to agents in other systems (e.g. web servers) (Figure 6.28b).

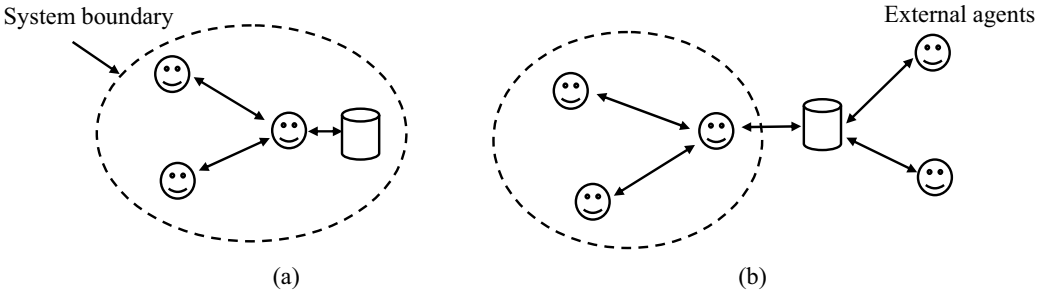


Figure 6.28 – Internal resources (a) and external resources (b)

Resources in a MAS can be identified by investigating the System Task Model or Role Model. These models specify the functionality of the target application, thereby revealing the major resources that accompany with, or are required by, the target

[51] Resources are to be distinguished from infrastructure facilities which provide system-specific services such as naming service or message transport service. The specification of infrastructure facilities is discussed in Step 3 of "*Architecture Design*" activity – "*Specify MAS Infrastructure Facilities*" (Section 6.6.3).

system. For example, the potential resources for the illustrative Product Search MAS are various external databases, web servers and search engines on cars (cf. Section 6.1.4.1). However, more resources may be uncovered during the detailed design of agents' behaviour (i.e. "*Agent Internal Design*" activity – Section 6.4). Thus, the Resource Diagram needs to be iteratively revised.

6.3.3.2. Notation of Resource Diagram

A Resource Diagram should display the following notational elements.

- **Resources**: Each resource is depicted as a rectangle with multiple compartments.
  - The top compartment, marked with keyword **resource**, specifies the resource's name.
  - Each remaining compartment describes the resource from a different dimension. The keyword in each compartment indicates its respective dimension. Some potential dimensions are:
    - ***Resource type***: which defines the category of the resource, e.g. "database", "web server" or "processing application system"; and
    - ***Resource Application Ontology***: which states the name(s) of the Resource Application ontology(ies) that conceptualises the application provided by the resource. The construction of these ontologies is discussed in Step 4 of this phase – "*Extend Ontology Model to include Resource Application ontologies*" (cf. Section 6.3.4).

  MOBMAS does *not* impose the above set of dimensions as a fixed template for resource modelling. These dimensions may vary largely from one type of resource to another, and/or from one development project to another. Therefore, the developer is free to adapt the Resource Diagram to fit the project at hand.

- **Agent classes that wrap around resources** (i.e. "wrapper" agent classes): A resource is typically accessed by agents via a dedicated "wrapper" agent (Jennings and Wooldridge 1995; FIPA 2001a). In the Resource Diagram, a wrapper agent class is represented as a rectangle with keyword **agent class**, followed by its name and roles.

- **Connections between agent classes and resources**: Each resource is linked with its wrapper agent class via an undirected line marked with keyword **wrap**.

An example Resource Diagram for the Product Search MAS is presented in Figure 6.29.

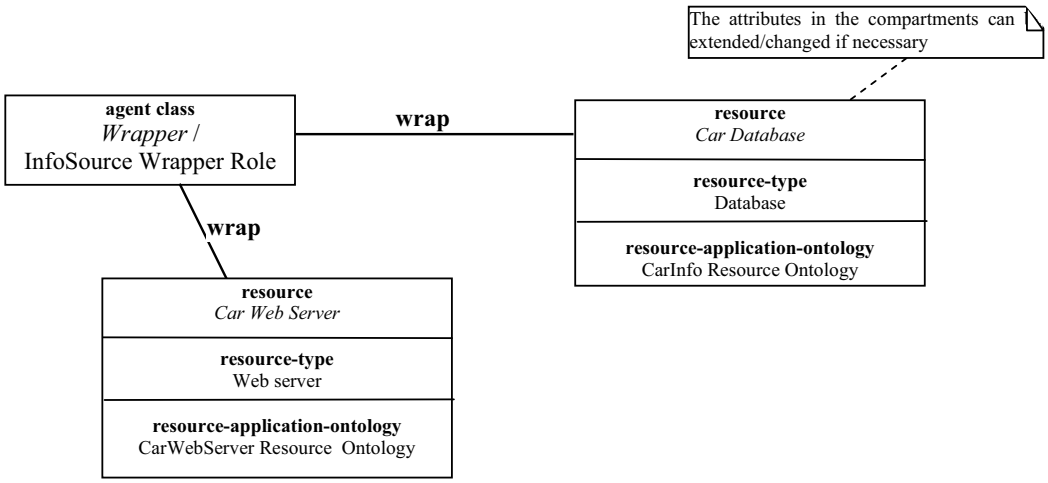


Figure 6.29 – Resource Diagram of Product Search MAS

6.3.3.3. Revise Role Model

Since each resource requires an associated wrapper agent class, the developer should revise the *Role Model* to check if all necessary wrapper role(s) has been identified. Each different type of resource may require a different wrapper role if the responsibilities involved in assessing each resource type are largely different from each other, e.g. "Database Wrapper" role or "Expert System Wrapper" role.

In addition, if the target MAS is open and frequently adds new resources and agents, the developer should consider introducing a "Resource Broker" role. Agents playing this role are responsible for brokering the available resources to interested agents (FIPA 2001a, O'Brien and Nicol 1998). Any newly added wrapper agents (as a result of the addition of new resources) can register its services with the "Resource Broker" agent. A client agent (who may be newly added to the system) can contact the "Resource Broker" to find out which wrapper agents can satisfy its requests. The client agent can then directly interact with the identified wrapper agents for services. The "Resource Broker" agent may also assume the responsibilities of negotiating over the terms and conditions of resource usage, or authorizing the client agents before giving them details of wrapper agents (FIPA 2001a). Note that, as with "Ontology Manager" agent, "Resource Broker"

agent is an application-independent component that is generally provided by the implementation framework (e.g. FIPA-based platforms such as JACK, JADE, FIPA-OS and ZEUS). The developer therefore does not have to design one from scratch, but can fine-tune the provided specification of the "Resource Broker" agent to suit the application at hand. Nevertheless, the Role Model should be extended to include the new "Resource Broker" role.

Figure 6.30 presents the updated Role Diagram for the Product Search MAS (cf. Figure 6.18).

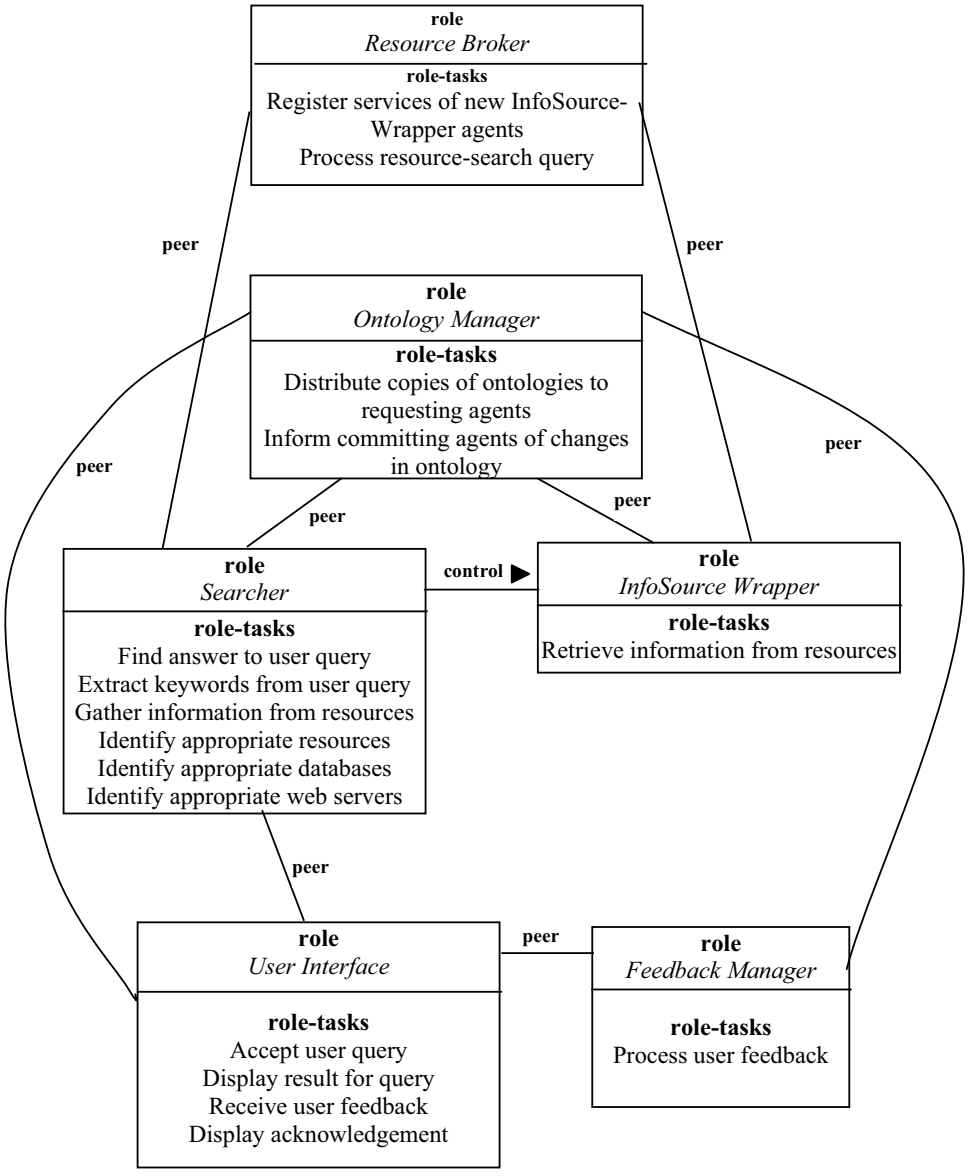


Figure 6.30 – Updated Role Diagram for Product Search MAS (cf. Figure 6.18)

6.3.3.4. Update Agent Class Model

Agent Relationship Diagram of MOBMAS' Agent Class Model Kind should be extended to show newly identified resources and their connections with wrapper agent classes.

- Each resource is depicted as a rectangle marked with keyword **«resource»**. For simplicity, the Agent Relationship Diagram only shows the names of the resources without specifying their internal configuration. This information has been captured in the Resource Diagram of Resource Model Kind.
- The connection between resources and their wrapper agent classes is represented as an undirected line marked with keyword **wrap**.

In addition, if the Role Model has been changed, both *Agent Class Diagram* and *Agent Relationship Diagram* need to be updated to show new agent classes and/or new role assignments to existing agent classes.

Figure 6.31 presents the updated Agent Relationship Diagram for the Product Search MAS (cf. Figure 6.27).

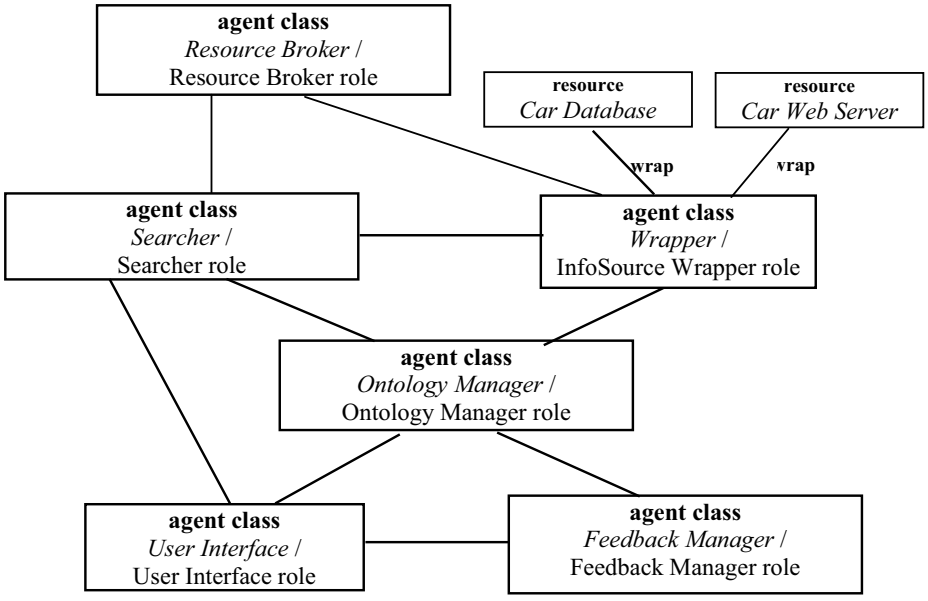


Figure 6.31 – Updated Agent Relationship Diagram for Product Search MAS

6.3.4. Step 4 – Extend Ontology Model to Include Resource Application Ontologies (Optional)

If the target MAS contains resources, the developer needs to extend the **Ontology Model** to include Resource Application ontologies that conceptualise the applications provided by these resources. As stated in Section 6.2.4:

- if the resource is a *processing application system* (e.g. a legacy system), its Resource Application 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 Resource Application ontology should capture all the concepts and relations that conceptualise the information stored in the resource. This Resource Application 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 Resource Application ontology. The development of Resource Application ontologies is not part of MOBMAS. The developer is referred to other research work on Resource Application ontology development, e.g. Hwang (1999), Pazzaglia and Embury (1998), Mars et al. (1994), Decker et al. (1999) and FIPA (2001b).

The following section discusses the issue of ontology mapping between Resource Application ontologies and MAS Application ontologies.

6.3.4.1. Specify ontological mappings between Resource Application ontologies And MAS Application ontologies

Ontological mappings between Resource Application 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 Resource Application 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 as an inter-lingua (cf. Section 2.3.2.1).

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. If otherwise 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 inter-lingua.

Figure 6.32 presents an example Resource Application ontology for the Car Database used by the Product Search MAS, named "CarInfo Resource Ontology" (Figure 6.29). The figure also shows the ontological mappings between the CarInfo Resource Ontology and Car MAS Application Ontology (cf. Figure 6.14).

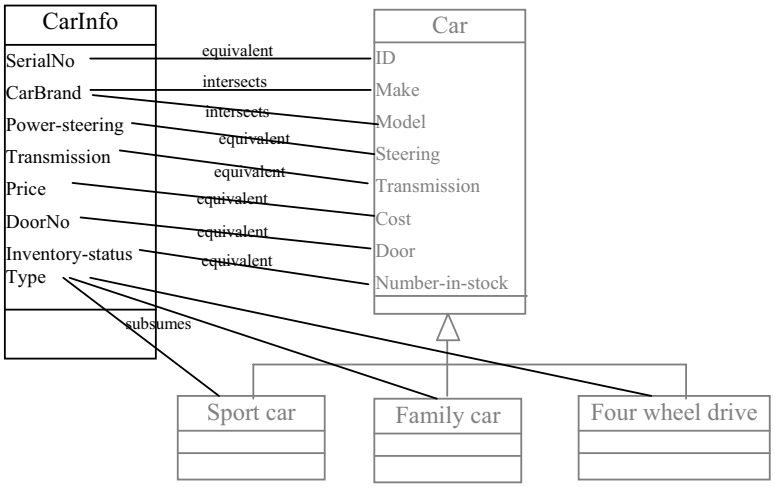


Figure 6.32 – CarInfo Resource Ontology and its mappings to Car MAS Application Ontology

6.4. AGENT INTERNAL DESIGN ACTIVITY

This activity of MOBMAS deals with the internal design of each agent class, namely the specification of each agent class' belief conceptualisation, agent goals, events, plan templates and reflexive rules.

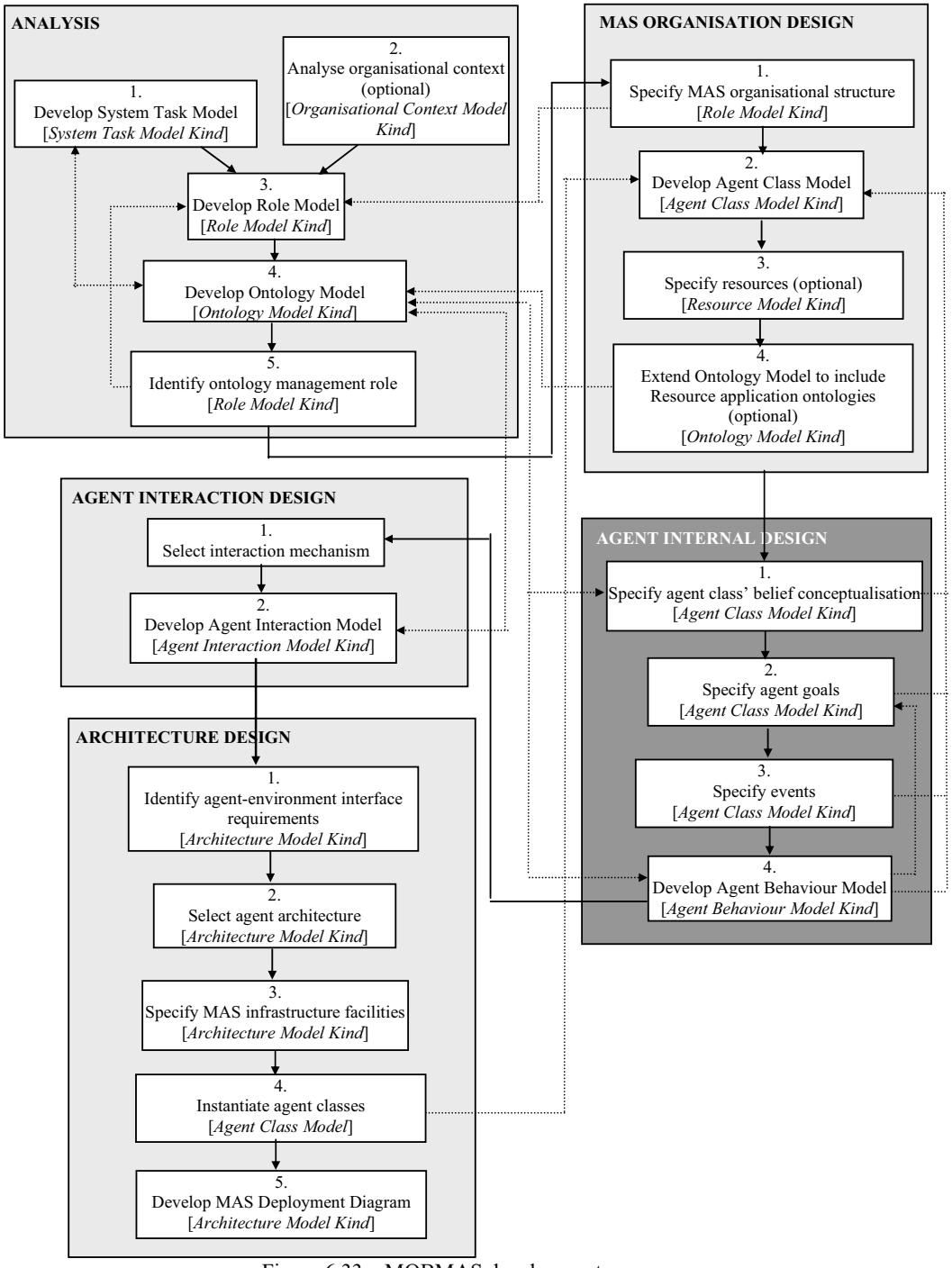


Figure 6.33 – MOBMAS development process

6.4.1. Step 1 – Specify Agent Class' Belief Conceptualisation

Agent beliefs refer to the information that an agent holds about the world (Shoham and Cousins 1994; Rao and Georgeff 1995). Agent beliefs exist at two levels of abstraction: *Belief State* and *Belief Conceptualisation* (Kinny and Georgeff 1996; Agent Oriented Software 2004).

- **Belief State**: corresponds to an agent's knowledge about a *particular state* of the world (in the past, present or future) (Shoham 1993) (Figure 6.34). It captures the *run-time facts* about the state of entities that exist in the agent's application (i.e. domains and tasks) and the environment (i.e. resources and other agents).
- **Belief Conceptualisation**: corresponds to the agent's knowledge about the *conceptualisation* of the world, particularly the conceptualisation of the entities referred to in the Belief State (Figure 6.35).

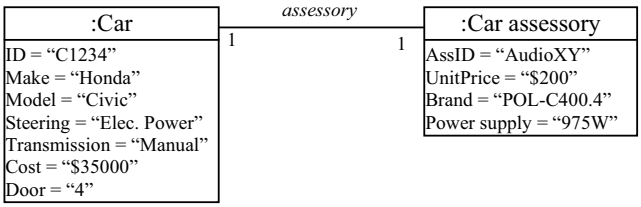


Figure 6.34 – Agent Belief State

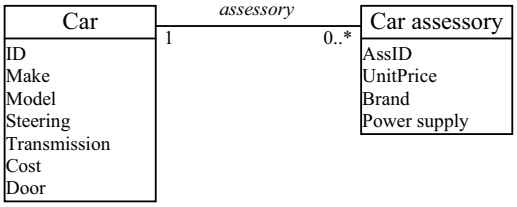


Figure 6.35 – Agent Belief Conceptualisation

At design time, it is only feasible to define the Belief Conceptualisation for each agent class, or more exactly, the *initial start-up* Belief Conceptualisation (because agents in each class may dynamically update their conceptualisation of the world during their lifetime). Belief States are run-time knowledge and therefore can only be populated when the agents of each class interact with the environment to obtain factual information about the world.

6.4.1.1. Specify belief conceptualisation of agent classes

Since an agent class' Belief Conceptualisation stores *conceptual* knowledge of the agent class' world, it should be composed of those **ontologies** which conceptualise the agent class' knowledge of its application (i.e. domains and tasks) and/or wrapped resources' applications[52]. Even though at run-time, agents also maintain beliefs about other agents in the environment, the conceptualisation of the "agent" component is normally imposed by the agent implementation platform and implicitly embedded into the agent coding. For example, JACK (Agent Oriented Software 2004) conceptualises the "agent" component in terms of "name", "capability", "event" and "database", while JADE (Fabio et al. 2004) defines each agent in terms of "name", "address" and "resolver". MOBMAS therefore only investigates the conceptualisation of "application", namely MAS application and resources' applications. As such, the specification of an agent class' Belief Conceptualisation comes down to the determination of **which (part of[53]) MAS Application ontologies and/or Resource Application ontologies the agent class should commit**.

6.4.1.1.a. Identify ontology commitments of agent classes

In general, an agent class needs to commit to a particular (part of) ontology if the agent class' functionality is related to the domain, task or resource that this (part of) ontology conceptualises. In MOBMAS, an agent class' functionality is reflected via its *roles* and *role-tasks*. For example, the "Searcher" agent class in the Product Search MAS plays the "Searcher" role, thereby being responsible for processing car-search queries (Figure 6.30). Accordingly, the "Searcher" agent class should commit to the Car MAS Application Ontology and Query MAS Application Ontology in order to know about car-related concepts (e.g. "Make", "Model" and "Transmission"; Figure 6.14) and querying-related concepts (e.g. "Keyword", "Result list" and "Hit"; Figure 6.15).

---

[52] Only agent classes that directly wrap around the resources need to commit to the corresponding Resource Application ontologies. Other agent classes in the system which wish to use the resources can interact with the wrapper agent classes using ACL messages formulated in MAS Application ontologies (Jennings and Wooldridge 1995; FIPA 2001a).

[53] In many cases, the agent class only needs to commit to a fragment of a particular MAS Application ontology or Resource Application ontology to do its work.

In addition, an agent class' functionality and its required ontologies may also be identified by investigating:

- *resources* wrapped by the agent class (cf. Agent Relationship Diagram of Agent Class Model or Resource Diagram of Resource Model Kind); and/or
- the *acquaintances* of the agent class and other agent classes in the system (cf. Agent Relationship Diagram of Agent Class Model Kind).

For example, the "Wrapper*"* agent class of the illustrative Product Search MAS needs to commit to the CarInfo Resource Ontology and CarWebServer Resource Ontology, since it wraps around the Car Database and Car Web Server resources (Figure 6.29). Besides, the "Wrapper*"* agent class also needs to commit to the Car MAS Application Ontology, because it needs to communicate car-related messages with the "Searcher" agent class (Figure 6.31).

It should be noted that not all ontological commitments of an agent class are apparent at this stage. The developer should proceed to the specification of agent classes' behaviour (i.e. Step 4 of this activity – "*Develop Agent Behaviour Model*"; Section 6.4.4) and agent classes' interactions (i.e. "*Agent Interaction Design*" activity; Section 6.5) in order to get more insight into the knowledge requirements of each agent class. Consequently, the development of each agent class' Belief Conceptualisation is an ongoing process.

At run-time, the initial start-up Belief Conceptualisation specified at design time for each agent class may be dynamically modified. Agents of each class may *extend* their Belief Conceptualisations to include the conceptualisation of new domains, tasks or resources, and/or *update* their Belief Conceptualisations with a new conceptualisation of their existing domains, tasks or resources. The extension of a Belief Conceptualisation normally involves the addition of new (parts of) MAS Application ontologies or Resource Application ontologies into the Belief Conceptualisation, while the update of a Belief Conceptualisation typically requires the modification of the existing MAS Application ontologies or Resource Application ontologies. When an ontology has been modified, all other agents in the MAS that commit to the same ontology should be notified of this modification, so that they can accordingly update their Belief

Conceptualisations. The mechanism of how the modification of ontologies can be propagated across agents at run-time is largely dependent on how ontologies are managed in the MAS. In particular, if an "Ontology Manager" agent class is used to take care of the distribution and maintenance of the ontologies (cf. Section 6.2.5), any agent which wants to modify an ontology can send the modification (or request to modify) to the "Ontology Manager". The "Ontology Manager" then multicasts this modification to all other agents that commit to the modified ontology. Otherwise, if no "Ontology Manager" exists, agents in the MAS will have to communicate the modifications directly to each other, probably in a serial manner. It should be noted that when an existing ontology has been modified, the *Belief State* of an agent committing to that ontology must also be modified to adjust the recorded run-time facts to the new conceptual structure. This modification requires the mapping/revision of knowledge that is not part of MOBMAS.

6.4.1.2. Update Agent Class Model to show belief conceptualisation

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to specify the ontologies that each agent class commits. The developer only needs to show the *names* of the ontologies in the **belief conceptualisation** compartment (Figure 6.36). Ontologies themselves are modelled in the Ontology Model Kind.

Figure 6.36 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown.

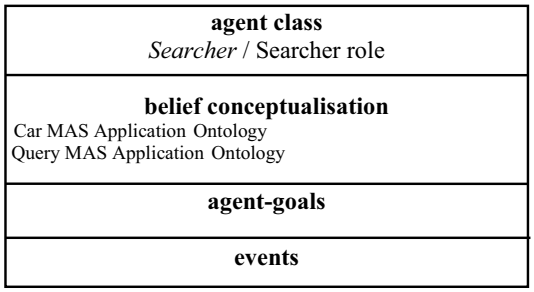


Figure 6.36 – Updated Agent Class Diagram for Product Search MAS ("Searcher" agent class)

6.4.2. Step 2 – Specify Agent Goals

An **agent-goal** is a state of the world that an agent class would like to achieve or satisfy (Silva and Lucena 2004; Wooldridge 1999). It signifies the purpose of existence of an

agent class. In MOBMAS, agent-goals are derived directly from role-tasks, since role-tasks describe what the agent class is responsible for fulfilling when playing its roles. The state of the world that each role-task seeks to achieve, satisfy or maintain indicates an agent-goal. For example, role-tasks "Accept user query" and "Display result for query" of "User Interface" agent class (Figure 6.30) indicate two agent-goals, "Incoming user query is accepted" and "Available result for query is displayed" respectively. Meanwhile, two role-tasks "Extract keywords from user query" and "Find answer to user query" of "Searcher" agent class (Figure 6.30) result in two agent-goals "Keywords are extracted from user query" and "Answer is found for user query" respectively.

Note that two different agent classes may have an identical agent-goal if they are mutually in charge of a "*joint task*". Recall that a joint task is one that requires the collective effort of more than one role (cf. Section 6.2.3.1.a). For example, in the illustrative Conference Program Management application, two roles "PC Chair" and "PC Member" are mutually in charge of a joint task "Distribute papers among members" (Figure 6.9). This joint task needs to be mapped to an agent-goal "Papers are distributed among members" in *each* of the two agent classes "PC Chair" and "PC Member". As such, the two agent classes aim to achieve an identical agent-goal. Since all joint tasks have been highlighted with the adornment ***(J)*** in the Role Diagram of Role Model Kind (cf. Section 6.2.3.3), it should be easy at this stage to identify the existence of all the joint tasks in the target system.

6.4.2.1. Update Agent Class Model to show agent-goals

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to show the agent-goals of each agent class in the **agent-goals** compartment. At design time, agent-goals can be specified in an informal natural language. Since agent-goals represent states, they should be defined in the form "*something is achieved/satisfied*", not as a phrase starting with an imperative as in system-tasks or role-tasks.

Figure 6.37 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown (cf. Figure 6.30).

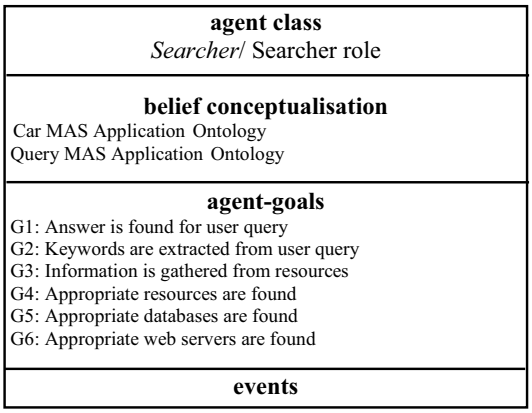


Figure 6.37 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

6.4.2.2. Develop Agent Goal Diagram (Optional)

If a particular agent class is found to pursue multiple agent-goals and these agent-goals are related, an **Agent Goal Diagram** can be developed to capture the relationships among the agent-goals. This diagram is an optional notational component of the **Agent Behaviour Model Kind**, which is examined in Step 3 of this activity – "*Develop Agent Behaviour Model*" (Section 6.4.4).

The Agent Goal Diagram of each agent class may capture the following types of agent-goal relationships.

- *Decomposition relationship*: an agent-goal of an agent class may be decomposed into sub-agent-goals if the role-task from which it is derived has sub-role-tasks. For example, in Figure 6.38, agent-goal "G1-Answer is found for user query" is decomposed into sub-agent-goals "G2-Keywords are extracted from user query" and "G3-Information is gathered from resources", because agent-goal "G1" is derived from role-task "Find answer for user query", which has sub-role-tasks "Extract keywords from user query" and "Gather information from resources" that indicate sub-agent-goals "G2" and "G3" respectively (Figure 6.30). The decomposition of an agent-goal can be an AND-decomposition or an OR-decomposition, depending on the nature of the decomposition of the role-task from which the agent-goal is derived.

- *Goal conflict relationship*: agent-goals of a particular agent class may be in conflict with each other (i.e. intra-agent conflicts[54]). For example, regarding a MAS for library management, agent-goals "User's book extension request is satisfied" and "Reserved book is recalled" of a "Librarian" agent class is in conflict with each other when the book requested for extension is also a reserved book.

  Conflicts between agent-goals may be resulted from the conflicts between system-tasks that these agent-goals aim to achieve[55]. For example, the two conflicting agent-goals "User's book extension request is satisfied" and "Reserved book is recalled" of the library management MAS aim to achieve two conflicting system-tasks "Maintain long borrowing period" and "Maintain regular availability" (cf. Section 6.2.1). The developer should trace through the *Role Diagram* of Role Model Kind and *System Task Diagram* of System Task Model Kind to identify any potential conflicts among agent-goals.

The notation of Agent Goal Diagram is presented below. Many notational elements are reused from the System Task Diagram (cf. Section 6.2.1.1). Each Agent Goal Diagram should be labelled with the name of the respective agent class.

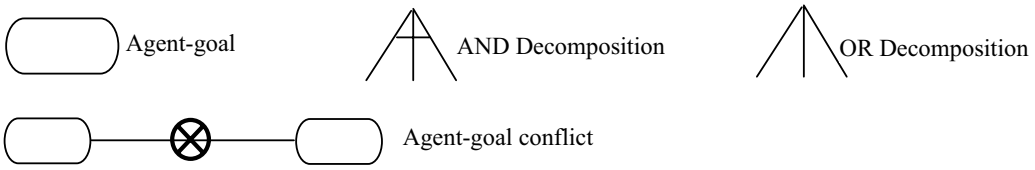


Figure 6.38 illustrates the Agent Goal Diagram for the "Searcher" agent class in the Product Search MAS (cf. Figure 6.37).

[54] The issue of inter-agent conflicts (i.e. conflicts between agent-goals of different agent classes) will be discussed in the "*Agent Interaction Design*" activity (Section 6.5).
[55] Recall that agent-goals are derived from role-tasks, which are in turn mapped from system-tasks (cf. Section 6.2.3.2).

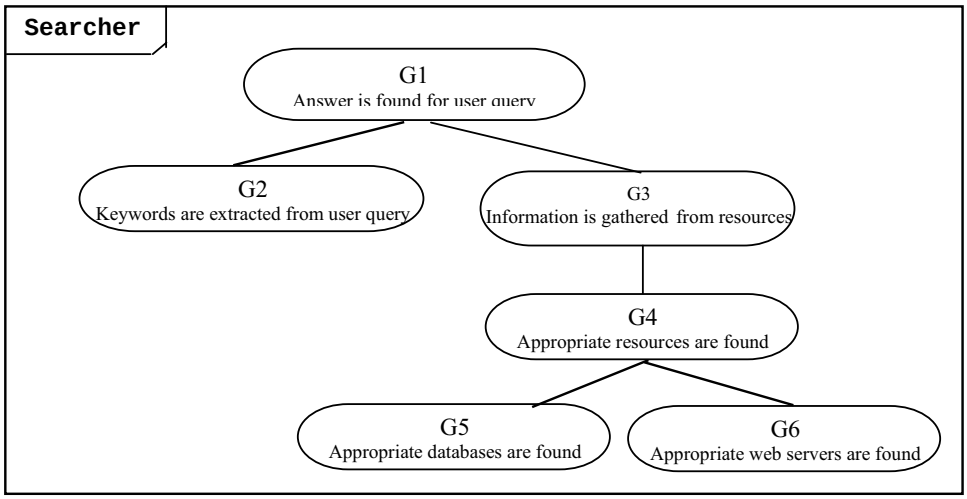


Figure 6.38 – Agent Goal Diagram of "Searcher" agent class of Product Search MAS

6.4.3. Step 3 – Specify Events

At run-time, even though agent-goals of a particular agent may be activated[56] proactively by the agent itself (i.e. the agent takes the initiative to pursue the agent-goal[57]), an agent-goal may also be activated by an event coming from the environment. For example, agent-goal "Answer is found for user query" of a "Searcher" agent is activated by a communication message sent by an "User Interface" agent. In addition, an agent's course of actions to achieve a particular agent-goal may also be affected by certain changes in the environment, for example, changes that cause an agent-goal to be "deactivated"[58], or that cause the current course of actions to be no longer applicable. As a result, an agent in a MAS is required to constantly perceive the environment and respond to relevant events within it (Wooldridge 1999).

An **event** is defined as a significant occurrence in the environment that an agent may respond (Winikoff et al. 2001). It may be generated in various ways: by agents via the execution of their actions, by human users via their inputs into the system, or by resources via the execution of their services (Silva and Lucena 2004).

---

[56] The term "activate" is used to mean that the agent starts carrying out some processing to satisfy an agent-goal. Accordingly, an active agent-goal is one that is being actively pursued or satisfied.
[57] For example, agent-goals "Keywords are extracted from user query" and "Appropriate resources are identified" of the "Searcher" agent class are proactive because they can be activated by the "Searcher" agent itself.
[58] The term "deactivate" is used to mean that the agent stops its processing to pursue or satisfy an active agent-goal.

For each agent class, the developer should identify those events that agents of that class need to respond at run-time. These events can typically be derived from stimuli in the environment[59] which:

- *activate agent-goals of the agent class*: For example, agent-goal "User query is accepted" of the "User Interface" agent class is activated by the event "Input of user query". Meanwhile, agent-goal "Information is retrieved from resource" of the "Wrapper" agent class is activated by the event "Incoming message from Searcher agent"; or
- affect the agents' course of actions to fulfil the agent-goals. For example, agent-goal "Answer is found for user query" of the "Searcher" agent class is cancelled if a cancel request is received from the human user (i.e. an event "Input of cancel message from user").

6.4.3.1. Update Agent Class Model to show events

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to show the identified events for each agent class in the **events** compartment. Figure 6.37 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown.

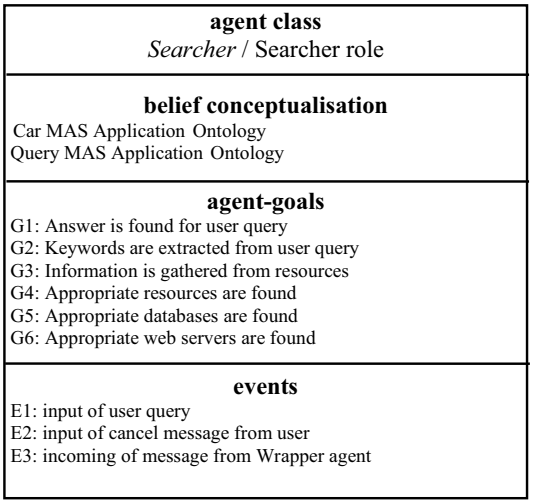


Figure 6.39 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

[59] Only stimuli from the environment are modelled as events because events are meant to reflect agent reactivity, which in turn is defined as the ability to perceive the environment and respond accordingly (Wooldridge 1999). Stimuli that occur from within the agent are not classified as events but as internal processing triggers.

6.4.4. Step 4 – Develop Agent Behaviour Model

Agent behaviour refers to the way an agent behaves in order to achieve/satisfy its agent-goals. Two major styles of behaviour have been commonly implemented for agents: "*planning*" and "*reflexive acting*"[60] (Wooldridge and Jennings 1994; Chelberg et al. 2001; Stone and Veloso 2000; Vidal et al. 2001). Planning requires an agent to carry out logical (or at least pseudo-logical) symbolic reasoning to dynamically choose among potential courses of actions for achieving an agent-goal, taking into account the current state of the environment, events occurring during the process of agent-goal achievement, and the failure/success of past actions (Russell and Norvig 2003; Wooldridge and Jennings 1994). Reflexive acting, on the other hand, frees the agent from complex symbolic reasoning by allowing it to behave in a hard-wired situation-action manner, similar to reflexes (Wooldridge 1999; Nareyek 2001). The agent simply follows pre-defined situation-action rules to determine which actions it should execute for achieving/satisfying an agent-goal. These rules, referred to as "reflexive rules" in MOBMAS, can be represented in if-then logic.

Each style of agent behaviour has its strengths and weaknesses (Nareyek 2001; Chelberg et al. 2001). The strength of planning is that it allows an agent to deal with unforeseen situations via reasoning. The developer does not have to predict at design time all the possible situations that the agent may encounter at run-time, or the actions to be executed in these situations. Nevertheless, the weakness with planning is its lack of speed. Every time the situation at hand is different from that anticipated (e.g. when an event occurs), a new plan must be formed, resulting in delays in the fulfilment of the respective agent-goal. Reflexive acting, on the other hand, allows agents to act fast, as the actions to be executed are already defined via reflexive rules. Nevertheless, the problem of reflexive behaviour is that every possible situation at run-time must be known and considered in advance. If the developer fails to foresee a particular event, the reflexive rules dealing with that event will not be defined, resulting in the agents not knowing how to act, or acting in an undesirable manner.

---

[60] Planning and reflexive acting are often referred to as "deliberative" and "reactive" behaviour respectively in the AOSE literature (e.g. Wooldridge and Jennings 1994; Chelberg et al. 2001; Stone and Veloso 2000; Vidal et al. 2001). However, MOBMAS avoids using these terms because "deliberative" may indicate the need for collaboration (which is not unique to planning agents, since reflexive behaviour may also involve collaboration), and "reactive" may refer to the mode of being triggered (which is not unique to reflexive behaviour, since planning agents may also be triggered).

Thus, for *each* agent-goal of each agent class, the developer should determine the style of agent behaviour to be adopted for the agent-goal. Each agent class may adopt different styles of behaviour for different agent-goals. Considering the strengths and weaknesses of each behavioural style, MOBMAS recommends the developer to consider the following characteristics of an agent-goal before making the decision.

- *Complexity of reasoning required by the agent-goal*: If an agent-goal can be satisfied by executing a simple, straightforward, pre-definable sequence of actions (i.e. no complex reasoning is required), the agent class can adopt reflexive behaviour to achieve the agent-goal. Otherwise, if the achievement of the agent-goal requires a complicated, dynamic set of actions where various alternative courses of actions are available, and logical reasoning is needed to decide which course of actions to follow (or which alternative course of actions to switch to when the chosen course of actions fails or when the situation changes), planning would be necessary.
- *Real-time requirement of the agent-goal*: Since planning agents cannot react well in real time, planning behaviour may not be appropriate to agent-goals which need to be achieved in a timely, immediate manner.
- *Predictability of environment situations*: If the developer can foresee each and every situation that may apply during the agent-goal achievement process, and can pre-define situation-action rules for the agent class to achieve the agent-goal, reflexive behaviour is applicable. Otherwise, planning is required to deal with unforeseen situations via logical reasoning.

In the illustrative Product Search MAS, agent-goal "User query is accepted" of the "User Interface" agent class can be achieved with reflexive behaviour, because it pertains to a simple, pre-definable course of actions. This agent-goal also needs to be fulfilled in a timely manner. On the other hand, agent-goal "Answer is found for user query" of the "Searcher" agent class calls for planning behaviour, because the developer cannot pre-define all of the potential courses of actions for achieving the agent-goal. This agent-goal also does not need to be achieved in an immediate manner, thus allowing for planning to take place.

It should be noted that an agent-goal may call for *both* planning and reflexive behaviour. Planning takes care of high-level, long-term reasoning for the agent-goal, while reflexive rules handle decisions about minor plan steps (Nareyek 2001).

MOBMAS models the behaviour of all agent classes in the **Agent Behaviour Model Kind**. This model kind is represented by three notational components:

- **Agent Goal Diagram**: which has been mentioned in Section 6.4.2.2;
- **Agent Plan Template**: which models the planning behaviour of a particular agent class for a particular agent-goal; and
- **Reflexive Rule Specification**: which models the reflexive behaviour of a particular agent class for a particular agent-goal.

Section 6.4.4.1 discusses the development of Agent Plan Templates, while Section 6.4.4.2 deals with Reflexive Rule Specifications.

6.4.4.1. Develop Agent Plan Templates

Normally, agent architectures and implementation platforms that support planning behaviour will offer a "planner" (or a "reasoner" or a "means-end analyser") which takes care of the formation of plans at run-time for agents, e.g. STRIPS (Fikes and Nilsson 1971), IPEM (Ambros-Ingerson and Steel 1988), AUTODRIVE (Wood 1993) and IRMA (Bratman et al. 1988). MOBMAS therefore does not address the issue of plan formation during run-time. Rather, it supports planning by *specifying the pieces of information that are used by planners to formulate plans* (Figure 6.40). This information is captured in the **Agent Plan Template**.

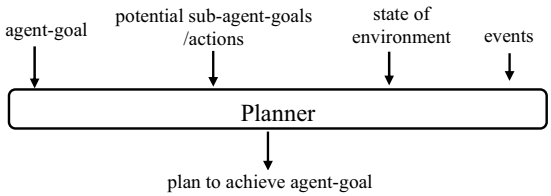


Figure 6.40 – Formation of plans by planner (Wooldridge 2002)

Any agent-goal that requires planning should be associated with an Agent Plan Template. Each Agent Plan Template should specify the following elements.

❖ **Target agent-goal**

This is the agent-goal that a plan derived from the Agent Plan Template aims to achieve. The agent-goal must have been listed in the **agent-goals** compartment of the respective agent class in the Agent Class Diagram of Agent Class Model Kind.

❖ **Triggering event (optional)**

This is the event that activates the target agent-goal, thereby triggering the planning process. This event must have been listed in the **events** compartment of the respective agent class in the Agent Class Diagram of Agent Class Model Kind. Note that an agent-goal may be proactively activated by the agent, in which case no "triggering event" exists.

❖ **A set of sub-agent-goals and/or actions**

To achieve the target agent-goal, the agent may pursue sub-agent-goals and/or actions. Accordingly, the Agent Plan Template should specify a set of sub-agent-goals, or a set of sub-agent-goals and actions, or a set of actions only (Figure 6.41). Each sub-agent-goal (if exist) should be accompanied by its own Agent Plan Template.

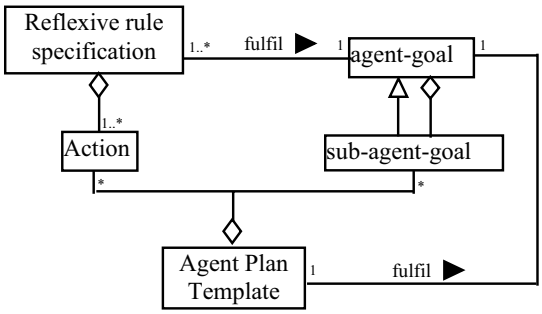


Figure 6.41 – Agent Plan Template and Reflexive Rule Specification (represented in UML)

The need for *sub-agent-goals* in an Agent Plan Template can be identified by investigating the Agent Goal Diagram of the corresponding agent class. This diagram shows the decomposition structure of the agent-goals (cf. Section 6.4.2.2). For example, Agent Plan Template for the agent-goal "Information is gathered from resources" of the "Searcher" agent class (Figure 6.38) should specify a sub-agent-goal "Appropriate resources are found". The Agent Plan Template for this sub-agent-goal should in turn specify two sub-agent-goals "Appropriate databases are found" and "Appropriate web servers are found".

It should be noted that, during the process of Agent Plan Template development, new sub-agent-goals may be discovered which have not been identified in the *Agent Class Diagram* of Agent Class Model Kind and *Agent Goal Diagram* of Agent

Behaviour Model Kind. This indicates the need for iterative development of these three notational components.

An *action* in an Agent Plan Template is an atomic unit of work that an agent class can perform, for example, carrying out some calculation or reasoning, changing the state of an entity in the environment, activating another agent-goal or sending a message to another agent class (Shoham 1993). MOBMAS defines each action in terms of:

- *pre-condition*: which specifies a state that must be true before an action can be executed;
- *post-condition*: which specifies a state resulted from the execution of the action; and
- *action name and parameter list*.

Note that "pre-condition" and "post-condition" constructs are necessary for the selection and sequencing of actions by planners at run-time (Russell and Norvig 2003).

If an action of an agent class involves the sending of an ACL message to another agent class, it is referred to as a "*communicative action*" in MOBMAS. Communicative actions are needed if the achievement of the target agent-goal requires inputs from, and/or provides outputs to, the other agent classes. The developer can refer to the *Agent Relationship Diagram* of Agent Class Model Kind and *Role Diagram* of Role Model Kind to obtain an overview of the agent acquaintances and dependencies[61]. In the case when two or more agent classes aim to achieve an identical agent-goal (cf. Section 6.4.2), they are likely to engage in "distributed planning" to achieve the agent-goal in a joint manner. In such case, the Agent Plan Template of each agent class should contain many communicative actions to allow for distributed planning[62]. Note that the specification of communicative actions may uncover acquaintances that have not been captured in

[61] Inter-agent dependencies are reflected via inter-role authority relationships in Role Diagram of Role Model Kind.

[62] Distributed planning is a complicate research issue by itself. MOBMAS refers the developer to other research work on distributed planning for more techniques, such as desJardins et al. (1999), Conry et al. (1988) and Durfee (1999).

the Agent Relationship Diagram or Role Diagram, thus resulting in a refinement of these two notational components.

The identification of actions for an agent class can be assisted by the investigation of those *MAS Application ontologies* that the agent class commits. These ontologies may define concepts that correspond directly to actions. For example, MAS Application ontology committed by a "Soccer player" agent class defines concepts such as "move", "kick", "turn" and "search-ball", which signify the basic actions of the agent class.

❖ **Events that affect the agent's course of actions**

As mentioned in Section 6.4.3, during the process of agent-goal achievement, certain events may occur that affect an agent's course of actions. For example, a cancel request from a human user will result in the "Searcher" agent class in the Product Search MAS forfeiting its agent-goal "Answer is found for user query".

At design time, it is not always feasible to determine the new course of actions for the agent given the occurrence of these events. This task is delegated to the built-in planners, which use complicated planning algorithms to determine (on the fly) the next best alternative course of actions for the agent (Russell and Norvig 2003). Nevertheless, to facilitate this run-time replanning, MOBMAS recommends the developer to identify the *potential events* that may affect the agent's run-time course of actions.

All events identified here should be listed in the **events** compartment of the corresponding agent class in the Agent Class Diagram of the Agent Class Model Kind. Likewise, all events listed in the **events** compartment of the Agent Class Diagram should be considered in the Agent Plan Template.

❖ **Commitment strategy**

At run-time, the planning process of agents may be largely affected by the way agents are committed to achieving the target agent-goal. For example, agents may persist on pursuing the agent-goal until it is satisfied, or are willing to forfeit the agent-goal after some time. At design time, MOBMAS recommends the developer

to identify the desirable commitment strategy for each agent class with respect to each agent-goal, so as to facilitate the agent class' planning process at run-time. Some example commitment strategies are proposed by Rao and Georgeff (1991).

- "blind or fanatical commitment": the agent will continue pursuing an agent-goal until it believes the agent-goal has actually been achieved.
- "single-minded commitment": the agent will continue pursuing an agent-goal until it believes that either the agent-goal has been achieved, or else that it is no longer possible to achieve the agent-goal.
- "open-minded commitment": the agent will pursue an agent-goal as long as it is still believed possible.

MOBMAS suggests the developer to consider the following factors when selecting the commitment strategy for a particular agent-goal.

- The importance of the agent-goal: For example, agent-goal "Appropriate resources are found" in Figure 6.38 needs to be achieved in order for agent-goal "Information is gathered from resources" to be achieved.
- The existence of events that make the achievement of the agent-goal impossible (e.g. input of user's cancel request): The existence of these events means that the adopted commitment strategy should *not* be blind or fanatical.

❖ **Conflict resolution strategy (optional)**

If certain agent-goals of an agent class are in conflict with each other (i.e. intra-agent conflict), the agent's actions at run-time must be selected in such a way as to minimise or resolve these conflicts. At design time, MOBMAS recommends the developer to identify the desirable *conflict resolution strategy* for the agent class in order to facilitate the agent class' planning process at run-time. There exists a vast amount of work in the area of conflict resolution. Some example strategies for intra-agent conflict resolution[63] are priority conventions (Ioannidis and Sellis 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981) and evidential reasoning (Carver and Lesser 1995).

[63] The issue of inter-agent conflicts (i.e. conflicts between agent-goals of different agent classes) will be discussed in the "*Agent Interaction Design*" activity (Section 6.5).

6.4.4.1.a. Notation of Agent Plan Template

Any representation languages for classical planners can be adopted for Agent Plan Template, e.g. STRIPS (Fikes and Nilsson 1971) and ADL (Pednault 1989). However at design time, it is acceptable to represent Agent Plan Templates in an informal natural language. MOBMAS suggests the following schema for Agent Plan Template.

Initial state: *state definition*
Target agent-goal: *state definition*
Commitment strategy: e.g. *blind, single-minded* or *open-minded*
List of sub-agent-goals (if any): *state definition and name of the Agent Plan Template that achieves the sub-agent-goal*
List of actions (if any): *action name and parameter list*
    Pre-condition: *state definition*
    Post-condition: *state definition*
Events: *list of events*
Conflict resolution strategy (if applicable): *strategy name for each agent-goal*

Figure 6.42 – Agent Plan Template

The definition of *states* in Agent Plan Templates (namely the initial states, agent-goals, sub-agent-goals, pre-conditions and post-conditions of actions) may contain variables. Datatypes of these variables should be defined. For example, the initial state of Agent Plan Template for the agent-goal "Information is gathered from resources" is

"keywords**: User query.Keyword** are known",

with "keywords" being a variable and "User query.Keyword" being the datatype of this variable (interpreted as "Keyword" of "User query"). "User query" and "Keyword" are application-specific concepts that are defined in a MAS Application ontology, namely, the Query MAS Application Ontology (Figure 6.15). Note that a datatype may be a "basic" concept that is known to every agent class without being defined in a MAS Application ontology, e.g. Integer or String datatypes.

Parameters of actions may be constants or variables. Again, datatypes of variable parameters should be defined. For example, an action in the Agent Plan Template for the agent-goal "Information is gathered from resources" is

"recordResultFromResource(carID**:Car.ID**, carModel**: Car.Model**, carStock**: Car.Number-in-stock**)".

"CarID", "carModel" and "carStock" are variables while "Car.ID", "Car.Model" and "Car.Number-in-stock" are datatypes of these variables respectively. Note that "Car", "ID" "Model" and "Number-in-stock" are application-specific concepts defined in the Car MAS Application Ontology (Figure 6.14).

Figure 6.43 shows the Agent Plan Template for achieving agent-goal "Information is gathered from resources" of "Searcher" agent class (cf. Figure 6.38).

Initial state: keywords:User query.Keyword are known
Target agent-goal: Information is gathered from resources
Commitment Strategy: single-minded
Sub-agent-goal: "Appropriate resources are found", cf. sub-plan X
Action 1: sendQueryToWrapper (keywords:User query.Keyword)
Pre-condition: Sub-agent-goal "Appropriate resources are found" is achieved successfully and resourceNo: Integer > 0
Effect: Event 2 incurs
Action 2: recordResultFromResource(carID:Car.ID, carModel: Car.Model, carStock: Car.Number-in-stock)
Pre-condition: message:Result list is received from Wrapper agents and carResultArray: [Car]* is empty
Effect: carResultArray: [Car]* is populated
Action 3: cancelSearch()
Pre-condition: Event 1 incurs
Effect: agent-goal is forfeited
Event 1: input of user's cancel request
Event 2: incoming message from Wrapper agents

Figure 6.43 – Agent Plan Template for agent-goal "Information is gathered from resources" of "Searcher" agent class in Product Search MAS

Even though the selection and sequencing of sub-agent-goals and actions for agents at run-time is delegated to built-in planners, if there exists a *tentative* course of sub-agent-goals/actions for achieving a particular agent-goal, this sequence can be captured in an **Agent Plan Diagram**. The notation of Agent Plan Diagram is borrowed from Kinny et al. (1996). It is an extended UML Statechart diagram where each state represents a sub-agent-goal or an action (Figure 6.44). Transition from one state to another occurs when an event happens and/or a condition applies.

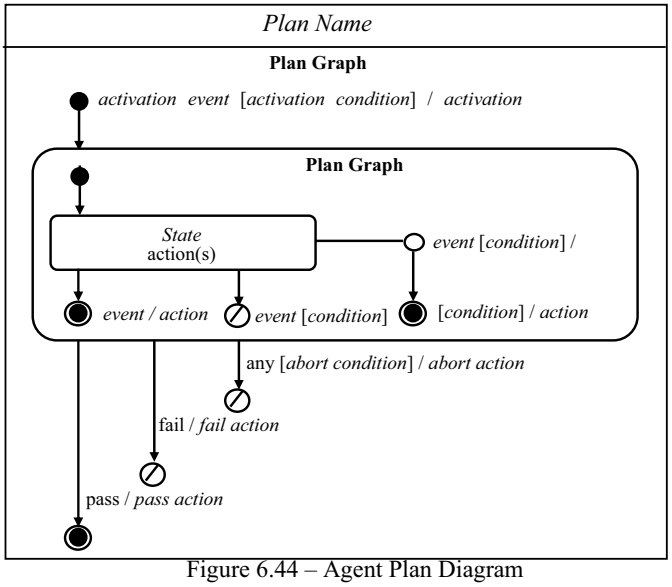


Figure 6.44 – Agent Plan Diagram

Figure 6.45 illustrates the Agent Plan Diagram for the plan that achieves agent-goal "Information is obtained from resources" (cf. Figure 6.43).

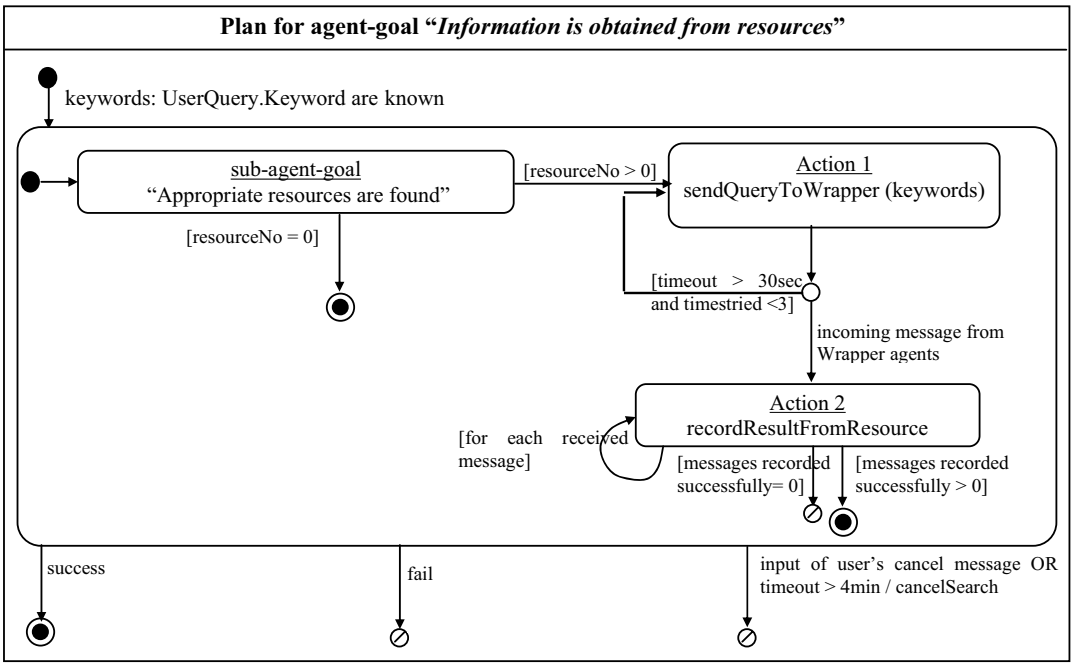


Figure 6.45 – Agent Plan Diagram for agent-goal "Information is gathered from resources" of "Searcher" agent class in Product Search MAS

6.4.4.2. Develop Reflexive Rule Specifications

Reflexive rules are basically (sequences of) "if-then" rules that couple stimuli and/or states of the environment with actions to be executed by an agent class. Each reflexive rule specifies either a complete course of actions to achieve an agent-goal, or a set of actions that works towards the achievement of the agent-goal. Each agent-goal requires one or more Reflexive Rule Specifications, each of which documents the following information.

❖ **Target agent-goal**

This is the agent-goal that the reflexive rule aims to achieve or satisfy. The goal must have been listed in the **agent goals** compartment of the corresponding agent class in the Agent Class Diagram of Agent Class Model Kind.

❖ **Course of actions**

Actions in a reflexive rule are analogous to actions in an Agent Plan Template (cf. Section 6.4.4.1). The only difference is that the sequence of actions is known at design time. The developer is referred to Section 6.4.4.1 for more discussion on actions.

❖ **Events and/or internal processing triggers**[64]

These are the events and/or internal processing triggers that initiate an action in the reflexive rule. The events must have been listed in the **events** compartment of the corresponding agent class in the Agent Class Diagram of the Agent Class Model Kind.

❖ **Guard conditions**

These are the states that make certain actions applicable for execution.

6.4.4.2.a. Notation of Reflexive Rule Specification

MOBMAS borrows the notation from *UML Activity diagrams* for Reflexive Rules Specification. Each action is depicted as an UML activity, while events, internal processing triggers and guard conditions are specified alongside the transition flows between activities just as events[65] and guard-conditions in UML Activity diagrams (Object Management Group 2003). However, each Reflexive Rule Specification should the Reflexive Rule Specification of "User Interface" agent class to satisfy its agent-goal "User query is accepted".

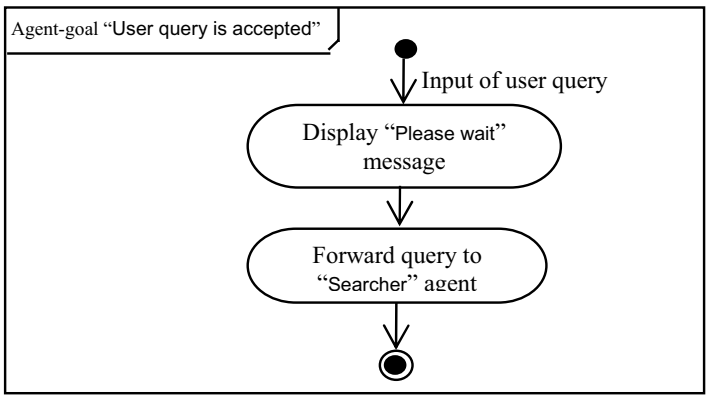


Figure 6.46 – Reactive Rule Specification

[64] Internal processing triggers are stimuli generated from within the agent itself.
[65] For representation simplicity, internal processing triggers are represented in the same way as events.

6.4.4.3. Verify Agent Behaviour Model against Ontology Model

Both Agent Plan Templates and Reflexive Rule Specifications of Agent Behaviour Model Kind contain the definition of *states* and *actions*. The *states* (namely, initial states of Agent Plan Templates, agent-goals, sub-agent-goals, pre-conditions, post-conditions and guard conditions of actions) normally refer to the states of entities/concepts that exist in the agent class' application and wrapped resources' applications. Likewise, parameters of actions in an Agent Plan Template or Reflexive Rule Specification often involve entities/concepts that exist in the agent class' application and wrapped resources' applications. Thus, Agent Plan Templates and Reflexive Rule Specifications can be used to verify the completeness of the content of MAS Application ontologies and Resource Application ontologies. In the other way around, MAS Application ontologies and Resource Application ontologies can be used to verify Agent Plan Templates and Reflexive Rule Specifications.

Specifically, the *datatypes* of all variables specified in Agent Plan Templates and Reflexive Rule Specifications (particularly, in the *states* and *actions' parameters*) should have been defined in a particular MAS Application ontology or Resource Application ontology, with the exception of basic datatypes such as Integer or String. Meanwhile, only concepts defined in the MAS Application ontologies and Resource Application ontologies should be used to define the datatypes of variables in Agent Plan Templates and Reflexive Rule Specifications.

For example, in the following action of the "Searcher" agent class in the Product Search MAS:

"recordResultFromResource(carID**:Car.ID**, carModel**: Car.Model**,
carStock**: Car.Number-in-stock**)",

concepts "Car", "ID", "Model" and "Number-in-stock" should have been defined in the Car MAS Application Ontology (Figure 6.14).

In another example, consider the following action of the "Wrapper" agent class:

"getPrice(carID**: CarProduct.SerialNo**)
Pre-condition: productPrice: **CarProduct.Price** = unknown
Post-condition: productPrice: **CarProduct.Price** = known".

The action aims to retrieve the price of a particular car product from the Car Database resource of the Product Search MAS (Figure 6.29). Concepts "CarProduct", "SerialNo" and "Price" should have been defined in the CarInfo Resource Ontology (Figure 6.15).

The above guidelines imply the need to *reciprocally* and *iteratively* develop the Agent Behaviour Model and Ontology Model. More specifically, the developer should:

- use the developed ontologies as inputs to define states and actions' parameters for the Agent Plan Templates and Reflexive Rule Specifications; and
- examine the states and actions' parameters of Agent Plan Templates and Reflexive Rule Specifications to determine if any concepts have *not* been defined in the developed ontologies, thereby verifying the content of these ontologies.

6.4.4.4. Verify Agent Behaviour Model against Agent Class Model

Since an agent needs to know about the entities/concepts that are mentioned in its Agent Plan Templates and Reflexive Rule Specifications, the conceptualisation of these entities/concepts should have been defined in its Belief Conceptualisation. Accordingly, the developer should check each agent class' Belief Conceptualisation to confirm that it contains all those *ontologies* which conceptualise the entities/concepts in the Agent Plan Templates and Reflexive Rule Specifications (namely those specified in the *states* and *actions' parameters*).

6.5. AGENT INTERACTION DESIGN ACTIVITY

This activity of MOBMAS models the interactions between agent instances by selecting a suitable interaction mechanism for the target MAS, thereafter specifying the patterns of data exchanges between agents given the chosen interaction mechanism.

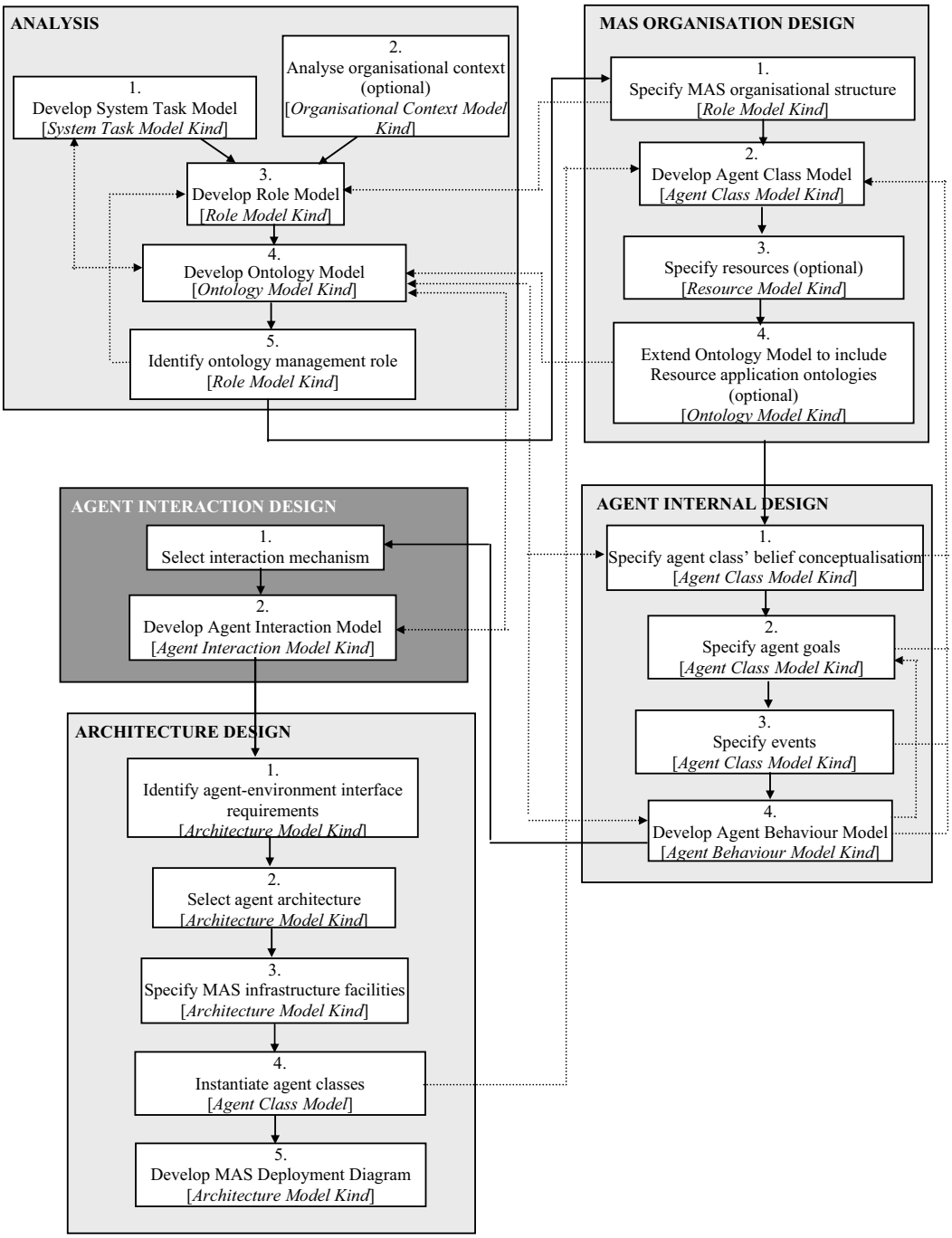


Figure 6.47 – MOBMAS development process

6.5.1. Step 1 – Select Interaction Mechanism

6.5.1.1. Overview of interaction mechanisms

"Interaction" refers to the exchange of data amongst agents, either two-way or multi-way (Goldin and Keil 2004). This exchange can be conducted using either of the two interaction mechanisms: direct interaction and indirect interaction (Weyns et al. 2004; Bandini et al. 2004; Goldin and Keil 2004).

- In **direct interaction**, agents exchange data by sending communication messages directly to each other. These messages are typically expressed in *ACL*, such as KQML or FIPA-ACL. The specification of message contents can be made using content languages such as KIF, FIPA-SL, LOOM and Prolog. Generally, the exchanges of ACL messages between agents need to conform to "*interaction protocols*", which are *allowed* communication patterns between the interacting agents. They specify the possible sequences of exchanged messages and the constraints on the content of these messages (Odell et al. 2000b). Some examples of interaction protocols are FIPA Contract Net, Simulated Trading, Request, Query and Subscribe Protocols (FIPA 2002).

- In **indirect interaction**, agents exchange data indirectly through some kind of communication abstraction. A well-known indirect interaction mechanism is *tuplespace interaction*, where agents interact by inserting "*tuples*"[66] into, and removing them from, a shared tuplespace in an associative way. Recently, this mechanism has progressed into a more advanced form, *tuple-centre interaction*. A tuple-centre is no longer a mere communication channel like a tuplespace, but a *programmable reactive interaction medium*, which is equipped with computational capacity to react to events (Omicini and Denti 2001; Ciancarini et al. 1999). Example middleware systems or models built upon the tuple-centre interaction mechanism are TuCSoN (Cremonini et al. 1999), LuCe (Denti and Omicini. 2001), $\mathcal{ACLT}$ (Omicini et al. 1995), LIME (Picco et al. 1999), Berlinda (Tolkdorf 1997) and MARS-X (Cabri et al. 2000). Other indirect interaction mechanisms for agents are

[66] Each tuple is an ordered collection of heterogeneous information chunks.

stigmergy and *spatially founded* interaction. The term "stigmergy" is used by biologists to refer to the coordination of insects through "pheromone" – a chemical substance deposited into the environment and sensed by the individual insects. Agents adopt this indirect interaction mechanism by generating and detecting "artificial pheromone objects" in an artificial dissipation environment (Valckenaers et al. 2002; Klugl 2001). Spatially founded interaction mechanism, meanwhile, is related strongly to the spatial structure of the environment. Co-Fields (Mamei and Zambonelli 2004) is an example interaction model that employs this mechanism. It aims to support agents' "motion" coordination by representing the agents' operational environment into "computational fields". These fields are a sort of spatial data structures that can be propagated across the environment by some network infrastructure. An agent can make its movement decisions by examining the shape of the computational fields, just as a physical mass moves in accord to the gravitational field. The movement of this agent may induce changes to the shape of some specific fields, which in turn affect the movement of other agents.

6.5.1.2. Select interaction mechanism

The developer should decide which interaction mechanism is best suited to the target MAS. MOBMAS supports this decision by presenting a comparison of the different mechanisms, thereafter providing recommendations on when to use which mechanism.

The direct interaction mechanism is different from the indirect mechanism in terms of the following key aspects (Goldin and Keil 2004; Weyns et al. 2004).

- *Early binding of recipient*: The direct interaction mechanism requires an agent to know its target interaction partner before the interaction can take place. Meanwhile, the indirect mechanism allows the identity of the target partner to be determined after the sending of tuples/pheromones/fields.
- *Name and location coupling*: With the direct interaction mechanism, the interacting agents have to know about each other's configuration and location before the interaction can take place, while with the indirect mechanism, the interacting agents do not have to hold this knowledge.
- *Time coupling*: The direct exchange of messages between agents in the direct interaction mechanism requires the interacting agents to exist and be available to

communicate at the same time. On the other hand, in the indirect mechanism, there can be a delay between the sending of a tuple/pheromone/field and its observation.

Among the various mechanisms of indirect interaction listed in Section 6.5.1.1, stigmergy and spatially-founded mechanisms are very limited in their applicability. Stigmergy coordination is mainly suited to those domains that involve some kind of attraction to specific locations, or attraction to move in a specific direction (e.g. synthetic ecosystem, network routing) (Biegel 2002; Bonabeau et al. 1998; Brueckner 2000). Likewise, spatially-founded mechanism should only be considered when space is an essential factor in agent interactions (e.g. motion coordination). Thus, MOBMAS focuses only on the *tuplespace*/*tuple-centre* indirect interaction mechanism. In the following section, a comparison between the direct interaction mechanism via ACL and the indirect mechanism via tuplespace/tuple-centre is presented.

6.5.1.2.a. Comparison between direct interaction mechanism and tuplespace/tuple-centre indirect interaction mechanism

Most of the existing tuplespace and tuple-centre frameworks are based upon the LINDA model (Papadopoulos 2001). They therefore use Linda-like communication primitives such as out, in, rd, inp and rdp to specify communication messages (Omicini and Zambonelli 1999). These primitives offer a very low level of semantics for the communication language. Meanwhile, the direct interaction mechanism employs complicated ACLs that offer a large range of expressive *speech-act performatives*, e.g. inform, tell, query-if, ask-if, ask-all, advertise, achieve, refuse and failure (FIPA. n.d.a).

With regard to popularity, the direct interaction mechanism is far more commonly used by the existing MAS development projects than the tuplespace/tuple-centre mechanism (Bergenti and Ricci 2002). The main reasons for its popularity are that:

- interaction protocols have been widely adopted in the OO paradigm. The developer can therefore borrow techniques from OO modelling to specify agent interaction protocols; and
- numerous interaction protocol patterns have been catalogued for reuse, e.g. those developed by FIPA (FIPA 2002).

Nevertheless, with its programmable behaviour, the tuple-centre mechanism offers various advantages over the direct interaction mechanism.

- *Decoupling of computation and coordination concerns*: The behaviour of the tuple-centre can be programmed in such a way as to embody any rules that govern the agent coordination[67] (referred to as "coordination rules"). Agents can thus be freed of the load of coordination and focus on their individual computation during the interaction process (Cremonini et al. 1999; Omicini and Denti 2001; Bergenti and Ricci 2002; Ciancarini et al. 2000). In particular, the interacting agents can simply be concerned with providing inputs to, and obtaining outputs from, the interaction process. The tuple-centre can be programmed to:
  - ensure that all the coordination rules governing the interaction process are satisfied; and
  - carry out some (low level) processing to assist in the fulfilment of the target coordinating task, thereby taking some processing load off the interacting agents.

  On the other hand, in the direct interaction mechanism, computation and coordination concerns are merged into the design of agents (Bergenti and Ricci 2002; Ciancarini et al. 2000). The interacting agents cannot abstract away from the coordination concerns, but have to embed the coordination rules into their interaction protocols, which are in turn embedded in their codes (Omicini and Denti 2001). This may be very difficult to implement if the number of interacting agent classes is large or the coordination rules are complex.

- *Support for modification*: With the tuple-centre interaction mechanism, any changes to the coordination rules may only lead to changes in the behaviour of the tuple-centre. The direct interaction mechanism, on the other hand, may require an update of the design of all interacting agents (Ciancarini et al. 2000; Omicini and Denti 2001).

- *Security control*: In MAS, security-related concerns include authentication (i.e. how agents are identified) and authorisation (i.e. what are agents allowed to do) (Cremonini et al. 1999). With the tuple-centre interaction mechanism, authentication

[67] "Coordination" refers to the management of interactions (Nwana et al. 1996; Wegner 1996)

and authorisation activities can be delegated to the tuple-centre. Meanwhile, with the direct interaction mechanism, the direct exchange of messages between agents will mean that the individual agents need to implement their own authentication and authorisation activities.

Nevertheless, the tuple-centre interaction mechanism exhibits a strong centralised design due to the tuple-centre's essential role in the interaction process (Bergenti and Ricci 2002). This centralisation may seriously compromise the robustness of the system, e.g. when the tuple-centre experiences downtime. The direct interaction mechanism, in contrast, spreads the locus of control over the interacting agents, hence avoiding the robustness problem if a particular agent goes down[68].

In summary, considering its popularity and reusability, MOBMAS recommends the direct interaction mechanism to most MASs. However, in various situations, the tuplespace or tuple-centre interaction mechanism is perceived more appropriate than the direct interaction mechanism, namely,

- *when the MAS environment is open and dynamic*: With its support for late binding of recipient, name decoupling, location decoupling and time decoupling, the tuplespace/tuple-centre interaction mechanism is able to facilitate flexible and robust interaction in open and dynamic systems (Zambonelli et al. 2001b; Bergenti and Ricci 2002). In addition, by embedding the coordination rules into the tuple-centre, the tuple-centre interaction mechanism can help preventing illegitimate or self-interested behaviour in newly added agents; or

- *when many agent classes aim to achieve identical agent-goals* (cf. Section 6.4.2): The tuple-centre interaction mechanism is particularly suited to the interactions amongst agents of these classes because:
  - the joint achievement of the identical agent-goals often requires many coordination rules to be enforced on the interacting agents. The tuple-centre can take charge of enforcing these rules; and

[68] In this case, other instances of the same agent class may serve as an substitute for the problematic agent.

- the tuple-centre can carry out some processing to assist in the achievement of the joint agent-goals.

For example, consider the agent-goal "Papers are distributed among members" jointly achieved by "PC Chair" and "PC Member" agent classes (cf. Figure 6.7). Some example coordination rules governing the interactions between agents of these classes are:

- "Each paper must be distributed to a required number of members";
- "Each member must collect a required number of papers";
- "Each member must not collect the same paper twice"; and
- "Members can only start collecting papers after all other members have viewed the Title List of all papers"[69].

All of these coordination rules can be enforced by the tuple-centre, which is programmed in such a way as to check and control the tuples sent from the "PC Chair" and "PC Member" agents (cf. Figure 6.57). The tuple-centre can also carry out some processing to identify which "PC Member" agent has not collected the required number of papers, thereby posting reminder tuples to these agents. This processing helps to enforce the rule "Each member must collect a required number of papers".

6.5.2. Step 2 – Develop Agent Interaction Model

The Agent Interaction Model Kind of MOBMAS captures the patterns of data exchanges between agent instances when they interact using the chosen interaction mechanism. Section 6.5.2.1 discusses the specification of Agent Interaction Model Kind for the direct interaction mechanism, while Section 6.5.3.2 examines the Agent Interaction Model Kind for the tuplespace/tuple-centre mechanism.

6.5.2.1. Develop Agent Interaction Model for Direct Interaction Mechanism

With the direct interaction mechanism, the exchanges of ACL messages between agents need to be governed by *interaction protocols*, each of which defines an allowed communication pattern between the interacting agents during a particular conversation

[69] This rule is needed to ensure fairness in paper selection by members.

(Odell et al. 2000b). Accordingly, the task of developing Agent Interaction Model for the direct interaction mechanism includes the task of defining interaction protocols for the agent conversations. Agent Interaction Model Kind of this mechanism is represented by a set of ***Interaction Protocol Diagrams***, each graphically describing an interaction protocol for an inter-agent conversation.

6.5.2.1.a. Define interaction protocols

To define interaction protocols for the target MAS, the developer should examine the *Agent Behaviour Model*, namely the Agent Plan Templates and Reflexive Rule Specifications of each agent class in the system.

- Each *communicative action* in the Agent Plan Templates and Reflexive Rule Specifications indicates a message being sent from one agent to another; and
- Any *event* that is specified in an Agent Plan Template or that triggers a Reflexive Rule Specification may itself be a message sent from one agent to another.

As recommended by FIPA (2001c), each ACL message should be defined in terms of:

- **Predecessor**: which defines the order by which a message is sent in relation to its concurrent messages (if ordering is important). For example, "1/inform(…)" denotes an *inform* message that has to be sent first, before any other concurrent messages are sent;
- **Guard-condition**: which defines the condition in which a message is applicable to be sent, e.g. [CarMake = unknown];
- **Sequence-expression**: which specifies the constraint of message sending. For example, "n..m" denotes that a message is sent *n* up to *m* times, while "broadcast" denotes a broadcast sending of a message.
- **Performative**: which defines the type of *speech-act* that the sender wishes to perform, e.g. query-if, inform, refuse, failure; and
- **Arguments**: which are pieces of information that a message conveys. Each argument can be a constant or a variable. For variable arguments, their datatypes must be defined. For example, in the following ACL messages

  "query-if (carCost:**Car.Cost** < custPrice: **UserQuery.Price**)" and

  "inform (carStockNo: **Car.Number-in-stock**)",

variables "carCost", "custPrice" and "carStockNo" are of types "Car.Cost", "UserQuery.Price" and "Car.Number-in-stock" respectively. Note that concepts "Car", "Cost", "Number-in-stock", "User query" and "Price" are application-specific concepts that are defined in the MAS Application ontologies which are shared between the communicating agents (in this case, Car MAS Application Ontology and Query MAS Application Ontology).

MOBMAS recommends the developer to *reuse* and *customize* the patterns of interaction protocols provided by the various libraries and catalogues. FIPA (2002), for example, offers a large range of interaction protocol patterns, supporting both cooperative-style interaction (e.g. Contract Net, Query, Request and Brokering) and negotiation-style interaction (e.g. English/Dutch Auction). If necessary, a complex interaction protocol can be built from a few basic pre-defined protocol definitions.

It should be noted that when developing interaction protocols for agents that aim to achieve an identical agent-goal (cf. Section 6.4.2), the developer should ensure that all the *coordination rules* governing the successful achievement of the agent-goal are embedded in either the *interaction protocols*, or in the *agent class' individual behaviour*, which is modelled in the Agent Behaviour Model Kind.

For example, consider the agent-goal "Papers are distributed among members" jointly achieved by the "PC Chair" and "PC Member" agent classes (cf. Figure 6.7). The rule "Members can only start collecting papers after all other members have viewed the Title List of all papers" (cf. Section 6.5.1.2) can be embedded in the definition of the interaction protocol between the "PC Chair" and "PC Member" agents. Specifically, the protocol can specify the pattern of exchanged messages in such a way that the "PC Member*"* agent is not allowed to send a "Paper-Request" message to a "PC Chair" agent until the "PC Chair" agent has sent a "Title-List" message to all "PC Member*"* agents. In other words, the sending of "Paper-Request" messages is sequenced after the sending of "Title-List" messages (Figure 6.56).

On the other hand, the rule "Each member must not collect the same paper twice" can be enforced by defining the behaviour of the "PC Chair" agent. Specifically, a "PC Chair" agent should not approve the paper request from a "PC Member" agent if that agent has

previously requested the same paper. This behaviour can be defined in the "PC Chair" agent's Agent Plan Template or Reflexive Rule Specification. As a result, when defining interaction protocols for agent classes that achieve identical agent-goals, the developer may have to revise the Agent Behaviour Model.

In addition, the developer should also identify the potential conflicts between the interacting agent classes and define the interaction protocols in such a way as to deal with these conflicts (note that the issue of intra-agent conflicts has been discussed in the "*Agent Internal Design*" activity; Section 6.4). In the context of MOBMAS, two agent classes may be in conflict if their agent-goals have been derived from conflicting system-tasks[70]. The developer should therefore trace through the *Agent Class Model*, *Role Model* and *System Task Model* to identify any potential conflicts between agent classes.

To date, there is a vast amount of research work in the area of conflict resolution. Some example conflict resolution strategies are negotiation (Sycara 1988), voting (Ephrati and Rosenschein 1991), priority conventions (Ioannidis and Sellis 1989), assumption surfacing (Mason and Johnson 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981), evidential reasoning (Carver and Lesser 1995), and standardization and social rules (Shoham and Tennenholtax 1992).

Moreover, the developer should specify the synchronisation mode engaged by each agent in each interaction protocol. Two common modes of agent synchronisation are (Mishra and Xie 2003; Weyns and Holvoet, 2003):

- *synchronous interaction*: i.e. when an agent yields a thread of control after sending a message (i.e. wait semantics). In other words, the agent will wait for a reply ACL message from its interaction partner after it sends a message to that agent; or
- *asynchronous interaction*: i.e. when an agent sends messages without yielding any control. The agent basically continues with its processing right after the sending of messages.

Generally, asynchronous interaction is the most common mode of synchronisation used in agent interaction (Odell et al. 2000a).

---

[70] Recall that agent-goals are derived from role-tasks, which are in turn derived from system-tasks (cf. Section 6.2.3.2).

6.5.2.1.b. Notation of Interaction Protocol Diagrams

The Agent Interaction Model Kind for the direct interaction mechanism is represented by a set of Each ***Interaction Protocol Diagrams*****,** of the Agent Interaction Model Kind each graphically describesing an interaction protocol for an inter-agent conversation. MOBMAS reuses the notation of AUML Sequence Diagram for the Interaction Protocol Diagram (Odell and Huget 2003; Odell et al. 2000a). Major notational rules of the AUML Interaction Protocol Diagram are presented below. The developer can refer to Odell and Huget (2003) and Odell et al. (2000) for a more extensive documentation[71].

- Each lifeline represents an agent and the role(s) that the agent plays during the conversation. The rectangle at the top of each lifeline should be specified in the format

:agent-class-name / role-name1, role-name2…

  The colon in front of the agent-class-name signifies an instance of the agent class.

- Arrows represent the sending of messages between agents. An "asynchronous" message is drawn as ⟶, while a "synchronous" message is shown as ⟶. If there is a delay between the time a message is sent and the time it is received (e.g. "mobile communication"), the message arrow is drawn as ↘.
- If an agent belongs to a class that plays multiple roles (cf. Section 6.3.2.1.a), the dynamics of the agent's role-playing behaviour during the conversation should be modelled.
  - If the agent's role-playing behaviour is *static* (i.e. if the agent plays some particular roles statically throughout its lifetime), the rectangular box at the top of the lifeline should specify *all* of the agent's roles (Bauer 2001b) (Figure 6.48a).
  - If the agent's role-playing behaviour is *dynamic* (i.e. if the agent dynamically changes its active role(s) from one time to another), the rectangular box should *only* specify the name of the active role. If a change of role occurs during the

[71] At the time of this research, the notation of AUML Sequence Diagram is in its preliminary version. The developer should check for updated versions (if exist) at http://www. auml.org.

conversation, this change can be represented as a **«role change»** stereotyped arrow (Bauer, B. 2001b) (Figure 6.48b).

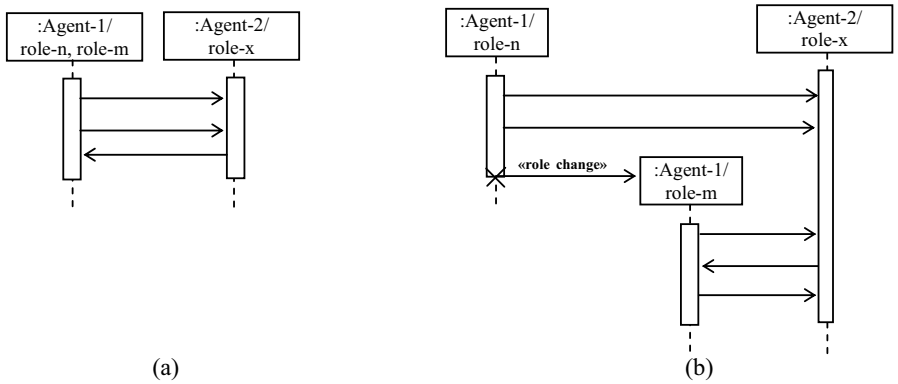


Figure 6.48 – AUML notation for the dynamics of agents' role-playing behaviour (Bauer 2001b)

- Concurrent threads of communication are modelled as shown in Figure 6.49. Figure 6.49a indicates that all messages are sent concurrently. Figure 6.49b includes a decision box to indicate that a decision will be made regarding which messages (zero or more) will be sent (i.e. *inclusive OR*). Figure 6.49c describes an *exclusive OR* (i.e. exactly one message will be sent).

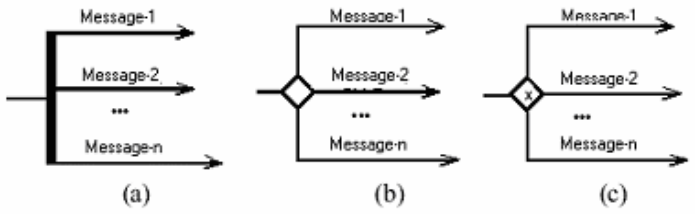


Figure 6.49 – AUML notation for concurrent threads of interaction

- Concurrent threads of processing in the recipient agent are modelled as either a *split of lifeline* (Figure 6.50a) or *activation bars* appearing on top of each other (Figure 6.50b).

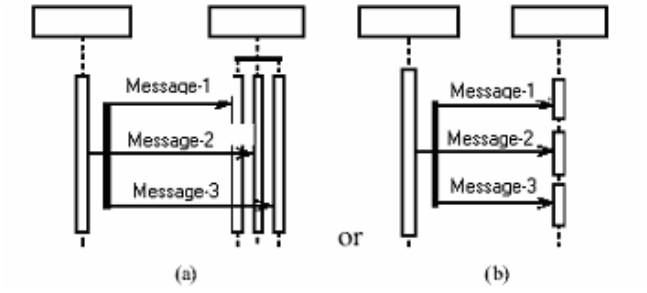


Figure 6.50 – AUML notation for concurrent threads of processing

Figure 6.51 presents an example Interaction Protocol Diagram for the illustrative Product Search MAS. The diagram describes a conversation between a "Searcher" agent and a "Wrapper" agent (cf. Figure 6.33 and Figure 6.41).

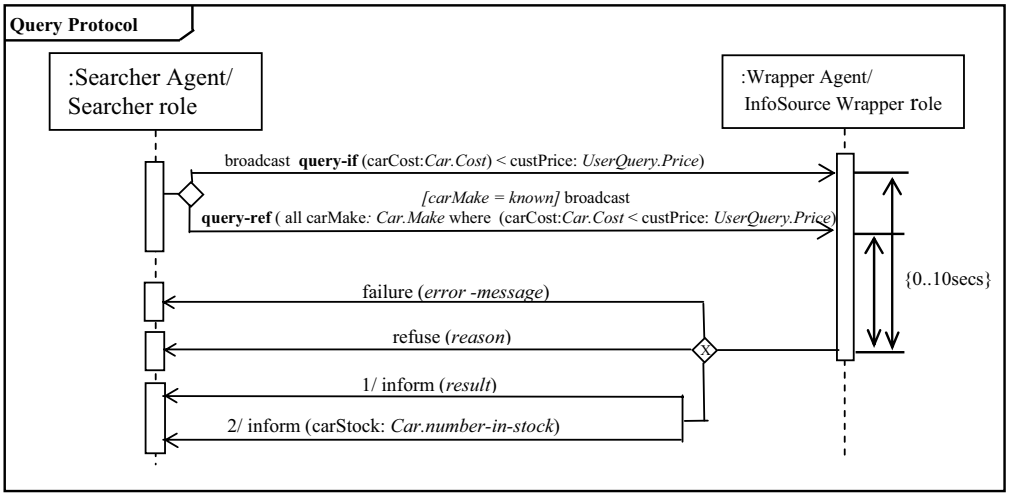


Figure 6.51 – Interaction Protocol Diagram for Product Search MAS

6.5.2.1.c. Update Agent Class Model and Role Model

The *Agent Relationship Diagram* of the *Agent Class Model Kind* should be updated to show:

- any new acquaintances between agent classes that have not been identified; and
- various descriptive information about *each* acquaintance, namely:
  - the *identity of the Interaction Protocol Diagrams* that govern the conversations between the acquainted agents; and
  - the *MAS Application ontology(ies)* which governs the semantics of the messages exchanged during the conversation and which must be shared between the communicating agents[72].

  This descriptive information is modelled as *UML notes* attached to each acquaintance (cf. Figure 6.22).

Figure 6.52 presents the updated Agent Relationship Diagram for the Product Search MAS (cf. Figure 6.28).

[72] That is, this MAS Application ontology must exist in both agents' Belief Conceptualisation.

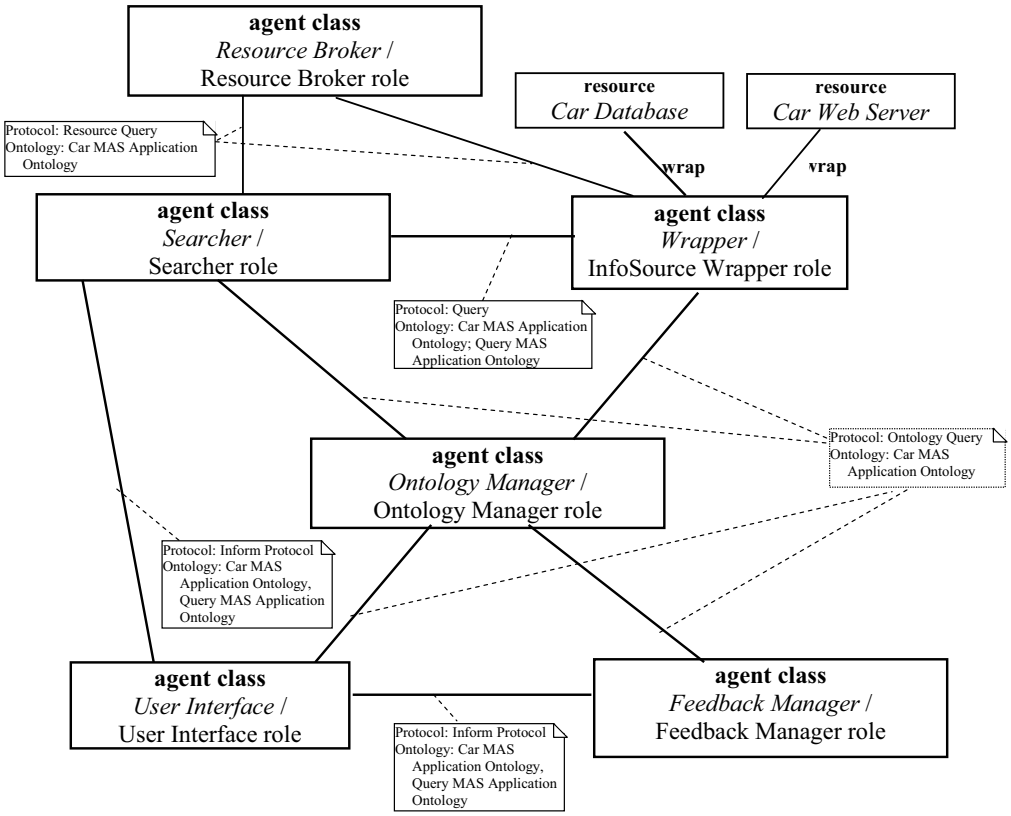


Figure 6.52 – Updated Agent Relationship Diagram for Product Search MAS

The *Role Diagram* of the *Role Model Kind* should also be checked to ensure that all the acquaintances between roles have been captured, and all the authority relationships are still valid. For example, two roles that are initially thought to be in a peer-to-peer relationship may turn out be in a superior-subordinate relationship after an in-depth investigation of the agents' interactions.

6.5.2.1.d. Conceptualise interation protocols with ontology (Optional)

So far, the interaction protocols governing the potential agent conversations in the target MAS have been identified and defined, particularly by step "*6.5.2.1.a. Define interaction protocols*". Traditionally, these protocols will be directly hard-coded into the agents at the implementation time, allowing any agents embedding the appropriate protocols to be able to participate in the respective conversations. Such an implementation mechanism, however, is only suitable to a closed or semi-open MAS environment, where the agents taking part in the interactions are known in advance and can be controlled. In an open environment, on the other hand, this mechanism is not sufficient. An open system would allow new agents to frequently enter the system and

join any existing conversations (such as an auction or an open marketplace). If the interaction protocols are hard-coded into the agents, joining a conversation whose protocol an agent does not know would mean that the agent has to either use one of the protocols it already knows, or else go off-line to be re-programmed. Similarly, in a dynamic environment where the interaction protocols can change over time, the hard-coding of protocols into agents will imply the need for re-coding of all affected agents when the protocols change.

Thus, MOBMAS proposes this optional design step, "*Conceptualise interaction protocols with ontology*", to avoid the above issues at run-time. This step is recommended to be performed if:

- the conversations in the future MAS are expected to be open; that is, any agents are allowed to join thethe pre-existing conversations; and/or
- the interactions in the future MAS are expected to be dynamic; that is, the interaction protocols governing the conversations can change overtime at run-time.

The techniques for this step are based upon Tamma et al.'s approach to ontology-based agent negotiation (Tamma et al. 2005; Tamma et al. 2002a; Tamma et al. 2002b). This approach suggests that agents do not have to hold priori knowledge about the interaction protocols. Instead, when an agent joins a conversation, it will then be provided with the protocol's definition, which is expressed in terms of an *ontology* shared between the agents participating in the conversation. Accordingly, the only priori knowledge that an agent needs to hold is this shared ontology. The definition of the protocol itself would be owned by some agents in-charge (such as auctioneer agents in an auction MAS) and distributed to the requesting agents at run-time. By committing to Ththe protocol'se ontology, the requesting agents would be able will provide the basic vocabulary for the agent to acquireunderstand the acquired and understand the definition of the protocols at run-time. . If the protocol changes, a new protocol definition could be sent to all participating agents.

For example, considering the "Query Protocol" governing the conversation between "Searcher" agents and "Wrapper" agents in the illustrative Product Search MAS (Figure 6.51). The traditional approach would have the protocol's definition directly hard-coded into the "Searcher" agents. Meanwhile, the ontology-based approach by Tamma et al.

(2005) suggests that only the *ontology* conceptualising the protocol should be hard-coded. The protocol's definition itself, which is built upon the ontology, would be given to the "Searcher" agents at run-time (for example, by the "Wrapper" agents).

This approach implies the need for the following design tasks:

- firstly, to define the *ontology* that conceptualises the protocols. MOBMAS refers to this ontology as the "***Protocol Ontology***";
- secondly, to develop *ontology-based definitions of the interaction protocols*; that is, to describe the protocols in terms of the Protocol Ontology.

These design tasks are sub-steps of the step "*Conceptualise interaction protocols with ontology*". The following sections describe these sub-steps.

It should be noted that not all interaction protocols in the MAS need to be conceptualized. Some conversations may be open and dynamic, but some others are closed, static and fixed. Protocols of the latter can be directly hard-coded into the agents without being conceptualised via any ontology.

Define Protocol Ontology

The Protocol Ontology should provide a set of generic concepts and relationships that can be used as the vocabulary for describing interaction protocols, for example, concepts "Protocol", "Participating party", "Message" and "Rule". Even though the different protocols can be largely different in term of their specifications, the vocabulary underlying these specifications should often be the same. Accordingly, it is desirable to have only one Protocol Ontology for the whole MAS. In that way, all agents in the MAS can commit to this single Protocol Ontology and can still obtain definitions of many different protocols at run-time.

The designer can either define the Protocol Ontology from scratch, or adopt/adapt an existing ontological work. In either case, the Protocol Ontology should contain all (and only) the concepts needed to define the relevant interaction protocols. Tamma et al. (2005; 2002a,b) have made a pioneering effort in developing an ontology for negotiation protocols. The ontology, called "Negotiation Ontology", provides the basic vocabulary for describing any negotiation protocols (as claimed by the authors).

The Protocol Ontology can be graphically represented by an Ontology Diagram like other types of ontologies. The Protocol Ontology Diagram should be added to the Ontology Model. As a result, the Ontology Model for a particularly MAS can embrace 3 types of ontologies in total: MAS Application ontologies, Resource Application ontologies and Protocol ontology.

Figure 6.53 presents a simple Protocol Ontology for the illustrative Product Search MAS. The ontology is based upon Tamma et al.'s (2005) Negotiation Ontology.

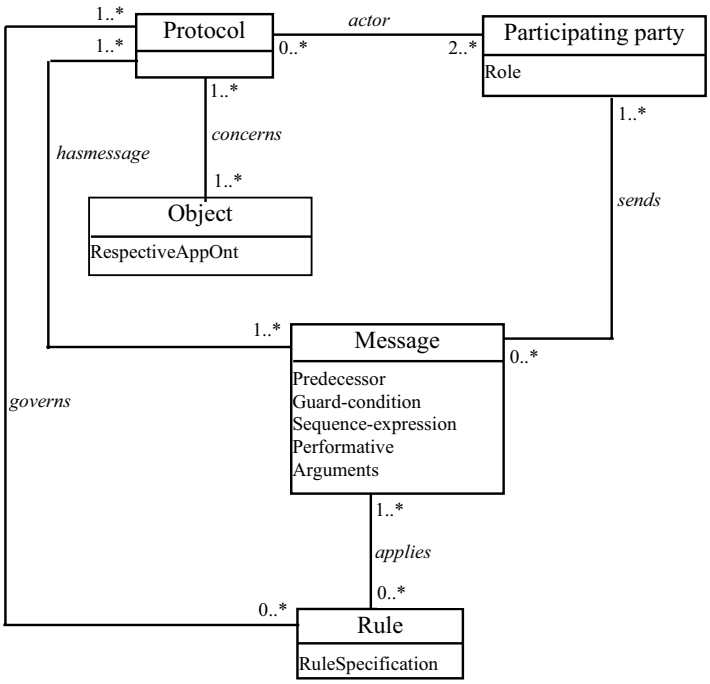


Figure 6.53 – Protocol Ontology

Specify ontology-based definitions of interaction protocols

Previously, step "*6.5.2.1.a. Define interaction protocols*" has produced the definitions of potential interaction protocols for the target MAS. In this sub-step, these protocol definitions are re-expressed in terms of the Protocol Ontology, thereby generating "ontology-based definitions" for the protocols. In order to do this, the designer should *instantiate* the concepts of the Protocol Ontology with specific values, so as to describe a particular protocol. For example, concept "Protocol" in the Protocol Ontology can be instantiated with "Query Protocol", and concept "Participating party" with "Requester" and "Informant", so as to define a query protocol. The underlying aim is to reproduce the protocol definition specified by step "*6.5.2.1.a. Define interaction protocols*" by using the vocabulary provided by the Protocol Ontology. At the implementation time, these

ontology-based definitions of protocols would be coded into some selected agents, who are in charge of distributing the definitions to other agents at run-time, for example, auctioneer agents in an auction MAS, or the "Wrapper" agents in the illustrative Product Search MAS.

Figure 6.54 illustrates the ontology-based definition of the "Query Protocol" in the illuastrative Product Search MAS. This protocol is previously defined in Figure 6.51. Ontology-based definitions of protocols can be represented as Object Diagrams, which is built upon Protocol Ontology's Class Diagram.

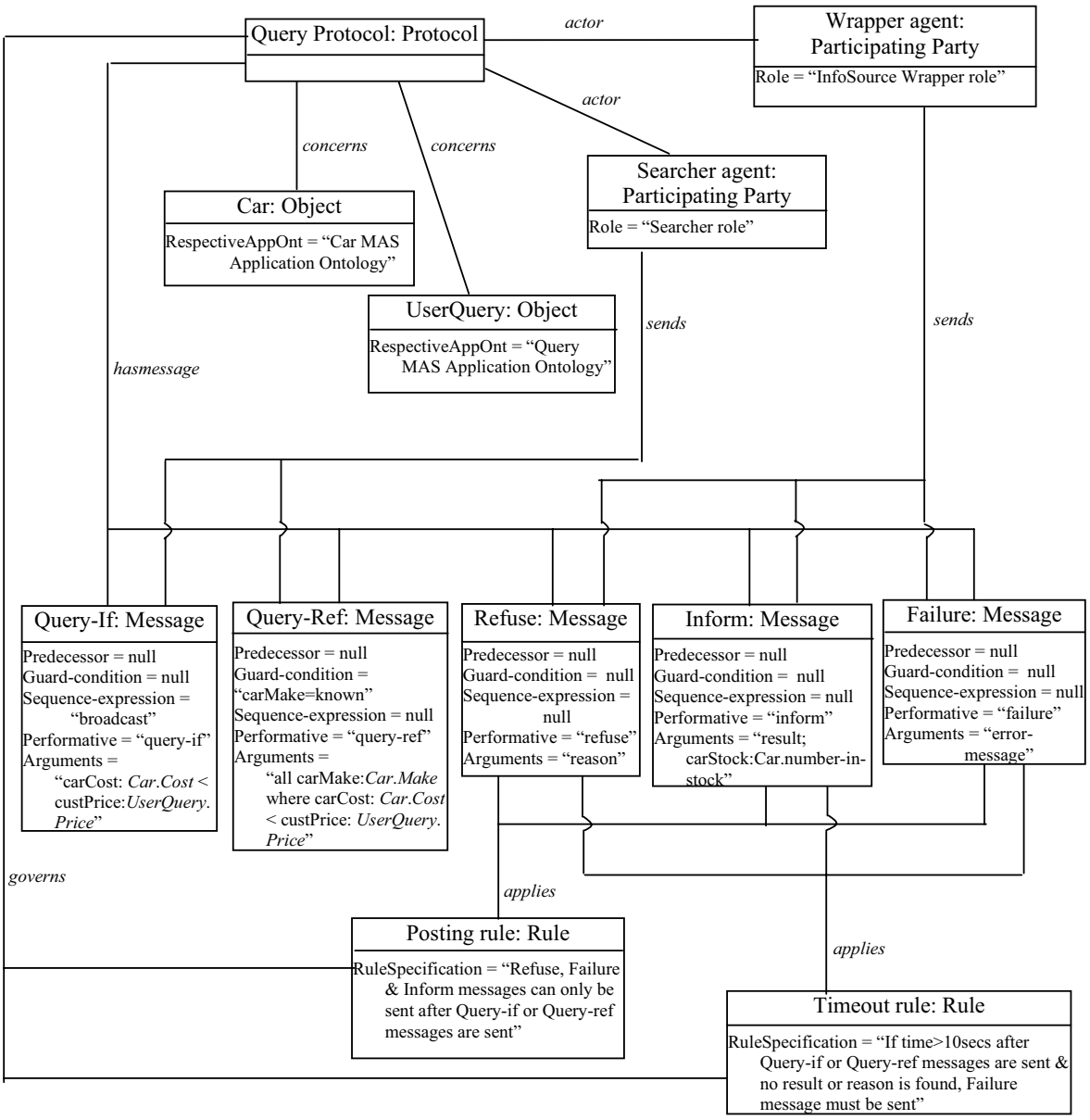


Figure 6.54 – Ontology-based definition of "Query Protocol" (c.f. Figure 6.51)

Update Agent Class Model

Any agent that potentially joins a conversation whose protocol is conceptualised should hold knowledge of the Protocol Ontology that conceptualises this protocol. This means that the Protocol Ontology should be part of the Belief Conceptualisation of the agent.

Figure 6.55 shows the updated Agent Class Diagram of "Searcher" agent class, where the Protocol Ontology (c.f. Figure 6.53) has been added to the Belief Conceptualisation compartment of the agent class. This will allow "Searcher" agents to acquire and understand the ontology-based definition of the "Query Protocol" at run-time.

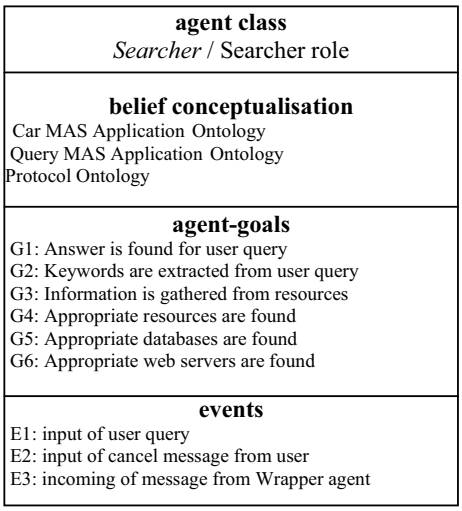


Figure 6.55 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

Introduce new interactions to Agent Interaction Model

Since new agents would need to acquire an appropriate ontology-based protocol definition from an agent in-charge before joining a conversation, there should be some initial interactions between the new agents and the agent in-charge so as to allow for this acquisition. These initial interactions, while simple, should be specified in the Agent Interaction Model. Naturally, protocols governing these interactions should be simple and fixed, hence can be directly hard-coded into the agents.

6.5.2.2. Develop Agent Interaction Model for Tuplespace/Tuple-Centre Interaction Mechanism

In the tuplespace/tuple-centre interaction mechanism, agents interact by inserting, inspecting and removing tuples from a shared tuplespace or tuple-centre. Accordingly,

the Agent Interaction Model Kind of this mechanism is represented by a set of ***Agent-TC[73] Interaction Diagrams***, each of which models the interactions between particular agents and the tuplespace/tuple-centre during a conversation. In addition, for the *tuple-centre* interaction mechanism, since the tuple-centre exhibits programmable reactive behaviour, the Agent Interaction Model Kind should also contain ***Tuple-Centre Behaviour Diagrams***, which model the behaviour of the tuple-centre. Section 6.5.2.2.a discusses the development of Agent-TC Interaction Diagrams, while Section 6.5.2.2.b examines TC Behaviour Diagrams.

6.5.2.2.a. Develop Agent-TC Interaction Diagrams

To identify the potential interactions between the agents in MAS and the share tuplespace/tuple-centre, the developer should examine the *Agent Behaviour Model* of the MAS, namely the Agent Plan Templates and Reflexive Rule Specifications of each agent class.

- Each *communicative action* in the Agent Plan Templates and Reflexive Rule Specifications indicates a tuple being sent from an agent to the tuplespace/tuple-centre.
- Any *event* that is specified in an Agent Plan Template or that triggers a Reflexive Rule Specification may be resulted from a tuple being sent from an agent to the tuplespace/tuple-centre.

Each exchanged tuple should be defined in terms of:

- a communication primitive; and
- definition of the tuple's content.

Regarding the *communication primitives*, most of the existing tuplespace/tuple-centre middleware/models are built upon the LINDA framework (Papadopoulos 2001). As such, they adopt Linda-like communicative primitives such as ***out***, ***in***, ***rd***, ***inp*** and ***rdp*** (Omicini and Zambonelli 1999).

- out writes a tuple to the tuplespace/tuple-centre.

[73] TC stands for "tuple-centre".

- in and rd send a tuple template to the tuplespace/tuple-centre and expect the tuplespace/tuple-centre to return a tuple that match the template, either deleting it or not from the tuplespace/tuple-centre, respectively.
- inp and rdp work analogously to in and rd, however while the latter wait until a matching tuple becomes available, inp and rdp fail if no such tuple is found.

Regarding *the content of tuples*, MOBMAS describes each tuple's content in terms of:

- ***a descriptive name***: e.g. "paperTuple", "all-paper-title-listTuple", "req-num-of-distributionTuple"[74] and "req-num-of-paperTuple"[75]; and
- ***arguments***: which represent the pieces of information that the tuple conveys. An argument can be a *constant* or a *variable*. For variable arguments, their datatypes must be defined. For example, in the following tuples of the illustrative Conference Program Management MAS,

  "req-num-of-distributionTuple(numberDist**: Integer**)" and

  "paperTuple(paperID**: Paper.ID**, paperTitle**: Paper.Title**, paperContent**: Paper.Content**)",

  "numberDist" is an "Integer" variable, while "paperID", "paperTitle" and "paperContent" are variables of datatypes "Paper.ID", "Paper.Title" and "Paper.Content" respectively[76]. "Paper", "ID", "Title" and "Content" are application-specific concepts that are defined in the MAS Application ontology which is shared between the communicating agents to govern the communication's semantics. Note that basic datatypes such as Integer or String are assumed known to every agent in the MAS, without having to be defined in a MAS Application ontology.

Given the above conventions, an example tuple sent by a "PC Chair" agent in the Conference Program Management MAS to the shared tuplespace/tuple-centre is

"**out**(*paperTuple*(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**))",

while a tuple sent by a "PC Member" agent to the tuplespace/tuple-centre is

"**inp**(*paperTuple*(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**)).

---

[74] That is, the required number of members to whom each paper must be distributed.
[75] That is, the required number of papers that each member must collect.
[76] These datatypes should be interpreted as, "ID" of "Paper", "Title" of "Paper" and "Content" of "Paper" respectively.

In the tuplespace/tuple-centre interaction mechanism, all interactions are asynchronous, due to the decoupling in agent identity, location and time during interactions (cf. Section 6.5.1.2).

Notation of Agent-TC Interaction Diagrams

Each Agent-TC Interaction Diagram graphically specifies an allowed pattern of tuple exchanges between agents and a shared tuplespace/tuple-centre during a conversation. MOBMAS reuses the notation of AUML Sequence Diagram for Agent-TC Interaction Diagram. The only difference is that the exchanged elements are tuples instead of ACL messages. Figure 6.56 presents an Agent-TC Interaction Diagram for a conversation between a "PC Chair" agent, a "PC Member" agent and the shared tuplespace/tuple-centre in the Conference Program Management MAS.

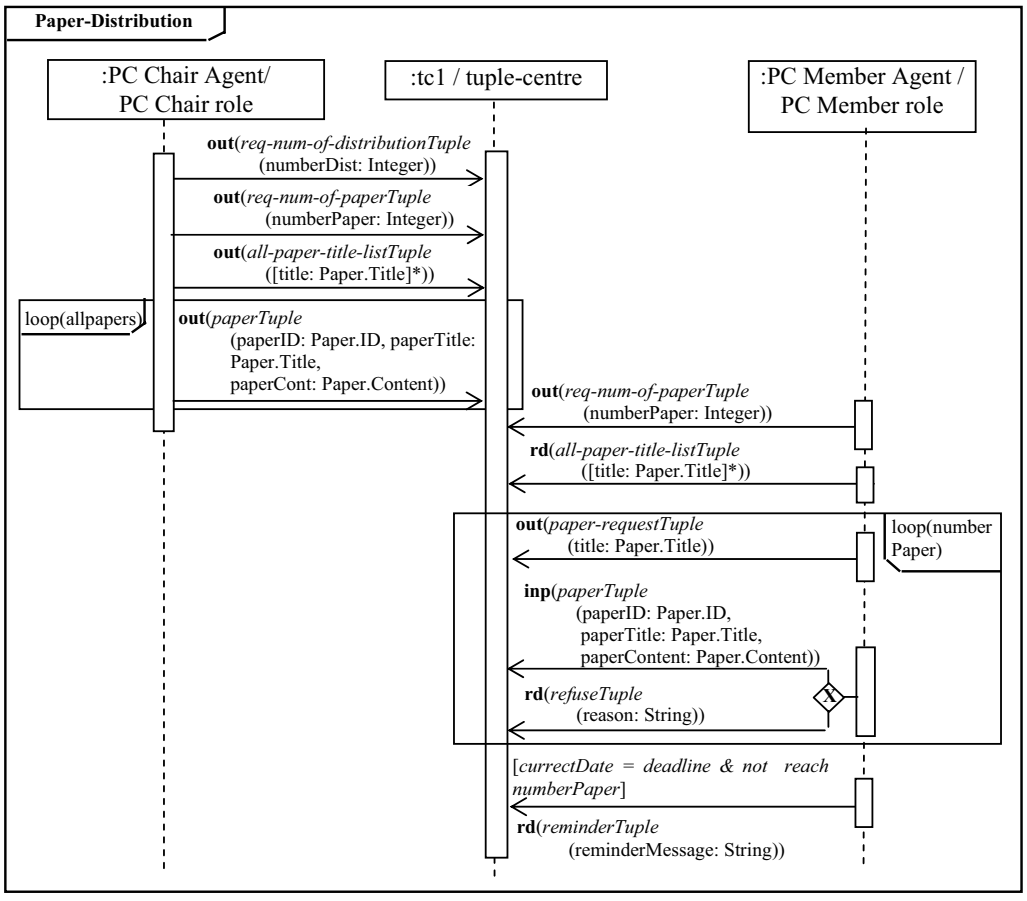


Figure 6.56 – Agent-TC Interaction Diagram for Conference Program Management MAS

6.5.2.2.b. Develop Tuple-Centre Behaviour Diagram (Optional)

This step is applicable only to the *tuple-centre interaction mechanism*. A tuple-centre behaves by *reacting* to the incoming tuples from agents (Omicini and Zambonelli 1999). Each "reaction" is a set of non-blocking actions which, if successfully executed, will change the state of the tuple-centre from one state to another. Otherwise the reaction will yield no transition in the tuple-centre's state at all (Dente et al. 1998).

MOBMAS recommends the following guidelines for the definition of the tuple-centre's reactions.

- *The reactions should allow the tuple-centre to enforce all the necessary coordination rules*: For example, consider agent-goal "Papers are distributed among members" jointly achieved by the "PC Chair" and "PC Member" agent classes in the Conference Program Management MAS (cf. Figure 6.7). One of the coordination rules governing the achievement of the agent-goal is that "Each PC member must not collect the same paper twice" (cf. Section 6.5.1.2). The tuple-centre can enforce this rule by defining a reaction that checks whether a particular "PC Member" agent is eligible to obtain a particular "Paper" tuple (Figure 6.57).

  Similarly, to enforce another coordination rule "Members can only start collecting papers after all other members have viewed the Title List of all papers" (cf. Section 6.5.1.2), the tuple-centre can make *n* copies of the "Title List" tuple, where n is the total number of the "PC Member" agents in the system. Then, the tuple-centre will not allow any "PC Member" agents to consume a "Paper" tuple until all copies of the "Title List" tuples have been consumed by all "PC Member" agents (Figure 6.57).

- *The reactions should allow the tuple-centre to carry out some (low level) processing to help fulfilling the target coordinating task, thereby taking some processing load off the interacting agents*: For example, to assist in the achievement of the agent-goal "Papers are distributed among members", the tuple-centre can try to determine which "PC Member" agent has *not* collected the required number of papers on the due date, thereby posting "Reminder" tuples to these agents.

- *The reactions should help the tuple-centre to deal with* "*inter-agent conflicts*": The issue of inter-agent conflicts have been discussed in Section 6.5.2.1.a.

Notation of Tuple-Centre Behaviour Diagram

MOBMAS adopts UML Statechart Diagram for the Tuple-Centre Behaviour Diagram.

- Each state of the diagram represents a reaction of the tuple-centre. A state can either be passive (i.e. idle, denoted as ⬭) or active. If active, it should contain one or more actions to be executed sequentially by the tuple-centre.
- Transitions between states occur when an event happens, e.g. when a tuple is sent by an agent.

At design time, the developer can specify reactions and state transitions using natural language and descriptive method names. These reactions and transitions can be formally coded using a "behaviour specification language" at implementation, e.g. ReSpecT (Denti et al. 1998).

Figure 6.57 presents the Tuple-Centre Behaviour Diagram for the tuple-centre of the Conference Program Management MAS during a conversation between the "PC Chair" and "PC Member" agents (cf. Figure 6.56).

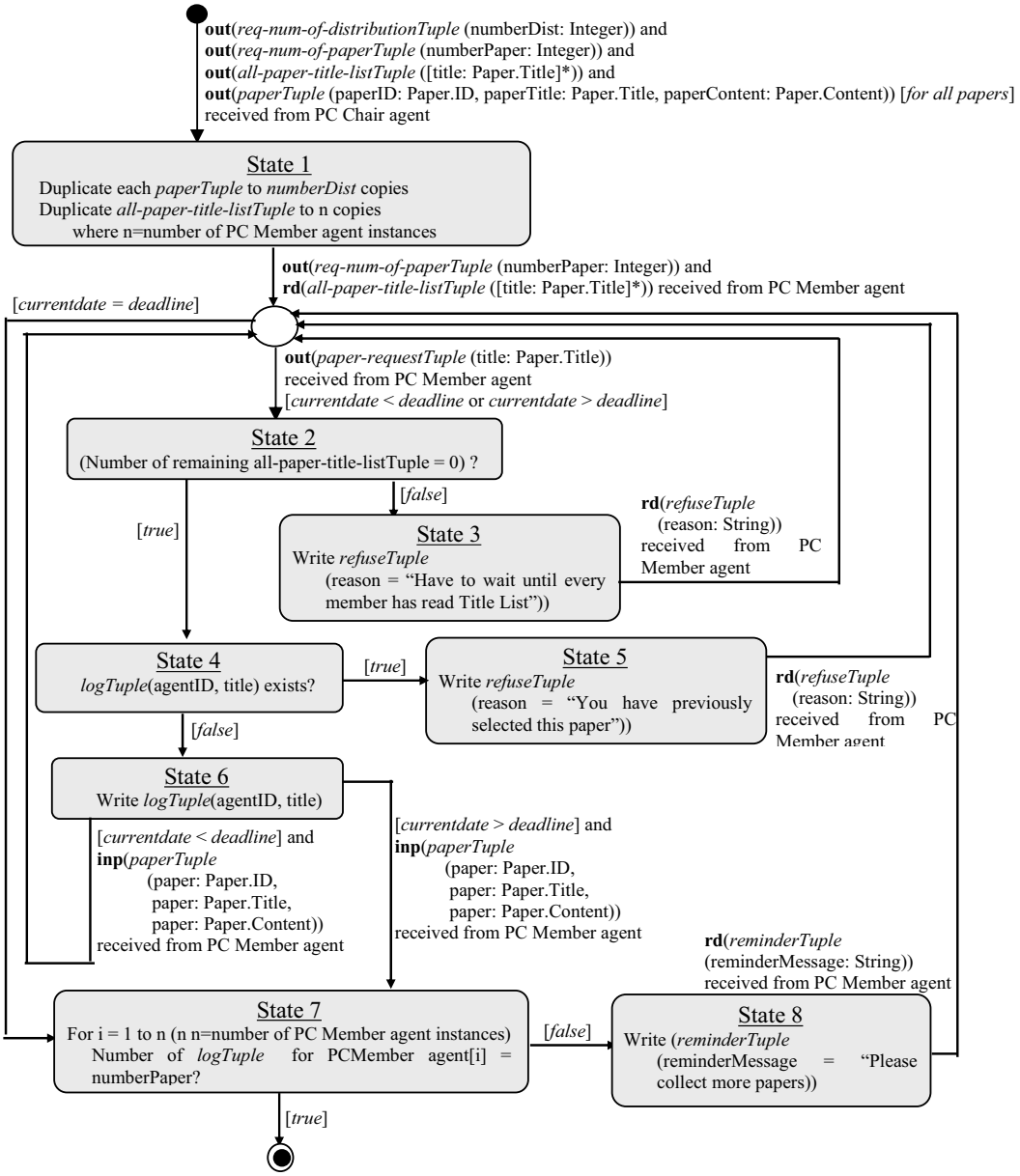


Figure 6.57 – Tuple-Centre Behaviour Diagram for Conference Program Management MAS

6.5.2.2.c. Update Agent Class Model and Role Model

The *Agent Relationship Diagram* of the *Agent Class Model Kind* should be updated to show:

- any new acquaintances between agent classes that have not been identified; and
- various descriptive information about *each* acquaintance, namely:

- the *identity of the Agent-TC Interaction Diagrams* that govern the conversations between the acquainted agents and the shared tuplespace/ tuple-centre; and
- the *MAS Application ontology(ies)* that governs the semantics of the tuples exchanged during the conversation and which must be shared between the communication agents[77].

This descriptive information is modelled as *UML notes* attached to each acquaintance (cf. Figure 6.22).

Figure 6.50 presents the Agent Class Diagram for the Conference Program Management MAS (cf. Figure 6.7).

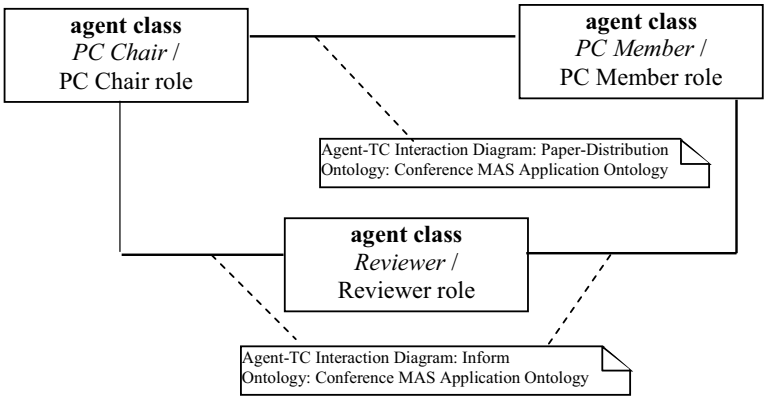


Figure 6.58 – Updated Agent Class Diagram of Conference Program Management MAS

The *Role Diagram* of *Role Model Kind* should also be checked to ensure that all the acquaintances between roles have been captured, and all the authority relationships are still valid. For example, two roles which are initially thought to be in a peer-to-peer relationship may turn out be in a superior-subordinate relationship after an in-depth investigation of the agents' interactions.

6.5.2.3. Verify Agent Interaction Model against Ontology Model and Agent Internal Model

In both the direct interaction mechanism via ACL and indirect interaction mechanism via tuplespace/tuple-centre, the *semantics* of the information conveyed in the ACL messages and tuples must be *consistently* interpreted by the interacting agents. To ensure this consistency, the ***datatypes*** of all variable arguments in the exchanged ACL

[77] That is, this MAS Application ontology must exist in both agents' Belief Conceptualisation.

messages or tuples *must* be defined using the concepts obtained from the MAS Application ontology *shared* between the communicating agents. Being "shared" means that this ontology must exist in the Belief Conceptualisation of each communicating agent. Vice versa, only concepts defined in the *shared* MAS Application ontology can be used to define the datatypes of the variable arguments in the ACL messages or tuples. Basic datatypes such as Integer or String are assumed known to every agent and thus do not need to be defined in an ontology.

For example, in the ACL message sent from a "Searcher" agent to a "Wrapper" agent,

"query-if (carCost:**Car.Cost** < custPrice: **User query.Price**)",

concepts "Car", "Cost", "User query" and "Price" should have been defined in the Car MAS Application Ontology and Query MAS Application Ontology that are shared between the "Searcher" and "Wrapper" agents (cf. Figures 6.13 and 6.14).

Similarly, in the tuple sent from a "PC Chair" agent to a "PC Member" agent via a shared tuplespace/tuple center,

"paperTuple(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**)"

concepts "Paper", "ID", "Title" and "Content" must be defined in a MAS Application Ontology which is shared between the "PC Chair" and "PC Member" agents.

The above guidelines highlight the need for *reciprocal* and *iterative* development of the Agent Interaction Model and Ontology Model. More specifically, the developer should:

- use the concepts defined in the MAS Application ontologies to formulate the content of the exchanged ACL messages and tuples; and
- examine the content of the ACL messages and tuples to determine if any concepts have *not* been defined in the developed MAS Application ontologies, thereby verifying the content of these ontologies.

The developer should also verify the Agent Interaction Model against the Agent Internal Model, to ensure that the ontology(ies) governing the semantics of communication between each pair (or group) of agent classes is indeed listed in the Belief Conceptualisation of each agent class (that is, the ontology is *shared* by the

communicating agent classes). If the communicating agents do not yet share a common ontology, such an ontology should be built and added to each communicating agent's Belief Conceptualisation. This ontology should contain concepts that serve as the inter-lingua between the agents' local (heterogeneous) ontological concepts.

6.6. ARCHITECTURE DESIGN ACTIVITY

This activity of MOBMAS deals with various design issues relating to the architecture of agents and MAS, namely the identification of agent-environment interface requirements, the selection of agent architecture, the identification of required infrastructure facilities, the instantiation of agent classes and the deployment configuration of the agent instances. The product of this activity is an **Architecture Model Kind**, which is represented by the following notational components.

- **Agent-Environment Interface Requirements Specification**: documents any special requirements of agents' sensor, effector and communication modules.
- **Agent Architecture Diagram**: provides a schematic view of the agent architecture(s).
- **Infrastructure Facilities Specification**: documents the specifications of the infrastructure facilities needed to support the target MAS' operation.
- **MAS Deployment Diagram**: shows the deployment configuration of the target MAS, including the allocation of agents to nodes and the connections between nodes.

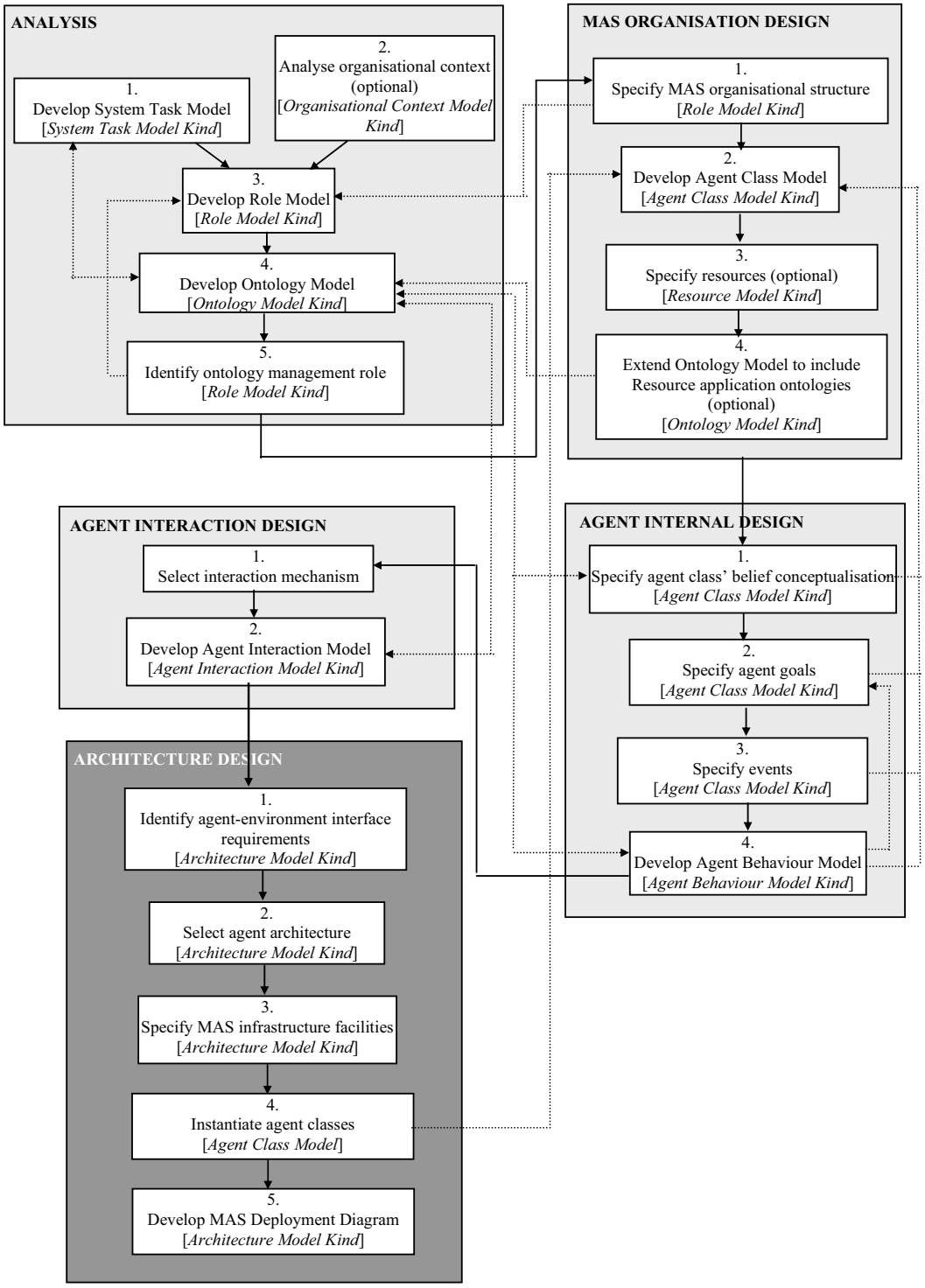


Figure 6.59 – MOBMAS development process

6.6.1. Step 1 – Identify Agent-Environment Interface Requirements

An agent may interact with its environment via (van Breemen 2002):

- *Perception*: which is an activity of observing or sensing the state of the environment;
- *Effect*[78]: which is an activity of changing the state of the environment; and
- *Communication*: which is an activity of exchanging ACL messages or tuples with other agents.

MOBMAS recommends the developer to investigate the characteristics of the perception, effect and communication activities to be performed by each agent class, so as to facilitate the selection of sensors, effectors, agent architectures and/or implementation platform.

Specifically, with regard to **perception** and **effect**, the developer should consider:

- whether they are related to the *physical world* (e.g. observing and changing the location of a soccer ball) or the *virtual world* (e.g. observing and changing the price of a car product) (Russell and Norvig 2003; Weyns et al. 2004). Perception and effect on the physical world typically require hardware components such as thermostats, infra-red sensor, wheels and grippers. In such cases, the sensor and effector of the agents must be able to connect and control these hardware components (probably by communication with the hardware's driver software). Meanwhile, for virtual perception and effect, the sensor and effector of the agents can directly perceive and impact on the environment without the use of hardware components;

- the *degree of complexity* of the perceptual inputs and/or effect outputs. If the agents operate in an open, dynamic and fast changing environment overloaded with sensor information, they may require robust perception mechanisms and efficient perception strategies (e.g. filtering) (Wray et al. n.d.). The developer should also consider employing *sensor/effector objects* to take care of the perception and effect activities for the agents, thereby relieving the agents from some workload. For

[78] "Effect" is referred to as "action" in van Breemen (2002). However, the term "action" is not used here because it may be confused with "actions" in Agent Plan Templates and Reflexive Rule Specification. The latter may or may not result in a change in state of the environment.

example, a sensor object can monitor percepts from the environment, thereupon alerting the agent if certain conditions exist. The sensor object may also serve as a "historical sensor" that watches for trends and patterns in the incoming percepts (Kendall et al. 1995); and

- *interaction with human user*: If an agent is required to engage in intensive interaction with human users, its sensor needs to be connected to an elaborate user-interface component that provides efficient means for inputs from, and outputs to, the users.

Regarding **inter-agent communication**, most existing agent architectures and implementation platforms provide built-in support for basic communication operations, e.g. message dispatching/receiving, tuple inserting/reading/removing, and message/tuple transport services. However, if the target MAS has some special communication requirements, for example, encryption of exchanged messages, mobile and ubiquitous communication, or support for binary data exchanged (such as rich multimedia objects), these requirements should be documented. Note that the interaction mechanism adopted by the target MAS may also impose certain requirements that are not commonly supported by existing agent architectures and implementation platforms. For example, spatially-founded interaction mechanism requires an implementation platform that can provide a spatially structured environment in which agents can be placed and communicate, e.g. SWARM (Minar et al. 1996).

At the end of this step, the *Agent-Environment Interface Requirement Specification* of the *Architecture Model Kind* should be developed to document the followings:

- any special requirements of the agents' sensor and effector (e.g. the need for connecting to hardware components, or the need for using sensor/effector objects); and
- any special requirements of the agents' communication activities.

All of these specifications can be documented in an informal natural language.

6.6.2. Step 2 – Select Agent Architecture

An agent architecture is a structural model of the *modules* that constitute an agent and the *interconnections* between these modules (Lind 1999). Abstract constructs of each agent class (namely, belief conceptualisation, agent-goals, plans and reflexive rules) will be mapped onto these concrete modules during implementation, which is not part of MOBMAS.

Given the availability of a large number of agent architectures, MOBMAS does not address the issue of agent architecture design, but instead presents guidelines on how to select the most appropriate agent architecture(s) for the target MAS.

6.6.2.1. Select agent architecture

Agents in a MAS may adopt the same agent architecture or require different architectures to support their different functional requirements. To select the most appropriate architectures for agents, the developer should consider the following factors.

- *Style of agent behaviour*: The style of agent behaviour is reflected in the Agent Behaviour Model Kind. If an agent class adopts *planning behaviour* to fulfil its agent-goals, the chosen agent architecture must be able to support planning activities via the use of planners, reasoners, analysers, or the like. Some well-known planning architectures are STRIPS (Fikes and Nilsson 1971), IPEM (Ambros-Ingerson and Steel 1988), IRMA (Bratman et al. 1988), Homer (Vere and Bickmore 1990) and SOAR (Newell 1990). If otherwise the agent class adopts solely reflexive behaviour, the chosen agent architecture does not need to support complex symbolic reasoning. Several well-known architectures for reflexive agents are subsumption architecture (Brook 1986), PENGI (Chapman and Agre 1986), AuRA architecture (Arkin and Balch 1997) and situated automata (Kaelbling 1991). In cases when an agent class exhibits hybrid behaviour (e.g. planning behaviour for some agent-goals and reflexive behaviour for some other agent-goals), a hybrid agent architecture which employs a *layered structure* to support both planning and reflexive behaviour is required. Example hybrid architectures are RAPs (Firby 1989), ATLANTIS (Gat 1991), TouringMachines (Ferguson 1992), INTERRAP (Muller and Pischel 1993),

Prodigy (Carbonell et al. 1991), PRS (Myers 1997; Georgeff and Lansky 1987) and dMARS (d'Inverno et al 1997).

- *Required agent behavioural capabilities*: The selected agent architecture should allow an agent to implement its desirable behavioural capabilities. For example, if an agent class is required to learn, the selected agent architecture should be able to support learning capability. Or, if the agent class needs to be time-persistent, its architecture must allow the knowledge structures to be maintained over time and over unavoidable downtimes (Wray n.d.).

- *Style of Control*: An agent class' desirable style of control can help to determine the required agent architecture. Some example styles of control are (Wray n.d.; Lind 1999):
  - asynchronous versus synchronous: The former requires an agent architecture that supports asynchronous processing threads while the latter does not; and
  - static versus dynamic: The former can be supported by an agent architecture that implicitly hardcodes the control flow, while the latter requires an architecture that allows the control flow to be explicitly specified (e.g. in plan scripts).

- *Knowledge representation mechanism*: The desirable mechanism of agent knowledge representation should be matched against the mechanism provided by the agent architecture. For example, an agent architecture may allow the agent knowledge to be explicitly stored in a knowledge base, or requires the knowledge to be implicitly embedded in the agent coding (Lind 1999).

- *Complexity of sensor input*: Agents that operate in a dynamic, open and fast-changing environment often faces overloads in sensor information. They should therefore adopt an architecture that supports robust perception mechanisms and efficient sensing strategies (e.g. sensor inputs filtering).

- *Support for scalability*: Some agent architectures such as HOMER experience a processing slow-down when its episodic knowledge base increases in size (Vere and Bickmore 1990). Thus, the developer should select an agent architecture that can

accommodate the agents' expansion of knowledge (e.g. via the support for easy-upsizing of the knowledge base) (Wray n.d.).

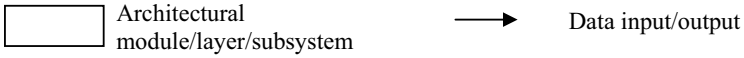
- *Agent-environment interaction requirements*: The sensor, effector and communication modules of the selected agent architecture should be able to support the requirements specified in the Agent-Environment Interface Requirements Specification (cf. Section 6.6.1).

6.6.2.2. Develop Agent Architecture Diagram

As mentioned previously, *Agent Architecture Diagram* is one of the four notational components of the *Architecture Model Kind*. It aims to provide a schematic view of an agent architecture. If agents in the target MAS are homogeneous in architecture, only one Agent Architecture Diagram would be needed. Otherwise, multiple diagrams are required.

An Agent Architecture Diagram should specify:

- *the* modules or layers or subsystems of the architecture. These modules/layers/subsystems should be represented as boxes; and
- the potential flows of data between these modules/layers/subsystems. These flows are represented as arrows.



Figures 6.60 and 6.61 show the Agent Architecture Diagrams for the TouringMachines and INTERRAP architectures.

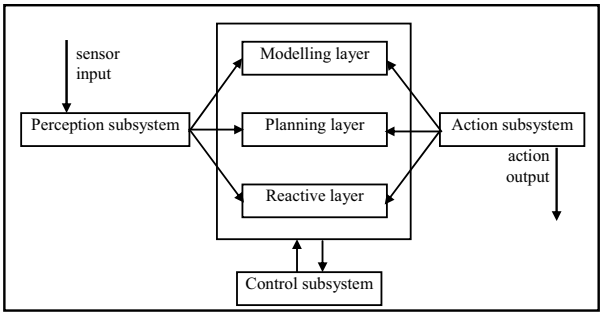


Figure 6.60 – Agent Architecture Diagram for TouringMachines architecture (Ferguson 1992)

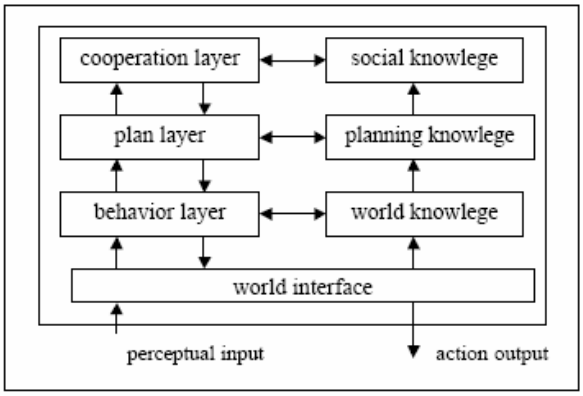


Figure 6.61 – Agent Architecture Diagram for INTERRAP architecture (Wooldridge 1999)

6.6.3. Step 3 – Specify MAS Infrastructure Facilities

As in other computing systems, MAS needs to be supported by various infrastructure facilities in order to operate. The developer should thus identify these infrastructure facilities and determine how these facilities can be provided and managed in the target MAS.

Various potential infrastructure facilities for a MAS system are (Iglesias et al. 1998; Shen et al. 1999):

- *Network facilities*: e.g. agent naming service, agent creation/deletion service, agent migration service, security service and accounting service;
- *Coordination facilities*: e.g. agent directory/yellow-page/white-page service, message transport service, protocol servers and "police" facilities to detect misbehaviours in agents or misusage of common resources; and
- *Knowledge facilities*: e.g. ontology servers, problem-solving methods servers and language translation service.

Some of these facilities are only necessary if the MAS is open and/or deployed in a distributed environment, e.g. agent migration services, security service and agent directory services. For many applications such as simulation, the MAS can be closed and run on a single platform, thus they do not require network-related infrastructure facilities.

MOBMAS suggests two common mechanisms for providing and managing infrastructure facilities.

- To employ the "built-in" infrastructure facilities provided by the MAS implementation platform, without managing them by any dedicated agents. For example, the kernel of MADKIT platform provides built-in "message transport services" (MADKIT 2002), while the kernel of AgentTcl offers built-in "agent migration services" (Gray 1995).

- To provide and manage customized infrastructure facilities via the use of some dedicated agents. For example, "directory services" may be offered by a "Facilitator/Broker" agent, while "security services" can be handled by a specialised "Police" agent. These agent classes may have already been defined in the *Agent Class Model*, or are now introduced. In both cases, the developer should refine the Agent Class Model to include new agent classes or to update the specification of the existing agent classes. All related models, including *Agent Behaviour Model* and *Agent Interaction Model*, should be accordingly revised.

An *Infrastructure Facility Specification* should be developed and included in the *Architecture Model Kind* to document the specifications of all the identified infrastructure facilities. These specifications can be documented in an informal natural language.

6.6.4. Step 4 – Instantiate Agent Classes

Each agent class should be instantiated into concrete agent instances. Common types of cardinality for agent instantiation are (Wooldridge et al. 2000):

- *n* cardinality: i.e. where each agent class is instantiated with exactly *n* agents;
- *m..n* cardinality: i.e. where each agent class is instantiated with no less than *m* and no more than *n* agents; and
- + cardinality: i.e. where each agent class is instantiated with one or more agents.

The instantiation cardinality of each agent class is represented as an annotation next to the agent class name in the **Agent Relationship Diagram** of the **Agent Class Model**

Kind (cf. Figure 6.25). Figure 6.62 presents the updated Agent Relationship Diagram for the illustrative Product Search MAS.

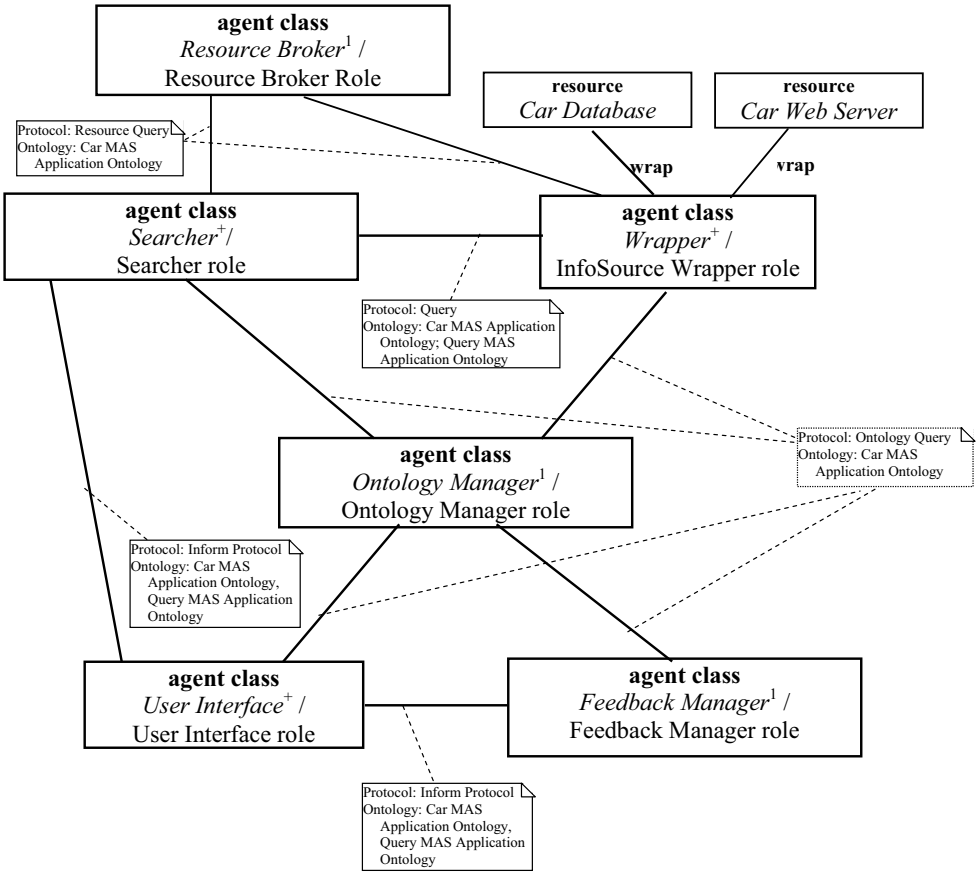


Figure 6.62 – Updated Agent Relationship Diagram of Product Search MAS

6.6.5. Step 5 – Develop MAS Deployment Diagram

In MOBMAS, the logical architecture of the target MAS is revealed in the *Agent Relationship Diagram* of the *Agent Class Model Kind*. This diagram shows all the agent classes composing the MAS system, the resources existing in the system, the acquaintances between agent classes, the connections between agent classes and resources, and the instantiation of agent classes. In this step, a *MAS Deployment Diagram* should be developed to describe how this logical MAS architecture can be actuated in the operational environment (i.e. how the MAS components can be located, distributed and connected). This diagram is particularly necessary for highly distributed MASs where it is important to visualise the system's physical topology. The MAS

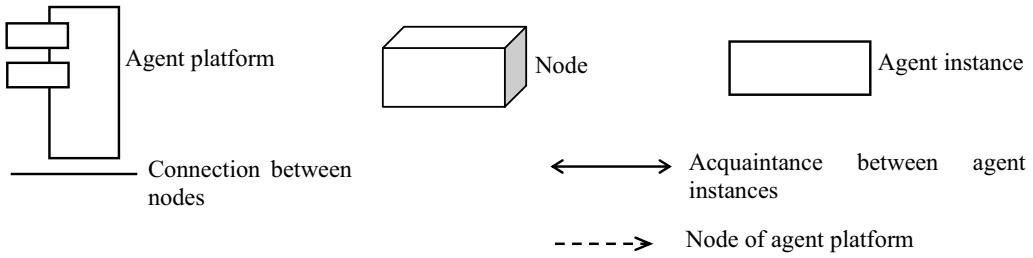
Deployment Diagram is one of the four notational components of the Architecture Model Kind as mentioned previously.

Major configuration details to be specified in the MAS Deployment Diagram include:

- *agent platforms*: i.e. the infrastructure on which agents are deployed. Agent platforms can be single processes containing lightweight agent threads, or fully built platforms around proprietary or open middleware standards (FIPA 2003);
- *nodes*: i.e. hosts on each agent platform;
- *agent instances located at each node*;
- *connections between nodes*; and
- *acquaintances between agent instances*.

The deployment configuration of MAS can be determined by investigating various factors, including the message traffic between nodes (estimated from *Agent Interaction Model*) and the required processing power of each node to accommodate the behaviour of agents (estimated from the *Agent Behaviour Model*).

MOBMAS reuses the notation of AUML Deployment Diagram (FIPA 2003) for its MAS Deployment Diagram[79].



The naming of agent platforms, nodes and agent instances should be specified in the format

instance-name : class-name

[79] AUML Deployment Diagram contains notation for agent mobility. However, since agent mobility is not discussed in MOBMAS, this notation is not presented.

Figure 6.63 presents an example MAS Deployment Diagram for the Product Search MAS.

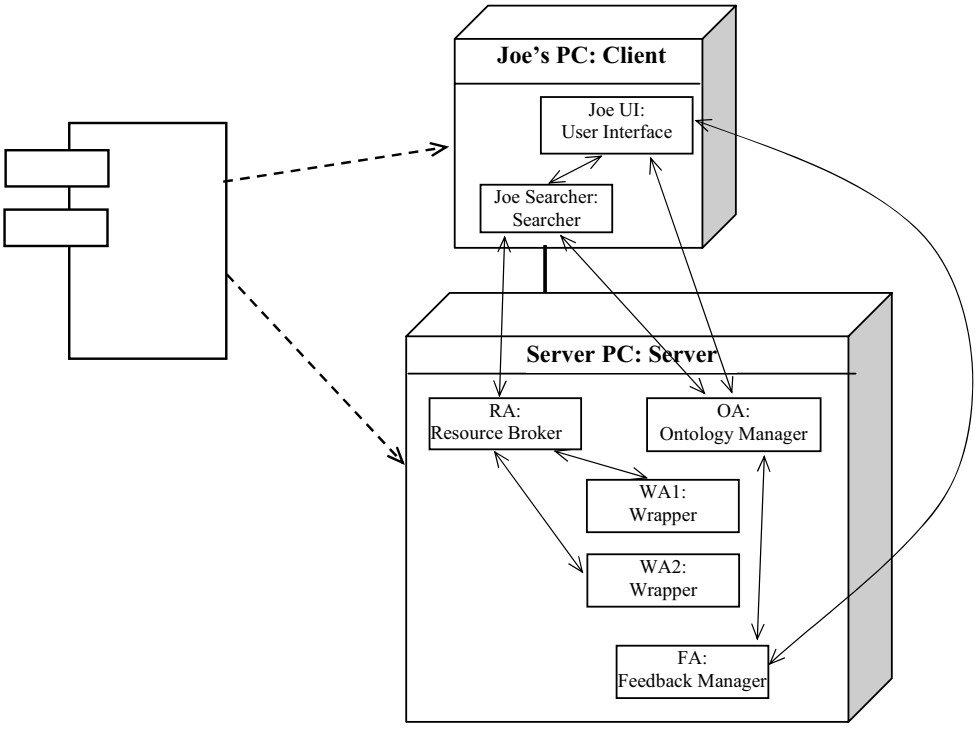


Figure 6.63 – MAS Deployment Diagram for Product Search MAS

6.7. SUMMARY

This chapter has presented the full documentation of MOBMAS – a "***M***ethodology for ***O***ntology-***B***ased ***MAS*** development". In total, MOBMAS consists of five activities, twenty steps and nine model kinds. An overview of MOBMAS' conceptual framework, development process and model kinds can be found in Section 6.1. Figure 6.2, in particular, depicts the five activities of MOBMAS, their associated steps, the iterative flow between these steps, and the model kinds produced or refined by each step. A detailed description of each MOBMAS' activity is provided in Sections 6.2 to 6.6.

In the next chapter, Chapter 7, the evaluation and refinement of MOBMAS are documented. These evaluation and refinements progressively led to the final version of MOBMAS as presented in this chapter.

CHAPTER 7

EVALUATION AND REFINEMENT OF MOBMAS

7.1. INTRODUCTION

This chapter documents the evaluation and refinement of MOBMAS as resulted from the execution of Research Activity 3 – "*Evaluate and Refine MOBMAS*" (cf. Section 4.3). The process of evaluating and refining MOBMAS consisted of three sequential steps (cf. Section 4.6):

- *Step 1* - Obtaining expert reviews;
- *Step 2* - Using MOBMAS on a test application; and
- *Step 3* - Performing a feature analysis on MOBMAS.

The first two steps resulted in the most refined version of MOBMAS which is presented in Chapter 6, while the third step evaluated this final version. Sections 7.2, 7.3 and 7.4 respectively report on the performance and outcome of each step.

7.2. EXPERT REVIEWS

The objective of expert reviews was to gather experts' non-empirical evaluation of the initial version of MOBMAS, specifically their opinions on the strengths, areas for improvement of the methodology, and how to improve these areas. The obtained feedback was used to refine MOBMAS before the methodology was empirically used on a test application by external developers (Section 7.3).

Section 7.2.1 firstly describes the procedures of the expert reviews, followed by Section 7.2.2 which provides brief biographic information about each expert. Section 7.2.3 then documents the refinements of MOBMAS as a consequence of the expert reviews.

7.2.1. Expert Review Procedures

Each of the two experts was provided with a full documentation of MOBMAS and requested to:

- identify the strengths of the methodology (optional).
- identify the areas for improvement of the methodology; and
- recommend how to improve these areas (optional).

The evaluation was based on the experts' non-empirical investigation of the methodology's documentation, and documented informally as "comment notes" on the methodology's documentation.

The two expert reviews were collected *independently* and in a *sequential* order. Comments from the first expert were used to refine the initial version of MOBMAS before the second expert was asked to evaluate the refined version. This sequential and independent procedure

- prevented the possibility of two experts identifying the same areas for improvement; and
- helped to identify new areas of improvement that potentially arose from the refinement of the methodology after the first review. Modifying the methodology according to the first expert review might introduce new problematic areas, which would be undetected if the second review was not subsequently conducted.

A face-to-face meeting was then organised with each expert to get a walkthrough of his/her feedback. This face-to-face communication helped to get an in-depth understanding of, and avoid any possible misunderstanding of, the experts' comments.

After MOBMAS was refined in accordance with each expert's feedback, follow-up communication via email was carried out with each expert to ensure that the experts were satisfied with all the refinements made to the methodology.

7.2.2. Experts' Biography

The two experts who participted in the study were Prof. Brian Henderson-Sellers (first reviewer) and Prof. Mary-Anne Williams (second reviewer). Both experts have active research interests in agent technology in general and AO methodologies in particular. They both have made major research contributions to the area.

- Prof. Brian Henderson-Sellers is the Director of the Centre for Object Technology Applications and Research at the University of Technology, Sydney. His research interest in AO methodologies arose from his long-term involvement in OO methodologies. He co-organised a number of international workshops on AO methodologies (including Annual ACM Conference on Object-Oriented Programming, Systems, Languages & Applications – OOPSLA, and International Bi-Conference Workshop on Agent-Oriented Information Systems). He is currently co-leading a research project funded by the Australian Research Council on "Metamodel-based Methodology for Developing Agent-Oriented Systems". The project investigates how to extend the OPEN framework, which was initially designed for OO development and of which he was a co-founder, to offer support for AO system development.

- Prof. Mary-Anne Williams is the Director of the Innovation and Technology Research Laboratory at the University of Technology, Sydney. She has been actively involved in research on both agent technology (particularly in the context of e-business, interactive marketing and artificial intelligence) and ontology. She has also supervised PhD students whose research interests were agent technology and/or ontology. Two of her current research projects are "Agent-Oriented Concept Management" and "Information and Knowledge Integration", both of which focused on agents and ontologies. She has also been the co-organiser of the International Conference on Principles of Knowledge Representation and Reasoning (KR), which covers the research topic of ontologies. With regard to this PhD research project, prior to getting involved as an expert reviewer of the MOBMAS methodology, Prof. Mary-Anne Williams was involved in the research in two aspects: she helped to shape the broad topic area of this research by sharing her background knowledge of the agent literature, and she assisted in recruiting participants for the research's

survey (Section 5.3) by helping to post the survey's recruiment advertisement on the UMBC Agents-Digest mailing list and UMBC AgentNews newsletter. At no stage was Prof. Mary-Anne Williams involved in the design of the MOBMAS methodology itself. Thus, Prof. Mary-Anne Williams was a totally independent reviewer of the methodology.

7.2.3. Refinements of MOBMAS

This section documents the refinements made to MOBMAS as a consequence of each expert's review. Only the refinements made to the steps and model definitions of MOBMAS are presented. Refinements made to the wording to improve MOBMAS' comprehensibility, coherence and expressiveness are not listed. The experts' comments are presented in Appendix F.

7.2.3.1. Refinements of MOBMAS as a result of the first expert review

1. Re-define the origin of MOBMAS modelling notation. If the notation of a particular MOBMAS notational component is not applicable to be an extension of UML and AUML (i.e. when all valid UML and AUML extensibility mechanisms[80] are inapplicable), that notation should be documented as MOBMAS' *own* notation rather than UML's or AUML's extension. (The initial version of MOBMAS states that some notational components are an extension of UML/AUML while they are not). In the refined version of MOBMAS, the notation for the following notational components is re-defined as MOBMAS' own notation:
   - Role Diagram;
   - Agent Class Diagram[81];
   - Agent Relationship Diagram;
   - Agent Plan Template; and
   - Resource Diagram.

[80] Namely, "stereotype", "tagged value" and "constraint" (Object Management Group 2003).
[81] For the Agent Class Diagram, the notation of AUML Agent Class Diagram (Huget et al. 2003; Bauer et al. 2000; Bauer, 2001a; Bauer 2001b) had been considered. However, since much of the syntax and semantics of the AUML Agent Class Diagram notation has not yet been determined and finalized at the time of this research, it was not adopted.

2. Provide elaborate definition of the syntax and semantics of MOBMAS notation. Whenever UML or AUML notation was reused or extended (namely in Ontology Diagram, Agent Plan Diagram, Reflexive Rule Specification, Interaction Protocol Diagram, Agent-TC Interaction Diagram, Tuple-Centre Behaviour Diagram and MAS Deployment Diagram), the reused or extended notational elements were highlighted, followed by the clarification of how the syntax and semantics of UML/AUML notation have been reused or extended.

3. Document MOBMAS' conceptual framework, which defines the semantics of the main abstractions that underlie MOBMAS development process and model kinds. A diagram that shows the relationships between these abstractions was also added.

4. Extend the modelling of ontological relationships in the Ontology Model Kind to include the modelling of "composition" relationship. The difference in semantics between "composition" and "aggregation" was also highlighted.

5. Fix the following notational errors:
   - An idle state or a decision point in the Tuple-Centre Behaviour Diagram (which is basically a UML State Chart) should be represented as a circle ◯ and not a diamond ◇, to adhere to UML notation.
   - Each lifeline in the Interaction Protocol Diagram and Agent-TC Interaction Diagram of the Agent Interaction Model Kind should represent an agent instance and not an agent class. Accordingly, there should be a colon (**:**) in front of the agent class name in the rectangle above the lifeline.

7.2.3.2. Refinements of MOBMAS as a result of the second expert review

1. Revise step "Develop System Task Model" and step "Analyse Organisational Context" of the "Analysis" activity (Section 6.2) to avoid any overlap in the applicable conditions of these steps. Specifically, in the initial version of MOBMAS, step "Develop System Task Model" was recommended if

   > "…the target MAS is a processing application system that does not exhibit any specific and apparent human organisational structure",

while step "Analyse Organisational Context" was suggested if

> "… the target MAS aims to mimic or support a human-like organisation and the human organisational structure is clear".

In the refined version of MOBMAS, the developer was recommended to "Analyse Organisational Context" if

> "… the target application satisfies **all** of the following conditions.
>
> - The structure of the real-world organisation is known and clear.
> - The real-world organisational structure is well-established, not likely to change, and has proven or been accepted to be an effective way to function. Accordingly, it is desirable for the future MAS to mimic this existing structure."

and to "Develop System Task Model" if

> "…the target application does not satisfy all of the above conditions, or if the developer is unsure of whether the target application satisfies all of these conditions."

2. Change the naming convention of "system-task" from "*to do something*" (which sounds like an objective) to "*do something*" (which sounds more like an activity, e.g. "Receive user query" or "Get information from resources").

3. Extend step "Develop Agent Behaviour Model" of the "Agent Internal Design" and step "Develop Agent Interaction Model" of the "Agent Interaction Design" to consider the adoption, and allow the developer to adopt, various techniques for conflict resolution within agent classes and between agent classes. Recommended techniques included priority conventions (Ioannidis and Sellis 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981) and evidential reasoning (Carver and Lesser 1995), negotiation (Sycara 1988), voting (Ephrati and Rosenschein 1991), and standardization and social rules (Shoham and Tennenholtax 1992).

4. Extend step "Select Interaction Mechanism" of the "Agent Interaction Design" activity to add a new criterion for the comparison between the direct interaction mechanism via ACL and the indirect interaction mechanism via tuplespace/tuple-centre: security control. This criterion examines how the two mechanisms differ in their control and implementation of security.

5. Extend step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity to consider the modes of synchronisation in agent interaction. Two common modes of synchronisation were described: synchronous and asynchronous.

6. Extend step "Identify agent-environment interface requirements" of the "Architecture Design" activity to include:
   - description of the different types of interactions between an agent and its environment (namely, perception, effect and communication with other agents);
   - consideration of the differences between physical and virtual environments in term of the requirements of the agents' sensor and effector; and
   - consideration of the requirements of inter-agent communications.

7. Allow the developer to adopt any ontology modelling languages for the Ontology Model Kind. UML was only used by MOBMAS for illustration purpose.

8. Mention the possibility of a complex tree structure in the System Task Diagram of System Task Model Kind.

9. Extend step "Specify Resources" of the "MAS Organisation Design" activity to discuss the distinction between those resources which belong *internally* to the MAS system, and those resources which exist externally and are available to agents in other systems.

10. Change the notation of AND/OR decomposition in the System Task Diagram of System Task Model Kind from the uncommon notation introduced by TROPOS (Figure 7.2) to the well-known notation of AND/OR graphs (Figure 7.1).

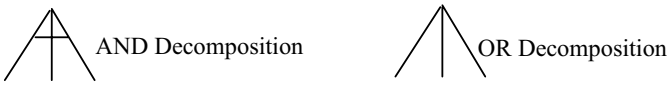


Figure 7.1 – Notation of AND/OR Graphs

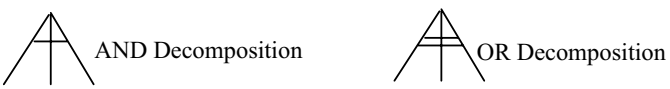


Figure 7.2 – TROPOS notation for AND/OR decomposition

7.3. APPLICATION OF MOBMAS

After MOBMAS was refined by Step 1 – "*Obtain expert reviews*" (Section 7.2), it was further evaluated and refined by Step 2 – "*Using MOBMAS on a test application*" (cf. Section 4.6). In this step, MOBMAS was given to two external developers who sequentially applied the methodology on a "Peer-to-Peer Information Sharing" application (Appendix H) and evaluated the methodology according to their experience of usage. The obtained feedback was investigated to refine MOBMAS into its final version as presented in Chapter 6. Compared to the expert reviews, the application of MOBMAS aimed to evaluate the methodology at a more detailed and elaborate level, specifically at *every step* and *every model kind* of the methodology.

Section 7.3.1 firstly describes the procedures of MOBMAS application. Section 7.3.2 then provides biographic information about each external developer. Section 7.3.3 finally reports on the refinements of MOBMAS as a consequence of the developers' evaluation.

7.3.1. Application procedures

Each of the two developers was provided with a full documentation of MOBMAS and requested to:

- apply it on a test application;
- produce a design of MAS by using the methodology and generate a number of major models to illustrate this design; and
- evaluate the methodology based on the usage of the methodology. Each developer was asked to:
  - identify the strengths of the methodology (optional);
  - identify the areas for improvement of the methodology;
  - recommend how to improve these areas (optional);
  - rate the "ease of understanding" and "ease of following" of each step of the methodology on a High-Medium-Low scale; and
  - rate the "ease of understanding" of each model kind of the methodology on a High-Medium-Low scale.

To help the developers to systematically and thoroughly record their evaluation, an evaluation form was designed. The form consists of five parts, each collecting the developers' evaluation of each activity of the methodology. The full evaluation form can be found in Appendix G.

MOBMAS was applied by the two developers in a *sequential* order. Evaluation from the first developer was obtained and used to refine MOBMAS before the second developer was asked to apply and evaluate this refined version. Reasons for this sequential application process are the same as those stated for the expert reviews in Section 7.2.1:

- to prevent the possibility of the two developers identifying the same areas for improvement of MOBMAS; and
- to help identifying new areas of improvement that potentially arose from the refinement of MOBMAS after the first application.

A face-to-face meeting was organised with each developer at the end of each application to get a walkthrough of his feedback. After MOBMAS was refined in accordance with each developer's feedback, follow-up meetings and email communication were carried out with the developer to ensure that he was satisfied with the refinement made to the methodology. In addition, the refinements made to MOBMAS as a result of the second developer's feedback were also discussed with the first developer to ensure that no conflicts of opinions incurred.

7.3.2. Developers' biography

The two developers who participated in the study were Dr. Ghassan Beydoun (first developer) and Dr. Cesar Gonzalez-Perez (second developer). Both developers have extensive knowledge and experience in software engineering in general and AO software engineering in particular.

- Dr. Ghassan Beydoun is currently a post-doc research fellow at the University of New South Wales, Sydney. He is working on a project funded by the Australian Research Council, entitled "FAME – Futuristic Agent-Oriented Method Engineering". The project aims to develop a repository of AOSE method components that can be used to build a MAS development methodology that suits

the application or organisation at hand. Dr. Beydoun's expertise also covers the realm of ontology, given his active research involvement in the areas of knowledge engineering, knowledge management, knowledge representation and acquisition. Although Dr Beydoun's research's location is the same as that of the conductor of this PhD study, he was not involved in the design and execution of this research in any way, except for being an independent tester.

- Dr. Cesar Gonzalez-Perez is currently a post-doc research fellow at the University of Technology, Sydney. He is another participant in the FAME project funded by the Australian Research Council, "Futuristic Agent-Oriented Method Engineering". His extensive knowledge and experience on software engineering include the involvement in developing the "International Standard Metamodel for Software Development Methodologies" (in progress) and "OPEN/Metis" - an integral, object-oriented software development framework. Dr. Gonzalez-Perez is also the founder and former technical director of NECO, a company based in Spain specialising in software development. While working at the same university as the two expert reviewers, Prof. Brian Henderson-Sellers and Prof. Mary-Anne Williams (c.f. Section 7.2.2), Dr Beydoun did not have any communication with these reviewers with regard to MOBMAS. His testing of the methodology was completely independent and unbiased by the preceeding expert reviews.

7.3.3. Refinements of MOBMAS

This section documents the refinements made to MOBMAS as a result of each developer's comments. Only the refinements made to the steps and model definitions of MOBMAS are presented. Refinements made to the writing style or wording to improve MOBMAS' comprehensibility, coherence and expressiveness are not listed. The two developers' comments are presented in Appendix G. These comments were recorded via an evaluation form. Appendix H documents the "Peer-to-Peer Information Sharing" application on which MOBMAS was applied, and the major models created by each developer for the application.

7.3.3.1. Refinements of MOBMAS as a result of Developer 1's comments

1. Make step "Develop System Task Model" a compulsory step in the "Analysis" activity (Section 6.2). Step "Analyse Organisational Context", however, remained optional. The applicable condition of the latter step was left unchanged.

2. Extend step "Develop Role Model" of the "Analysis" activity to highlight the need for spiral, iterative development of System Task Model and Role Model.

3. Extend step "Develop Agent Class Model" of the "MAS Organisation Design" and step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity to provide support for the modelling of agent class' dynamics.

4. Extend step "Develop Ontology Model" of the "Analysis" activity to include the discussion of the use of ontologies to validate System Task Model and Role Model.

5. Extend step "Identify Ontology Manage Roles" of the "Analysis" activity to describe an alternative design approach where no ontology manager is needed. Advantages and disadvantages of this alternative design were documented.

6. Move step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity in front of step "Develop Agent Class Model" (of the same activity) to make the steps easier to follow.

7. Extend step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity to consider the "hybrid" style of organisational structure.

8. Merge steps "Identify Resource Application Ontologies" and "Develop Resource Application Ontologies" of the "MAS Organisation Design" activity into one step "Extend Ontology Model To Include Resource Application Ontologies".

9. Make steps “Specify resources” and “Extend Ontology Model To Include Resource Application Ontologies” of the “MAS Organisation Design” activity optional, because these steps are only applicable to heterogeneous systems.

10. Remove step “Develop System Overview Diagram” from the “MAS Organisation Design” activity. Instead, a sub-step “Update Agent Class Model” was defined within step “Specify resources” of the same activity to show the overall architecture of MAS.

11. Simplify Agent Class Diagram by omitting various compartments which were initially created to store the names of Agent Plan Templates for an agent class. The Agent Behaviour Model Kind was instead used to capture the specification of these Agent Plans.

12. Rename the relationships between agent classes in the Agent Relationship Diagram from “association” to “acquaintance” to clarify that these relationships represent interaction pathways.

13. Extend step “Specify Agent Class’ Belief Conceptualisation” of the “Agent Internal Design” activity to discuss the dynamics of the belief conceptualisations and belief state of agents at run-time.

14. Revise step “Develop Agent Behaviour Model” of the “Agent Internal Design” activity to re-define the information to be specified in the Agent Plan Template. Instead of modelling the exact sequences of actions to be performed by an agent to achieve an agent-goal, an Agent Plan Template should model a repository of *potential* actions and/or sub-agent-goals that *may* be selected by the agent to perform to achieve the target agent-goal at run-time. The selection of which actions to perform, and the sequencing of these actions, are delegated to the planners or reasoners or means-end analysers that are built in to the agent architecture/platform. The sequencing of sub-agent-goals may be modelled for an Agent Plan if this sequence is fixed and determinable at design time.

15. Extend sub-step "Develop Agent Plan Templates" of step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity to specify:
    - "commitment strategy" for each Agent Plan Template; and
    - "conflict resolution strategy" to deal with conflicts between agent-goals (if any).

16. Extend step "Develop Agent Interaction Model" of the "Agent Interaction Design" step to:
    - address the identification of "communicative actions" in the Agent Plan Templates and Reflexive Rule Specifications; and
    - highlight the need for consistency checking between Role Model, Agent Class Model and Agent Interaction Model (i.e. checking whether all the acquaintances between roles and agent classes have been captured, and whether all the authority relationships previously defined for roles and agent are still valid).

17. Extend step "Identify agent-environment interface requirements" of the "Architecture Design" activity to consider two other characteristics of the perception and effect activities of agent classes: the degree of complexity of the perceptual inputs and effect outputs, and the interaction with human user.

18. Revise step "Select agent architecture" of the "Architecture Design" activity to remove the statement that all agent architectures listed in MOBMAS are commercial products. Some agent architectures are actually available to only the academic realm (e.g. SOAR).

19. Extend step "Specify MAS Infrastructure Facilities" of the "Architecture Design" activity to address the case when MASs are closed and run on a single platform.

20. Remove the notation for agent mobility from the MAS Deployment Diagram, since MOBMAS does not address mobile agents. However, the capability of the MAS Deployment Diagram to model agent mobility is noted in a footnote.

7.3.3.2. Refinements of MOBMAS as a result of Developer 2's comments

1. Provide an explicit definition for the term "(MAS) *environment*".

2. Use the term "*model kind*" to refer to the definition of a class of models. The term "*model*" is used to mean a particular work product that is created by the developer at the design time.

3. Use the term "*activity*" to refer to a group of steps in MOBMAS instead of the term "*phase*" as used in the earlier version of MOBMAS. This change helps to avoid the implication of temporal ordering between the activities (MOBMAS adopts an iterative and incremental development life cycle where the developer is allowed to move back and forth across activities).

4. Refine the diagram that shows MOBMAS' conceptual framework (i.e. Figure 6.1) to model the cardinality indicators of the relationships between MOBMAS' abstractions.

5. Rename the notational components of the Agent Behaviour Model Kind as "Agent Plan Template" and "Reflexive Rule Specification" instead of "Agent Plan" and "Reflexive Courses of Actions".

6. Use the term "authority relationship" to refer to the organisational relationships between roles (e.g. peer-to-peer relationship or superior-subordinate relationship) instead of the term "control regime" as used in the earlier version of MOBMAS.

7. Use the term "information sources" to refer to non-agent resources instead of "knowledge sources". The former term is more general than the latter because it encompasses resources such as databases and web servers that are not necessarily knowledge sources.

8. Use the term “belief conceptualisation” to refer to the conceptualisation of an agent class’ beliefs instead of the term “belief set”. The latter may be misunderstood as a set of beliefs rather than a conceptualisation of beliefs.

9. Revise step “Develop System Task Model” of the “Analysis” activity to delegate the identification of system-tasks to a separate Requirements Engineering effort (which is not part of MOBMAS) instead of identifying system-tasks from system goals as in the earlier version of MOBMAS. With this change, MOBMAS allows the developer to identify system-tasks from any constructs, thereby supporting the development of MASs that do not have clear goals.

10. Revise step “Analyse Organisational Context” of the “Analysis” activity to remove the use of the term “*real-world*” when referring to the structure of the MAS’ organisational context. This term may mislead the readers into thinking of only the “physical” world.

11. Refine the techniques for step “Develop Role Model” of the “Analysis” activity by:
    - making the heuristic “*strong internal coherence and loose coupling*” the primary criterion for grouping system-tasks to roles;
    - considering the mapping of system-tasks to roles when the system-tasks are fully and partially decomposed;
    - using the term “joint task” instead of “social task” to refer to a task that is collectively carried out by multiple parties. The term “joint task” helps to avoid any improper implications that may come with the term “social task”, e.g. the existence of subjective or intersubjective spaces; and
    - documenting the cardinality of the mapping relationship between system-tasks and role-tasks.

12. Revise Role Diagram and Agent Class Diagram to remove the use of guillemets in the label of each compartment in these diagrams. This removal helps to avoid any potential confusion with the UML stereotypes.

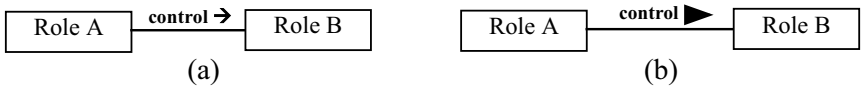
13. Revise step "Develop Ontology Model" of the "Analysis" activity to:
    - re-define an Application ontology as a *specialisation* of generic Domain ontologies and Task ontologies, not a composition of these two ontologies as inaccurately suggested in the earlier version of MOBMAS;
    - document the need for specifying cardinality of the relationships between ontological concepts in the Ontology Diagram;
    - provide examples for each type of ontological relationships; and
    - recommend the developer to study the extensive literature on "ontological mappings" for more options on the semantic correspondences between concepts besides the three basic correspondences – "equivalent", "subsumes" and "intersects" which are listed in MOBMAS.

14. Use UML "dependency relationship" to depict ontological mappings in the Ontology Diagram instead of UML "association relationship".

15. Refine step "Identify Ontology Management Role" of the "Analysis" activity to note that "ontology manager" agent classes may have been provided by the development platform as built-in components. These agent classes therefore do not have to be designed from scratch.

16. Extend step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity to add "modularity" and "non-functional requirements support" as additional factors to be considered when determining the organisational structure for the target MAS.

17. Change the notation of the superior-subordinate relationship between roles in the Role Diagram from Figure 7.3a to Figure 7.3b to improve its comprehensibility.



(a) (b)

Figure 7.3 – Old (a) and new (b) notation for superior-subordinate relationship between roles in Role Diagram

18. Revise step "Develop Agent Class Model" of the "MAS Organisation Design" activity to:
    - add "modularity" and "efficiency consideration" to the list of factors to be considered when assigning multiple roles to a single agent class; and
    - remove the consideration of the computation complexity of each agent class and the available processing power of each node (where the agent classes will be run) when assigning multiple roles to a single agent class. This removal is necessary because it is often unknown at the design time as to which hardware platform will implement the MAS system.

19. Modify Agent Class Diagram to model each agent class' dynamics as a property of the agent class instead of the property of the roles played by the agent class.

20. Modify Resource Diagram to remove the modelling of "communication properties" from the description of each resource, since these details (e.g. networking protocol and network address) are normally not available at the design time.

21. Revise step "Extend Ontology Model To Include Resource Application Ontologies" of the "MAS Organisation Design" activity to:
    - note that Resource Application ontologies can only be derived from the conceptual schema of the information stored in the resource, *not* directly corresponding to this conceptual schema; and
    - remove the guideline that, if a Resource Application ontology coincides with a MAS Application ontology, the MAS Application ontology can be used in place of the Resource Application ontology. Such a guideline will hinder the system's ability to accommodate future changes in the conceptual structure of the resource.

22. Extend step "Specify Agent Class' Belief Conceptualisation" of the "Agent Internal Design" activity to discuss the different ways by which the changes in an agent's belief conceptualisation can be propagated to other agents at run-time.

23. Revise step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity to rename the two major styles of agent behaviour as "planning" and "reflexive" instead of "proactive" and "reactive". This change helps to avoid any potential misunderstanding of the nature of these two styles. ("Proactive" and "reactive" may be understood as the two different modes of triggering of a piece of behaviour, while "planning" and "reflexive" refer to the two different levels of complexity in agent reasoning exhibited in a piece of behaviour).

24. Extend step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity and step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity to address the case when the datatypes of the variables/arguments are "basic" datatypes such as String and Integer. These datatypes do not need to be defined in a MAS Application ontology.

25. Revise the "Agent Interaction Design" activity to re-classify the two major mechanisms of agent interactions into "direct" and "indirect" mechanisms, instead of "direct" and "tuplespace/tuple-centre" mechanisms as in the earlier version of MOBMAS. The refined version of MOBMAS acknowledged that the tuplespace/tuple-centre interaction method is only one of many methods of indirect interaction, even though it is the most commonly used method.

26. Rename the last activity of MOBMAS as "Architecture Design" instead of "Deployment Design" as in the earlier version of MOBMAS. This new name describes more accurately the steps specified in this activity.

27. Refine step "Select Agent Architecture" of the "Architecture Design" activity by merging the criterion "*Size of knowledge base*" (which was suggested for consideration when selecting agent architecture for the target MAS) into criterion "*Support for scalability of agent*".

28. Introduce a new model kind, Resource Model Kind, which is depicted by the Resource Diagram, and extend the Architecture Model Kind to include Agent-Environment Interface Requirements Specification and Infrastructure Facilities Specification notational components. This modification removed the need for an

Environment Model Kind, which was originally defined to encompass the three notational components Resource Diagram, Agent-Environment Interface Requirements Specification and Infrastructure Facilities Specification.

29. Extend Agent Relationship Diagram to model agent classes' instantiation cardinality (via the use of annotations next to each agent class' name). This extension removed the need for a separate Agent Instantiation Diagram as recommended in the earlier version of MOBMAS.

30. Refine step "Identify agent-environment interface requirements" of the "Architecture Design" activity to note that, when interacting with a physical environment, the sensor and effector of an agent needs to able to connect to, and control, the drivers of hardware components such as thermostats, infra-red sensor, wheels and grippers.

7.4. FEATURE ANALYSIS OF MOBMAS

After MOBMAS was refined into its final version by Step 2 – "*Using MOBMAS on a test application*" (Section 7.3), Step 3 – "*Performing a feature analysis of MOBMAS*" was conducted on this final version. The objectives of this step were to (cf. Section 4.6):

- verify whether MOBMAS is able to achieve its goal, that is, to provide support for ontology-based MAS development and various other important AOSE methodological requirements that are documented in Section 5.4.3 (which include the required *features*, *AOSE steps* and *modelling concepts*);
- reveal the origin of MOBMAS' techniques and model definitions; and
- compare MOBMAS with the existing AOSE methodologies in term of specific evaluation criteria. These criteria had previously been used to evaluate the existing AOSE methodologies in Section 5.4. The comparison also highlights the various ontology-related strengths of MOBMAS, which are not provided, or provided to a lesser extent, by the existing AOSE methodologies.

Section 7.4.1 firstly examines the support of MOBMAS for its methodological requirements (including the support for ontology-based MAS development) and reports

on the origin of MOBMAS techniques and model definitions. Section 7.4.2 then compares MOBMAS with the existing AOSE methodologies.

7.4.1. MOBMAS' Support for Methodological Requirements

The methodological requirements of MOBMAS consist of the **features**, **steps** and **modelling concepts** that are desirable to the development process, techniques and model kinds of MOBMAS. These requirements have been shown in Tables 5.33, 5.34 and 5.35 of Chapter 5 respectively.

The clarification of MOBMAS' support for the required ***features*** is presented in Table 7.4. Feature "*Support for ontology-based MAS development*", in particular, is examined in detail in Section 7.4.1.1.

The support of MOBMAS for the required ***steps*** is documented in Table 7.5. The table shows the correspondences between MOBMAS' *actual* steps and the required AOSE steps (listed in Table 5.34). The various AOSE steps that were previously identified as desirable *ontology-related* steps (cf. Section 5.5) have been highlighted with an adornment *(O)*.

Table 7.6 lastly clarifies MOBMAS' support for the required *modelling concepts*. For each concept, the table shows the name of the model kind(s) of MOBMAS which represents or captures the concept.

In all three tables (Tables 7.4, 7.5 and 7.6), column "*Origins of MOBMAS techniques and modelling definitions*" provides an account of:

- the techniques and/or model definitions that have been **reused** by MOBMAS from the existing AOSE methodologies (i.e. "*REUSE*" subheading). The names of these source methodologies are shown under the "*Sources*" subheading. They are a subset of the "*potential sources of techniques and model definitions*" previously identified in Tables 5.33, 5.34 and 5.35 of Chapter 5;

- the **enhancements** that were made by MOBMAS on the reused techniques and/or model definitions (i.e. "*ENHANCEMENT*" subheading); and/or

- the **new** techniques and/or model definitions that were developed by MOBMAS and not found in the existing AOSE methodologies (i.e. "*NEW*" subheading). The need for some of these techniques and model definitions has previously been identified in Tables 5.33, 5.34 and 5.35 of Chapter 5.

Table 7.4 – MOBMAS' support for the required features (cf. Table 5.33)

Required features of MOBMAS development process	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
1. Specification of a system development lifecycle	MOBMAS adopts an iterative and incremental life cycle	**REUSE:** Allowing the development process to be iterative and incremental life cycle within and across all phases. *Sources: MASE, HLIM, PROMETHEUS, TROPOS and suggestions of survey's participants (Section 5.3).*
2. Support for verification and validation	MOBMAS performs consistency checking between many of its model kinds (e.g. between Agent Class Model Kind, Agent Behaviour Model Kind, Agent Interaction Model Kind and Ontology Model Kind)	**REUSE:** Supporting verification and validation via consistency checking of model kinds. *Sources: MASE, MASSIVE, INGENIAS, PROMETHEUS, PASSI, ADELFE and TROPOS.* **ENHANCEMENT:** Providing specific guidelines for the consistency checking between particular model kinds
3. Specification of steps for the development process	See Table 7.5	See Table 7.5
4. Specification of model kinds and/or notational components	See Table 7.6	See Table 7.6
5. Specification of techniques and heuristics for performing each process step and producing each model kind	See Table 7.5	See Table 7.5
6. Support for refinability	All models in MOBMAS are iteratively refined and expanded throughout of the system development process (e.g. Role Model, Agent Class Model, Environment Model and Ontology Model)	**REUSE:** Iteratively refining and expanding the different models of the methodology throughout different phases of the development process. *Sources: All existing methodologies except for SODA.*

Required features of MOBMAS model definitions	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
1. High degree of completeness/expressiveness	MOBMAS model kinds can represent the target system from both static aspects such as agent/role acquaintances, agent internal structure and system structure, and from dynamic aspects such as interaction protocols and agent behaviour. MOBMAS model kinds can also capture/represent a wide variety of concepts (cf. Table 7.3)	**REUSE:** Providing a comprehensive set of model kinds that represent the target system from both static aspects and dynamic aspects. The model kinds should also be able to capture/represent a wide variety of concepts. *Sources: MASE, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS and TROPOS.*
2. High degree of formalisation/preciseness	MOBMAS clearly defines the semantics and syntax of its notation. Examples are provided for all model kinds and notational components	**REUSE:** - Providing explanation of the semantics and syntax of the modelling notation. *Sources: MASE, GAIA, PROMETHEUS and TROPOS.* - Adopting UML and AUML notation which are familiar to most system developers. *Sources: PASSI and ADELFE.*
3. Provision of guidelines/logics for model derivation	MOBMAS provides techniques for transforming between its model kinds and notational components (e.g. from System Task Model Kind to Role Model Kind, from Role Model Kind and Ontology Model Kind to Agent Class Model Kind, and from Role Diagram and Resource Diagram to Agent Relationship Diagram)	**REUSE:** Allowing the developer to easily transform between the model kinds and notational components, or at least partially create a model kind or notational component from the information present in another. *Sources: All existing methodologies except CASSIOPEIA and MAS-CommonKADS.* **ENHANCEMENT:** Suggesting specific guidelines for the transformation between particular model kinds and notational components
4. Guarantee of consistency	MOBMAS provides rules for consistency checking among many of its model kinds (e.g. between Agent Behaviour Model Kind and Ontology Model Kind, between Agent Interaction Model Kind and Ontology Model Kind, between Agent Class Model Kind and Agent Behaviour Model Kind, and between System Task Model Kind, Role Model Kind and Ontology Model Kind)	**REUSE:** Documenting the need for the different model kinds and notational components which represent the same modelling concept(s) to be consistent with each other. *Sources: MASE, SODA, GAIA, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE and TROPOS.* **ENHANCEMENT:** Providing specific guidelines for the consistency checking between particular model kinds and notational components

Required features of MOBMAS model definitions	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
5. Support for modularity	MOBMAS models agent classes as entities that encapsulate roles, goals, agent plan templates, reflexive rule specifications and ontologies)	**REUSE:** - Modelling agent classes as entities that encapsulate roles. *Sources: MASE, MASSIVE, SODA, BDIM, GAIA, MESSAGE, INGENIAS and CASSIOPEIA.* - Modelling agent classes as entities that encapsulate goals. *Sources: TROPOS, MESSAGE, PASSI, HLIM, MEI and INGENIAS.* **NEW:** - Modelling agent classes' behaviour via modular agent plan templates and reflexive rule specifications - Modelling agent classes' belief conceptualisation as being composed of ontologies
6. Manageable number of concepts in each model kind and each notational component	Each MOBMAS model kind and notational component represents a manageable number of concepts	**REUSE:** Not capturing too many concepts in one single model kind. Adopting well-known notation. If new notation is used, it should have simple semantics and syntax. *Sources: All existing methodologies except SODA and INGENIAS.*
7. Model kinds expressed at various level of abstraction and detail	Many model kinds of MOBMAS can be developed at various levels of detail (e.g. Agent Class Model Kind, Role Model Kind and Ontology Model Kind)	**REUSE:** - Developing Role Model Kind at various levels of detail. *Sources: GAIA and MASE.* - Developing Agent Class Model Kind at various levels of detail. *Source: BDIM.* **NEW:** Developing Ontology Model Kind at various levels of detail
8. Support for reuse	MOBMAS allows the developer to reuse various modelling elements such as role patterns, protocol templates, organisational structures and ontologies	**REUSE:** Making it possible to reuse the following modelling elements: role patterns, protocol templates and organisational structure. *Sources: MASE, MASSIVE, GAIA, BDIM, PASSI and MAS-CommonKADS.* **NEW:** Promoting reusability with the use of ontology (cf. Section 7.4.2.4)

Agent properties required to be captured/represented by MOBMAS model kinds	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
1. Autonomy	MOBMAS models agent classes as entities with purposes (represented as agent roles and goals) and entities with internal control (represented as agent beliefs, plans and reflexive rules)	**REUSE:** - Modelling agent classes as entities with purposes, which are represented as agent roles and goals. *Sources: MASE, MASSIVE, SODA, GAIA, PROMETHEUS, PASSI, INGENIAS and ADELFE.* - Modelling agent classes as entities with internal control, which is captured via agent beliefs/knowledge and plans. *Sources: PROMETHEUS, MESSAGE, MAS-CommonKADS, TROPOS, BDIM, HLIM, MEI, COMOMAS and INGENIAS.*
2. Adaptability	MOBMAS discusses the possibility of agent's ontologies being modified and propagated to other agents, e.g. as a result of learning, and mentions the need to consider learning (amongst other required behaviour capabilities of an agent) when selecting an agent architecture	**REUSE:** Supporting adaptability by considering learning in selecting agent architecture. *Sources: MASSIVE.* **NEW:** Discussing the possibility of agent's ontologies being modified and propagated to other agents, e.g. as a result of learning.
3. Cooperative behaviour	MOBMAS supports the modelling of agent acquaintances and interaction protocols	**REUSE:** Capturing cooperative behaviour via agent acquaintances and interaction protocols. *Sources: All existing methodologies.* **NEW**: Supporting both direct interactions via ACL and indirect interactions via tuplespace/tuple-centre
4. Inferential capability	MOBMAS support s the modelling of agent plan templates and reflexive rules which help the agents to determine what to do to achieve its active goals at run-time	**REUSE:** Supporting inferential capability by developing agent plans. *Sources: PROMETHEUS, BDIM, HLIM, MEI and TROPOS.*
5. Knowledge-level communication ability	MOBMAS supports the modelling of ACL communication messages between agents, which are based upon speech-act performatives	**REUSE:** Modelling agents' exchanged messages using ACL speech-acts performatives. *Sources: BDIM, ADELFE, HLIM and MESSAGE.* **NEW:** Formulating agents' exchanged messages in terms of shared ontologies

Agent properties required to be captured/represented by MOBMAS model kinds	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
6. Reactivity	MOBMAS models events and agents' behaviour to react to these events	**REUSE:** - Modelling "events" that incur during agent interactions and in agent internal processing. *Sources: MASE, MESSAGE, INGENIAS, PASSI, ADELFE and MAS-CommonKADS.* - Developing agent behaviour to react to events. *Sources: PROMETHEUS, BDIM, HLIM, COMOMAS and TROPOS.*
7. Deliberative behaviour	MOBMAS models agent classes as entities with purposes (represented as agent goals) and supports the modelling of agent plans for accomplishing these purposes in a deliberative manner	**REUSE:** - Capturing deliberative behaviour via agents' goals. *Sources: BDIM and HLIM.* - Modelling agent plans to achieve the agent-goals. *Sources: BDIM, HLIM, TROPOS and PROMETHEUS.*
Required features of MOBMAS as a whole	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
1. Support for open systems	MOBMAS facilitates the operation of newly-added agents at run-time by explicitly modelling the resources offered by the target MAS environment and allowing the specification of a "resource broker" agent which brokers the available resources to the newly added agents. MOBMAS also allows for the use of indirect interaction mechanisms which are particularly suitable to open systems. MOBMAS also offers an option to conceptualize the agent interaction protocols during the design time. This explicit conceptualization will allow any new agents to join the pre-existing conversations, and allow the interaction protocols to change over time.	**REUSE:** Modelling resources of the MAS environment. *Sources: SODA and GAIA.* **NEW:** - Considering the design of a "*Resource Broker*" role/agent class - Considering the adoption of an indirect interaction mechanism, which is particularly suitable to open MASs - Considering system openness as a criterion when selecting the organisational structure style for target MAS
2. Support for dynamic systems	MOBMAS supports the dynamic role-playing behaviour of agents and offers model kinds to capture/represent this dynamics. MOBMAS also considers the issue of system dynamics when selecting the interaction mechanism for the target MAS.	**REUSE:** Supporting system dynamics by allowing for dynamic assignment of roles to agents. *Sources: MASSIVE, HLIM, MASE and PASSI.* **NEW:** Discussing the modelling of agent classes' dynamics in the Agent Class Model Kind and Agent Interaction Model Kind.

Required features of MOBMAS as a whole	**Support by MOBMAS**	**Origins of MOBMAS' techniques and model definitions**
3. Support for ontology-based MAS development	See Section 7.4.1.1	**REUSE:** - Addressing the modelling of MAS Application ontologies. *Sources: MAS-CommonKADS, PASSI, MESSAGE and MASE.* - Formulating agents' exchanged messages in accordance with a shared ontology. *Source: MASE and PASSI.* **NEW:** - Discussing the modelling of Resource Application ontologies - Considering the use of ontologies to verify other model kinds of MOBMAS, e.g. System Task Model Kind, Agent Behaviour Model Kind and Agent Interaction Model Kind - Modelling agents' belief conceptualisation as being composed of ontologies - Modelling Agent Plan Templates, Reflexive Rule Specification and exchanged messages in terms of concepts defined in ontologies - Discussing the mapping between ontologies - Specifying the steps when the developer should start constructing each type of ontology in the MAS development process, and providing some guidelines on the construction of the ontologies. *Source: suggestions of survey's participants in Chapter 7.*
4. Support for heterogeneous systems	MOBMAS deals with the modelling of non-agent resources apart from agent classes. The agent classes themselves are allowed to range from purely planning agents to purely reflexive agents, each with a different agent behavioural style.	**REUSE:** Supporting system heterogeneity by addressing the modelling of non-agent software components. *Sources: INGENIAS, PROMETHEUS, GAIA and MASSIVE.* **NEW:** - Discussing the modelling of Resource Application Ontologies to conceptualise the information and/or services provided by each resource - Addressing the design of both planning agents and reflexive agents

Table 7.5 – MOBMAS' support for the required steps (cf. Table 5.34)

Problem Domain Analysis steps	**Support by MOBMAS?**	**Origins of MOBMAS' techniques (for performing steps)**
1. Identify system functionality (O)	See step "Develop System Task Model" of the "Analysis" activity	**REUSE:** Incrementally decomposing system-tasks into sub-system-tasks. *Source: COMOMAS.* **NEW:** - Considering full versus partial decomposition of system-tasks - Considering conflicts between system-tasks - Using Ontology Model to validate System Task Model
2. Identify roles	See step "Develop Role Model" of the "Analysis" activity	**REUSE:** - Using system-tasks as inputs to identify roles. *Sources: MASSIVE and BDIM.* - Mapping a "joint" system-task to a group of roles. *Source: SODA.* - Applying the principle of "strong internal coherence and loose coupling" to group system-tasks to roles. *Sources: MASSIVE and PROMETHEUS.* - Grouping different system-tasks to one role if the tasks share a lot of common data or interact intensely with each other. *Source: PROMETHEUS.* - Assigning different system-tasks to different roles if they need to be executed on different processors and if there exist security and privacy requirements. - Analysing the structure of the MAS' organisational context (if applicable) to identify roles. *Source: GAIA.* **NEW:** - Considering the mapping of system-tasks to roles when the system-tasks are fully and partially decomposed. - Considering the identification of "Wrapper" roles, "Resource Broker" roles and "Ontology Manager" role.
3. Identify agent classes	See step "Develop Agent Class Model" of the "MAS Organisation Design" activity	**REUSE:** - Identifying agent classes from roles. *Sources: MASE, GAIA, MASSIVE, PASSI and suggestions of survey's participants in Section 5.3.* - Applying one-to-one mapping from roles to agent classes. *Sources: MASE and GAIA.* - Grouping roles to one agent class if roles interact intensely. *Sources: MASE and BDIM.* - Evaluating the coherence of each agent class by checking whether the agent class can be easily described by a single name without any conjunctions. *Source: PROMETHEUS.* **ENHANCEMENT:** Considering various modularity and efficiency issues in the assignment of roles to agent classes

Problem Domain Analysis steps	**Support by MOBMAS**	**Origins of MOBMAS' techniques (for performing steps)**
4. Model domain conceptualisation (O)	See step "Develop Ontology Model" of the "Analysis" activity	**REUSE:** Incrementally adding domain concepts as they arise throughout the MAS development process. *Source: MESSAGE.* **NEW:** Identify generic Domain ontologies and Task ontologies that can serve as inputs to the construction of MAS Application ontologies.
Agent Interaction Design steps	**Support by MOBMAS**	**Origins of MOBMAS' techniques (for performing steps)**
1. Specify acquaintances between agent classes	See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity	**REUSE:** Identifying agent acquaintances from acquaintances between roles. *Source: CASSIOPEIA.* **NEW:** Deriving agent acquaintances from communicative actions in Agent Plan Templates and Reflexive Rule Specifications.
2. Define interaction protocols	See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity	**REUSE:** - Specifying coordination rules to govern agents' interactions for achieving identical agent-goals. *Source: SODA.* - Allowing for reuse of interaction protocol templates from FIPA. *Sources: PROMETHEUS, MAS-CommonKADS and PASSI.* - Indicating the ontology used to govern each interaction protocol. *Source: PASSI.* **NEW:** Considering both direct interactions via ACL and indirect interaction mechanism via tuplespace/tuple-centre
3. Define content of exchanged messages (O)	See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity	**REUSE:** - Specifying speech-act performatives for each exchanged message. *Sources: PASSI and MAS-CommonKADS.* - Specifying arguments to be passed within each exchanged message. *Sources: MASE, PASSI and MAS-CommonKADS.* - Formulating exchanged messages using concepts defined in MAS Application ontologies. *Source: PASSI.* **ENHANCEMENT:** - Providing extensive techniques for the specification of exchanged messages in each interaction mechanism (i.e. direct interaction via ACL and indirect interaction via tuplespace/tuple-centre) - Specifying explicit rules to verify the content of the exchanged messages against the concepts defined in ontologies.

Agent Internal Design steps	**Support by MOBMAS**	**Origins of MOBMAS' techniques (for performing steps)**
1. Specify agent architecture	See step "Select Agent Architecture" of the "Architecture Design" activity	**REUSE:** Proposing a set of criteria for selecting agent architecture. *Source: MASSIVE.* **ENHANCEMENT:** Proposing a few additional criteria for selecting agent architecture that are not listed in MASSIVE, namely "required agent behavioural capabilities", "complexity of sensor input", "support for scalability" and "agent-environment interaction requirements". **NEW:** Listing various existing architectures from which the developer can consider for reuse
2. Define agent informational constructs (O)	See step "Specify Agent Class' Belief Conceptualisation" of the "Agent Internal Design" activity	**REUSE:** Discussing the differentiation between Belief conceptualisation and Belief States. *Source: BDIM.* **NEW:** - Defining agents' belief conceptualisation as being composed of ontologies - Providing detailed techniques for determining which ontologies each agent should commit
3. Define agent behavioural constructs (O)	See steps "Specify Agent Goals", "Specify Events" and "Develop Agent Behaviour Model" of the "Agent Internal Design" activity	**REUSE:** Developing agent plans to achieve agent-goals. *Source: BDIM.* **ENHANCEMENT:** - Providing guidelines to ensure consistency between Agent Plan Templates, Reflexive Rule Specification and Agent Belief Conceptualisation - Considering both planning behaviour and reflexive behaviour. **NEW:** Specifying the states and actions of Agent Plan Templates and Reflexive Rule Specification in accordance with the concepts defined in ontologies.
Overall System Design steps	**Support by MOBMAS**	**Origins of MOBMAS' techniques (for performing steps)**
1. Specify system architecture (i.e. overview of all system components & their connections)	See sub-step "Update Agent Class Model" of step "Specify resources" of the "MAS Organisation Design" activity	**REUSE:** Showing all system components and their connections. *Sources: INGENIAS, PROMETHEUS, PASSI and MAS-CommonKADS.*

Overall System Design steps	Support by MOBMAS	Origins of MOBMAS' techniques (for performing steps)
2. Specify organisational structure/inter-agent authority relationships	See step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity	**REUSE:** - Suggesting common styles of organisational structures which MAS may adopt, namely flat and hierarchical styles. *Source: MASSIVE.* - Suggesting the developer to investigate the structure of the MAS' organisational context when choosing the organisational structure for the target MAS. *Source: GAIA.* **ENHANCEMENT**: - Extend the common styles of MAS organisational structure to include federation and hybrid styles - Suggesting the developer to consider modularity issues, non-functional requirements and the number of roles in the system when choosing the organisational structure for the target MAS.
3. Model MAS environment (O)	See steps "Specify Resources" and "Extend Ontology Model to include Resource application ontologies" of the "MAS Organisation Design" activity, and step " "Specify MAS Infrastructure Facilities" of the "Architecture Design" activity	**REUSE** Modelling MAS environment by specifying resources offered by the environment. *Sources: SODA and GAIA.* **ENHANCEMENT:** Providing specific guidelines for the identification and modelling of resources **NEW:** - Modelling Resource Application ontology for each resource - Specifying infrastructure facilities for MAS
4. Specify agent-environment interaction mechanism	See step "Identify Agent-Environment Interface Requirements" of the "Architecture Design" activity	**REUSE:** Considering the use of conventional objects as sensors or effectors for agents. *Source: MEI.* **ENHANCEMENT:** Investigating all three different methods of agent-environment interaction: perception, effect and communication **NEW:** Suggesting various characteristics of the agents' perception, effect and communication that the developer should investigate when identifying the requirements of agent-environment interactions
5. Instantiate agent classes	See step "Instantiate Agent Classes" of the "Architecture Design" activity	**REUSE** Specifying cardinalities for agent classes' instantiation. *Source: GAIA.*
6. Specify agent instances deployment	See step "Develop MAS Deployment Diagram" of the "Architecture Design" activity	**REUSE:** Determining MAS deployment configuration by considering the communication traffic between agents, and the processing power available on particular machines or required by agents. *Source: MASE.* **ENHANCEMENT:** Proposing a list of configuration details that need to be specified for MAS deployment.

Table 7.6 – MOBMAS' support for the required modelling concepts (cf. Table 5.35)

Problem Domain concepts	**Support by MOBMAS**	**Origins of MOBMAS' modelling techniques and notation**
1. System functionality	System Task Model Kind	**REUSE:** - Showing a hierarchy of system-tasks. *Sources: MASSIVE, COMOMAS and MAS-CommonKADS.* - Notation for System Task Diagram. *Source: TROPOS.*
2. Role	Role Model Kind	**REUSE:** - Showing tasks of each role. *Sources: MASE, PASSI and GAIA.* - Model communication paths between roles. *Sources: MASE, GAIA, PASSI and INGENIAS.* **NEW:** Notation for Role Diagram (including the notation for "joint role-tasks" in Role Diagram)
3. Domain conceptualisation	Ontology Model Kind	**REUSE:** Representing ontological concepts as classes in UML Class Diagram, and ontological relations as relationships between UML classes. *Sources: MESSAGE, PASSI and MAS-CommonKADS.* **NEW:** Modelling mappings between ontologies
Agent concepts	**Support by MOBMAS**	**Origins of MOBMAS' modelling techniques and notation**
1. Agent-role assignment	Agent Class Model Kind	**REUSE:** - Showing roles of each agent class. *Sources: MASE, GAIA and MESSAGE.* - Modelling agent classes' dynamics in role playing behaviour. *Source: PASSI.* **NEW:** - Notation for Agent Class Diagram and Agent Relationship Diagram - Notation for the modelling of agent classes' dynamics in Agent Class Diagram
2. Agent goal/task	Agent Class Model Kind (Agent Class Diagram) and Agent Behaviour Model Kind (Agent Goal Diagram)	**REUSE:** - Modelling agent-goals as states that an agent class aims to achieve. Sources: INGENIAS and BDIM. - Notation for agent-goals. Source: TROPOS **NEW:** Showing a hierarchy of agent-goals for a particular agent class (if necessary) and the conflicts amongst these goals (if any)

Agent concepts	**Support by MOBMAS**	**Origins of MOBMAS' modelling techniques and notation**
3. Agent belief/knowledge	Agent Class Model Kind (Agent Class Diagram)	**REUSE:** Modelling belief conceptualisation for each agent class. *Source*: *BDIM* **NEW:** Modelling each agent class' belief conceptualisation in term of ontologies (which are in turn modelled in Ontology Model Kind)
4. Agent plan/reasoning rule/problem solving method	Agent Behaviour Model Kind	**REUSE:** - Modelling plans for agent classes. *Sources: BDIM, HLIM, MEI and TROPOS.* - Notation for Agent Plan Diagram. *Sources: BDIM and TROPOS.* - Specifying events and actions for each agent plan. *Sources: BDIM and PROMETHEUS.* **ENHANCEMENT**: Considering the modelling of both planning behaviour and reflexive behaviour for agent classes **NEW**: Notation for Agent Plan Template and Reflexive Rule Specification
5. Agent architecture	Architecture Model Kind (Agent Architecture Diagram)	**REUSE:** Showing the logical modules of agent architecture and potential flows of information between the modules. *Source: MASE*. **NEW**: Notation for Agent Architecture Diagram
Agent Interaction concepts	**Support by MOBMAS**	**Origins of MOBMAS' modelling techniques and notation**
1. Agent acquaintance	Agent Class Model Kind (Agent Relationship Diagram)	**REUSE:** Representing agent acquaintances as connections between agent classes in Agent Relationship Diagram. *Sources: MASE, GAIA, MESSAGE and PASSI.* **NEW:** Showing descriptive information about each agent acquaintance (e.g. interaction protocols and ontology that govern the acquaintance)
2. Interaction protocol	Agent Interaction Model Kind	**REUSE:** Using AUML Interaction Diagrams for the modelling of interaction protocols in "direct interaction mechanism". *Sources: MESSAGE, PROMETHEUS and PASSI*. **NEW:** - Adapting AUML Interaction Diagrams for the modelling of interactions between agents and a shared tuplespace/tuple-centre in the "indirect interaction mechanism" - Modelling the behaviour of tuple-centre in the "indirect interaction mechanism" using UML State Chart

Agent Interaction concepts	**Supported by MOBMAS?**	**Origins of MOBMAS' modelling techniques and notation**
3. Content of exchanged messages	Agent Interaction Model Kind	**REUSE:** Specifying speech-act performative for each exchanged message. *Sources: MESSAGE, TROPOS and MAS-CommonKADS.* **ENHANCEMENT:** Providing detailed techniques for modelling the content of each exchanged message (i.e. performatives, variables, data types) for each coordination mechanism (i.e. Tuple Centre Coordination Mechanism or Interaction-Protocol Coordination Mechanism)
Overall System Design concepts	**Supported by MOBMAS?**	**Origins of MOBMAS' modelling techniques and notation**
1. System architecture	Agent Class Model Kind (Agent Relationship Diagram)	**REUSE:** Showing an overview of all agent classes and resources in Agent Relationship Diagram. *Sources: PROMETHEUS, PASSI, INGENIAS and MAS-CommonKADS.* **ENHANCEMENT:** Showing "wrapping" relationships between agent classes and resources in Agent Relationship Diagram
2. Organisational structure/inter-agent authority relationship	Role Model Kind	**REUSE:** Showing authority relationships between roles in Role Diagram. *Sources: GAIA, MESSAGE, INGENIAS and HLIM.* **NEW:** Adapting UML dependency relationship for the modelling of inter-role authority relationships
3. Environment resource/facility	Resource Model Kind	**REUSE:** Modelling resources and their relationships with "wrapper" agent classes. *Source: INGENIAS.* **NEW:** - Notation for Resource Diagram - Allowing the Resource Diagram to be adaptable to the target MAS development project (i.e. the configuration dimensions can be changed) - Modelling Resource Application ontology for each resource in Ontology Model Kind - Modelling MAS infrastructure facilities in "Infrastructure Facility Specification"
4. Agent instantiation	Agent Class Model Kind (Agent Relationship Diagram)	**REUSE:** Showing instantiation cardinality for each agent class *Sources: MASE, GAIA and PROMETHEUS.* **NEW:** Notation for modelling agent instantiation in Agent Relationship Diagram
5. Agent instance deployment	Architecture Model Kind (MAS Deployment Diagram)	**REUSE:** - Showing locations of agents. *Source: MASE.* - Show processing nodes, agents at each node and connections between nodes/agents. *Source: PASSI.* **ENHANCEMENT:** Adopting UML Deployment Diagram for the modelling of agent instance deployment.

7.4.1.1. MOBMAS' support for ontology-based MAS development

MOBMAS offers extensive support for ontology-based MAS development, by using ontologies in various steps of the MAS development **process** and integrating ontologies into the model definitions of various **model kinds**.

Regarding the MAS *development process*, a large proportion of MOBMAS' steps are ontology-related. In fact, these MOBMAS' ontology-related steps correspond to those *desirable ontology-related AOSE steps* that were previously recommended for MOBMAS in Section 5.5. The following discussion recapitulates these desirable ontology-related AOSE steps (c.f. Section 5.5) and reveals their correspondences with MOBMAS' actual steps.

1. "*Identify system functionality*": This generic AOSE step is supported by MOBMAS via step "*Develop System Task Model*" (c.f. Section 6.2.1). Even though MOBMAS does not cover the process of system tasks elicitation, it suggests the developer to validate and refine the identified system tasks by examining the application ontologies captured in the Ontology Model (c.f. Section 6.2.4.1.c).

2. "*Model domain conceptualisation*": This desirable AOSE step corresponds to MOBMAS' step "*Develop Ontology Model*" (c.f. Section 6.2.4). This step produces an Ontology Model to capture all of the application ontologies of the target MAS and the semantic mappings between these ontologies.

3. "*Define content of exchanged messages*": This AOSE step is performed as part of step "*Develop Agent Interaction Model*" in MOBMAS (c.f. Section 6.5.2). MOBMAS uses ontological concepts to formulate the content of the exchanged messages (particularly the datatypes of the exchanged variables), and requires the developer to validate the formulated messages against the ontologies in the Ontology Model and vice versa.

4. "*Define agent information constructs*": This desirable AOSE step corresponds to MOBMAS' step "*Specify agent class' belief conceptualisation*" (c.f. Section 6.4.1). Ontologies are used by this step as the building blocks for modelling agents' conceptual beliefs.

5. "*Define agent behavioural constructs*": This generic AOSE step is supported via three MOBMAS' steps: "*Specify agent goals*", "*Specify events*" and "*Develop Agent Behaviour Model*" (c.f. Sections 6.4.2, 6.4.3 and 6.4.4). All of these steps refer to the concepts defined in the application ontologies whenever appropriate to define agents' goals, plans, reflexive rules and actions. Ontologies are also used to help identify and validate the agents' actions.

6. "*Model MAS environment*": This AOSE step is covered via three steps in MOBMAS: "*Specify resources*", "*Extend Ontology Model to include Resource application ontologies*" and "*Specify MAS infrastructure facilities*" (c.f. Sections 6.3.3, 6.3.4 and 6.3.3). The former two[82] involve identifying the ontologies which conceptualise each resource of MAS, and updating the Ontology Model to include these ontologies (together with their mappings).

As can be seen above, all desirable ontology-related steps that were identified in Section 5.5 are supported by MOBMAS. Consequently, it can be said that MOBMAS is capable of realising all the benefits of ontology in MAS design and operation as listed in Section 2.3.2. These capabilities will be further confirmed in Section 7.4.2.4. In addition, MOBMAS also addresses how the MAS development process can assist in the engineering of ontology. Specifically, the investigation of system's functionality, agent goals, plans, reflexive rules, actions and exchanged messages helps to identify and validate the concepts to be included in ontologies.

Regarding its *model definitions*, MOBMAS integrates ontologies into five of its nine model kinds, namely:

- *Ontology Model Kind*;
- *Agent Class Model Kind*;
- *Resource Model Kind*;
- *Agent Behaviour Model Kind*; and
- *Agent Interaction Model Kind.*

These model kinds are the direct products of the previously listed MOBMAS' ontology-related steps.

---

[82] Note that step "*Specify MAS infrastructure facilities*" does not need to involve ontologies.

7.4.2. Comparison of MOBMAS and Existing AOSE Methodologies

This section documents the comparison between MOBMAS and the existing AOSE methodologies, which uses the same evaluation framework as that used in the feature analysis of the existing AOSE methodologies in Section 5.4. This evaluation framework consists of 38 criteria (cf. Section 5.4.1), namely:

- 36 evaluation criteria on *features* (Table 5.21);
- one criterion on *steps* (Table 5.22); and
- one criterion on *modelling concepts* (Table 5.23).

The comparison between MOBMAS and the existing methodologies in term of these evaluation criteria is presented in Sections 7.4.2.1, 7.4.2.2 and 7.4.2.3. It should be noted that, some of the criteria actually deal with features, steps and modelling concepts that are *not* required from MOBMAS (i.e. those identified as "potential requirements" of MOBMAS but which were not eligible to become MOBMAS' "actual requirements" in Section 5.4.3). Consequently, MOBMAS' support for these criteria may be weak. These criteria are annotated with symbol "#" in Tables 7.4, 7.5, 7.6 and 7.7.

Section 7.4.2.4 finally provides an account of the important ontology-related strengths of MOBMAS. These strengths are either not provided, or provided to a lesser extent, by existing AOSE methodologies due to their lack, or low level, of support for ontology.

7.4.2.1. Comparison of support for Features

- **Comparison of support for features relating to AOSE process:**
  Of the fourteen criteria in this category (Table 5.21), only eight criteria are examined in this section (Table 7.7). The comparison of the remaining six criteria[83] will be discussed in Section 7.4.2.2 alongside criterion "*Support for steps*", because these six criteria need to use the list of AOSE steps presented in Table 5.22 as a common yardstick.

[83] That is, criteria "*Specification of model kinds and/or notational components*", "*Definition of inputs and outputs for steps*", "*Specification of techniques and heuristics*", "*Ease of understanding of techniques*", "*Usability of techniques*" and "*Provision of examples for techniques*".

Justification of MOBMAS' support for the criteria "*Specification of a system development lifecycle*", "*Support for verification and validation*", "*Specification of steps for the development process*" and "*Support for refinability*" of Table 7.7 can be found in Table 7.4. For criteria "*Ease of understanding of the development process*" and "*Usability of the development process*", the evaluation of MOBMAS was obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G). The assessment of the first developer is denoted as "*D1*", while that of the second developer is labelled with "*D2*". For criterion "*Approach for MAS development*", MOBMAS is Role-Oriented (denoted as "*RO*") because it uses "role" as the building block for defining agent classes.

- **Comparison of support for features relating to AOSE model definitions:**
  Nine evaluation criteria were used to conduct this comparison (Table 5.21). The comparison results are presented in Table 7.. The support provided by MOBMAS for each criterion has been justified in Table 7.4, except for criterion "*Ease of understanding of model definitions*", whose evaluation was obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G).

- **Comparison of support for agent properties:**
  Nine agent properties were investigated in total (Table 5.21). The comparison between MOBMAS and the existing methodologies in term of the support for these properties is shown in Table 7.9. The justification for MOBMAS' support can be found in Table 7.4.

- **Comparison of support for features relating to the methodology as a whole**:
  There are six high-level, supplementary features that pertain to a MAS development methodology as a whole. The comparison between MOBMAS and the existing methodologies with regard to these features is presented in Table 7.10. The justification for MOBMAS' evaluation can be found in Table 7.4. For criterion "*Support for agility and robustness*", MOBMAS was given a "*possibly*" evaluation because it implicitly models the exceptional situations and the exception-handling

behaviour of agents, through the specification of interaction protocols in the "*Agent Interaction Design*" activity (Section 6.5).

Table 7.7 – Comparison of support for features relating to AOSE process

	Specification of a system development (dev.) lifecycle	Support for verification & validation	Specification of steps for the dev. process	Support for refinability	Ease of understanding of the dev. process	Usability of the dev. process	Approach for MAS dev.
MASE	Iterative across all phases	Yes	Yes	Yes	High	High	RO
MASSIVE	Iterative View Engineering process	Yes	Yes	Yes	High	Medium	RO
SODA	Not specified	No	Yes	No	High	Low	RO
GAIA	Iterative within each phase but sequential between phases	No	Yes	Yes	High	Medium	RO
MESSAGE	Rational Unified Process	Mentioned as future enhancement	Yes	Yes	High	Medium	RO
INGENIAS	Unified software development process	Yes	Yes	Yes	High	High	RO
BDIM	Not specified	No	Yes	Yes	High	Low	RO
HLIM	Iterative within and across the phases	No	Yes	Yes	High	High	RO
MEI	Not specified	No	Yes	Yes	High	High	NRO
PROMETHEUS	Iterative across all phases	Yes	Yes	Yes	High	High	NRO
PASSI	Iterative across and within all phases	Yes	Yes	Yes	High	High	RO
ADELFE	Rational Unified Process	Yes	Yes	Yes	High	High	NRO
COMOMAS	Not specified	No	Yes	Yes	High	Medium	NRO
MAS-CommonKADS	Cyclic risk-driven process	Mentioned but no clear guidelines	Yes	Yes	High	Medium	NRO
CASSIOPEIA	Not specified	No	Yes	Yes	High	Medium	RO
TROPOS	Iterative and incremental	Yes	Yes	Yes	High	Medium	NRO
MOBMAS	Iterative and incremental	Yes	Yes	Yes	D1: High D2: High	D1: High D2: High	RO

Table 7.8 – Comparison of support for features relating to AOSE model definitions

	Completeness/ expressiveness	Formalization/ preciseness	Provision of guidelines/ logics for model derivation	Guarantee of consistency	Support for modularity	Management of complexity	Levels of abstraction	Support for reuse	Ease of understanding of model definitions
MASE	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Yes	High
MASSIVE	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Yes	High
SODA	Medium	a. Medium b. Yes	Yes	a. Yes b. Yes	Yes	Yes	No	Possibly	High
GAIA	Medium	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Yes	High
MESSAGE	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Possibly	High
INGENIAS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	No	Yes	Possibly	Medium
BDIM	Medium	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Yes	High
HLIM	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
MEI	Medium	a. Low b. Yes	Yes	a. No b. Yes	Yes	Yes	Yes	Possibly	Medium
PROMETHEUS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
PASSI	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	No	Yes	High
ADELFE	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
COMOMAS	High	a. High b. Yes	Yes	a. No b. Yes	Yes	Yes	No	Possibly	High
MAS-CommonKADS	High	a. Medium b. Yes	No	a. No b. Yes	Yes	Yes	Yes	Yes	Medium
CASSIOPEIA	Medium	a. High b. Yes	NA	a. No b. NA	Yes	Yes	No	Possibly	High
TROPOS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Possibly	High
MOBMAS	High	a. High b. Yes	Yes	a. Yes b. Yes	Yes	Yes	Yes	Yes	D1: Medium D2: High

Table 7.9 – Comparison of support for agent properties

	Autonomy	Adaptability	Cooperative behaviour	Inferential capability	Knowledge-level communication ability	Personality#	Reactivity	Deliberative behaviour
MASE	Yes	No	Yes	Possibly	Yes	No	Yes	Yes
MASSIVE	Yes	Yes	Yes	No	No	No	Possibly	Yes
SODA	Yes	No	Yes	No	No	No	Possibly	Yes
GAIA	Yes	No	Yes	No	No	No	Possibly	Yes
MESSAGE	Yes	No	Yes	Yes	Yes	No	Yes	Yes
INGENIAS	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
BDIM	Yes	No	Yes	Yes	Yes	No	Yes	Yes
HLIM	Yes	No	Yes	Yes	Yes	No	Yes	Yes
MEI	Yes	No	Yes	Yes	No	No	Possibly	Yes
PROMETHEUS	Yes	No	Yes	Yes	Yes	No	Yes	Yes
PASSI	Yes	No	Yes	Yes	Yes	No	Yes	Yes
ADELFE	Yes	No	Yes	Yes	Yes	No	Yes	Yes
COMOMAS	Yes	No	Yes	Yes	No	No	Yes	Yes
MAS-CommonKADS	Yes	No	Yes	Yes	Yes	No	Yes	Yes
CASSIOPEIA	Yes	No	Yes	No	No	No	Possibly	No
TROPOS	Yes	No	Yes	Yes	Yes	No	Yes	Yes
MOBMAS	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes

Table 7.10 – Comparison of support for features relating to the methodology as a whole

	Support for open systems	Support for dynamic systems	Support for agility & robustness#	Support for heterogeneous systems	Support for mobile agents#	Support for ontology-based MAS development
MASE	No	Possibly	Yes	Yes	No	Yes
MASSIVE	Possibly	Yes	Yes	Yes	No	No
SODA	Yes	No	No	No	No	No
GAIA	Yes	No	No	Yes	No	No
MESSAGE	No	No	No	No	No	Yes
INGENIAS	No	No	No	Yes	No	No
BDIM	No	No	No	No	No	No
HLIM	No	Yes	No	No	No	No
MEI	No	No	No	No	No	No
PROMETHEUS	No	No	Yes	Yes	No	No
PASSI	No	Yes	No	No	Yes	Yes
ADELFE	Yes	No	Yes	No	No	No
COMOMAS	No	No	No	No	No	Yes
MAS-CommonKADS	No	No	Yes	No	No	No
CASSIOPEIA	No	Yes	No	No	No	No
TROPOS	No	No	No	No	No	No
MOBMAS	Yes	Yes	Possibly	Yes	No	Yes

7.4.2.2. Comparison of support for Steps

The assessment of MOBMAS with regard to the criterion "*Support for steps*" (Table 5.22) is presented in Table 7.11. This table also documents the evaluation of MOBMAS with regard to six other criteria which relate to the AOSE process but had not been examined in Section 7.4.2.1, namely:

- "*Specification of model kinds and/or notational components*";
- "*Definition of inputs and outputs of steps*";
- "*Specification of techniques and heuristics*";
- "*Ease of understanding of techniques*";
- "*Usability of techniques*"; and
- "*Provision of examples for techniques*".

All of these criteria use the list of AOSE steps presented in Table 5.22 as yardstick. This list had previously been used in the feature analysis of the existing methodologies in Section 5.4.2 and Appendix D. The meaning of all abbreviations in Table 7.11 is the same as that in Tables D.1 to D.14 in Appendix D. It should be noted that, the assessment of MOBMAS' support for the criteria "*Ease of understanding of techniques*" and "*Usability of techniques*" were obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G). The assessment of each developer is denoted as "*D1*" and "*D2*" respectively.

To compare MOBMAS with the existing 16 AOSE methodologies, readers are referred to Tables AppendixD.1 to AppendixD.16 in Appendix D, where the individual assessment of each existing methodology is documented. Since the evaluation findings of all methodologies are presented in the same format, a direct comparison between them can easily be made. A bird-eye view of the comparison between all methodologies is shown in Table 7.12. This table shows only the assessment of the criterion "*Usability of techniques"*.

Table 7.11 – MOBMAS' support for steps

MOBMAS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	System Task Model Kind	B	See Section 6.2.1 – "Develop System Task Model"	See Section 6.2.1.1 – "Notation of System Task Diagram"	D1: H D2: M	D1: H D2: H	Y
2. Specify use case scenarios #								
3. Identify roles	E	Role Model Kind	B	See Section 6.2.3.1 – "Identify roles" and Section 6.2.3.2 – "Specify role-tasks"	See Section 6.2.3.3 – "Notation of Role Diagram"	D1: H D2: M	D1: H D2: H	Y
4. Identify agent classes	E	Agent Class Model Kind	B	See Section 6.3.2.1 – "Identify agent classes"	See Section 6.3.2.2 – "Notation of Agent Class Model Kind"	D1: H D2: H	D1: H D2: M	Y
5. Model domain conceptualisation	E	Ontology Model Kind	B	See Section 6.2.4.1 – "Develop MAS Application ontologies"	See Section 6.2.4.2 – "Language of Ontology Model Kind"	D1: M D2: H	D1: M D2: H	Y
6. Specify acquaintances between agent classes	E	Agent Class Model Kind (Agent Relationship Diagram)	B	See Sections 6.5.2.1.c and 6.5.2.2.c – "Update Agent Class Model and Role Model"	See Section 6.3.2.2 – "Notation of Agent Class Model"	D1: H D2: H	D1: H D2: H	Y
7. Define interaction protocols	E	Agent Interaction Model Kind	B	See Section 6.5.2 – "Develop Agent Interaction Model"	See Section 6.5.2 – "Develop Agent Interaction Model"	D1: H D2: H	D1: H D2: H	Y
8. Define content of exchanged messages	E[7]	Agent Interaction Model Kind	B	See Section 6.5.2 – "Develop Agent Interaction Model"	See Section 6.5.2 – "Develop Agent Interaction Model"	D1: H D2: H	D1: H D2: H	Y
9. Specify agent communication language#	I[8]			Assume the use of any ACL that use speech-act performatives (such as FIPA-ACL or KQML)				
10.Specify agent architecture	E	Architecture Model Kind (Agent Architecture Diagram)	B	See Section 6.6.2.1 – "Select agent architecture"	See Section 6.6.2.2 – "Develop Agent Architecture Diagram"	D1: H D2: H	D1: H D2: H	Y
11.Define agent informational constructs (i.e. beliefs)	E	Agent Class Model Kind (Agent Class Diagram); Ontology Model Kind	B	See Section 6.4.1.1 – "Specify Belief Conceptualisation of Agent Classes"	See Section 6.4.1.2 – "Update Agent Class Model To Show Belief Conceptualisation"	D1: M D2: H	D1: H D2: H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	Agent Behaviour Model Kind; Agent Class Model Kind (Agent Class Diagram)	B	See Section 6.4.2 – "Specify agent-goals", Section 6.4.3 – "Specify events" and Section 6.4.4 – "Develop Agent Behaviour Model"	See Section 6.4.2 – "Specify agent-goals", Section 6.4.3 – "Specify events" and Section 6.4.4 – "Develop Agent Behaviour Model"	D1: H D2: M	D1: M D2: H	Y
13.Specify system architecture (i.e. overview of all components and their connections)	E	Agent Class Model Kin d (Agent Relationship Diagram)	B	See Section 6.3.3.4 – "Update Agent Class Model"	See Section 6.3.3.4 – "Update Agent Class Model"	D1: H D2: H	D1: H D2: H	Y
14.Specify organisational structure/inter-agent authority relationships	E	Role Model Kind	B	See Section 6.3.1.1. – "Determine MAS Organisational Structure"	See Section 6.3.1.2 – "Update Role Model"	D1: H D2: H	D1: M D2: H	Y
15.Model MAS environment	E	Resource Model Kind	B	See Section 6.3.3 – "Specify resources"	See Section 6.3.3.2 – "Notation of Resource Diagram"	D1: H D2: H	D1: H D2: H	Y
16.Specify agent-environment interaction mechanism	E	Architecture Model Kind (Agent-Environment Interface Requirements Specification)	B	See Section 6.6.1 – "Identify agent-environment interface requirements" and Section 6.6.3 – "Specify MAS infrastructure facilities"	See Section 6.6.1 – "Identify agent-environment interface requirements" and Section 6.6.3 – "Specify MAS infrastructure facilities"	D1: M D2: H	D1: M D2: H	Y
17.Specify agent inheritance and aggregation#								
18.Instantiate agent classes	E	Agent Class Model Kind (Agent Relationship Diagram)	B	See Section 6.6.4 – "Instantiate agent classes"	See Section 6.6.4 – "Instantiate agent classes"	D1: H D2: H	D1: H D2: H	Y
19.Specify agent instances deployment	E	Architecture Model Kind (MAS Deployment Diagram)	B	See Section 6.6.5 – "Develop MAS Deployment Diagram"	See Section 6.6.5 – "Develop MAS Deployment Diagram"	D1: H D2: H	D1: M D2: H	Y

Table 7.12 – Comparison re criterion "Usability of techniques"

	1. Identify system functionality	2. Specify use case scenarios#	3. Identify roles	4. Identify agent classes	5. Model domain conceptualisation	6. Specify acquaintances between agent classes	7. Define interaction protocols	8. Define content of exchanged messages	9. Specify agent communication language#	10. Specify agent architecture	11. Define agent mental attitudes	12. Define agent behavioural interface	13. Specify system architecture	14. Specify organisational structure/inter-agent social relationships	15. Model MAS environment	16. Specify agent-environment interaction mechanism	17. Specify agent inheritance & aggregation#	18. Instantiate agent classes	19. Specify agent instances deployment
MASE	H	H	H	H	H	H	H	H		M	M							H	H
MASSIVE	H		H	L		L	H			H			M	H	H	M			
SODA	L		M	M			H								M				
GAIA	M		H	H		M	H					H		H	M		H	H	
MESSAGE	H		M	M	M	M	H	L	M	H	H		H	H	L			L	
INGENIAS	H	H	M	H		H	H	M			H		H	H	H	H			
BDIM			L	H		L		L			H	M					H	M	
HLIM	H	H	M	M		M	H	M			H	L		H					
MEI	H	H		H		H	H				M					H			
PROME-THEUS	H	H		H		H	H	L		H	H	H	H		M	H		L	
PASSI	H	H	H	M	M	H	H	H		H	M	H	H						L
ADELFE	H	H		H		M	M	M	M	H	H	M	H		H	L	M		
COMOMAS	M			M			L				M			L					
MAS-COMMON AKDS	H	H		M	M	H	H	H		L	M	L	M	L	M		M	L	
CASSIO-PEIA	H		M	M		H	L							M					
TROPOS	H			H		M	H	M			M			H	H				
MOBMAS	H H		H H	H M	M H	H H	H H	H H		H H	H H	M H	H H	M H	H H	M H		H H	M H

7.4.2.3. Comparison of support for Modelling Concepts

In this section, MOBMAS is compared with the existing AOSE methodologies in term of the criterion "*Support for concepts*" (Table 7.13). This criterion uses the list of AOSE modelling concepts presented in Table 5.23 as yardsticks. This list had previously been used in the feature analysis of the existing methodologies in Section 5.4 and Appendix D.

In Table 7.13, if a methodology provides support for a particular modelling concept, this support is represented by a tick✓. Readers are referred to Tables 5.29a and 5.29b for the names of the model kinds and/or notational components that model the concepts in each existing methodology. For MOBMAS, this information can be found in Table 7.6.

Table 7.13 – Comparison of support for modelling concepts

	1. System functionality	2. Use case scenario#	3. Role	4. Domain conceptualisation	5. Agent-role assignment	6. Agent goal/task	7. Agent belief/knowledge	8. Agent plan/reasoning rule/problem solving method	9. Agent capability/service#	10. Agent percept/event#	11. Agent architecture	12. Agent acquaintance	13. Interaction protocol	14. Content of exchanged messages	15. System architecture	16. Organisational structure/inter-agent authority relationship	17. Environment resource/facility	18. Agent aggregation relationship#	19. Agent inheritance relationship#	20. Agent instantiation	21. Agent instance deployment
MASE	✓	✓	✓	✓	✓			✓			✓	✓	✓	✓						✓	✓
MASSIVE	✓		✓		✓					✓	✓	✓	✓		✓	✓					
SODA	✓		✓		✓							✓	✓				✓			✓	✓
GAIA	✓		✓		✓				✓			✓	✓			✓	✓	✓		✓	
MESSAGE	✓		✓	✓	✓							✓	✓	✓	✓	✓	✓				
INGENIAS	✓	✓	✓		✓	✓	✓	✓				✓	✓	✓	✓	✓	✓				
BDIM						✓	✓	✓				✓		✓				✓	✓	✓	
HLIM	✓	✓	✓			✓	✓	✓	✓			✓	✓	✓		✓					
MEI	✓	✓				✓		✓					✓								
PROMETHEUS	✓	✓				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	
PASSI	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓						✓
ADELFE	✓	✓				✓	✓		✓		✓	✓	✓	✓	✓		✓	✓			
CASSIOPEIA			✓		✓							✓									
COMOMAS	✓					✓	✓						✓			✓					
MAS-COMMONKADS	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
TROPOS	✓					✓	✓	✓			✓	✓	✓	✓		✓					
MOBMAS	✓		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓	✓

7.4.2.4. Ontology-related strengths of MOBMAS

By using ontologies in the MAS development process and integrating ontologies into the MAS model definitions, MOBMAS is able to exploit ontologies to enhance its *MAS development process* and *MAS development product*[84] with many important ontology-related strengths. These strengths are either not provided, or provided to a lesser extent, by the existing AOSE methodologies due to their lack, or low level, of support for ontology.

The following sections identify and justify the various ontology-related strengths of MOBMAS, organised into *process*-related strengths and *product*-related strengths. These strengths include those ontology's benefits that are identified in Section 2.3.2, and those additional benefits provided by MOBMAS.

Ontology-related strengths of MOBMAS *development process*

- *Highly reliable system analysis* (cf. Section 2.3.2.3): By acknowledging that an effective ontology analysis would facilitate the understanding of a particular domain, MOBMAS recommends using the ontological analysis effort of knowledge engineers to facilitate and validate the system analysis effort of system developers. In particular, ontologies can be used to help identify and validate the identification of system tasks in the System Task Model (cf. Section 6.2.4.1.c). In all other existing AOSE methodologies, the system analysis process is conducted solely by the system developer himself and not supplemented by any other analysis effort.

- *Effective modelling of application domain*: As discussed in Section 2.3.2.3, ontology provides a structured, human-readable and reusable representation mechanism for modelling application domain of a given MAS system. MOBMAS accordingly recommends the developer to use ontologies as the modelling mechanism for application domains (i.e. "MAS application ontologies"; cf. Section 6.2.4.1). Amongst the existing 16 AOSE methodologies, only MAS-CommonKADS, MESSAGE, MASE and PASSI exploit ontologies for application domain modelling (cf. Section 3.3.2).

[84] "Product" refers to the final MAS system whose design is developed by MOBMAS.

- *Well-structured, modular modelling of agents' local knowledge* (cf. Section 2.3.2.3): In MOBMAS, ontologies serve as the building blocks for defining agents' conceptual knowledge (cf. Section 6.4.1). This ontology-based modelling mechanism results in agent knowledge models that are much more structured and modular than those produced by the existing AOSE methodologies. The latter do not organise agents' knowledge into any modular conceptual structures.

- *Systematic modelling of agent local knowledge*: If an agent wishes to hold beliefs about a particular domain or resource, the local knowledge of that class should contain the corresponding ontology. By implementing this modelling mechanism, MOBMAS helps the developer to systematically and effectively design the knowledge model for each agent class (cf. Section 6.4.1). The process of agent knowledge modelling in the other existing AOSE methodologies is not as simple and effective, because these methodologies identify agent knowledge in a "bit-by-bit" manner, e.g. by investigating each agent goal, plan, interaction and/or use cases (e.g. BDIM, MESSAGE, INGENIAS, HLIM, COMOMAS and MAS-CommonKADS; cf. Appendix D).

- *Reliable specification of agents' behaviour*: While the existing AOSE methodologies only examine constructs such as agent roles, goals, interactions and use cases to identify agents' potential actions (e.g. GAIA, BDIM, HLIM, PROMETHEUS and MAS-CommonKADS; cf. Appendix D), MOBMAS also examines the ontologies committed by each agent (cf. Section 6.4.4). By using ontologies as an additional input, the developer may uncover actions that would otherwise be missed if he only investigates agent roles, goals, interactions and use cases.

- *Extensive verification and validation*: An ontology development effort is closely related and supplementary to a MAS development effort, because both involve a detailed investigation of the target application. As noted in Section 2.3.2.3, a strong ontological analysis leads to a more complete and accurate understanding of the target application. Accordingly, MOBMAS recommends that the developer should

exploit application ontologies to verify and validate the correctness and completeness of its MAS analysis and design models, namely System Task Model, Role Model, Agent Behaviour Model and Agent Interaction Model (cf. Sections 6.2.4.1.c, 6.4.4.1, 6.4.4.3 and 6.4.2.3). Since application ontologies are often constructed by a separate development team (e.g. domain experts or knowledge engineers), they can serve as a reliable tool for verification and validation. When examining the few existing AOSE methodologies that offer support for verification and validation (e.g. MASE, INGENIAS, PASSI, PROMETHEUS and TROPOS; cf. Table 5.24), it was found that each methodology simply uses its MAS analysis and design models to verify and validate against themselves. This mechanism of verification and validation is undoubtedly less reliable than the use of a separately-developed ontology model as seen in MOBMAS.

- *Support for (distributed) team-based development*: Sharing an ontology is basically sharing the same conceptual knowledge of a particular application domain, task or resource. This consensual knowledge is important to a MAS development project where multiple, distributed developers are involved. In MOBMAS, where ontologies are used as a major information source for many MAS analysis and design steps (e.g. "Develop System Task Model" step, "Specify Agent Class' Belief Conceptualisations" step, "Develop Agent Behaviour Model" step and "Develop Agent Interaction Model" step; cf. Sections 6.2.1, 6.4.1, 6.4.4 and 6.5.2 respectively), the different developers who engage in different development steps can share the same knowledge base when performing their individual work, thereby generating consistent work products despite of the distributed development contexts. In addition, as mentioned in Section 2.3.2.3, the mappings between different ontologies identify the associations amongst the different application domains, tasks and/or resources. This identification allows the developers to combine/integrate their work if each had focused on a different domain, task or resource.

Ontology-related strengths of MOBMAS' development product

- *Support for interoperability* (cf. Section 2.3.2.1): The MASs resulted from MOBMAS can strongly support interoperability between heterogeneous agents and between heterogeneous resources, because in these MASs, the knowledge of heterogeneous agents has been explicitly conceptualised by ontologies (cf. Section 6.4.1), the information/applications of heterogeneous resources have also been explicitly conceptualised by ontologies (cf. Section 6.3.4), and the semantic mappings between these ontologies have been explicitly specified (cf. Sections 6.2.4.2 and 6.3.4.1). A detailed discussion of how these factors can support interoperability is already presented in Section 2.3.2.1. Amongst the existing AOSE methodologies, INGENIAS, PROMETHEUS, GAIA and MASSIVE are the only ones that mention the existence of non-agent resources in MAS. However, they do not discuss how the heterogeneous components of MAS can be interoperated. Meanwhile, even though MASE considers the use of ontology to support interoperability between heterogeneous agents, it does not mention the case of non-agent resources.

- *Support for reusability* (cf. Section 2.3.2.2): The Ontology Model of a MAS designed by MOBMAS offers a detailed description of the target application. Therefore, any future MAS development projects can simply examine this model to determine whether, and which part(s) of, a past MAS design can be reused. Moreover, since the core design models of MOBMAS are composed in terms of ontologies and ontological concepts (namely, Agent Belief Conceptualisation, Agent Behaviour Model and Agent Interaction Model), the developer can adapt the past MAS design models to a new application by simply changing the ontologies involved. In addition, MOBMAS has implemented the idea of using ontologies to decouple the modelling of agents' domain knowledge from agents' behavioural/problem-solving knowledge[85], thereby supporting the reuse of these two knowledge components across agents. Lastly, MOBMAS provides extensive support for interoperability (as discussed above). It therefore shows how legacy agents and/or resources can be reused by the current MAS system. In summary, compared

[85] In MOBMAS, agents' domain knowledge is captured via ontologies in Agent Belief Conceptualisations (cf. Section 6.4.1), while agents' behavioural constructs (i.e. plans, reflexive rules and actions) are defined in Agent Behaviour Model (cf. Section 6.4.4).

to the existing AOSE methodologies, MOBMAS discovers new ways of supporting reusability through its use of ontologies in MAS development.

- *Support for semantically-consistent communication between agents* (cf. Section 2.3.2.4): Ontology is essential to the successful communication between agents. If agents use the same ontology to compose and interpret the exchanged messages, they can convey the information in a uniform and consistent manner. With this understanding, MOBMAS requires the developer to "datatype" the variables in all exchanged ACL messages (or tuples) with concepts defined in the ontologies shared between the communicating parties (cf. Section 6.5.2). With this rule, agents in the resulting MASs will always be able to interpret the exchanged messages, and interpret them in a consistent manner. All of the existing AOSE methodologies, except for MASE and PASSI, do not provide this insurance, since they fail to recognise the importance of ontology in agent communication (cf. Section 3.3.2).

- *Support for communication between agents and resources* (cf. Section 2.3.2.4): An explicit conceptualisation of a resource will allow the wrapper agents to determine which vocabulary they should use to formulate the queries/commands to the resource and to interpret the queries' results, without having to access the resource's internal structure. This ontology-related benefit is naturally offered by a MAS produced by MOBMAS, because MOBMAS addresses the modelling of resources' conceptualisations through ontologies and the specification of mappings between resources' ontologies and MAS application ontologies (cf. Section 6.3.4). Even though four of the existing AOSE methodologies show some consideration for ontology (i.e. MAS-CommonKADS, MESSAGE, MASE and PASSI), they do not discuss the modelling of resources' conceptualisations, thus failing to use ontologies to facilitate agent-resource communication.

- *Support for agent reasoning* (cf. Section 2.3.2.4): In MOBMAS, the specification of agents' behavioural constructs (i.e. plans, reflexive rules and actions) makes reference to the agents' ontology-based knowledge (wherever appropriate) to allow for the agents' problem-solving knowledge to be linked with the agents' ontology-based domain-related knowledge. For example, ontological concepts are used to

define the knowledge requirements of each agent' plans and actions; cf. Section 6.4.4.3). This enables agents' reasoning (which operationalises the agents' problem-solving knowledge) to utilize the ontology-based domain-related knowledge of agents at run-time. No existing AOSE methodologies are found to explicitly associate agents' problem-solving knowledge with agents' ontological knowledge at design time. Accordingly, they cannot illustrate whether, and how, agent reasoning can utilize ontology-based knowledge at run-time.

- *Support for maintainability*: A MAS system produced by MOBMAS can easily be maintained, even by someone other than the original developer, because the specification of the underlying application domains, tasks and wrapped resources has been formally documented in the ontologies of the Ontology Model, and because the other core MAS design models such as Agent Belief Conceptualisation, Agent Behaviour Model and Agent Interaction Model are consistently defined in term of these ontologies. This support for maintainability is not demonstrated in the existing AOSE methodologies.

- *Support for extendibility*: When a MAS designed by MOBMAS needs to cover new domains, tasks or resources, its agents can easily extend their knowledge by adding new ontologies to their knowledge models. New Agent Plan Templates, Reflexive Rule Specifications and Interaction Diagrams can also be created by referring to the concepts defined in the new ontologies. Such ease of extendibility is not demonstrated in the existing AOSE methodologies.

- *High likelihood of a correct system*: This is the direct result of MOBMAS' extensive support for verification and validation during the design of core models such as Agent Behaviour Model (cf. Section "*Ontology-related strengths of the MAS development process of MOBMAS*").

7.5. SUMMARY

This chapter has documented the process of evaluating and refining MOMBAS, which progressively led to the final version of MOBMAS presented in Chapter 6. The

progressive evaluation and refinements of MOBMAS were conducted through the collection of two expert reviews, the use of MOBMAS on a test application by two external developers, and a feature analysis of MOBMAS. This feature analysis includes the justification of MOBMAS' comprehensive support for ontology-based MAS development and various other important AOSE methodological requirements, the comparison between MOBMAS and the existing AOSE methodologies, and the clarification of MOBMAS' ontology-related strengths.

CHAPTER 8
CONCLUSIONS

8.1. INTRODUCTION

This chapter concludes the thesis by recapitulating the contributions of this research to the literature on AOSE (Section 8.2). It also identifies the limitations of the process of conducting the research (Section 8.3) and suggests directions for future research (Section 8.4). The chapter is closed with some concluding remarks (Section 8.5).

8.2. CONTRIBUTIONS OF THE RESEARCH

The main contribution of this research is *the proposal of an AOSE methodology for the analysis and design of ontology-based MASs, which is named "MOBMAS" – "Methodology for Ontology-Based Multi-Agent Systems"*. MOBMAS offers a **software engineering process** comprising of *activities* and *steps* to conduct the analysis and design of an ontology-based MAS, **techniques** to perform these steps and **model kinds** to represent the software artifacts. The methodology is capable of supporting *ontology-based MAS development* and various other *AOSE methodological requirements* which are important to an AOSE methodology but which may not be well-supported by the existing methodologies.

- With regard to the **support for ontology-based MAS development**, MOBMAS surpasses all of the existing AOSE methodologies. Even though four of the existing methodologies were found to integrate ontologies into MAS design, they fail to identify and implement the diverse potential ways in which ontologies can be used in the MAS development process and/or included in the MAS model definitions (cf. Section 3.3.2). Meanwhile, MOBMAS, with its comprehensive acknowledgement of ontology's significant benefits to interoperability, reusability, MAS development activities (particularly system analysis and agent knowledge modelling) and MAS

operation (specifically communication and agent reasoning), has extensively incorporated ontologies into its MAS development process and model definitions.

- Regarding the *MAS development process*, MOBMAS makes use of application ontologies to facilitate the process of constructing and validating its MAS analysis and design models. In particular, application ontologies are used to help identify and validate the system tasks of the target MAS, actions of agent classes and exchanged messages between agents. Moreover, MOBMAS also enables the MAS development process to, in return, support the development of application ontologies. Specifically, the analysis of MAS system tasks and the detailed design of agent classes' goals, plans, actions and exchanged messages help to identify and validate the concepts defined in the application ontologies.

- Regarding the *MAS model definitions*, MOBMAS dedicates one of its model kinds, namely "Ontology Model Kind", to the representation of application ontologies. This model kind captures all of the application ontologies that are necessary for agents in the target MAS to operate. Agents' knowledge is then modelled in term of these ontologies. Agent behaviour modelling and interaction modelling are also based upon ontologies: concepts in the application ontologies are used to formulate agent classes' goals, plans, actions and content of communication messages. MOBMAS also models the conceptualisation of non-agent resources (i.e. Resource Application ontologies), and the mappings between these Resource Application ontologies and the MAS Application ontologies.

By extensively exploiting ontology as described above, MOBMAS is able to enhance its *MAS development process* and *MAS development product* with many important ontology-related strengths (cf. Section 7.4.2.4). These strengths include those widely-acknowledged benefits of ontology to MASs (i.e. support for interoperability, reusability, system analysis, agent knowledge modelling, communication and agent reasoning; cf. Section 2.3.2), and those additional benefits of ontology as uncovered by MOBMAS (e.g. support for verification and validation, maintainability, extendibility and reliability). These ontology-related strengths are either not provided, or provided to a lesser extent, by the existing AOSE methodologies due to their lack, or low level, of support for ontology (cf. Sections 3.3.2 and 7.4.2.4).

- With regard to the **general support for MAS analysis and design**, no individual AOSE methodology was found to address *all* of the important methodological requirements of an AOSE methodology (which were identified by this research from an investigation of the AOSE literature and confirmed by the practitioners and researchers in the field). MOBMAS, on the other hand, endeavours to support all of these requirements by combining the strengths of the existing AOSE methodologies (i.e. by reusing and enhancing the various strong techniques and model definitions of the existing methodologies where appropriate) and proposing new techniques and model definitions where necessary.

Given the above major improvements of MOBMAS over the existing AOSE methodologies, this research helps to cultivate the maturity of the AOSE paradigm, which is still far away from reaching the maturity level of other conventional software engineering paradigms such as OO software engineering.

In addition, apart from MOBMAS, this research also makes other notable contributions to the literature on AOSE.

- It recommends a list of *methodological requirements for an AOSE methodology*. These requirements consist of a set of *features* that an AOSE methodology should support, a set of *steps* that the MAS development process should include, and a set of *modelling concepts* that MAS development model kinds should represent. They were identified by investigating the literature on AOSE, namely, the various evaluation frameworks on AOSE methodologies and conventional system development methodologies, as well as the documentation of the various existing AOSE methodologies. These requirements were also validated by conducting a survey on practitioners and researchers in the field.

  The identification of AOSE methodological requirements has a significant contribution to the future research in AOSE, because it establishes a sensible starting point for the development of new AOSE methodologies, namely new AOSE development process, techniques and model definitions. To date, no study has been found that attempts to identify these AOSE methodological requirements. This research therefore represents a pioneering effort in this area.

- This research also proposed a comprehensive and multi-dimensional *feature analysis framework* for the evaluation and comparison of AOSE methodologies. Developed from the synthesis of various existing evaluation frameworks (both for AOSE methodologies and for conventional system development methodologies), the novelty of the proposed framework lies in the high degree of its completeness and relevance. The framework consists of evaluation criteria that assess an AOSE methodology from both the dimensions of system engineering and those specific to AOSE. It also pays attention to all three major elements of a system development methodology: development process, techniques and model definitions.

8.3. LIMITATIONS OF THE RESEARCH

8.3.1. Limitations of the survey on practitioners and researchers

The survey was conducted by this research to validate the methodological requirements proposed for an AOSE methodology (Section 5.3). Since the AOSE paradigm is still very young, the number of practitioners and researchers who participated in the survey was expected to be small. Eventually the survey sample was 41. Although this is not a small number, a larger sample size would provide a more reliable assessment of the professional opinions on the importance of the proposed AOSE requirements.

8.3.2. Limitations of the feature analysis on the existing AOSE methodologies

This research evaluated the 16 existing AOSE methodologies in term of their support for each AOSE methodological requirement (Section 5.4). This investigation was based on the published documentation of each methodology. Even though this material provided a relatively comprehensive description of each methodology, it might omit discussion of the methodology's support for some particular features (e.g. which software development lifecycle the methodology adopts, or whether a methodology is capable of supporting dynamic systems). Accordingly, this research sometimes had to deduce a methodology's support for a particular methodological requirement from the

published documentation (if possible), or concluded that the methodology does not support the feature. This evaluation may be subject to error.

In addition, with regard to the evaluation of criteria "*Usability of the development process*" and "*Usability of techniques*", this research arrived at its assessment by *non-empirically* reviewing the methodology's steps and model definitions. A more reliable evaluation would be to empirically apply the 16 methodologies on an (identical) application. Unfortunately, the constraints in time and resources prevented this research from conducting such an empirical evaluation.

8.3.3. Limitations of the comparison between MOBMAS and the existing AOSE methodologies

The comparison of MOBMAS and the 16 existing methodologies in term of criteria "*Usability of the development process*" and "*Usability of techniques*" is potentially biased, because MOBMAS' usability was *empirically* assessed by the two external developers who used the methodology on an application, while the usability of the existing methodologies was *non-empirically* assessed by the researcher. As such, the evaluators were different and the methods of usability evaluation were also different. To obtain an ideal comparison, the two developers who evaluated MOBMAS should also use all of the 16 existing AOSE methodologies on the same application, thereby comparing the usability of all methodologies. However, the constraints in time and resources prevented this research from conducting such an empirical comparison.

8.4. SUGGESTIONS FOR FUTURE RESEARCH

Since the main output of this research is the proposal of an AOSE methodology, there are basically two general directions of future research: making extensions to the proposed methodology, and applying the methodology to a variety of applications.

8.4.1. Extending MOBMAS

MOBMAS may be extended in two major ways.

- *Adding more techniques and/or modelling notation to support the currently provided features, steps and modelling concepts*: As research on AOSE continually grows, new techniques and/or modelling notation may arise for supporting the features, steps and modelling concepts that are currently addressed in MOBMAS (e.g. new techniques for the identification of agent classes, new mechanisms for agent interactions, or new notation for ontology modelling). These new ideas should be recognised and included in MOBMAS (if applicable) to improve MOBMAS' powerfulness.

- *Adding support for new features, steps and modelling concepts*: Currently, MOBMAS provides support for a variety of features, steps and modelling concepts that have been determined to be important to an AOSE methodology. However, to extend MOBMAS' capability and applicability, new features, steps and modelling concepts can be added to MOBMAS. For example, new techniques and model definitions can be introduced to provide support for the development of MASs with mobile agents, or agents with personality.

8.4.2. Applying MOBMAS to a variety of applications

In this dissertation, MOBMAS has been applied to a "Peer-to-Peer Information Sharing" application by two external developers (Appendix H). To further validate MOBMAS, the methodology should be tested on other demonstrative applications, and/or employed in many real-world development projects. Preferably, MOBMAS should be tested and/or used on applications of diverse domains, sizes and/or degrees of complexity. In addition, a potential revenue for future research is to apply MOBMAS on the same application as that previously used by an existing MAS development methodology(ies). This would facilitate a reliable comparison between MOBMAS and the existing methodology(ies) in term of usability (as had been discussed in Section 8.3.3).

8.5. CONCLUDING REMARKS

In summary, this research has proposed a software engineering methodology for the analysis and design of ontology-based MASs. This methodology improves on the existing AOSE methodologies in terms of its comprehensive support for ontology-based MAS development, and its support for various other important features, steps and modelling concepts of MAS analysis and design that are not well-supported by the existing methodologies. The proposed methodology has been applied to a "Peer-to-Peer Information Sharing" application by two external developers. It is hoped that the methodology will be applied to many other MAS development projects and widely recognised and adopted by the AOSE community.

REFERENCES

Acronymics Inc. 2004. *AgentBuilder*. http://www.agentbuilder.com/. (accessed March 17, 2003).

Agent Lab. 2000. *Multi-Agent Modeling Language*. http://www.maml.hu/. (accessed February 25, 2003).

Agent Oriented Software. 2004. *JACK™ Intelligent Agents Manual*. http://www.agent-software.com/shared/demosNdocs/JACK_Manual.pdf. (accessed May 21, 2004).

Ambros-Ingerson, J. and S. Steel. 1988. Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88), St. Paul, USA*.

Anton, A.I., and C. Potts. 1998. The use of goals to surface requirements for evolving systems. In *Proceedings of 20thInternational Conference on Software Engineering (ICSE'98)*, *Kyoto, Japan*, 157-166.

Anton, A.I., J. H. Dempster, and D. F. Siege. 1996. Deriving Goals from a Use Case Based Requirements Specification for an Electronic Commerce System. In *Proceedings of the 6th International Workshop on Requirements Engineering: Foundations for Software Quality, Stockholm, Sweden*.

Anton, A.I., W.M. McCracken, and C. Potts. 1994. Goal decomposition and scenario analysis in business process reengineering. In *Proceedings of the 6th International Conference on Advanced Information Systems Engineering (CaiSE'94), Utrecht, The Netherlands*, 136-144.

Arkin, R., and T. Balch. 1997. AuRA: Principles and Practice in Review. *Journal of Experimental and Theoretical Artificial Intelligence* 2-3: 175-189.

Awad E M 1985. *Systems Analysis and Design*. Illinois: Richard D. Irwin.

Baader, F., D.L. Calvanese, D. McGuinness, D. Nardi and P.F. Patel-Schneider, eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. New York: Cambridge University Press.

Bandini, S., S. Manzoni, and G. Vizzari. 2004. A Spatially Dependent Communication Model for Ubiquitous Systems. In *Proceedings of the 1st International Workshop on Environments for Multiagent Systems, New York, USA*.

Bauer, B. 2001a. UML Class Diagrams revisited in the context of agent-based systems. In *Proceedings of Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada*, 1-8.

Bauer, B. 2001b. UML Class Diagrams and Agent-Based Systems. In *Proceedings Autonomous Agents 2001, Montreal, Canada*, 104-105.

Bauer, B., J.P. Muller, and J. Odell. 2000. Agent UML: A Formalism for Specifying Multiagent Software Systems. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick*, Ireland, 91-103.

Bayardo, R.J., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. 1997. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of data, Tucson, USA*, 195-206.

Bechhofer, S., C. Goble, and I. Horrocks. 2001. DAML+OIL is not enough. In *Proceedings of the 1st Semantic Web Working Symposium*, *Stanford, USA*, 151-159.

Benjamins, R. 1995. Problem solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications* 2(8): 93-120.

Benjamins, R., L.N. de Barros, and A. Valente. 1996. Constructing planners through problem-solving methods. In *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, *Banff, Canada.*

Bergenti, F., and A. Ricci. 2002. Three approaches to the coordination of multiagent systems. In *Proceedings of the 2002 ACM symposium on applied computing*, *Madrid, Spain*, 367-372.

Bergenti, F., and A. Poggi. 2001. Agent-oriented Software Construction with UML. In *Handbook of Software Engineering and Knowledge Engineering* Vol 2, ed. S.K. Chang, 757-770. Singapore: World Scientific Publishing Co.

Bergenti, F., and A. Poggi. 2002. Supporting Agent-Oriented Modelling with UML. *International Journal of Software Engineering and Knowledge Engineering* 12(6): 605-618.

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. *The Semantic Web*. http://www.sciam.com/2001/0501issue/0501berners-lee.html. (accessed February 26, 2002).

Bernon, C., M.P. Gleizes, G. Picard, and P. Glize. 2002a. The ADELFE methodology for an intranet system design. In *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto, Canada*.

Bernon, C., M.P. Gleizes, S. Peyruqueou, and G. Picard. 2002b. ADELFE, a methodology for Adaptive Multi-Agent Systems Engineering. In *Proceedings of the 3rd International Workshop on Engineering Societies in the Agents World (ESAW-2002), Madrid, Spain*.

Beydoun, G., G. Low, C. Conzalez-Perez, and B. Henderson-Sellers. 2005. Synthesis of a Generic MAS Metamodel. In *Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05) St. Louis, USA*, 27-31.

Biegel, G. 2002. Cooperation through the Environment: Stigmergy in CORTEX. In *CORTEX: Preliminary Definition of the Interaction Model*, ed. J. Kaiser, 31-38. http://cortex.di.fc.ul.pt/Deliverables/WP2-D3.pdf. (accessed January 16, 2005)

Bleyer, M. 1998. *Multi-Agent Systems for Information Retrieval on the World Wide Web*. Master Thesis, University of Ulm, Germany.

Boer, F. S. 2000. *Methodology for Agent-Oriented Software Design*. http://www.cs.uu.nl/people/frankb/nwo.html. (accessed November 15, 2003).

Boicu, M., G. Tecuci, M. Bowman, D. Marcu, S.W. Lee, and K. Wright. 1999. A Problem-Oriented Approach to Ontology Development. In *Proceedings of the 16th National Conference on Artificial Intelligence Workshop on Ontology Management, Orlando, Florida*.

Bonabeau, E., F. Henaux, S. Guerin, D. Snyers, P. Kuntz, and G. Theraulaz. 1998. Routing in Telecommunications Networks with "Smart" Ant-Like Agents. In *Proceedings of the 2nd International Workshop on Agents in Telecommunications Applications (IATA '98), Paris, France*.

Booch, G. 1994. *Object-oriented Analysis and Design*. 2nd ed. Massachusetts: Addison-Wesley.

Bordart, F., A. Flory, M. Leonard, A. Rochefeld, C. Rolland, and H. Tardieu. 1983. Evaluation of CRIS 1 I.S. Developments Methods Using a Three Cycles. In

Information Systems Design Methodologies - A Feature Analysis, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Borgida, A., R.J. Brachman, D.L. McGuinness, and L.A. Resnick. 1989. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, USA*, 59-67.

Brachman, R.J., and J.G. Schmolze. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2): 191-216.

Brandt, I. 1983. A Comparative Study of Information Systems Design Methodologies. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Bratman, M.E., D.J. Israel, and M.E. Pollack. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349-355.

Bresciani, P., P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. 2004. TROPOS: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems* 8(3): 203-236.

Brueckner, S. 2000. *Return from the Ant.* PhD thesis, Humboldt-Universität zu Berlin, Germany.

British Telecommunications. 2002. *Zeus*. http://more.btexact.com/projects/agents/zeus/index.htm. (accessed June 5, 2003).

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2: 14-23.

Burrafato, P., and M. Cossentino. 2002. Designing a multi-agent solution for a bookstore with the PASSI methodology. In *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto, Canada*.

Cabri, G., L. Leonardi, and F. Zambonellli. 2000. XML Dataspaces for mobile agent coordination. In *Proceedings of the 2000 ACM symposium on Applied computing, Como, Italy*, 181-188.

Cairo, O. and J.C. Alvarez. 2004. The KAMET II Approach for Knowledge-Based system Construction. In *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Wellington, New Zealand*, 1227-1234.

Calvanese, D., G. De Giacomo, and M. Lenzerini. 2001. A framework for ontology integration. In *Proceedings of the 1st International Semantic Web Working Symposium, Stanford, USA*, 303–317.

Carbonell, J.G., C.A. Knoblock, and S. Minton. 1991. PRODIGY: An Integrated Architecture for Prodigy. In *Architectures for Intelligence*, ed. K. VanLehn, 241-278. New Jersey: Lawrence Erlbaum Associates.

Cardelli, L. 1994. *Obliq: A Language with Distributed Scope*. Technical report, Digital Equipment Corp, Systems Research Center, California, USA.

Carver, N., and V. Lesser. 1995. The DRESUN Testbed for research in FA/C Distributed situation assessment: extensions to the model of external evidence. In *Proceedings of the 1st International Conference on Multi-Agent Systems*, *San Francisco, USA*, 33-40.

Castro, J., M. Kolp, and J. Mylopoulos. 2001. A Requirements-Driven Development Methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering CAiSE 01, Interlaken, Switzerland.*

Castro, J., M. Kolp, and J. Mylopoulos. 2002. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems* 27: 365-389.

Ceccaroni, L. 2001. What if a wastewater treatment plant were a town of agents. In *Proceedings of the workshop Autonomous Agents 2001 - W03: Ontologies in Agent Systems, Montréal, Canada*.

Cernuzzi, L., and G. Rossi. 2002. On the Evaluation of Agent-Oriented Modelling Methods. In *Proceedings of the OOPSLA Workshop on Agent-Oriented Methodologies*, *Seattle, USA*, 21-33.

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

Chapman, D., and P. Agre. 1986. Abstract reasoning as emergent from concrete activity. In *Proceedings of the 1986 Workshop on Reasoning About Actions &Plans, Los Altos, USA*, 411-424.

Chatley, R. n.d. *Hybrid Deliberative/Reactive Agents*. http://www.iis.ee.ic.ac.uk/~frank/surp99/article2/rbc97/. (accessed May 21, 2002).

Cheikes, B.A. 1995. GIA: An Agent-Based Architecture for Intelligent Tutoring Systems. In *Proceedings of the CIKM'95 Workshop on Intelligent Information Agents, Baltimore, USA*.

Chelberg, D., L. Welch, A. Lakshmikumar, G. Matthew, and Q. Zhou. 2001. Meta-Reasoning For a Distributed Agent Architecture. In *Proceedings of the South-Eastern Symposium on System Theory, Ohio, USA,* 377-381.

CHI Software Inc. 2003. *iGEN The Cognitive Agent Software Toolkit*. http://www.cognitiveagent.com/. (accessed April 28, 2002).

Ciancarini, P. 1996. Coordination models and languages as software integrators. *ACM Computing Surveys* 28(2): 300-302.

Ciancarini, P., O. Nierstrasz, and R. Tolksdorf. 1998. *A case study in coordination: Conference Management on the Internet*. ftp://cs.unibo.it/pub/cianca/ coordina.ps.gz. (accessed April 20, 2004).

Ciancarini, P., A. Omicini, and F. Zambonelli. 1999. Multiagent System Engineering: the Coordination Viewpoint. In *Proceedings of the 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL), Orlando, USA*, 250-259.

Collinot, A., and A. Drogoul. 1998. Using the Cassiopeia Method to Design a Soccer Robot Team. *Applied Artificial Intelligence Journal* 12(2-3): 127-147.

Collinot, A., A. Drogoul, and P. Benhamou. 1996. Agent Oriented Design of a Soccer Robot Team. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96), Kyoto, Japan*, 41-47.

Conri, S.E., R.A. Meyer, and V.R. Lesser. 1988. Multistage negotiation in distributed planning. In *Distributed Artificial Intelligence*, ed. A.H. Bond and L. Gasser, 367-384. California: Morgan Kaufmann Publishers Inc.

Cossentino, M. 2002. Different perspectives in designing multi-agent systems. In *Proceedings of Agent Technology and Software Engineering Workshop (AGES '02), Erfurt, Germany*.

Cossentino, M., and M. Potts. 2002. A CASE tool supported methodology for the design of multi-agent systems. In *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02)*.

Cranefield, S., and M. Purvis. 1999. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden*.

Cranefield, S., S. Hausteiny, and M. Purvis. 2001. UML-based ontology modelling for software agents. In *Proceedings of Ontologies in Agent Systems Workshop*, 21-28.

Cranefield, S., M. Purvis, M. Nowostawski, and P. Hwang. 2002. Ontologies for interaction protocols. In *Proceedings of the 2nd International Workshop on Ontologies in Agent Systems, Bologna, Italy*.

Cremonini, M., A. Omicini, and F. Zambonelli. 1999. Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain*, 77-88.

Cuesta, P., A. Gómez, J.C. González, and F.J. Rodríguez. 2002. The MESMA approach for AOSE. In *Proceedings of 4th Iberoamerican Workshop on Multi-Agent Systems (Iberagents'2002), Málaga, Spain*.

Cuppari, A., P.L. Guida, M. Martelli, V. Mascardi, and F. Zini. 1999. Prototyping Freight Trains Traffic Management Using Multi-Agent Systems. In *Proceedings of IEEE International Conference on Information, Intelligence and Systems, Bethesda, USA*, 646-653.

Dardenne, A., A. van Lamsweerde, and S. Fickas. 1993. Goal-Directed Requirements Acquisition. *Science of Computer Programming* 20: 3-50.

Davis, D. 1995. *A design for the robot crèche scenario*. http://citeseer.nj.nec.com/davis95design.html. (accessed October 25, 2002).

de Bruijn, J. 2003. *Using Ontologies*. *Enabling Knowledge Sharing and Reuse on the Semantic Web*. Technical Report, Digital Enterprise Research Institute, Ireland.

Decker, K.S., V.R. Lesser, M.V. Nagendra-Prasad, and T. Wagner. 1995. MACRON: an architecture for multi-agent cooperative information gathering. In *Proceedings of the CIKM-95 Workshop on Intelligent Information Agents, Baltimore, USA*.

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

Degirmenciyan, I., F. Marc, and A. ElFallah-Seghrouchni. 2003. Modeling multi-agent plans with hybrid automata. In *Proceedings of the workshop FAMAS 03 ETAPS 03, Warsaw, Poland.*

DeLoach, S.A. 1999. Multiagent Systems Engineering: A methodology and language for designing agent systems. In *Proceedings of Agent-Oriented Information Systems (AOIS'99), Seattle, USA*, 45-57.

DeLoach, S.A. 2005. Multiagent Systems Engineering of Organization-based Multiagent Systems. In *Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05) St. Louis, USA*, 5-11.

DeMarco, T. 1978. *Structured Analysis and System Design*. New York: Yourdon Press.

Demazeau, Y., and A.C.R. Costa. 1996. Populations and organizations in open multi-agent systems. In *Proceedings of the 1st National Symposium on Parallel and Distributed AI, Hyderabad, India*.

Dennis, A., and B. Wixom. 2003. *Systems analysis design*. 2nd ed. New York: J. Wiley.

Denti, E., and A. Omicini. 2001. LuCe: A tuple-based coordination infrastructure for Prolog and Java agents. *Autonomous Agents and Multi-Agent Systems* 4(1/2):139–141.

Denti, E., A. Natali, and A. Omicini. 1998. On the expressive power of a language for programming coordination media. In *Proceedings of the 1998 ACM Symposium on Applied Computing, Atlanta, USA*, 169-177.

desJardins, M.E., E.H. Durfee, C.L. Ortiz, and M.J. Wolverton. 2000. A Survey of Research in Distributed, Continual Planning. *AI Magazine* 4: 13-22.

DiLeo, J., T. Jacobs, and S. DeLoach. 2002. Integrating Ontologies into Multiagent Systems Engineering. In *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Bologna, Italy*.

Ding, Y. 2001. IR and AI: The role of ontology. In *Proceedings of the 4th International Conference of Asian Digital Libraries (ICADL 2001), Bangalore, India*.

d'Inverno, M., D. Kinny, M. Luck, and M. Wooldridge. 1997. A formal specification of dMARS. In *Proceedings of the 4th International Workshop on Agent Theories, Architectures and Languages, Providence, USA*, 155-176.

Durfee, E.H. 1999. Distributed Problem Solving and Planning. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, ed. G. Weiss, 121-164. London: The MIT Press.

Duursma, C. 1993. *Task Model definition and Task Analysis process*. ESPRIT Project P5248 KADS-II KADS-II/M5/VUB/RR/004/1.1c, Vrije Universiteit Brussel.

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

Elammari, M., and W. Lalonde. 1999. An Agent-Oriented Methodology: High-Level and Intermediate Models. In *Proceedings of the 1st Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'99)*, *Heidelberg, Germany*.

Eliason, A.L. 1990. Syst*ems development: analysis, design and implementation*. 2nd ed. Glenview: Scott, Foresman/Little, Brown Higher Education.

Ephrati, E., and J.S. Rosenschein. 1991. The Clarke Tax as a consensus mechanism among automated agents. In *Proceedings of the 9th National Conference on Artificial Intelligence, San Jose, USA*, 173-178.

Erdur, R.C., O. Dikenelli, and H. Sengonca. 1999. A Multiagent System for Searching and Retrieving Reusable Software Components. In *Proceedings of 14th International Conference on Computer and Information Sciences - ISCIS XIV, Kusadasi, Turkey*.

Eurescom. 2001a. *MESSAGE: Methodology for Engineering Systems of Software Agents – Final Guidelines for the Identification of Relevant Problem Areas where Agent Technology is Appropriate*. http://www.eurescom.de/public/projectresults/P900-series/907d2.asp. (accessed Oct 7, 2003).

Eurescom. 2001b. *Methodology for Agent-Oriented Software Engineering*. http://www.eurescom.de/public/projectresults/P900-series/907ti1.asp. (accessed March 21, 2004).

Fabio, B., G. Caire, T. Trucco, and G. Rimassa. 2004. *JADE Programmer's Guide*. http://jade.tilab.com/doc/programmersguide.pdf. (accessed Dec 14, 2004).

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

Falbo, R.A., C.S. Menezes, and A.R.C. Rocha. 1998. A Systematic Approach for Building Ontologies. In *Proceedings of the 6th Ibero-American Conference on AI: Progress in Artificial Intelligence*, 349-360.

Falbo, R.A., G. Giancarlo, and K.C. Duarte. 2002. An Ontological Approach to Domain Engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy*, 351-358.

Falkenberg, E., G.M. Nijssen, A. Adams, L. Bradley, P. Bugeia, A.L. Campbell, M. Carkeet, G. Lehmann, and A. Shoesmith. 1983. Feature Analysis of ACM/PCM, CIAM, ISAC and NIAM. In *Information Systems Design Methodologies - A*

Feature Analysis, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

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

Farquhar, A., R. Fikes, and J. Rice. 1996. The Ontolingua Server: A tool for collaborative ontology construction. Technical Report 926-26, Knowledge Systems Laboratory, Stanford University.

Fensel, D. 1997. An Ontology-Based Broker: Making Problem-Solving Method Reuse Work. In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems held in conjunction with the 15th International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Japan,* 23-29.

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

Fensel, D., E. Motta, S. Decker, and Z. Zdrahal. 1997. Using Ontologies For Defining Tasks, Problem-Solving Methods and Their Mapping. In *Proceedings of 10th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW-97), Heidelberg, Germany*, 113-128.

Ferber, J., and O. Gutknecht. 1998. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98), Paris, France*, 128-135.

Ferguson, I. A. 1992. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, University of Cambridge, UK.

Fernandez, M., A. Gomez-Perez, and N. Juristo. 1997. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of the AAAI Spring Symposium on Ontological Engineering, California, USA*, 33-40.

Fikes, R.E., and N. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2): 189-208.

Finkelstein, A. 1998. Interoperable Systems: An introduction. In *Information Systems Interoperability*, ed. B.J. Kramer, M.P. Papazoglou and H.-W. Schmidt, 1-9. England: Research studies press.

FIPA. n.d.a. *FIPA Agent Communication Language Specifications*. http://www.fipa.org/repository/aclspecs.html. (accessed January 6, 2003).

FIPA. n.d.b. *The Foundation for Intelligent Physical Agents*. http://www.fipa.org/. (accessed February 24, 2002).

FIPA. 2001a. *FIPA Agent Software Integration Specification*. http://www.fipa.org/specs/fipa00079/XC00079B.html. (accessed June 25, 2004).

FIPA. 2001b. *FIPA Ontology Service Specification*. http://www.fipa.org/specs/fipa00086/XC00086D.html. (accessed June 10, 2002).

FIPA.2001c. *FIPA Interaction Protocol Library Specification*. http://www.fipa.org/specs/fipa00025/XC00025E.html. (accessed January 21, 2003).

FIPA. 2002. *FIPA Interaction Protocols Specifications*. http://www.fipa.org/repository/ips.php3. (accessed September 3, 2002).

FIPA. 2003. *FIPA Modeling Area: Deployment and Mobility*. http://www.auml.org/auml/documents/DeploymentMobility.zip. (accessed August 24, 2003).

FIPA. 2004. *FIPA Agent Management Specification*. http://www.fipa.org/specs/fipa00023/SC00023K.html. (accessed May 19, 2003).

Firby, R.J. 1989. *Adaptive Execution in Dynamic Domains*. PhD thesis, Yale University, Computer Science Department, USA.

Firesmith, D.G., and Henderson-Sellers, B. 2002. *The OPEN Process Framework: An Introduction*. London: Addison-Wesley.

Fisher, M., J. Muller, M. Schroeder, G. Staniford, and G. Wagner. 1997. Methodological Foundations for Agent-Based Systems. *The Knowledge Engineering Review* 12(3): 323-329.

Flores-Mendez, R.A. 1999. Towards a Standardization of Multi-Agent System Frameworks. *ACM Crossroads Student Magazine* 5(4). http://www.acm.org/crossroads/xrds5-4/multiagent.html. (accessed January 8, 2002).

Franklin, S. and A. Graesser. 1996. Is it an agent or just a program?: A Taxonomy for Intelligent Agents. In *Proceedings of the 3rd International Workshop on Agent Theories, Architectures and Languages, Budapest, Hungary*, 21-35.

Franzén, T., S. Haridi, and S. Janson. 1992. An Overview of the Andorra Kernel Language. In *Proceedings of the 2nd Workshop on Extensions to Logic Programming, Stockholm, Sweden*, 163-180.

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

Gat, E. 1991. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. *SIGART Bulletin 2*: 70-74.

General Magic Inc. 1995. *The Telescript Language Reference*. http://www.science.gmu.edu/~mchacko/Telescript/docs/telescript.html. (accessed September 23, 2002).

Genesereth, M.R., and N.J. Nilsson. 1987. *Logical Foundation of Artificial Intelligence*. California: Morgan Kaufmann.

Genesereth, M.R., and R.E. Fikes. 1992. Knowledge Interchange Format Version 3.0 Reference Manual. Logic Group Technical Report Logic-92-1, Stanford University Logic Group, Standford University, USA.

Gennari, J.H., S.W. Tu, T.E. Rotenfluh, and M.A. Musen. 1994. Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41: 399-424.

Georgeff, M.P. 1994. *Distributed multi-agent reasoning systems (dMARS)*. Technical report, Australian Artificial Intelligence Institution, Melbourne, Australia.

Georgeff, M. P., and A.L. Lansky. 1987. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, *Seattle, USA*, 677-682.

Girardi, R. and C.G. de Faria. 2004. An Ontology-Based Technique for the Specification of Domain and User Models in Multi-Agent Domain Engineering. *Clei Electronic Journal* 7(1).

Girardi, R., C.G. de Faria, and L. Balby. 2004. Ontology-based Domain Modeling of Multi-Agent Systems. In *Proceedings of the 3rd International Workshop on Agent-Oriented Methodologies at OOPSLA 2004*, Vancouver, Canada, 51-62.

Glaser, N. 1996. *Contribution to Knowledge Acquisition and Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*. PhD Thesis, University of Navy 1, France.

Glaser, N. 1997a. The CoMoMAS Approach: From Conceptual Models to Executable Code. In *Proceeding of the 8th European Workshop On Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-97), Ronneby, Sweden*.

Glaser, N. 1997b. The CoMoMAS Methodology and Environment for Multi-Agent System Development. In *Multi-Agent Systems – Methodologies and Applications*, ed. C. Zhang and D. Lukose, 1-16. Berlin: Springer-Verlag.

Glass, G. 1998. ObjectSpace Voyager – The Agent ORB for Java. In *Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications*, *Tsukuba, Japan*, 38-55.

Goldin, D., and D. Keil. 2004. Toward Domain-Independent Formalization of Indirect Interaction. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04)*, *University of Modena and Reggio Emilia, Italy,* 393-394.

Gray, R.S. 1995. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents, 4th International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, USA.

Gruber, T. 1993a. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies* 43(5-6): 907-928.

Gruber, T. 1993b. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* 5(2):199-220.

Grüninger, M., and M.S. Fox. 1995. Methodology for the Design and Evaluation of Ontologies. In *Proceedings of IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada*.

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

Guarino, N. 1998. Formal Ontology and Information Systems. In *Proceedings of the 1st International Conference on Formal Ontology in* I*nformation Systems, Trento, Italy*.

Guarino, N., and P. Giaretta. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, ed. N. Mars, 25-32. Amsterdam: IOS Press.

Guessoum, Z., and J.P. Briot. 1999. From active objects to autonomous agents. *IEEE Concurrency* 7(3): 68-76.

Guilfoyle, C., and E. Warner. 1994. *Intelligent agents: The new revolution in software*. Ovum Report.

Gutierrez-Casorran, C., J.T. Fernandez-Breis, and R. Martinez-Bejar. 2001. Ontological modelling of natural categories-based agents: an ant colony. In *Proceedings of the*

Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents, Montreal, Canada.

Haumer, P., K. Pohl, and K. Weidenhaupt. 1998. Requirements Elicitation and Validation with Real World Scenes. *IEEE Transactions on Software Engineering* 24(12): 1036-1054.

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

Herrero, P. and de Antonio, A. 2002. A Human Based Perception Model for Cooperative Intelligent Virtual Agents. In *Proceedings of the 10th International Conference on Cooperative Information Systems (CoopIS 2002)*, California. USA, 195-212.

Hevner, A.R., S.T. March, J. Park, J., and S. Ram. 2004. Design science in information systems research. *MIS Quarterly*, 28(1): 75-106.

Honavar, V. 1999. *Intelligent Agents and Multi Agent Systems - Tutorial at IEEE CEC*. http://www.cs.iastate.edu/%7Ehonavar/agent99.pdf. (accessed October 10, 2002).

Horlait, E. 2003. *Mobile Agents for Telecommunication Applications (Innovative Technology Series: Information Systems and Networks)*. England: Kogan Page Science.

Horrocks, I., and F. van Harmelen. 2001. *Reference Description of the DAML+OIL Ontology Markup Language*. Technical report. http://www.daml.org/2001/03/reference.html. (accessed May 16, 2004).

Huget, M.P., B. Bernhard, J. Odell, R. Levy, P. Turci, R. Cervenka, M. Nodine, S. Cranefield, and H. Zhu. 2003. *FIPA Modeling: Agent Class Diagrams*. http://www.auml.org/auml/documents/main.shtml. (accessed September 5, 2003).

Huhns, M.N. and L.M. Stephens. 1999. Multiagent Systems and Societies of Agents. Journal of Applied Artificial Intelligence. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, ed. G. Weiss, 79-120. London: The MIT Press.

Huhns, M.N., and M.P. Singh. 1997. Ontologies for agents. *IEEE Internet Computing* 1(6): 81-83.

Huhns, M.N., and M.P. Singh, eds. 1998. *Readings in Agents*. California: Morgan Kaufmann Publishers Inc.

Humphreys, B.L, and D.A. Lindberg. 1993. The UMLS project: making the conceptual connection between users and the information they need. *Bulletin of Medical Library Association* 81(2):170-177.

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.

IBM. 2002. *Aglets*. http://www.trl.ibm.com/aglets/. (accessed April 25, 2003).

IEEE – Institute of Electrical and Electronics Engineers. 1990. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York.

Iglesias, C.A., M. Garijo, and J.C. Gonzalez. 1999. A survey of agent-oriented methodologies. In *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98), Paris, France*, 317-330.

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

Iglesias, C.A., M. Garijo, J.C. Gonzalez, and J.R. Velasco. 1998. Analysis and Design of Multi-Agent Systems using MAS-CommonKADS. In *Intelligent Agents IV (LNAI Volume 1365)*, ed. M.P. Singh, A. Rao, and M. Wooldridge, 313-326. Berlin: Springer-Verlag.

Iivari, J. and P. Kerola. 1983. A Sociocybernetic Framework for the Feature Analysis of Information Systems Design Methodologies. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Institut de Recherche en Informatique de Toulouse. n.d. *ADELFE: Atelier de Développement de Logiciels à Fonctionnalité Emergente*. http://www.irit.fr/ADELFE/. (accessed July 25, 2002).

Ioannidis, Y.E., and T.K. Sellis. 1989. Conflict Resolution of rules assigning values to virtual attributes. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data, Portland, USA*, 205-214.

JAFMAS Java-Based Framework for Multi-Agent Systems. Ohio: University of Cincinnati. http://www.ececs.uc.edu/~abaker/JAFMAS/. (accessed April 25, 2003).

Jayaratna, N. 1994. *Understanding and Evaluating Methodologies - NIMSAD A Systematic Framework*. England: McGraw-Hill.

Jennings, N. R. 1993. Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems* 2(3):289-318.

Jennings, N.R. 2001. Building complex, distributed systems: the case for an agent-based approach. *Communications of the ACM* 44(4): 35-41.

Jennings, N.R., and M. Wooldridge. 1995. Applying Agent Technology. *Applied Artificial Intelligence* 9 (4): 351-359.

Jennings, N.R., and M. Wooldridge. 1998. Applications of Intelligent Agents. In *Agent Technology: Foundations, Applications, and Markets*, ed. N. Jennings, and M. Wooldridge, 3-28. Berlin: Springer-Verlag.

Jennings, N.R., and M. Wooldridge. 2001. Agent-Oriented Software Engineering. In *Handbook of Agent Technology*, ed. J. Bradshaw. USA: AAAI/MIT Press.

Jennings, N.R., S. Sycara, and M. Wooldridge. 1998. A Roadmap of Agent Research and Development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1): 7-38.

Kaelbling, L.P. 1991. A situated automata approach to the design of embedded agents. *SIGART Bulletin* 2(4): 85-88.

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

Kankaanpää, T. 1999. *Design and Implementation of a Conceptual Network And Ontology*. Master thesis, Helsinki University of Technology, Finland.

Karp, P. D., V.K. Chaudhri, and J. Thomere. 1999. *XOL: An XML-Based Ontology Exchange Language*. Technical report version 3. ftp://smi.stanford.edu/pub/bio-ontology/xol.doc. (accessed May 17, 2004)

Kendall, E.A. 1999. Role modelling for agent system analysis, design, and implementation. In *Proceedings of the 1st International Symposium on Agent Systems and Applications, Palm Springs, California*, 204-218.

Kendall, E.A. 2000. Agent Software Engineering with Role Modelling. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland*, 163-170.

Kendall, K. E., and J. E. Kendall. 2002. *Systems Analysis and Design*, 5th ed. New Jersey: Prentice Hall.

Kendall, E.A., and L. Zhao. 1998. Capturing and Structuring Goals. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'98, Vancouver, Canada.*

Kendall, E.A., M.T. Malkoun, and C. Jiang. 1995. A methodology for developing Agent based Systems for Enterprise Integration. In *Proceedings of the 1st Australian Workshop on Distributed Artificial Intelligence: Architecture and Modelling, Canberra, Australia*, 85-99.

Khan, L.R. 2000. *Ontology-based information selection*. PhD thesis, University of South California, USA.

Kifer, M., G. Lausen, and J. Wu. 1995. Logical foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM* 42(4): 741 - 843.

Kinny, D., and M. Georgeff. 1996. Modelling and Design of Multi-agent Systems. In *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages (ATAL'96), Budapest, Hungary*, 1-20.

Kinny, D., M. Georgeff, and A. Rao. 1996. A Methodology and Modelling Technique for Systems of BDI Agents. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, *Eindhoven, The Netherlands*, 56-71.

Klampanos, I.A., J.J. Barnes, and J.M. Jose. 2003. Evaluating Peer-to-Peer Networking for Information Retrieval within the Context of Meta-Searching. In *Proceedings of the 2003 European Colloquium on IR Research*, *Pisa, Italy*, 528-536.

Klampanos, I.A., and J.M. Jose. 2003. An Architecture for Peer-to-Peer Information Retrieval. *SIGIR'03*, 401-402.

Knoblock, C.A., A. Arens, and C.N. Hsu. 1994. Cooperating Agents for Information Retrieval. In *Proceedings of the 2nd International Conference on Cooperative Information Systems, Toronto, Canada.*

Knowledge Based Systems Inc. 1994. *IDEF5 Method Report*. http://www.idef.com/Downloads/pdf/Idef5.pdf. (accessed October 16, 2001).

Kolp, M. , Castro, J. and Mylopoulos, J. 2001. A social organization perspective on software architectures. In *Proceedings of the 1st International Workshop from Software Requirements to Architectures (STRAW'01), Toronto, Canada*, 5-12.

Kotonya, G, and I. Sommerville. 1998. *Requirements Engineering: Processes and techniques*. John Wiley & Sons.

Kung, C.H. 1983. An Analysis of Three Conceptual Models with Time Perspective. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Leach, C. 1979. *Introduction to statistics: a nonparametric approach for the social sciences*. New York: Wiley.

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

Lesser, V.R. 1996. Cooperative Multi-agent systems: A personal View of the State of the Art. *IEEE Transactions on Knowledge and Data Engineering* 11(1): 133 - 142.

Lind, J. 1999. *MASSIVE: Software Engineering for Multiagent Systems*. PhD Thesis, University of Saarbrucken, Germany.

Lind, J. 2000a. The MASSIVE development method for Multiagent Systems. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agents (PAAM2000)*, *Manchester*, *UK*.

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

Luck, M., P. McBurney, and C. Preist. 2003. *Agent Technology: Enabling Next Generation Computing: A roadmap for Agent Based Computing*. AgentLink

Lueg, C., and Salomon, R. 1997. A New AI Perspective on Software Agents: Preliminary Report. In *Proceedings of the 2nd German Workshop on Artificial Life (GWAL 97), Dortmund, Germany*, 59-60.

Macaulay, L. 1996. *Requirements Engineering*. Berlin: Springer-Verlag.

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

MADKIT. 2002. http://www.madkit.org/. (accessed July 20, 2002).

Maes, P. 1991. The agent network architecture. *SIGART Bulletin* 2(4): 115-120.

Mahalingam, K., and M.N. Huhns. 1997. An ontology tool for query formulation in an agent-based context. In *Proceedings of the 2nd IFCIS International Conference on Cooperative Information Systems (CoopIS '97), Kiawah Island, USA*, 170-178.

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

Mamei, M., and F.Zambonelli. 2004. Motion Coordination in the Quake 3 Arena Environment: A Field-based Approach. In *Proceedings of the 1st International Workshop on Environments for Multiagent Systems, New York, USA*.

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

Mars, N.J.I., W.G. Ter Stal, H. De Jong, P.E. Van Der Vet, and P.-H. Speel. 1994. Semi-automatic Knowledge Acquisition in Plinius: An Engineering Approach. In *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-based Systems Workshop, Banff, Canada*, 4.1-4.15.

Mason, C.L., and R.R. Johnson. 1989. DATMS: a framework for distributed assumption based reasoning. In *Distributed Artificial Intelligence II*, ed. L. Gasser and M.N. Huhns. London: Pitman/Morgan Kaufman

Masuoka, R., and A. Sato. 1999. *Agent Description Ontology - Version 2 (CFP6 014).* http://www.flacp.fujitsulabs.com/~rmasuoka/papers/fipa99-nice-proposal.pdf. (accessed July 10, 2003).

Mena, E., A. Illarramendi, V. Kashyap, and A. Sheth. 2000. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. International Journal on Distributed and Parallel Databases 8(2): 223--271.

Miles, S., M. Joy, and M. Luck. 2000. Designing Agent-Oriented Systems by Analysing Agent Interactions. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland*, 171-184.

Mine, T., D. Matsumo, A. Kogo, and M. Amamiya. 2004. Design and Implementation of Agent Community Based Peer-to-Peer Information Retrieval Method. In *Proceedings of the 8th International Workshop on Cooperative Information Agents, Erfurt, Germany*, 31-46.

Mishra, S., and P. Xie. 2003. Interagent Communication and Synchronization Support in the DaAgent Mobile Agent-Based Computing System. *IEEE Transactions on Parallel and Distributed Systems* 14(3): 290-306.

Mitsubishi Electric Research Laboratories. 2004. *Concordia*. http://www.merl.com/projects/concordia/. (accessed April 15, 2003).

Moukas, A., and P. Maes. 1998. Amalthaea: an evolving multi-agent information filtering and discovery system for the WWW. *Autonomous Agents and Multi-agent Systems* 1(1): 59-88.

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

Mueller, H.J. 1997. Towards agent systems engineering. *International Journal on Data and Knowledge Engineering (Special Issue on Distributed Expertise)* 23: 217-245.

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

Muller, J.P. 1999. Architectures and applications of intelligent agents: A survey. *Knowledge Engineering Review* 13(4):353-380.

Muller, J.P., and M. Pischel. 1993. *The Agent Architecture InteRRaP: Concept and Application*. Technical Report RR-93-26, German Research Center for Artificial Intelligence, Saarbrucken, Germany.

Myers, K.L. 1997. *User Guide for the Procedural Reasoning System*. Technical Report, Artificial Intelligence Center, California, USA.

Nareyek, A. 2001. EXCALIBUR: Adaptive Constraint-Based Agents in Artificial Environments. http://www.ai-center.com/projects/excalibur/documentation/ (accessed December 10, 2004)

Newell, A. 1990. *Unified Theories of Cognition.* Massachusetts: Harvard University Press.

Newell, A. and H. Simon. 1963. GPS: A program that simulates human thought. In *Computers and Thought*, ed. E.A. Feigenbaum and J. Feldman. New York: McGraw-Hill.

Nissen, H.E. 1983. Subject Matter Separability in Information Systems Design Methods. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Nodine, M. H., and A. Unruh. 1997. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Language*s, 281-295.

Noy, N.F., and D.L. McGuinness. 2001. *Ontology Development 101: A Guide to Creating Your First Ontology*. Technical report KSL-01-05, Stanford Knowledge Systems Laboratory.

Nwana, H., and M. Wooldridge. 1996. Software agent technologies. *BT Technology Journal* 14(4): 68-79.

Object Agency Inc. 1995. *A Comparison of Object-Oriented Development methodologies*. http://www.toa.com/smnn?mcr.html. (accessed November 9, 2002)

Object Management Group. 2000. *Agent Technology Green Paper version 1*. http://www.jamesodell.com/ec2000-08-01.pdf. (accessed May 27, 2003).

Object Management Group. 2003. *OMG Unified Modeling Language Specification*. http://www.omg.org/technology/documents/formal/uml.htm. (accessed October 20, 2004).

O'Brien, P.D., and R.C. Nicol. 1998. FIPA - Towards a Standard for Software Agents. *BT Technology Journal* 16(3): 51-59.

Odell, J., and M.-P. Huget. 2003. *FIPA Modeling: Interaction Diagrams*. http://www.auml.org/auml/documents/ID-03-07-02.pdf. (accessed December 17, 2003).

Odell, J., H.V.D Parunak, and B. Bauer. 2000a. Extending UML for Agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Austin, USA*, 3-17.

Odell, J., H.V.D Parunak, and B. Bauer. 2000b. Representing agent interaction protocols in UML. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland*, 121-140.

Odell, J., H.V.D. Parunak, S. Brueckner, and J. Sauter. 2003b. Temporal aspects of dynamic role assignment. In *Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering (AOSE 2003), Melbourne, Australia*.

Olive, A. 1983. Analysis of Conceptual and Logical Models in Information Systems Design Methodologies. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Olle, T.W., H.G. Sol, and C.J. Tully, eds. 1983. *Information Systems Design Methodologies - A Feature Analysis*. Amsterdam: Elsevier Science Publishers.

O'Malley, S.A., and S.A. DeLoach. 2001. Determining When to Use an Agent-Oriented Software Engineering Paradigm. In *Proceedings of the 2nd Workshop on Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada*, 188-205.

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

Omicini, A., and E. Denti. 2001. From tuple spaces to tuple centres. *Science of Computer Programming* 41(3): 277-294.

Omicini, A., and F. Zambonelli. 1999. Coordination for Internet Application Development. *Autonomous Agents and Multi-Agent Systems* 2(3): 251-269.

Omicini, A., E. Denti, and A. Natali. 1995. Agent coordination and control through logic theories. In *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence, Florence, Italy*, 439 - 450.

Padgham, L., and M. Winikoff. 2002a. Prometheus: A methodology for developing intelligent agents. In *Proceedings of 3rd International Workshop on Agent-Oriented Software Engineering (AOSE-2002), Bologna, Italy*.

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

Papadopoulos, G.A. 2001. Models and technologies for the coordination of Internet agents: A survey. In *Coordination of Internet Agents: Models, Technologies, and Applications*, ed. A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, 25-56. London: Springer-Verlag.

Papadopoulos, G.A., and F. Arbab. 1998. Coordination models and languages. *Advances in Computers* 46: 329-400.

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

Parrott, L, R. Lacroix, and K. M. Wade. 2003. Design considerations for the implementation of multi-agent systems in the dairy industry. *Computers and Electronics in Agriculture* 38(2): 79-98.

Pavon, J., and J. Gomez-Sanz. 2003. Agent Oriented Software Engineering with INGENIAS. In *Proceedings of 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic,* 394-403.

Pavon, J., J. Gomez-Sanz, and R. Fuentes. 2005. The INGENIAS Methodology and Tools. In *Agent-Oriented Methodologies*, ed. B. Henderson-Sellers and P. Giorgini. Pennsylvania: Idea Group Publishing (in press).

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

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Canada*, 324-332.

Picco, G.P., A.L. Murphy, and G.-C. Roman. 1999. LIME: Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99), Los Angeles, USA*, 368-377.

Poggi, A., G. Rimassa, and P. Turci. 2002. Engineering CoMMA Multiagent System with Agent UML. In *Proceedings of the Workshop from Objects to Agents*, *Milan, Italy*.

Potts, C. 1999. ScenIC: A Strategy for Inquiry-Driven Requirements Determination. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering, Limerick, Ireland*, 58-65.

Rao, A.S., and M.P. Georgeff. 1991. Modelling rational agents within a BDI architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, *Cambridge, USA*, 473-484.

Rao, A.S., and M.P. Georgeff. 1995. BDI agents: from theory to practice. In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA*, 312-319.

Richards, D. 2000. The Reuse of Knowledge: A User-Centered Approach. *International Journal of Human Computer Studies* 52(3): 553-579.

Robinson, D.J. 2000. *A Component Based Approach to Agent Specification*. Master thesis, Air Force Institute of Technology, USA.

Rolland, C., C. Souveyet, and C.B. Achour. 1998. Guiding Goal Modeling Using Scenarios. *IEEE Transactions on Software Engineering* 24(12): 1055-1071.

Russell, S., and P. Norvig. 1995. *Artificial Intelligence: A Modern Approach*. 2nd ed. New Jersey: Prentice Hall.

Russell, S., and P. Norvig. 2003. *Artificial Intelligence: A Modern Approach*. 2nd ed. New Jersey: Prentice Hall.

Sabas, A., M. Badri, and S. Delisle. 2002. A Multidimensional Framework for the Evaluation of Multiagent System Methodologies. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2002), Orlando, USA*, 211-216.

Sargent, P. 1992. Back to school for a brand new ABC. *The Guardian*, March 12: 28.

Sathi, A., and M.S. Fox. 1989. Constraint-directed negotiation of resource reallocations. In *Distributed Artificial Intelligence II*, ed. L. Gasser and M.N. Huhns. London: Pitman/Morgan Kaufman.

Schreiber, A.T., B. J. Wielinga, R. de Hoog, J. M Akkermans, and W. Van de Velde. 1994. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert* 9(6): 28-37.

Shave, M.J.R. 1997. Ontological Structures for Knowledge Sharing. *New Review of Information Networking* 3: 125-133.

Shehory, O., and A. Sturm. 2001. Evaluation of modeling techniques for agent-based systems. In *Proceedings of the 5th International Conference on Autonomous agents, Montreal, Canada*, 624-631.

Shen, W., and D.H. Norrie. 1999. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems* 1(2): 129-156.

Shen, W., D.H. Norrie, and R. Kremer. 1999. Towards an Infrastructure for Internet Enabled Collaborative Agent Systems. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada.

Sheth, A.P., and J.A. Larson. 1990. Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys* 22(3): 183-236.

Shoham, Y. 1993. Agent-oriented programming. *AI* 60(1): 139-159.

Shoham, Y., and M. Tennenholtax. 1992. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the 10th National Conference on Artificial Intelligence*, Menlo Park, California, 276-281.

Shoham, Y., and S.B. Cousins. 1994. Logics of mental attitudes in AI: A very preliminary survey. In *Foundations of Knowledge Representation and Reasoning*, ed. G. Lakemeyer and B. Nebel, 296-309. Berlin, Heidelberg: Springer Verlag.

Siau, K., and M. Rossi. 1998. Evaluation of Information Modeling Methods – A Review. In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences*, *Hawaii, USA,* 314-322.

Silva, V., A. Garcia, A. Brandao, C. Chavez, C. Lucena, and P. Alencar. 2003. Taming Agents and Objects in Software Engineering. In *Software Engineering for Large-Scale Multi-Agent Systems – Lecture Notes in Computer Science*, ed. A. Garcia, C. Lucena, J. Castro, A. Omicini, and F. Zambonelli, 1-26. Springer Verlag.

Silva, V.T.D., and C.J.P.D. Lucena. 2004. From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language. *Autonomous Agents and Multi-Agent Systems* 8: 1-45.

Singh, D. 2000. *An agent based architecture for query planning and cost modelling of web sources*. Master Thesis, University of Georgia.

Sowa, J.F. 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove: Brooks/Cole Thompson Learning.

Standards Australia. 2004. Standard metamodel for software development methodologies (AS-4651-2004). http://www.standards.com.au/.

Steep, R., S. Cammarata, F.A. Hayes-Roth, P.W. Thorndyke, and R.B. Wesson. 1981. *Architectures for distributed intelligence for air fleet control*. Technical report R-2728-ARPA, Rand Corporation, Santa Monica, California.

Studer, R., H. Eriksson, J. H. Gennari, S. Tu, D. Fensel, and M. Musen. 1996. Ontologies and the Configuration of Problem-Solving Methods. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada*.

Stone, P., and M. Veloso. 2000. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots* 8(1): 345-383.

Stumme, G., and A. Maedche. 2001. Ontology merging for federated ontologies on the semantic web. In *Proceedings of Workshop on Ontologies and Information Sharing (IJCAI' 01), Seattle, USA*.

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

Sundsted, T. 1998. An introduction to agents. *Javaworld*, June 6. http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html. (accessed August 25, 2002).

Sycara, K. 1998a. Resolving goal conflict via negotiation. In *Proceedings of the 7th National Conference on Artificial Intelligence, Minnesota, USA*, 245-250.

Sycara, K. 1998b. Multiagent Systems. *AI Magazine* 19(2): 79-92.

TACOMA – Tromso and Cornell Moving Agents. Tromso: University of Tromso. http://www.tacoma.cs.uit.no/. (accessed July 10, 2003).

Tahara, Y., A. Ohsuga, and S. Honiden. 1999. Agent system development based on agent patterns. In *Proceedings of the 21st International Conference on Software Engineering, California, USA*, 356–367.

Tamma, V., M. Wooldridge, and I. Dickinson. 2002a. An ontology for automated negotiation. In *Proceedings of the International Workshop on Ontologies in Agent Systems* (OAS'02), Bologna, Italy.

Tamma, V., M. Wooldridge, and I. Dickinson. 2002b. An ontology based approach to automated negotiation. In *Proceedings of the 4th Workshop on Agent Mediated Electronic Commerce (AMEC IV)*, Bologna, Italy.

Tamma, V., S. Phelps, I. Dickinson, and M. Wooldridge. 2005. Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence* 18: 223-236.

Taveter, K. 1998. Intelligent Information Retrieval Based on Interconnected Concepts and Classes of Retrieval Domains. In *Proceedings of the 8th DELOS Workshop User Interface in Digital Libraries, Stockholm, Sweden*, 39-43.

Telecom Italia Lab. 2004. *Java Agent Development Framework – an Open Source platform for peer-to-peer agent based applications*. http://jade.tilab.com/. (accessed August 14, 2002).

Thangarajah, J., L. Padgham, and J. Harland. 2002. Representation and Reasoning for Goals in BDI Agents. In *Proceedings of the twenty-fifth Australasian conference on Computer science - Volume 4, Melbourne, Australia,* 259–265.

Tolkdorf, R. 1997. Berlinda: an object-oriented platform for implementing coordination languages in Java. In *Proceedings of the 2nd International Conference on Coordination Languages and Models, Berlin, Germany*, 430-433

Tout, H. 2001. An Informational Model for Cooperative Information Gathering. In *Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents, Montreal, Canada*.

Tran, Q.N., G. Low, and M.A. Williams. 2003. A Feature Analysis Framework for Evaluating Multi-agent System Development Methodologies. In *Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems ISMIS'03, Maebashi, Japan*, 613-617.

Tran, Q.N., G. Low, and M.A. Williams. 2004. A Preliminary Comparative Feature Analysis of Multi-agent Systems Development Methodologies. In *Proceedings of the 6th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2004), Riga, Latvia*, 386-397.

Tran, Q.N., and G. Low. 2005. Comparison of Methodologies. In *Agent-Oriented Methodologies*, ed. B. Henderson-Sellers and P. Giorgini. Pennsylvania: Idea Group Publishing (in print).

Tveit, A. 2001. A survey of Agent-Oriented Software Engineering. In *Proceedings of the 1st NTNU Computer Science Graduate Student Conference, University of Science and Technology, Norway*.

UMBC Lab for Advanced Information Technology. n.d.a. *UMBC KQML Web*. http://www.cs.umbc.edu/kqml/. (accessed January 6, 2003).

UMBC Lab for Advanced Information Technology. n.d.b. *UMBC AgentNews*. http://agents.umbc.edu/agentnews/ (accessed April 22, 2003).

UMBC Lab for Advanced Information Technology. n.d.c. *The Software Agents Mailing List*. http://www.csee.umbc.edu/agentslist/ (accessed May 5, 2003).

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

Uschold, M., and M. King. 1995. Towards A Methodology for Building Ontologies. In *Proceedings of IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada*.

Valckenaers, P., H. Van Brussel, H. Karuna, M. Kollingbaum, O. Bochmann. 2002. Multi-Agent Manufacturing Control Using Stigmergy. In *Proceedings of the 15th IFAC World Congress on Automatic Control, Barcelona, Spain*.

van Breemen, A.J.N. 2002. Integrating Agents in Software Applications. In *Proceedings of Workshops at Net.Objectdays, Erfurt, The Netherlands*, 278-289.

van Heijst, G., A. Schreiber, and B. Wielinga. 1997. Using Explicit Ontologies in KBS Development. *International Journal of Human computer studies* 46: 183-292.

van Lamsweerde, A. 2001. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, *Toronto, Canada*, 249-263.

Vere, S., and T. Bickmore. 1990. A Basic Agent. *Computational Intelligence* 6: 41-60.

Vidal, J.M., P.A. Buhler, and M.N. Huhns. 2001. Inside an Agent. IEEE Internet Computing 5(1): 82-86.

Vlado, K. 1998. *Multi-agent systems for Internet information retrieval using natural language processing*. Master thesis., University of Waterloo.

Wache, H., U. Visser, U., and T. Scholz. 2001. Ontology construction – an iterative and dynamic task. In Proceedings *of the 15th International Florida Artificial Intelligence Research Society Conference, Florida, USA*, 445-449.

Wasserman, A.I., Freeman, P. and M. Porcella. 1983. Characteristics of Software Development Methodologies. In *Information Systems Design Methodologies - A Feature Analysis*, ed. T.W. Olle, H.G. Sol and C.J. Tully, 63-85. Amsterdam: Elsevier Science Publishers.

Wegner, P. 1996. Coordination as constrained interaction. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION '96)*, *Cesena, Italy,* 28-33.

Weiss, G., eds. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Massachusetts: MIT Press.

Weyns, D., and T. Holvoet. 2003. Synchronous versus Asynchronous Collaboration in Situated Multi-agent Systems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia*, 1156-1157.

Weyns, D., H.V.D. Parunak, F. Michel, T. Holvoet, and J. Ferber. 2004. Environments for Multiagent Systems State-of-the-Art and Research Challenges. In *Proceedings of the 1st International Workshop on Environments for Multiagent Systems, New York, USA*.

Wiegers, K. 2003. *Software Requirements*. 2nd ed. Washington: Microsoft Press.

Winikoff, M., and L. Padgham. 2004. The Prometheus Methodology. In *Methodologies and Software Engineering for Agent Systems*. *The Agent-Oriented Software*

Engineering handbook, ed. F. Bergenti, M.P. Gleizes, and F. Zambonelli, Chapter 11. Kluwer Academic Publishers.

Winikoff, M., L. Padgham, and J. Harland. 2001. Simplifying the Development of Intelligent Agents. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, *Adelaide, Australia*, 557-568.

Wood, S. 1993. *Planning and Decision Making in Dynamic Domains*. Chchester: Ellis Horwood.

Wood, M. F. 2000. *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*. Master thesis, Air Force Institute of Technology, USA.

Wood, M., and S.A. DeLoach. 2000a. An Overview of the Multiagent Systems Engineering Methodology. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland*, 207-221.

Wood, M., and S.A. DeLoach. 2000b. Developing Multiagent Systems with agentTool. In *Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages, Boston, USA*, 46-60.

Wood, B., R. Pethia, L.R. Gold, and R. Firth. 1988. *A Guide to the Assessment of Software Development Methods*. Technical Report CMUSEI-88-TR-8, SEI, Software Engineering Institute, Carnegie Mellon University.

Wooldridge, M. 1997. Agent-based software engineering. *IEEE Proceedings on Software Engineering* 144(1): 26-37.

Wooldridge, M. 1999. Intelligent Agents. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, ed. G. Weiss, 27-77. London: The MIT Press.

Wooldridge, M. 2002. *An Introduction to MultiAgent Systems*. Chichester: John Wiley & Sons.

Wooldridge, M., and N.R. Jennings. 1995. Agent Theories, Architectures and Languages: a survey. In *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents, Amsterdam, The Netherlands*, 1-39.

Wooldridge, M., and N.R. Jennings. 1998. Pitfalls of Agent-Oriented Development. In *Proceedings of the 2nd International Conference on Autonomous Agents*, *Minneapolis, USA*, 385-391.

Wooldridge, M., and P. Ciancarini. 2000. Agent-Oriented Software Engineering: The State of the Art. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, *Limerick, Ireland*, 1-28.

Wooldridge, M., N.R. Jennings, and D. Kinny. 1999. A Methodology for Agent-Oriented Analysis and Design. In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents '99), Seattle, Washington*, 69-76.

Wooldridge, M., N.R. Jennings, and D. Kinny. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems* 3(3): 285-312.

Wray, R., R. Chong, J. Phillips, S. Rogers, and B. Walsh. n.d. *A Survey of Cognitive and Agent Architectures*. http://ai.eecs.umich.edu/cogarch0/index.html. (accessed April 20, 2002)

Wu, X., and A.C. Esterline. 1999. Representing Multi-agent Plans Using Statecharts with Explicit Aggregation. In *Proceedings of ADMI-99 Minority Institutions Computing Conference, Duluth, USA*.

Xu, D., R. Volz, T. Ioerger, and J. Yen. 2002. Modeling and verifying multi-agent behaviors using predicate/transition nets. In *Proceedings of the 14th international conference on Software Engineering and Knowledge Engineering, Ischia, Italy*, 193-200.

Yan, Q., L.J. Shan, and X.J. Mao. 2003. *RoMAS: A Role-Based Modeling Method for Multi-Agent System*. In *Proceedings of International Conference on Active Media Technology, Chongqing, China*.

Yourdon, E. 1989. *Modern Structured Analysis*. Englewood Cliffs, New Jersey: Yourdon Press.

Yu, E. 1995. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Canada.

Yuan, S.T. 1999. Ontology-Based Agent Community for Information Gathering and Integration. *Proceedings of the National Science Council: Physical Science and Engineering (Part A)* 23(6): 766-780.

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

Zambonelli, F., N. Jennings, and M. Wooldridge. 2001a. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering* 11(3): 303-328.

Zambonelli F., N. Jennings, and M. Wooldridge. 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology* 12(3): 317-370.

Zambonelli, F., N.R. Jennings, A. Omicini, and M. Wooldridge. 2001b. Agent-oriented software engineering for Internet applications. In *Coordination of Internet Agents*, ed. A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, 326-346. New York: Springer-Verlag.

Zhou, J., M. Zhou, and Q. Wu. 2000. An Agent Framework Based on Distributed Object. In *Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia'00), Xi'an, China*, 188-194.

APPENDIX A

ADVERTISEMENT FOR SURVEY RECRUITMENT

Dedicated Agent Researchers and Developers Needed!

If you have knowledge and/or experience in agent-oriented software engineering, please take your time to complete a Multi-Agent Development Methodology Survey, which is available at

http://129.94.244.146/personal/numi+tran/surveyq.nsf/survey/.

Access password: MAS

The survey is part of a doctoral research project and its purpose is to gather your professional opinions and suggestions on what generic features, process steps and modelling concepts should be part of a methodology for developing Multi-Agent Systems. The features, steps and modelling concepts must be ranked and rated with regard to their importance. The survey can be completed in stages and will take approximately 30-40 minutes to finish. The Closing Date is 31 Jan 2003.

The survey is demanding, but the acquired information will prove to be invaluable to the Agent community. Participation is completely voluntary and you can remain anonymous. Contact: Quynh Nhu Numi Tran mailto:numitran@unsw.edu.au.

APPENDIX B

ONLINE SURVEY QUESTIONNAIRE

START-UP PAGE

Methodology for Multi-Agent Systems Development

Please enter password:

Password protection is implemented to prevent unauthorized access. It is NOT used for identification purposes.

Continue

WELCOME PAGE[86]

Methodology for Multi-Agent Systems Development

(For those who wish to continue their partially completed survey, please click here[87])

Firstly, thank you in anticipation for participating in this survey. We appreciate you giving up some of your time to assist us in this study.

PURPOSE OF THE SURVEY

The survey is the basis for a doctoral research project at the School of Information Systems, Technology and Management - The University of New South Wales. The research's aim is to **propose a software engineering methodology for developing Multi-Agent Systems** (MAS). The intention is to reuse and enhance the existing techniques and model definitions offered by the current agent-oriented software engineering methodologies where appropriate, and introduce new techniques and model definitions where necessary.

[86] This page is loaded when button "Continue" in "Start-up page" is clicked.
[87] This is a hyperlink which when clicked will load "Survey Return page".

The survey aims to gather your professional opinions and suggestions on what **features**, **steps** and **modelling concepts** should be supported by an Agent-Oriented Software Engineering (AOSE) methodology for developing MAS.

If you wish, the findings of the survey and the final results of the research will be forwarded to you when available.

WHO SHOULD PARTICIPATE IN THE SURVEY

- Project managers, system analysts, system designers or system developers who have been involved in developing at least one MAS.
- Researchers/academics whose area of interest is MAS development.

RESEARCH CONTACTS

If you have any questions on the survey, please contact:

Miss **Quynh-Nhu (Numi) Tran**
School of Information Systems, Technology and Management
The University of New South Wales
numitran@unsw.edu.au

Prof. **Graham Low**
Head of School
School of Information Systems, Technology and Management
The University of New South Wales
g.low@unsw.edu.au

Continue

INSTRUCTIONS PAGE[88]

Methodology for Multi-Agent Systems Development

GUIDELINES FOR QUESTIONNAIRE

The survey questionnaire consists of 5 parts.

- **Part 1** collects your demographic and background information.
- **Part 2** gathers your opinions on a list of *features* in terms of how important these features are to a "standard" MAS development methodology.
- **Part 3** seeks your opinions on a list of *steps* with regard to how important these steps are to a "standard" MAS development process.

[88] This page is loaded when button "Continue" on "Welcome page" is clicked.

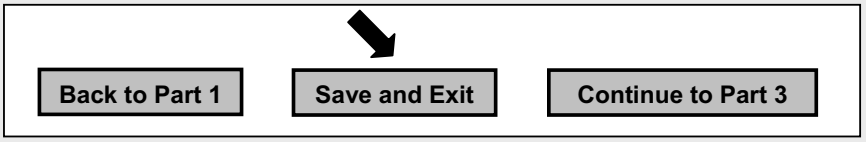
- **Part 4** obtains your opinions on a list of *concepts* with respect to how important these concepts are to models of a "standard" MAS development methodology.
- **Part 5** asks for your recommendations on various issues relating to the construction of a MAS development methodology.

The whole questionnaire takes approximately 30-40 minutes in total to complete.

IMPORTANT NOTES

1. You do NOT have to complete the whole questionnaire in one go. After starting the survey, you can leave the questionnaire at any point and come back later for further completion.

To save a partially completed questionnaire, you just need to click on the button "**Save and Exit Survey**" at the end of each part (as shown on the picture below).



When you leave a partially completed survey, you will be given an ID Number which allows you to return to the questionnaire later. You can save and go back to your partially completed questionnaire as many times as you like until you finish the survey.

2. It is required to **enable Javascript** on your browser to allow the questionnaire to function.

3. Please navigate between the questionnaire parts using the **navigation buttons at the end of each part**, and ***not*** the browser's "Back" and "Forward".

Start

SURVEY PART 1 PAGE[89]

Methodology for Multi-Agent Systems Development

PART 1

This part of the survey questionnaire aims to gather some background information about you and your experience with Multi-Agent Systems (MAS) and Multi-Agent System development. You can remain anonymous if you wish.

[89] This page is loaded when button "Start" on "Instructions page" is clicked.

If you would like to remain anonymous, please tick below:

☐ Anonymous

1. Name: ☐
2. Organisation: ☐
3. Department: ☐
4. Email: ☐

(required if wish to receive feedback on the survey's findings and/or final result of the research)

5. Please tick if you would like to be informed of

☐ Survey's findings ☐ Research's final result (i.e. documentation of the proposed MAS development methodology

Please provide the following information even if you wish to remain anonymous.

6. What is your IT role of work? (multiple choices are allowed)

☐ Project manager ☐ Programmer
☐ System analyst ☐ Researcher/Academic
☐ System developer/developer ☐ Other. Please specify below ☐

7. How would you describe your current theoretical knowledge of Multi-Agent Systems (MAS)?

Low ○1 ○2 ○3 ○4 ○5 ○6 ○7 Extensive

8. How would you describe your current industrial experience with MASs?

Low ○1 ○2 ○3 ○4 ○5 ○6 ○7 Extensive

9. How would you describe your current theoretical knowledge of MAS development?

Low ○1 ○2 ○3 ○4 ○5 ○6 ○7 Extensive

10. How would you describe your current industrial experience with MAS development?

Low ○1 ○2 ○3 ○4 ○5 ○6 ○7 Extensive

11. Have you been involved in developing any MAS? ○Yes ○No

If Yes,

11a. How many MAS development projects have you been involved with? ☐

11b. What is the number of agents in these MASs? (multiple choices are allowed)

☐ Fewer than 10 ☐ 10-50 ☐ 51-99 ☐ 100 or more

11c. How do you perceive the average level of complexity of these MASs? (e.g. in terms of agent cognitive ability and intelligence, agent interactions and system dynamics)

Very low ○1 ○2 ○3 ○4 ○5 ○6 ○7 Very high

11d. What is/are the application area(s) of these MASs? (multiple choices are allowed)

☐ Personal Assistance ☐ Information Gathering & Management
☐ Electronic Commerce ☐ Simulation
☐ Automated Control/Monitoring ☐ System and Network Management
☐ Others. Please specify ☐

11e. Did you follow any system development methodology in any of these projects?

○ Yes ○ No

If yes, what is/are the methodology(ies)? (Please provide as many details as you can on the methodology's names, authors and/or references)

11f. Have you been involved in developing Ontology-Based MASs (i.e. MASs whose design specification explicitly includes ontologies, and ontologies are used by agents at run-time to facilitate the operation of MAS)

○ Yes ○ No

Save and Exit **Continue to Part 2**

(4 more parts to go)

SURVEY PART 2 PAGE[90]

Methodology for Multi-Agent Systems Development

PART 2

This part of the survey questionnaire aims to gather your opinions and suggestions on *what generic features should be offered* by an AOSE methodology for MAS development. (From here on, the term "AOSE methodology" will be used to mean "AOSE methodology for MAS development").

INSTRUCTIONS

Below you will find a list of features that may be offered by an AOSE methodology. Although all of them are important, some of the features are more important to be supported by an AOSE methodology than the others. Thus, we ask for your opinion on this prioritisation.

Desirable features of an AOSE process

Please order rank the following features in terms of their importance to be provided by an AOSE process. Please also indicate the rating of their importance.

Note: Please try to give each feature a unique ranking. But if you cannot differentiate, features can be ranked equally.

[90] This page is loaded when button "Continue to Part 2" on "Survey Part 1 page" is clicked.

	Most important 1st	2nd	3rd	4th	5th	*Least important* 6th	*Rating of importance*
1. Specification of a system development lifecycle	○	○	○	○	○	○	
2. Support for verification and validation (more) [92]	○	○	○	○	○	○	
3. Specification of steps for the development process	○	○	○	○	○	○	
4. Specification of models and/or notational components to be generated from each process step	○	○	○	○	○	○	
5. Specification of techniques and heuristics for performing each process step and for producing each model	○	○	○	○	○	○	
6. Support for refinability (more)	○	○	○	○	○	○	

Desirable features of AOSE model definitions

Please order rank the following features in terms of their importance to be provided by AOSE model definitions. Please also indicate the rating of their importance.

Note: Please try to give each feature a unique ranking. But if you cannot differentiate, features can be ranked equally.

	Most important 1st	2nd	3rd	4th	5th	6th	*Least important* 7th	8th	*Rating of importance*
1. High degree of completeness/expressiveness (more)	○	○	○	○	○	○	○	○	
2. High degree of formalisation/preciseness (more)	○	○	○	○	○	○	○	○	
3. Provision of guidelines/logics for model derivation (more)	○	○	○	○	○	○	○	○	
4. Guarantee of consistency (more)	○	○	○	○	○	○	○	○	
5. Support for modularity (more)	○	○	○	○	○	○	○	○	
6. Manageable number of concepts in each model and each notational component	○	○	○	○	○	○	○	○	
7. Models expressed at various levels of abstraction and detail	○	○	○	○	○	○	○	○	
8. Support for reuse	○	○	○	○	○	○	○	○	

Agent properties desirable to be captured/represented by AOSE models

Please order rank the following agent properties in terms of their importance to be captured/represented by AOSE models. Please also indicate the rating of their importance.

Note: Please try to give each agent property a unique ranking. But if you cannot differentiate, agent properties can be ranked equally.

---

[91] This is a combo box which contains 5 possible ratings: "Very High", "High", "Medium", "Low" and "Very Low".

[92] "more" is a programmed hyperlink which when clicked will open a small pop-up screen to show more explanation about a particular feature.

	Most important							*Least important*	*Rating of importance*
	1st	2nd	3rd	4th	5th	6th	7th	8th	
1. Autonomy (more)	○	○	○	○	○	○	○	○	
2. Adaptability (more)	○	○	○	○	○	○	○	○	
3. Cooperative behaviour (more)	○	○	○	○	○	○	○	○	
4. Inferential capability (more)	○	○	○	○	○	○	○	○	
5. Knowledge-level communication ability (more)	○	○	○	○	○	○	○	○	
6. Personality (more)	○	○	○	○	○	○	○	○	
7. Reactivity (more)	○	○	○	○	○	○	○	○	
8. Deliberative behaviour (more)	○	○	○	○	○	○	○	○	

Desirable features of an AOSE methodology as a whole

Please order rank the following features in terms of their importance to be supported by an AOSE methodology. Please also indicate the rating of their importance.

Note: Please try to give each feature a unique ranking. But if you cannot differentiate, features can be ranked equally.

	Most important				*Least important*		*Rating of importance*
	1st	2nd	3rd	4th	5th	6th	
1. Support for open systems (more)	○	○	○	○	○	○	
2. Support for dynamic systems (more)	○	○	○	○	○	○	
3. Support for agility and robustness (more)	○	○	○	○	○	○	
4. Support for heterogeneous systems (more)	○	○	○	○	○	○	
5. Support for mobile agents (more)	○	○	○	○	○	○	
6. Support for ontology-based MAS development (more)	○	○	○	○	○	○	

YOUR SUGGESTIONS ON FEATURES

If you have any suggestions on other desirable features to be supported by a MAS methodology, please provide details below:

Back to Part 1 **Save and Exit** **Continue to Part 3**

(3 more parts to go)

SURVEY PART 3 PAGE[93]

`Methodology for Multi-Agent Systems Development

PART 3

Typically, an AOSE methodology should present a system development process, which involves *steps* to guide the system developers through the process. This part of the survey questionnaire aims to gather your opinions and suggestions on *what steps should be included* in an AOSE process.

INSTRUCTIONS

Please order rank the following steps in terms of their importance to be provided by an AOSE process. Please also indicate the rating of their importance.

Note: Please try to give each step a unique ranking. But if you cannot differentiate, steps can be ranked equally.

Problem Domain Analysis steps

	Most important 1st	2nd	3rd	4th	*Least important* 5th	*Rating of importance*
1. Identify system functionality	○	○	○	○	○	
2. Specify use case scenarios	○	○	○	○	○	
3. Identify roles	○	○	○	○	○	
4. Identify agent classes	○	○	○	○	○	
5. Model domain conceptualisation	○	○	○	○	○	

Agent Interaction Design steps

	Most important 1st	2nd	3rd	*Least important* 4th	*Rating of importance*
1. Specify acquaintances between agent classes	○	○	○	○	
2. Define interaction protocols	○	○	○	○	
3. Define content of exchanged messages	○	○	○	○	
4. Specify agent communication language	○	○	○	○	

[93] This page is loaded when button "Continue to Part 3" on "Survey Part 2 page" is clicked.

Agent Internal Design steps

	Most important 1st	2nd	*Least important* 3rd	*Rating of importance*
1. Specify agent architecture	○	○	○	
2. Define agent informational constructs (i.e. beliefs)	○	○	○	
4. Define agent behavioural constructs (e.g. goals, plans, actions, services)	○	○	○	

Overall System Design steps

	Most important 1st	2nd	3rd	4th	5th	6th	*Least important* 7th	*Rating of importance*
1. Specify system architecture (more)	○	○	○	○	○	○	○	
2. Specify organisational structure/inter-agent control regimes	○	○	○	○	○	○	○	
3. Model MAS environment (more)	○	○	○	○	○	○	○	
4. Specify agent-environment interaction mechanism	○	○	○	○	○	○	○	
5. Specify agent inheritance and aggregation	○	○	○	○	○	○	○	
6. Instantiate agent classes	○	○	○	○	○	○	○	
7. Specify agent instances deployment	○	○	○	○	○	○	○	

YOUR SUGGESTIONS ON STEPS

If you have any suggestions on other desirable steps to be included in an AOSE process, please provide details below:

Back to Part 2 **Save and Exit** **Continue to Part 4**

(2 more parts to go)

SURVEY PART 4 PAGE[94]

Methodology for Multi-Agent Systems Development

PART 4

Typically, besides a system development process, a MAS methodology should also present a set of model definitions which capture/represent various concepts. This part of the survey questionnaire aims to gather your opinions and suggestions on *what concepts should be captured/ represented* in AOSE models.

[94] This page is loaded when button "Continue to Part 4" on "Survey Part 3 page" is clicked.

INSTRUCTIONS

Please order rank the following concepts in terms of their importance to be captured/ represented in AOSE models. Please also indicate the rating of their importance.

Note: Please try to give each step a unique ranking. But if you cannot differentiate, concepts can be ranked equally.

Problem Domain concepts

	Most important 1st	2nd	3rd	*Least important* 4th	*Rating of importance*
1. System functionality	○	○	○	○	
2. Use case scenario	○	○	○	○	
3. Role	○	○	○	○	
4. Domain conceptualisation	○	○	○	○	

Agent concepts

	Most important 1st	2nd	3rd	4th	5th	6th	*Least important* 7th	*Rating of importance*
1. Agent-role assignment	○	○	○	○	○	○	○	
2. Agent goal/task	○	○	○	○	○	○	○	
3. Agent belief/knowledge	○	○	○	○	○	○	○	
4. Agent plan/reasoning rule/problem solving method	○	○	○	○	○	○	○	
5. Agent capability/service	○	○	○	○	○	○	○	
6. Agent percept/event	○	○	○	○	○	○	○	
7. Agent architecture	○	○	○	○	○	○	○	

Agent Interaction concepts

	Most important 1st	2nd	*Least important* 3rd	*Rating of importance*
1. Agent acquaintance	○	○	○	
2. Interaction protocol	○	○	○	
3. Content of exchanged message	○	○	○	

Overall System Design concepts

	Most important 1st	2nd	3rd	4th	5th	6th	7th	*Least important* 8th	*Rating of importance*
1. System architecture	○	○	○	○	○	○	○	○	
2. Organisational structure/ inter-agent control regimes	○	○	○	○	○	○	○	○	
3. Environment resource/facility	○	○	○	○	○	○	○	○	
4. Agent aggregation relationship	○	○	○	○	○	○	○	○	
5. Agent inheritance relationship	○	○	○	○	○	○	○	○	
6. Agent instantiation	○	○	○	○	○	○	○	○	
7. Agent instance deployment	○	○	○	○	○	○	○	○	

YOUR SUGGESTIONS ON STEPS

If you have any suggestions on other desirable concepts to be captured/represented by AOSE models, please provide details below:

Back to Part 3 **Save and Exit** **Continue to Part 5**

(1 more part to go)

SURVEY PART 5 PAGE[95]

Methodology for Multi-Agent Systems Development

PART 5

Please provide your opinions and recommendations on the following issues.

1. If an AOSE methodology must incorporate a system development lifecycle (SDLC), which SDLC do you think it should be (e.g. waterfall)?

Please list reasons for your answer (if any)

2. Please indicate the importance of an AOSE methodology to commit to a particular agent architecture (e.g. BDI architecture).

Rating of importance[96] [▼]

Please list reasons for your answer (if any)

[95] This page is loaded when button "Continue to Part 5" on "Survey Part 4 page" is clicked.

[96] This is a combo box which contains 5 possible ratings: "Very High", "High", "Medium", "Low" and "Very Low".

3. Which approach do you think an AOSE methodology should adopt for the development of MAS?

○ Role-oriented approach (more)

○ Non-role-oriented approach (more)

If you selected the second choice, please indicate the constructs

Please list reasons for your answer (if any)

Back to Part 4 | Save and Exit | Submit Survey

THANK YOU PAGE[97]

Thank You!

Thank you for your time and effort in completing this survey! Your contribution is highly appreciated, and will enable us to develop an effective software engineering methodology for developing Multi-Agent Systems.

If you chose to be contacted for follow-up session(s) or survey/research findings, look forward to contact you again.

Research Contacts:

Miss **Quynh-Nhu (Numi) Tran**
School of Information Systems, Technology and Management
The University of New South Wales
numitran@unsw.edu.au

Prof. **Graham Low**
Head of School
School of Information Systems, Technology and Management
The University of New South Wales
g.low@unsw.edu.au

[97] This page is loaded when button "Submit Survey" on "Survey Part 5 page" is clicked.

SURVEY RETURN PAGE[98]

Methodology for Multi-Agent Systems Development

To return to your partially completed survey questionnaire,
please enter your ID Number:

Return to Survev

SURVEY SAVE AND EXIT PAGE[99]

Methodology for Multi-Agent Systems Development

Thank you for your partial completion of the survey. Your responses have been saved. Please return at a later time to continue with your saved survey. Your ID Number is:

[An ID Number is to be shown here]

Return to Survey

[98] This page is loaded when hyperlink "here" on "Welcome page" is clicked. It will direct the respondent back to the survey part which he had partially completed.
[99] This page is loaded whenever button "Save and Exit" on other pages is clicked.

APPENDIX C

DEMOGRAPHIC AND PROFESSIONAL CHARACTERISTICS OF SURVEY RESPONDENTS

This appendix presents the descriptive statistics of seven variables that pertained to the demographic and professional characteristics of survey respondents. The other four demographic variables, namely "*Theoretical knowledge of MAS*", "*Theoretical knowledge of MAS development*", "*Industrial experience with MAS*" and "*Industrial experience with MAS development*", are analysed in Section 5.3.4.1.

Variable "Field of work"

A majority of the respondents worked in the field of research/academia (Figure AppendixC.1). Four respondents were involved in multiple fields, including two who worked as both researcher and system developer/developer, one who worked as both researcher and project manager, and two who worked concurrently as researcher, programmer and system developer/developer.

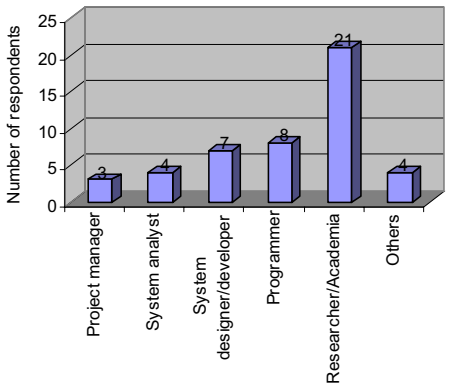


Figure AppendixC.1 – Survey respondents' field of work

Variable "Involvement in MAS development projects"

A high proportion of the respondents (33 out of 41) had participated in at least one MAS development project (Figure AppendixC.2). Out of these respondents, twenty-six had engaged in 1-5 projects, while two had participated in more than 10 projects.

An exploratory analysis was conducted to discover if "*Involvement in MAS development projects*" had any significant impact on the respondents' "rating of importance" and "order ranking" of features, steps and modelling concepts in Parts 2, 3 and 4 of the survey. For this analysis, Mann-Whitney Tests[100] were performed to compare the "ratings of importance" and "order ranks" from two different groups of respondents – those who had participated in at least one MAS project and those who had not. The comparisons were carried out for all features, steps and modelling concepts. However, no significant difference was detected between the two groups at a significance level of 5%. Consequently, the data obtained from both groups of respondents was combined to form the final survey data set.

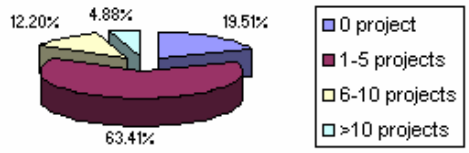


Figure AppendixC.2 – Survey respondents' involvement in MAS development projects

Variable "Size of past MAS projects"

This variable, as well as the succeeding four variables, were collected from respondents who had been involved in at least one MAS development project (i.e. 33 respondents). Most respondents had developed small-sized to medium-sized MASs, i.e. MASs with less than 10 agents and from 10 to 50 agents respectively (Figure AppendixC.3). Six respondents were involved in multiple MAS projects of different sizes.

[100] Mann-Whitney Test was chosen because it is a well-known test for comparing two independent samples with continuous ordinal data (Leach 1979). The two samples in this case were the rating (or order ranking) data of the respondents who have involved in MAS projects and those who have not, with regard to a particular feature, step or concept. The samples are thus independent and the collected data is ordinal and continuous.

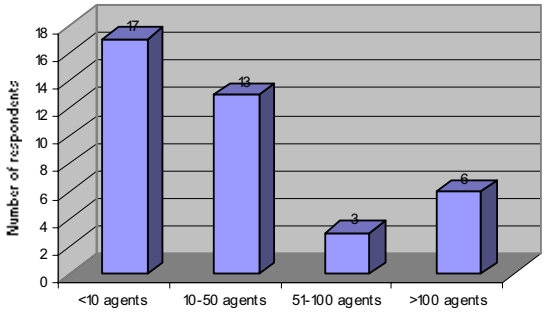


Figure AppendixC.3 – Size of past MAS projects

Variable "Level of complexity of past MAS projects"

The *median* complexity of the past MAS development projects that the respondents had been involved in was "5" (on a 7-point Likert scale ranging from "Very low" to "Very high"), indicating an average level of medium complexity for the involved MAS development projects (Figure AppendixC.4).

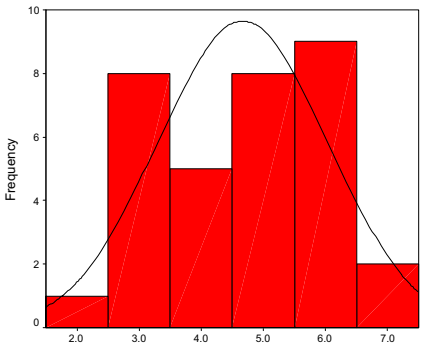


Figure AppendixC.4 – Level of complexity of involved MAS projects

Variable "Application areas of past MAS projects"

A large number of involved MAS projects were in the areas of Information Gathering/Management, Simulation and Personal Assistant (Figure AppendixC.5). Nine respondents had been involved in projects of two different application areas, while six respondents were involved in three different application areas.

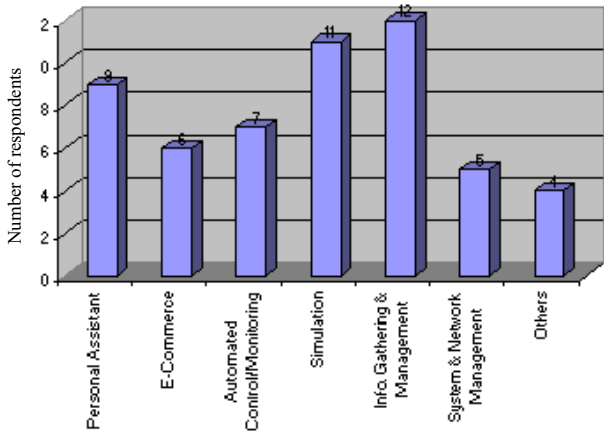


Figure AppendixC.5 – Application areas of involved MAS projects

Variable "Adoption of AOSE methodologies in past MAS projects"

A large proportion of the respondents (26 out of 33[101]) did not follow any AOSE methodology or framework in their past MAS projects. Of the respondents that did follow a methodology or framework, the listed AOSE methodologies and frameworks were:

- PROMETHEUS (Padgham and Winikoff 2002a; Padgham and Winikoff 2002b);
- GAIA (Wooldridge et al. 1999; Wooldridge et al. 2000);
- INGENIAS (Pavon and Gomez-Sanz 2003; Pavon et al. 2005);
- RoMAS (Yan et al. 2003);
- MESMA (Cuesta et al 2002);
- JADE framework (Telecom Italia Lab 2004);
- FIPA specifications (FIPA n.d.b); and
- Adapted OO frameworks and techniques for agent-oriented development, including Rational Unified Process, UML and design patterns.

Variable "Involvement in Ontology-Based MAS development projects"

Of the respondents who had been involved in MAS development projects, only a small proportion had experienced the construction of Ontology-Based MASs (15 out of 33).

[101] The proportion was calculated out of the respondents who had been involved in at least one MAS development project, i.e. 33 respondents.

APPENDIX D

EVALUATION OF EXISTING MAS DEVELOPMENT METHODOLOGIES

This appendix presents the evaluation of 16 AOSE methodologies according to criterion "*Support for steps"* (cf. Table 5.22) and six other criteria relating to the AOSE process, namely (cf. Table 5.21):

- "*Specification of model kinds and/or notational components*";
- "*Definition of inputs and outputs of steps*";
- "*Specification of techniques and heuristics*";
- "*Ease of understanding of techniques*";
- "*Usability of techniques*"; and
- "*Provision of examples for techniques*".

All of these criteria used the list of steps in Table 5.22 as yardsticks.

If a methodology was found to provide "*support for [a particular] step*", this support was evaluated as "explicit" ("E") or "implicit" ("I") (cf. Tables 8.9a to 8.9p). The former applies if the methodology specifies the step as a distinct activity in its development process. The latter incurs when the step is implicitly fulfilled as part of another step, or only briefly mentioned by the methodology. If a step is specified as part of, or in conjunction with, another step, this other step is indicated in the square brackets [].

Evaluation for "*Definition of inputs and outputs of steps*" is denoted as "I" if only inputs are specified, "O" if only outputs are specified, and "B" if both inputs and outputs are defined. Criterion "*Specification of techniques and heuristics*" was assessed in two parts: "*Techniques used to perform each step*" and "*Techniques used to produce each model or notation component*" (cf. Table 5.21 in Section 5.4.1). "*Ease of understanding of techniques*" and "*Usability of techniques*" were evaluated as either "high" ("H"), "medium" ("M") or "low" ("L").

Table AppendixD.1 – Support for steps of MASE

MASE								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Goal hierarchy diagram	B	Analyze initial system specifications, e.g. technical documents, user stories, and se cases. Identify tasks that each role should perform to achieve goals	Hierarchically organise goals in the order of importance. All sub-goals should relate functionally to their parent. Attach tasks to roles in Extended role diagram	H	H	Y
2. Specify use case scenarios	E	Use case diagrams	B	Conventional OO techniques	Conventional OO techniques	H	H	Y
3. Identify roles	E	Role diagram	B	Typically one-to-one mapping between goals and roles	Show roles, their related goals, and communication paths between roles	H	H	Y
4. Identify agent classes	E	Agent class diagram	B	Group roles into agent classes	Show agent classes, related roles, and acquaintances between agents	H	H	Y
5. Model domain conceptualisation	E		B	Define purpose and scope of the ontology, collect data from the information domain, form the initial ontology, and finally refine the ontology into a complete version		H	H	Y
6. Specify acquaintances between agent classes	I [5]	Agent class diagram	B	Any communication paths between 2 roles indicate acquaintances between their respective agent classes		H	H	Y
7. Define interaction protocols	E	Communication class diagrams	B	Specify conversations between agents by analyzing inter-role interactions in use cases, and task descriptions in Task state diagrams	Produce a Communication class diagram (which is a finite state machine) for each participant in the conversation	H	H	Y
8. Define content of exchanged messages	E[8]	Communication class diagrams	B	Analyze inter-role interactions in use cases, and task descriptions in Task state diagrams	Model messages as transitions between states in Communication class diagram. Specify performatives and parameters	H	H	Y
9. Specify ACL								
10.Specify agent architecture	E	Agent class architecture diagram	O	Refer to the work of (Robinson 2000)	Refer to the work of (Robinson 2000)	H	M	Y
11.Define agent informational constructs (i.e. beliefs)	I[4]	Task state diagram	B	Specify how a role/agent can fulfill a task with a structured set of activities and communications. This implicitly represents an agent's plan for achieving tasks.	Depict the task processing as a finite state machine	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes	E	Deployment diagram	B	Similar to instantiating objects from object classes	Show numbers and types of agents	H	H	Y
19.Specify agent instances deployment	E [18]	Deployment diagram	B	Consider message traffic between agents, and processing power available on particular machines and required by particular agents	Show locations of agents (e.g. hostname and address)	H	H	Y

Table AppendixD.2 – Support for steps of MASSIVE

MASSIVE								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Task View	B	Analyze the intended workflow to specify what to be done. The functional decomposition of tasks can be supported by Structured Analysis	Construct the Task Tree following hierarchical decomposition. The granularity of decomposition depends on the specific problem, but should not become a specification of a particular algorithm	H	H	Y
2. Specify use case scenarios								
3. Identify roles	E	Role View	B	Group atomic activities (from Task View) into roles while satisfying the physical constraints of the operational environment		H	H	Y
4. Identify agent classes	I[3]	Role View, Architectural View	B			L	L	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	I[7]	Interaction View				L	L	Y
7. Define interaction protocols	E	Interaction View	B	Characterise the nature of agent interactions, thereby choosing appropriate interaction scheme and protocols		H	H	Y
8. Define content of exchanged messages								
9. Specify ACL	I[7]	Interaction View	O	Recommend KQML as a de-factor standard for ACL				Y
10.Specify agent architecture	E	Architectural View	B	Characterise the requirements of the agent architecture, thereby selecting a suitable architecture		H	H	Y
11.Define agent informational constructs (i.e. beliefs)								
12.Define agent behavioural constructs (e.g. goals, plans, actions services)								
13.Specify system architecture (i.e. overview of all system components and their connections)	E	Architectural View	B	Examine the overall nature of the system, then choose an architectural patterns that firs		H	M	Y
14.Specify organisational structure/inter-agent authority relationships	E	Society View	B	Characterise the target system society and design/choose an optimal social structure accordingly		H	H	Y
15.Model MAS environment	E	Environment View	B	Characterise MAS' environment from both the perspectives of the developer and of the system.	Characterise organisational context (e.g. accessible or inaccessible, deterministic or non-deterministic, episodic or non-episodic, static or dynamic) and runtime environment (programming model, programming language, and communication mode)	H	H	Y
16.Specify agent-environment interaction mechanism	E [15]	Environment View	O	Determine a generic model of sensors + effectors that allows agents to interact with the environment.		H	M	Y
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.3 – Support for steps of SODA

SODA								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	I[3]	Role Model	O		Specify each task in terms of its responsibilities, competences and required resources. Classify each task to either "individual" task or "social" task	H	L	N
2. Specify use case scenarios								
3. Identify roles	E	Role Model	B	Associate each "individual" task to an "individual" role, each "social" task to a group of "social" roles.	Define each role/role-group in terms of its individual and/or social tasks, permissions to resources (which are identified in Resource Model), and interaction protocols and rules (which are defined in Interaction Model).	H	M	N
4. Identify agent classes	E	Agent Model	B	Groups roles into agent classes	Define each agent by its individual/social tasks, permissions to resources, interaction protocols associated with its roles, cardinality, location and source (i.e. from inside or outside the system).	H	M	N
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	I[7]	Interaction Model						
7. Define interaction protocols	E	Interaction Model	O	Define the interaction protocols for roles and for resources, as well as interaction rules for role-groups	An interaction protocol specifies the information required/provided by a role to accomplish its tasks, or by a resource to invoke its services. An interaction rule for a role-group governs the interactions among social roles and resources so as to make the group accomplish its social task	H	H	N
8. Define content of exchanged messages								
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)								
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment	E	Resource Model, Environment Model	O	Identify resources offered by environment. Map these resources onto infrastructure classes. Specify the topological model of environment	Define resource in terms of services, access modes and permissions granted to roles and role-groups. Describe each infrastructure class is described in terms of services, interfaces, location, owner and cardinality	H	M	N
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes	I[4]	Agent Model						
19.Specify agent instances deployment	I[4]	Agent Model						

Table AppendixD.4 – Support for steps of GAIA

GAIA								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	I [3]	Role model	O	Specified as "liveness responsibilities" of roles	Each liveness responsibility is made up of "actions" and/or "protocols"	H	M	Y
2. Specify use case scenarios								
3. Identify roles	E	Role model	B	Identify roles from individuals, departments/offices, or sub-organisations in the target system	Model each role by its "responsibilities" (including "liveness" and "safety") and "permissions"	H	H	Y
4. Identify agent classes	E	Agent model	B	Typically one-to-one mapping between roles and agent classes	Show identifier of agent classes and their respective roles	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Acquaintance model	B	Identify acquaintances from Role, Agent and (inter-role) Interaction models	Show agent classes and communication paths between them	H	M	Y
7. Define interaction protocols	E	Interaction model	O	Only specifies protocols for inter-role interactions. Each protocol defines an institutionalized pattern of interaction with no detailed sequences of exchanged messages	Specify purpose, initiator, responder, inputs, outputs, and (informal) processing description for each inter-role protocol	H	H	Y
8. Define content of exchanged messages								
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)								
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	Service model	B	Identify agents' services by analyzing their roles' responsibilities, actions, and protocols.	Show inputs, outputs, pre-condition, and post-condition for each agent's service	H	H	Y
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships	E	Organisational structure model	B	Choose a structure that optimizes the organisational efficiency and simplicity, respects organisational rules, and reflects the structure of real world organisation.	Specify organisational dependencies between roles	H	H	Y
15.Model MAS environment	E	Environmental model	B	Identify abstract computational resources (e.g. tuples/variables) that are available to agents for sensing, effecting or consuming	Specify a symbolic name, types of actions permitted on each environmental resource, and their textual/structural descriptions	H	M	Y
16.Specify agent-environment interaction mechanism	I			Implicitly indicates that agents interact with environment via sensors and affectuators. No additional information provided				
17.Specify agent inheritance and aggregation	I[4]	Agent model	B	Aggregation occurs when an agent class is composed of the roles that make up other agent classes. Does not consider inheritance	Show an aggregate agent class as the parent of the children classes in the tree structure of Agent model	H	H	N
18.Instantiate agent classes	E [4]	Agent model	O		Specify numbers of instances for each agent class by annotating the class with qualifiers from Fusion	H	H	Y
19.Specify agent instances deployment								

Table AppendixD.5 – Support for steps of MESSAGE

MESSAGE								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Goal/Task view	B	Analyze organisation chart, company goals description, and business processes to identify goals. Identify services that can be performed by roles to satisfy these goals, and tasks that can be implemented to fulfill these services	Show a hierarchy of goal decomposition in Goal diagram. Describe the flow of tasks to achieve a service in a Workflow diagram	H	H	Y
2. Specify use case scenarios								
3. Identify roles	E	Agent /Role view	B		Associate roles to goals in Delegation structure diagram, and to services and tasks in Workflow diagrams. Describe each role with Role Schema	H	M	Y
4. Identify agent classes	E	Agent /Role view	B	Assign roles to agents based on the developer's experience and heuristics	Describe each agent with an Agent Schema and an Agent diagram (which shows the associated roles, goals, tasks and assessed data sources)	H	M	Y
5. Model domain conceptualisation	E	Domain view	B	Incrementally add domain concepts and relations to Domain view as they are needed in other views	Specify concepts as classes in UML class diagram	H	M	Y
6. Specify acquaintances between agent classes	E	Organisation View	B		Specify acquaintances between roles/agents in Organisation view.	H	M	Y
7. Define interaction protocols	E	Interaction view	B	Incrementally built from Analysis to Design. In Analysis phase, only need to highlight which, why and when roles communicate. In Design phase, elaborate each interaction considering the assignment of roles to agents and implementation of services in terms of tasks	May model each protocol with AUML protocol diagrams or UML statechart. Can model the behaviour of each agent/role in a protocol with statecharts	H	H	Y
8. Define content of exchanged messages	I[7]	Interaction view	O		Define each message in appropriate ACL in protocol diagrams.	H	L	N
9. Specify ACL	E		O	In *Agent-Platform Driven* design approach, the developers should choose an ACL and content language to use, e.g. KQML/KIF or FIPA-ACL/SL		H	M	N
10.Specify agent architecture	E		B	Select an architecture that suits the functional requirements of agents (e.g. cognitive versus reactive)	Specify architecture components/layers, depending on the chosen architecture	H	H	Y
11.Define agent informational constructs (i.e. beliefs)	E		B	Depending on the agent architecture, various categories of knowledge may need to be specified, including domain knowledge, social knowledge, and behavioural knowledge. These can be determined by analyzing the Domain view, Organisation view, and Goal/Task view.	Represent domain entities (for domain knowledge), social constraints (for social knowledge), and rules, objectives, and tasks (for behavioural knowledge) probably using UML notation	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)	E [14]	System architecture diagram	B	Derive system architecture from Organisation Model	Show all system components as a package structure	H	H	Y
14.Specify organisational structure/inter-agent authority relationships	E	Organisation view	B	Incrementally built from Analysis to Design. Start by analyzing organisation chart and business process documentation	Show stakeholders/users, agents/roles, resources, sub-organisations, and relationships bet them (e.g. power/peer-to-peer organisational relationships, acquaintances)	H	H	Y
15.Model MAS environment	I [14]	Organisation view	O	Identify resources that agents use, control or receive input from (e.g. databases, computational resources)	Show resources and their relationships with agents in Organisation view	H	L	N
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes	I[4]			Mentioned but no techniques/model kinds provided		L	L	N
19.Specify agent instances deployment								

Table AppendixD.6 – Support for steps of INGENIAS

INGENIAS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Goals and Tasks model	B	Identify goals from system requirements or objectives associable to agents. Derive tasks from system requirements or from goals	Show goals, goal-subgoals dependencies, tasks, tasks' pre-conditions, post-conditions, and goals-tasks associations	H	H	Y
2. Specify use case scenarios	E	Use case diagrams	O	Incrementally identified and refined	Conventional OO techniques	H	H	Y
3. Identify roles	I[4,15, 7]	Agent model, Organisation model, Interaction model	B	Identify roles from the analysis of workflows and tasks in Organisation model	Show roles as actors of workflows/tasks in Organisation model, as participants in Interaction model, and associated to agents in Agent model	H	M	Y
4. Identify agent classes	E	Agent model	B	Apply "rationality principle" on system components to identify agents.	Describe each agent in terms of its roles, goals, tasks, mental states, and control structure/process.	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Interaction model	B	In Analysis phase, identify significant interactions between actors (i.e. agents/roles) and initial schemes of exchanged info.	Show participants (agents/roles) and goals pursued by each interaction	H	H	Y
7. Define interaction protocols	E[6]	Interaction model	B	In Design phase, elaborate each interaction with detailed description of exchanged elements (e.g. messages, tuples, method calls)	Specify exchanged elements and order of their execution (e.g. iteration, concurrency, branching)	H	H	Y
8. Define content of exchanged messages	E[7]	Interaction model	B		For each message, show name of operation, parameters, guards, and annotation of sequence	H	M	Y
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)	E	Agent model	B	Determine agent's "mental states" from analysis of goals, tasks, and interactions. Define the "control" of agent to assure desired transitions between Its mental states	Represent mental states in terms of goals, tasks, facts, or any other entity that helps in state description. Agent goals can be modelled as initial state. Can model agent control as algorithms or complex deliberative process	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)	I[14]	Organisation model	B		Implicitly reflected in the Organisation model where all system components and their connections are shown	H	H	Y
14.Specify organisational structure/inter-agent authority relationships	E	Organisation model	B	Incrementally identify and refine the org. structure in terms of system components (i.e. agents, roles, resources, and applications), and social dependencies among them	Show how system components are grouped, their social dependencies (e.g. subordination and client-server relations), task/workflow assignment, and resources used/produced.	H	H	Y
15.Model MAS environment	E	Environment model	B	Identify resources and applications in the environment by analyzing system requirements and agent requirements	Model resources and applications as objects. Specify internal states and operations for applications, and initial states, category, and limit of consumption for resources.	H	H	Y
16.Specify agent-environment interaction mechanism	E[15]	Environment model	B	Determine how agents perceive outputs of applications in the environment. Possible perception mechanisms: sampling or notification	Represent agent's perception mechanism as a type of association relationship between the agent and an application in Environment model	H	H	Y
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.7 – Support for steps of BDIM

BDIM								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality								
2. Specify use case scenarios								
3. Identify roles	E		B	Roles can be organisational or functional; can be domain dependent or required by system implementation		M	L	Y
4. Identify agent classes	E	Agent Model	B	Group roles (that have common lifetime and intense interactions) into a draft agent hierarchy. Refine the hierarchy to introduce new abstract agent classes, new concrete agent classes and agent instances	Produce Agent Class Diagram and Agent Instance Diagram (may be combined into a single diagram if the number of agents is small)	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Interaction Model	B	Identify interactions between agents by analyzing the provision of services among agents.	Offer no modelling notation for Interaction Model Developers can use any notation that fits	M	L	N
7. Define interaction protocols								
8. Define content of exchanged messages	E [6]	Interaction Model		For each interaction, identify the speech acts required for the messages and the messages' information content.	Offer no modelling notation	M	L	N
9. Specify ACL								
10.Specify agent architecture	I			BDIM adopts a BDI agent architecture				
11.Define agent informational constructs (i.e. beliefs)	E	Goal Model, Belief Model, Plan Model	B	Identify agent goals from agent's service. For each goal, identify means for achieving the goal (i.e. plans) and the context, conditions, inputs and outputs of goals and plans (i.e. beliefs)	A Goal/Belief Model consists of 1 Goal/Belief Set and many Goal/Belief States. A Plan Model contains a set of plan diagrams	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E		B	For each agent, identify its responsibilities and the services provided/used to fulfill these responsibilities		H	M	Y
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation	E	Agent Model	B	Identify inheritance and aggregation relationships by examining similarity in lifetime, services and interaction interfaces of agents	Inheritance allows an agent to override/extend the Goal/Belief/Plan Model of its superclass(es). Aggregation allows for agents with independent Goal/Belief/Plan Models to be combined into an aggregate class	H	H	Y
18.Instantiate agent classes	E	Agent Model	B		Capture instantiation information (e.g. instance identification and cardinality) either in Agent Class Diagram or separately in Agent Instance Diagram	M	M	Y
19.Specify agent instances deployment								

Table AppendixD.8 – Support for steps of HLIM

HLIM								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E [2]	High Level Model	B	Identify system tasks as "responsibilities" appearing in use case scenarios	Each path in the UCM connects responsibilities, indicated by named points along paths	H	H	Y
2. Specify use case scenarios	E	High Level Model	B		Develop a Use Case Map (UCM) for each use case scenario, where the UCM's path traces a scenario from a start to finish	H	H	Y
3. Identify roles	E [2]	High Level Model	O		Roles are represented by "slots" (boxes with dashed lines) along UCM's paths	H	M	Y
4. Identify agent classes	E [2]	High Level Model	B	Identify agents as carrier of responsibilities in UCMs. Initial agents can be extracted from essential and active entities that exist in the problem domain.	Represent agents as boxes incorporating responsibilities along UCM's paths.	H	M	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	I[7]	Conversation Model	B	Derive necessary agent interactions from Agent Relationship Model and Internal Agent Model		M	M	Y
7. Define interaction protocols	E	Conversation Model	B	Each type of dependency relationships and jurisdictional relationships has a predefined interaction protocol associated with it	Express a protocol as a set of performatives that are specified in the exchanged messages	H	H	Y
8. Define content of exchanged messages	E	Conversation Model	B	Identify what messages are required to fulfill the dependency and jurisdictional relationships. The content of messages is determined by examining plans that satisfy the dependencies.	Use tabular format. A table for each agent. 3 columns: receive, send and comment. Each message contains a performative and parameters.	H	M	Y
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)	E	Internal Agent Model	B	Derive agents' goals, tasks, beliefs and plans directly from UCM's components in High Level Model	Use tabular template, where agent goals, tasks and beliefs are specified in columns. Plans are combinations of goals, tasks and beliefs (i.e. rows)	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	Contract Model	O		Services provided by each agent are captured in its contracts with other agents	M	L	Y
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships	E	Agent Relationship Model	B	Identify organisational relationships by analysis of path segments responsibilities in UCMs.	Model organisational relationships as Dependency and Jurisdictional relationships. Each type is captured in Dependency Diagram and Jurisdictional Diagram respectively	H	H	Y
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.9 – Support for steps of MEI

MEI								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	IDEF/CIMOSA Function Model	O	Borrow techniques from enterprise modelling	Borrow techniques from enterprise modelling	H	H	Y
2. Specify use case scenarios	E	Use Case Model	O	Borrow techniques from OOSE	Borrow techniques from OOSE, including use case extension and inheritance	H	H	Y
3. Identify roles								
4. Identify agent classes	E		B	Identify agents from actors in use cases and resources/mechanism in CIMOSA/IDEF function model		H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E		B	Agent collaboration exists if there is more than 1 actor per use case or more than 1 resource per enterprise function		H	H	Y
7. Define interaction protocols	E	Coordination Protocol Script	B	Derive protocols from event trace of use cases and information exchanges between resources	Use State Diagrams to model protocol scripts for each agent	H	H	Y
8. Define content of exchanged messages								
9. Specify ACL								
10.Specify agent architecture	I			MEI adopts a BDI-like model of agency, where each agent is composed of goals, plans and beliefs				
11.Define agent informational constructs (i.e. beliefs)	E	Goal-Plan Diagram, Plan State Diagrams	B	Derive agents' Goals and Plans from use cases and enterprise functions with control outputs. Context/ invocation conditions of plans can be derived from control inputs of enterprise functions, or input from actor and entity objects. Agent Beliefs correspond to domain objects in use cases and entities in IDEF Information Model.	An agent's goals and plans can be depicted as a tree structure, where goals are the root nodes and plans are leaves. Each plan can be further defined by a state diagram	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment								
16.Specify agent-environment interaction mechanism	I		O	Agents interact with environment via sensor and effector objects, which communicate with co-existing objects or sensor/effector objects of other agents	Each agent may have many sensor/effector objects	H	H	Y
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.10 – Support for steps of PROMETHEUS

PROMETHEUS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Functionality descriptor	B	Identify a set of functionalities by considering groupings of goals.	Describe each functionality in terms of its goals, actions, percepts/events and potential data read/written	H	H	Y
2. Specify use case scenarios	E	Use case descriptor	B	Identify sequences of steps that describe how the system achieves a goal or responds to an event	Annotate each step with associated functionality and data read/written	H	H	Y
3. Identify roles								
4. Identify agent classes	E	Agent class descriptor	B	Assign functionality to agent class based on the criteria of strong coherence and loose coupling	Describe each agent class in terms of its functionality, goals, events, actions, and data read/written, cardinality, agent lifetime).	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Interaction diagrams	B	Whenever there's a step in a use case that involves functionality from a new agent, there must be an interaction pathway from a previously involved agent and this new agent	Show the core interaction channels between agents using sequence diagrams	H	H	Y
7. Define interaction protocols	E [6]	Interaction protocols	B	Elaborate each complex interaction with protocol by analyzing use case scenarios	Show all variations of interaction sequences that are valid in the system. Each protocol may be split into smaller chunks	H	H	Y
8. Define content of exchanged messages	I [6, 7]	Interaction diagrams and protocols	B	Analyze use case scenarios		H	L	Y
9. Specify ACL								
10.Specify agent architecture	E	Agent overview diagram	B		Show the top-level view of an agent's internals, including top-level capabilities, events connecting these capabilities, and data objects internal to the agent	H	H	Y
11.Define agent informational constructs (i.e. beliefs)	E	Plan descriptor, Data descriptor	B	Recursively decompose each agent's capability into plans, events connecting plans, data read/written by plans, and sub-capabilities.	Describe each agent "plan" in terms of input/output events, actions, and messages. Describe each "data object" used by the agent with fields and methods.	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	Capability diagram	B	Identify agent "capabilities" by analyzing functionalities assigned to the agent	Describe each capability in terms of sub-capabilities, plans, events, and data read/written in Capability diagram	H	H	Y
13.Specify system architecture (i.e. overview of all system components and their connections)	E	System overview diagram	B	Describe how the system as a whole will function	Show an overview of all agent classes in the system, events connecting classes, and shared data objects	H	H	Y
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment	E	System overview diagram	O	There may be data objects existing in the environment that must be shared among agents (e.g. databases)	Show and link shared data objects to agents in System overview diagram	H	M	Y
16.Specify agent-environment interaction mechanism	E	Percepts descriptor, Actions descriptor	B	Specify raw data available to the system as "percepts", and activities performed by the system on the environment as "actions"	Specify percepts and actions for each system functionality in Percepts and Actions descriptors	H	H	Y
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes	I[4]	Agent class descriptor	O		Specify cardinality for each agent in its Agent class descriptor	H	L	N
19.Specify agent instances deployment								

Table AppendixD.11 – Support for steps of PASSI

PASSI								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	System requirements model	O	Follow standard requirements elicitation techniques in OO, or scenario-based teleological methods such as GBRAM	Specify functionality as use cases in Use case diagrams	H	H	Y
2. Specify use case scenarios	E [1]	System requirements model	B		Develop a hierarchical series of use case diagrams. The uppermost serves as a context diagram	H	H	Y
3. Identify roles	E	System requirements model, Agent society model	B	Identify roles for each agent by exploring all the possible scenarios of inter-agent interaction (captured in Agent identification diagram – step 5)	For each inter-agent interaction scenario, develop a Role identification diagram to specify the roles that agents play during the interaction. Describe roles with a Role description diagram, which shows their agents, role changes within an agent, roles' tasks, roles' interactions and dependencies.	H	H	Y
4. Identify agent classes	E	System requirement model	B	Package use cases into agents	Show agents, their respective use cases, and interaction paths between use cases in Agent identification diagram	H	M	Y
5. Model domain conceptualisation	E	Agent society model	O	Specify concepts/entities that define the domain's knowledge.	Represent domain ontology as an XML schema or class diagram in Domain ontology diagram	H	M	Y
6. Specify acquaintances between agent classes	I[4]	System requirement model	B		Agent acquaintances are reflected via the interaction paths between use cases in Agent identification diagram	H	H	Y
7. Define interaction protocols	E	Agent society model	B	Select and refine protocol for each agent acquaintance by consulting e.g. FIPA library. Should also specify the ontology used with the protocol	Document each protocol in Protocol description diagram (which may be AUML sequence diagram). Specify identifier of protocol and ontology for each agent acquaintance in Communication ontology diagram.	H	H	Y
8. Define content of exchanged messages	I	Agent implementation model	B	Specify messages' performatives as required by the interaction protocol and messages' contents by using concepts defined in Domain ontology diagram.	Model exchanged messages (including their performatives and contents) as transitions between agents in the Multi-agent behavior description diagram	H	H	Y
9. Specify ACL								
10.Specify agent architecture	E	Agent implementation model	B	Define agent structure as being composed of one main agent class and a set of inner classes, each representing a task of the agent	Specify data structures and methods of the agent and its tasks in the main agent class and task classes respectively.	H	H	Y
11.Define agent informational constructs (i.e. beliefs)	I [10]	Agent implementation model	B	Specify agent "knowledge" by using the concepts defined in Domain ontology diagram	Model knowledge as agent data structures in Single-agent structure definition diagram	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	System requirements model, Agent implementation model	B	Agent's capabilities are represented by its tasks, which can be identified by analyzing its roles and interactions described in Role identification diagrams	Show all tasks of an agent in a Task specification diagram. Further describe each method required to achieve each task in Single-agent behavior description (using flow charts, state diagrams, or text description)	H	H	Y
13.Specify system architecture	E	Agent implementation model	B		Show all agent classes, their knowledge, tasks, and connections with external actors in Multi-agent structure diagram	H	H	Y
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment	E	Deployment model	O		Show processing units, agents in each unit, agent movements, and units/agents connections in Deployment configuration diagram.	H	L	N

Table AppendixD.12 – Support for steps of ADELFE

ADELFE								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E [2]	Use case model	O	Identify the different functionalities the system has to carry out	Express each functionality as a use case	H	H	Y
2. Specify use case scenarios	E	Use case model	B	Apart from identifying use cases, need to also highlight the possible cooperation failures in the identified use cases	Conventional OO techniques	H	H	Y
3. Identify roles								
4. Identify agent classes	E	Software architecture document	B	First, decompose system into entities. Determine which entities fit to be agents, i.e. whether they are autonomous, goal-directed, dynamic, and need to deal with unpredictable events. If an agent needs to be adaptive/evolving, it should be decomposed into a collective of sub-agents.	Show entities and their relationships in Preliminary class diagram. Update this diagram to indicate which classes are agents	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Software architecture document	B	Identify potential interaction relationships between agents, and also between agents and non-agent active/passive entities	Model interaction relationships with sequence diagrams or collaboration diagrams	H	M	Y
7. Define interaction protocols	E	Interaction languages document	B	Analyze each use case and interaction scenarios	Elaborate each interaction relationship with a protocol diagram that specifies information exchanges between agents and between agents and non-entities	H	M	Y
8. Define content of exchanged messages	E [7]	Interaction languages document	B	As above. Also select the communication languages for specifying the messages		H	M	Y
9. Specify ACL	E [8]	Interaction languages document	O		ACL used to implement the exchanged messages is documented in Interaction Languages document	H	M	N
10.Specify agent architecture	E	Detailed architecture document	O	Agent architecture should contain 5 components: representations, social attitudes, interaction languages, aptitudes, and skills	Model each agent in terms of the 5 listed components	H	H	Y
11.Define agent informational constructs (i.e. beliefs)	E	Detailed architecture document	B	"Representations" are agent's beliefs about itself and environment. "Social attitudes" contain rules for dealing with non-cooperative situations. "Interaction languages" involve protocols used by the agent.	Specify attributes and methods for agent's representations. Select protocols for agent's interaction languages from the set defined in step 8. Specify non-cooperative situations and rules for cooperative attitudes.	H	H	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	E	Detailed architecture document	B	"Skills" are capabilities that an agent brings to its collective. "Aptitudes" are agent's capabilities on its knowledge.	Specify methods and/or attributes for agent's "skills" and "aptitudes"	H	M	Y
13.Specify system architecture (i.e. overview of all system components and their connections)	E	Detailed architecture document		Define the system architecture in terms of packages and classes (of agents and objects). Should use design patterns and/or re-usable components	Generate a Class diagram for each package	H	H	Y
14.Specify organisational structure/inter-agent authority relationships								
15.Model MAS environment	E	Environment definition document	B	Identify active and passive entities in the environment; characterise the system's environment as being accessible or not, deterministic or not, static or dynamic, and discrete or continuous		H	H	Y
16.Specify agent-environment interaction mechanism	I [11]	Detailed architecture document	O	Agents interact with environment via percepts and actions, implicitly specified in agents' "skills", "aptitudes" and "interactions".		H	L	N
17.Specify agent inheritance and aggregation	I	Detailed architecture document	O		Show aggregation relationships between agents in Class diagrams	H	M	Y
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.13 – Support for steps of COMOMAS

COMOMAS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Task Model	O	Identify tasks to be solved by the target MAS and data/control dependencies between them	Develop a task hierarchy, along with each task's details (i.e. input, output and control structure). Can use Conceptual Modelling Language (CML) as notation	H	M	Y
2. Specify use case scenarios								
3. Identify roles								
4. Identify agent classes	E	Agent Model	B	Identify agents by clustering the competencies for solving tasks (Expertise Model) while respecting the design requirements (Design Model)	Model each agent as a composition of knowledge structures obtained from other models. Can use CML as modelling notation	H	M	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes								
7. Define interaction protocols	E	Cooperative Model	O		Specify cooperation protocols, cooperation methods (e.g. data sharing or message exchange) and conflict resolution methods. Can use CML as modelling notation	H	L	Y
8. Define content of exchanged messages								
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)	E	Expertise Model	B	Determine agent competencies required to solve system tasks and social knowledge required to enable it to act smoothly during interaction	Competencies include "task knowledge" (i.e. agents' experience on previously solved tasks), "problem-solving knowledge" and "reactive knowledge" (i.e. agents' reactive responses to stimuli). "Social knowledge" includes roles, association between beliefs, commitments, intentions and goals. Can use CML as modelling notation.	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships	E	System Model	B			M	L	Y
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.14 – Support for steps of MAS-CommonKADS

MAS-CommonKADS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Task model	O	Decompose tasks following a top-down approach	Show tasks in an and/or tree. Describe each task in terms of inputs, outputs, task structure, required capabilities of performer, and preconditions	H	H	N
2. Specify use case scenarios	E	Use case	B	Perform user-centered analysis during Conceptualization phase to identify potential users and how the system processes a user request	Conventional OO techniques	H	H	Y
3. Identify roles								
4. Identify agent classes	E	Agent model	B	Analyze various sources e.g. use cases, statement problems, heuristics, initial Task and Expertise models	Describe each agent in terms of type, role, position, description, offered/used services, goals, plans, knowledge, collaborates, skills (sensors and effectors), reasoning capabilities, general capabilities norms, preferences and permissions.	H	M	Y
5. Model domain conceptualisation	E [11]	Expertise model	O	Specify domain conceptualisation as agent's domain knowledge	Represent concepts, properties, expressions, and relationships in the domain using e.g. class/object diagrams	H	M	Y
6. Specify acquaintances between agent classes	E	Coordination model	B	Identify prototypical conversations between agents by analyzing the results of techniques used for identifying agents (e.g. use cases, heuristics, task model, and CRC cards).	Model conversations by using Message Sequence Charts and Event flow diagrams	H	H	Y
7. Define interaction protocols	E [6]	Coordination model	B	Identify protocols for complex conversations by consulting existing libraries and reuse protocol definitions	Model protocols using high level Message Sequence Charts. Model the processing states of an agent during a protocol using State transition diagrams	H	H	Y
8. Define content of exchanged messages	E [7]	Coordination model	B	Analyze use cases and Expertise model	Model data interchanged in each interaction in terms of data structures specified in Expertise model	H	H	Y
9. Specify ACL								
10.Specify agent architecture	E	Design model	B	Select an appropriate architecture and map the elements defined in Coordination, Expertise, Agent, and Task models onto modules of the architecture. No techniques or models are discussed		H	L	N
11.Define agent informational constructs (i.e. beliefs)	E	Expertise model	B	Specify domain knowledge, task knowledge, inference knowledge, and problem solving knowledge for each agent	Describe each type of knowledge in Domain knowledge ontology, Inference diagrams, Task knowledge specification, or Problem solving method diagrams/templates.	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)	I	Agent model; Organisation model	O		Identify the services that an agent offers to other agents and document this in Agent Model and/or Organisation Model	H	L	Y
13.Specify system architecture (i.e. overview of all system components and their connections)	E	Organisation model	B		Show all agents, objects and their relationships (e.g. inheritance, association, agent-object relationship)	H	M	Y
14.Specify organisational structure/inter-agent authority relationships	I [13]	Organisation model			Agent organisational relationships are modelled as association relationships annotated with roles (of each involved agent)	H	L	Y
15.Model MAS environment	E	Reaction cases; Design model	B	Perform environment-centered analysis during Conceptualization phase to identify objects in the environment and potential events coming from each object and actions performed by agents on each object. Identify networking, knowledge and coordination facilities.	Describe the reaction cases coming from interaction of agents with objects in the environment	H	M	N
16.Specify agent-environment interaction mechanism								

17.Specify agent inheritance and aggregation	E	Organisation model	B	Specify aggregation relationships for agent groups, and inheritance relationships for agents that inherit from the values of the precedent agents		H	M	Y
18.Instantiate agent classes	I	Organisation model	O	Mentioned but no techniques discussed	Organisation model can be developed for both agent classes and agent instances	H	L	N
19.Specify agent instances deployment								

Table AppendixD.15 – Support for steps of CASSIOPEIA

CASSIOPEIA								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E		B	Employ existing functional or OO analysis techniques	Define system behaviour at a level of abstraction that makes sense to the achievement of the system's collective task	H	H	Y
2. Specify use case scenarios								
3. Identify roles	E	Coupling Graph	B	CASSIOPEIA identifies 3 layers of roles: domain-dependent roles, relational roles and organisational roles. Identify domain-dependent roles by grouping elementary behaviors needed to achieve the task. See steps 7 and 16 for other 2 types of roles		H	M	Y
4. Identify agent classes	E	Coupling Graph	B	Group roles into agents. One agent may play many roles (one of which is active at a point in time) and one role may be played by many agents		H	M	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Coupling Graph	B	Identify dependencies between domain-dependent roles, thereafter deriving dependencies/acquaintances between agents.	Specify "relational roles" for each agent, i.e. the role of an "influencing" agent or an "influenced" agent	H	H	Y
7. Define interaction protocols	E [6]		O	Specify "influence signs" (i.e. interaction messages) between influencing and influenced agents by analyzing the domain-dependent roles that each agent is playing.		H	L	Y
8. Define content of exchanged messages								
9. Specify ACL								
10.Specify agent architecture								
11.Define agent informational constructs (i.e. beliefs)								
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships	E		B	Analyze the agents' dependencies and relational roles to determine their "organisational roles", i.e. role of "group initiator" and "group participant". Also identify the "organisational behaviors" of agents when playing these organisational roles, i.e. group formation behaviour, commitment behaviour and dissolution behaviour.		H	M	Y
15.Model MAS environment								
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

Table AppendixD.16 – Support for steps of TROPOS

TROPOS								
Steps	**Supported?**	**Model kinds/ Notational components?**	**Inputs/ Outputs?**	**Techniques for step**	**Techniques for modelling**	**Ease of understanding**	**Usability**	**Examples**
1. Identify system functionality	E	Actor diagram, Rationale diagram	B	In Early Requirements, identify goals and softgoals of stakeholders, and perform means-end analysis to determine how these goals can be fulfilled. In Late Requirements, focus on the target system and how it can fulfill the assigned goals. Perform means-end analysis to identify tasks to achieve goals during both Early and Late Requirements	Show task dependencies among stakeholders and system in Actor diagram. Show how goals are achieved through tasks in Rationale diagram.	H	H	Y
2. Specify use case scenarios								
3. Identify roles								
4. Identify agent classes	E	Actor diagram	B	Depending on the chosen organisational structure, decompose the system into sub-actors, each of which can be recursively refined into sub-actors (can consult catalogues of agent patterns for this activity). Assign sub-actors to agents.	Show sub-actors within each system actor, and goal/task/resource dependencies among them	H	H	Y
5. Model domain conceptualisation								
6. Specify acquaintances between agent classes	E	Sequence diagrams, Collaboration diagrams	B	Identify interactions between agents to fulfill particular tasks		H	M	Y
7. Define interaction protocols	E [6]	Sequence diagrams	B	Elaborate each inter-agent interaction in greater detail (e.g. by introducing additional or refined exchanged messages)		H	H	Y
8. Define content of exchanged messages	I[6, 7]				Model each message as a communication act in ACL	H	M	Y
9. Specify ACL								
10.Specify agent architecture	I	BDI agent architecture		TROPOS adopts BDI model for agent architecture				
11.Define agent informational constructs (i.e. beliefs)	E	Plan diagrams, Agent class diagram	B	Define agent's "plans" to achieve a goal, perform a task, or respond to a (communicative) event. Identify resource entities that are incorporated in the agent's knowledge base	Specify Plan diagrams at a directly executable level. Represent resource entities as component classes of an agent class in Agent class diagram	H	M	Y
12.Define agent behavioural constructs (e.g. goals, plans, actions, services)								
13.Specify system architecture (i.e. overview of all system components and their connections)								
14.Specify organisational structure/inter-agent authority relationships	E	Non-functional requirement model	B	Select a suitable organisational structure style (e.g. from the set proposed by TROPOS) by evaluating its quality attributes against the system's softgoals.	Specify how well each alternative organisational structure style fulfils the system's softgoals	H	H	Y
15.Model MAS environment	I[1]	Actor diagram	B		Model environment via stakeholders, and their goal/ task/resource dependencies with the system	H	H	Y
16.Specify agent-environment interaction mechanism								
17.Specify agent inheritance and aggregation								
18.Instantiate agent classes								
19.Specify agent instances deployment								

APPENDIX E

MODELLING NOTATION OF MOBMAS

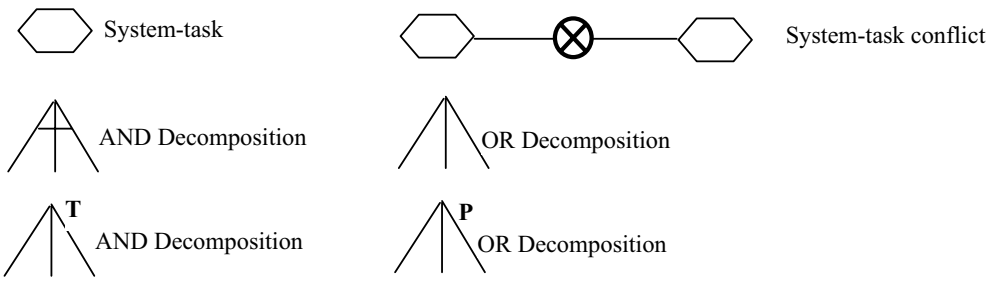
The modelling notation of MOBMAS is mostly reused or adapted from UML, AUML and other sources (such as the existing AOSE methodologies), except for the following notation components that are represented using MOBMAS' own notation:

- Role Diagram;
- Agent Class Diagram;
- Agent Relationship Diagram;
- Agent Plan Template; and
- Resource Diagram.

The notation proposed by MOBMAS has a similar syntax to UML. For example, an agent class or a role or a resource is represented as a rectangular box with multiple compartments, each specifying a different property of the target entity. The similarity in syntax between MOBMAS notation and UML is intentional, because it facilitates the use of MOBMAS by developers who are familiar with UML.

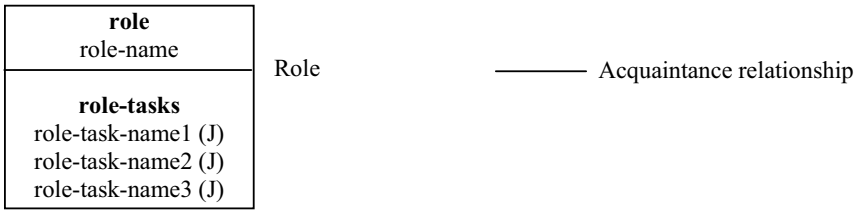
A summary of MOBMAS notational syntax is presented below.

SYSTEM TASK DIAGRAM



ORGANISATION CONTEXT CHART

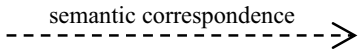
ROLE DIAGRAM



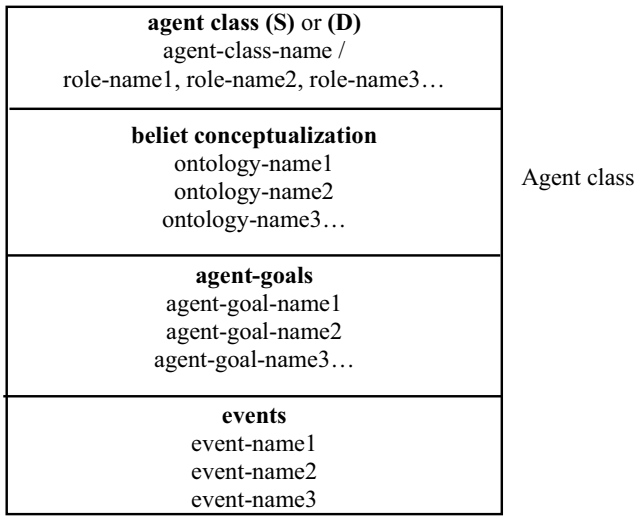
ONTOLOGY DIAGRAM

Adapt the notation of UML Class Diagram (Object Management Group 2003).

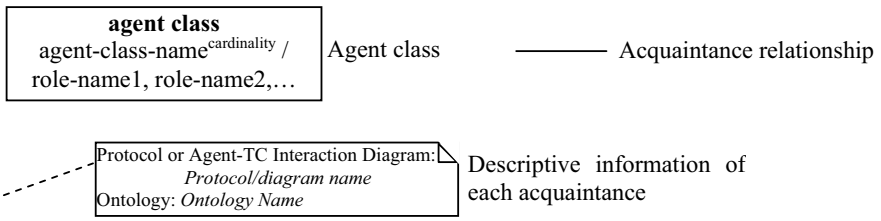
- Ontological concepts are represented as UML classes, attributes or predicates that describe the associations between concepts.
- Relations between ontological concepts are represented as UML relationships between classes, which can be specialisation, aggregation or association.
- Ontological mappings are represented as UML dependency relationships:



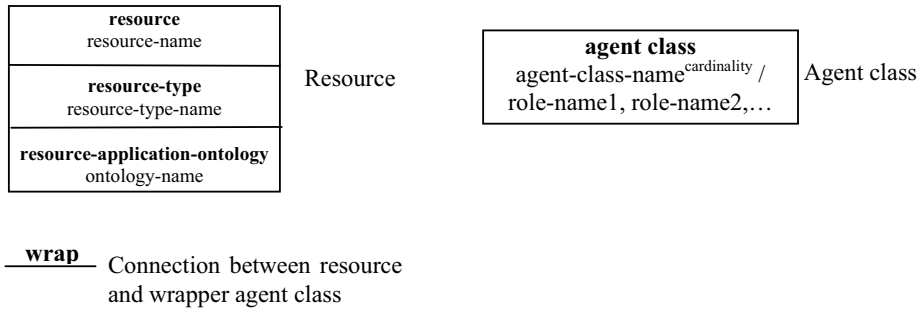
AGENT CLASS DIAGRAM



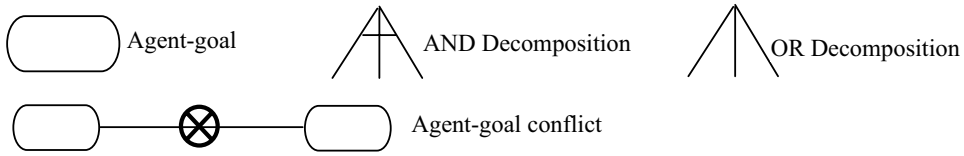
AGENT RELATIONSHIP DIAGRAM



RESOURCE DIAGRAM



AGENT GOAL DIAGRAM



AGENT PLAN TEMPLATE

Initial state: *state definition*

Target agent-goal: *state definition*

Commitment strategy: e.g. *blind, single-minded* or *open-minded*

List of sub-agent-goals (if any): *state definition and name of the Agent Plan Template that achieves the sub-agent-goal*

List of actions (if any): *action name and parameter list*

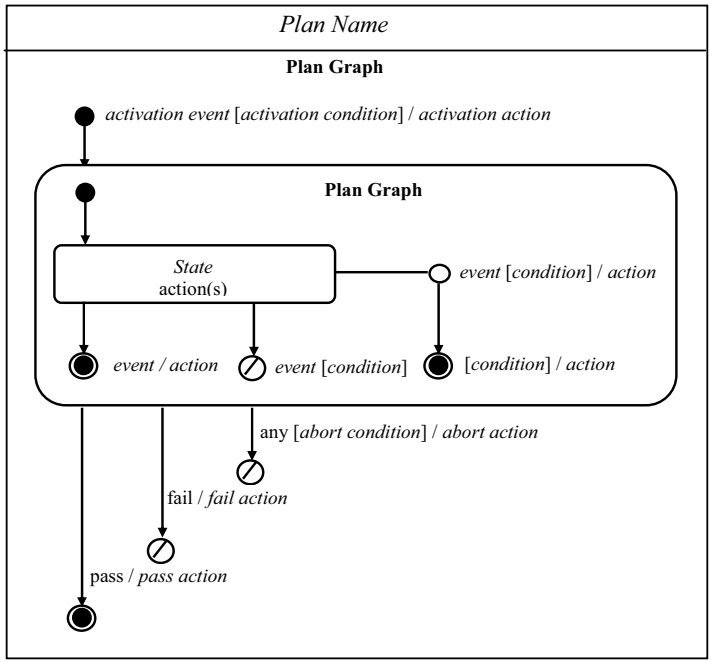
Pre-condition: *state definition*

Post-condition: *state definition*

Events: *list of events*

Conflict resolution strategy (if applicable): *strategy name for each agent-goal*

AGENT PLAN DIAGRAM



REFLEXIVE RULE SPECIFICATION

Adapt the notation of UML Activity Diagram (Object Management Group 2003).

- Actions are represented as UML activities.
- Events, internal processing triggers and guard conditions are represented as UML events and guard conditions.

INTERACTION PROTOCOL DIAGRAM / AGENT-TC INTERACTION DIAGRAM

Reuse AUML Interaction Diagrams

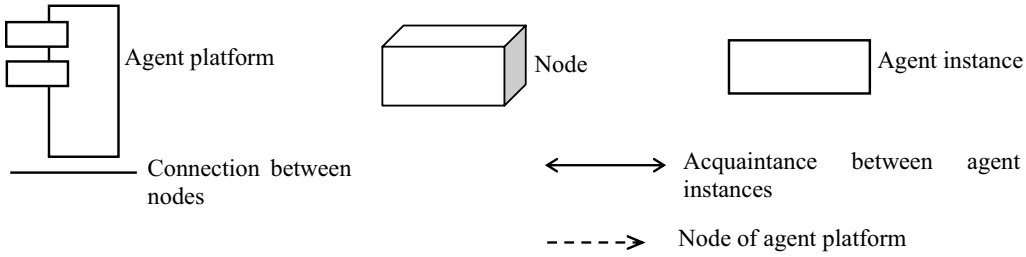
TUPLE-CENTRE BEHAVIOUR DIAGRAM

Adapt UML Statechart Diagram.

- Reactions are represented as states.
- Events are represented as transitions between states.

AGENT ARCHITECTURE DIAGRAM

MAS DEPLOYMENT DIAGRAM



APPENDIX F

EXPERT REVIEWS OF MOBMAS

This appendix documents the two expert reviews of MOBMAS which were obtained from Prof. Brian Henderson-Sellers and Prof. Mary-Anne Williams. Each review contained each expert's opinions on the strengths of the methodology, areas for improvement and how to improve these areas. These opinions were recorded informally as comment notes on MOBMAS' documentation which was initially given to each expert.

Expert Review 1

EXPERT: PROF. BRIAN HENDERSON-SELLERS

Strengths of MOBMAS

1. The overall methodology is easy to understand and appears to be easy to follow.
2. The methodology is comprehensive and offers support for diverse aspects of MAS development, covering from analysis to agent internal design to agent interaction design.
3. While the modelling notation of MOBMAS needs to be revised, the steps and techniques are mostly practical and comprehensive.
4. The methodology proposes a clear mapping from roles to agent classes, and from role-tasks to agent-goals and events, thus providing a smooth transition from MAS analysis to agent internal design. The classification of role-tasks into reactive and proactive tasks, thereby identifying agent-goals and events, is also original.

Areas for improvement and suggestions on how to improve these areas

1. Many notational components of MOBMAS are described as *extensions* of UML notational components. However, some extensions are inappropriate because the extended notation is too semantically distant from the original UML notation (namely, MOBMAS Role Diagram, Agent Class Diagram and Resource Diagram). Some other extensions are appropriate but the semantics of the extended modelling notation are not well-documented.

Therefore, it is necessary to determine whether a particular MOBMAS notational component is eligible to be an extension of a UML component. Valid UML extension mechanisms are "stereotypes", "tagged values" and "constraints". Thus, if a MOBMAS notational component cannot be mapped to an UML component via these permissible mechanisms, it should be regarded as MOBMAS' *own* modelling notation. For example, although the Agent Class Diagram of MOBMAS has a similar appearance to UML Class Diagram (with multiple compartments, each modelling a different property of the class), the semantics of each compartment in MOBMAS Agent Class Diagram is very different from the semantics of each compartment in the UML Class Diagram. The difference in semantics is too significant to be expressed by stereotypes, tagged values or constraints.

If a MOBMAS notational component is eligible to be considered as an extension of UML, the methodology should explicitly specify the extension mechanism adopted, and clearly define the semantics of the extended notation.

2. Throughout the development process, MOBMAS employs a variety of modelling concepts, including "system-task", "role-task", "agent class", "agent-goal" and "agent". Although the semantics of these concepts are defined at their first occurrence in the methodology, it is hard for the readers to recall the meaning of a particular concept, especially after many other concepts have been introduced (for example, concepts "system-task" and "role-task", or "agent class" and "agent" can be easily confused). Moreover, various modelling concepts in MOBMAS are closely linked (e.g. "system-task" is associated to "role-task", which is mapped onto "agent-goal"). Even though these linkages are highlighted in the documentation of the respective steps and model kinds, it is difficult for the readers to recall these associations when numerous associations exist.

   A meta-model of the core modelling concepts of MOBMAS should be developed. This meta-model will assist the readers in their understanding and remembering of the semantics and associations of these concepts.

3. The following errors in modelling notation should be fixed:
   - An idle state or a decision point in the tuple-centre Behaviour Diagram (which is basically a UML State Chart) should be represented as a circle ◯ and not a diamond ◇, to adhere to UML specification of state charts.
   - The actors in Interaction Diagrams of Agent Coordination Model should be agent instances instead of agent classes. Accordingly, the names of agent classes in the boxes above the lifelines should be preceded with a colon (**:**) to represent instances.

4. The modelling of relationships between ontological concepts in the Ontology Model should include the modelling of "composition" relationship apart from the "aggregation" relationship. The difference in semantics between these two relationships should be highlighted.

Expert Review 2

EXPERT: PROF. MARY-ANNE WILLIAMS

Strengths of MOBMAS

1. The methodology provides extensive support for the openness, heterogeneity and dynamics of MAS.
2. Ontology modelling is tightly incorporated into the analysis and design of MAS, with numerous two-way verification and input linkages between Ontology Model and other MAS analysis and design models (such as Agent Class Model, Agent Behaviour Model and Agent Interaction Model).
3. The methodology provides comprehensive support for MAS development, incorporating diverse analysis and design activities and modelling MAS from diverse aspects (from internal to external).
4. The internal and interaction design of agents are relatively detailed.

Areas for improvement and suggestions on how to improve these areas

1. Regarding steps "Develop System Task Model" and "Analyse Organisational Context" of MOBMAS, the applicable conditions of each step are not appropriate because they may be overlapped. Specifically, for step "Analyse Organisational Context", the recommended applicable condition is that

> "…the target MAS is a processing application system that does not exhibit any specific and apparent human organisational structure"

and for step "Develop System Task Model", the condition is that

> "if the target application exhibits a clear organisational structure, roles can later be identified directly from this structure, thus making the adoption of the Organisation Analysis Approach beneficial".

Accordingly, most MASs are eligible for step "Analyse Organisational Context" because they aim to support a human organisation whose structure is clear. But at the same time, these MASs are also eligible for step "Develop System Task Model" because they do not aim to adopt the human organisational structure.

Therefore, the application conditions of each step should be made clearer and more sensible, avoiding any potential overlaps. The methodology should also give consideration to whether or not the target MAS should adopt the existing human organisational structure.

2. For the naming of system-tasks, MOBMAS adopts the naming format of "*To do something*" (e.g. "*To receive user query"* or "*To get information from resources*"). This naming format makes system-tasks appear like an abstract objective. It may be more appropriate to name system-tasks as phrases starting with *imperatives* to signify activities/actions.

3. MOBMAS offers only one technique for the resolution of conflicts within agents and between agents: "priority conventions". The methodology should consider the adoption, or allow the developer to adopt, other techniques of conflict resolution.

4. When comparing between the direct interaction mechanism via ACL and the tuplespace/tuple-centre interaction mechanism in step "*Select interaction mechanism*", MOBMAS does not consider the issue of security support by the two coordination mechanisms. Since security is an important matter in agent interactions, it should be included as a criterion for the comparison between the two coordination mechanisms.

5. In step "*Develop Agent Interaction Model*", the issue of agent synchronisation is not discussed. This step should be extended to include the specification of agent synchronisation in the design of agent interactions.

6. For step "*Identify agent-environment interface requirements*", the provided techniques are insufficient because:
   - they do not address inter-agent communication; and
   - they only focus on "hardware" sensors and effectors (e.g. camera and wheels).

   The support for this step should be extended to account for the various categories of agent-environment interactions and various types of sensors and effectors.

7. In Ontology Model, MOBMAS considers the adoption of only UML notation for the representation of ontologies. This may limit MOBMAS' applicability because UML notation may not be sufficiently powerful for the modelling of highly complicated ontologies. The developer should be allowed to adopt other notation for Ontology Model if necessary.

8. For System Task Model, MOBMAS should mention the possibility of a complex tree structure where two or more system-tasks share the same sub-system-task(s).

9. In step "*Specify resources*", MOBMAS should distinguish between resources that are available to the agents within the system only and those that are available to other systems. This differentiation will help to clarify the system boundary during design.

10. The AND/OR decomposition of system-tasks and agent-goals should be represented using the well-known notation of AND/OR graphs (Figure AppendixF.1) rather than adopting the uncommon notation introduced by TROPOS (Figure AppendixF.2).

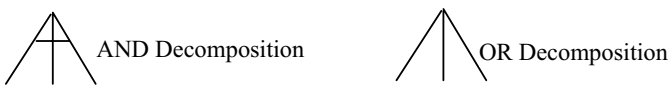


Figure AppendixF.1 – Notation of AND/OR Graphs

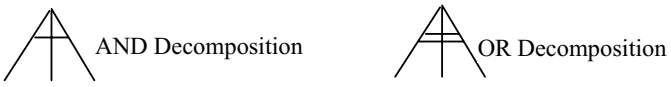


Figure AppendixF.2 – TROPOS notation for AND/OR decomposition

APPENDIX G

EXTERNAL DEVELOPERS' EVALUATION OF MOBMAS

This appendix documents the evaluation of MOBMAS by Dr. Ghassan Beydoun and Dr. Cesar Gonzalez-Perez, who used MOBMAS on a "Peer-to-Peer Information Sharing" application (cf. Appendix H). Each evaluation contained each developer's opinions on the strengths, areas for improvement, how to improve these areas, the ease of understanding and the ease of following of the steps and model kinds of MOBMAS. These opinions were recorded via a specially-designed evaluation form. It should be noted that the forms used by the two developers is slightly different from each other, because they were built upon the two different versions of MOBMAS[102]. Some steps and model kinds listed in the evaluation forms are also different from those specified in the final version of MOBMAS. This is because these forms were based upon the earlier versions of MOBMAS.

Evaluation of Developer 1

DEVELOPER: DR. GHASSAN BEYDOUN

Overall ease of understanding of the development process: High
Overall usability of the development process: High
Overall ease of understanding of model definitions: Medium

[102] Recall that MOBMAS was refined after the evaluation of the first developer.

Evaluation of Analysis Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of step and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Develop System Task Model (optional)	- MOBMAS overlooks the fact that organisational chart can be seen as a result of functional analysis. - Analysis of system functionality should be done for all applications. Analysis of real-world organisational structure can be optional.	High	Medium
STEP 2. "Analyse Organisational Context" (optional)	The real-world organisational structure does not always necessarily correlate with software agent roles.	High	High
STEP 3. Specify roles	MOBMAS does not accommodate dynamic roles.	High	High
STEP 4. Identify Application Ontologies		High	High
STEP 5. Develop Application Ontologies	How the role and task models can be used here is not articulated. There is a lot of focus on notation instead.	Medium	Medium
STEP 6. Identify ontology management roles	This step is described as an option, but there are not any guidelines of when to use this step.	High	Medium
Evaluation of models			
Models	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)	
System Task Model		High	
Real-World Organisational Chart		High	
Role Model	Relationships between this and organisational charts are not explicit.	High	
Ontology Model	Class diagrams are sufficiently powerful to express ontologies. How some of previous models can be used to develop parts of this model is mentioned briefly, without any specific procedures to follow.	High	

Comments on the strengths of Analysis steps and techniques for performing steps:

The first three analysis steps make the transition from the 'system requirement realm' (creating System Task Model) to the agent description realm (Role Model). Development steps offer two layers of abstractions (from system-tasks to roles), getting closer to the agent world design and implementation. The analysis phase also suggests steps from moving from layer 1 (system-tasks) to layer 2 (roles). This layered abstraction view is a useful and intuitive complexity management approach to analyse and implement MAS.

The development steps involved in the above 2-layered abstraction generate (as a by-product) parts of the MAS Application Ontology. Having a stream in the analysis effort focussed on ontology development can be used to verify the completeness of the two earlier models (system-tasks and roles).

Comments on the strengths of Analysis models and modelling techniques:

As earlier commented, the two models System Task Model and Role Model are logically related. The notation suggested for the System Task Model is intuitive (And-Or graph) for non programmers to follow (this notation has been used before in AI to represent declarative knowledge). The process of generating the two models produces parts of the MAS Application Ontology. The development of MAS Application Ontology can serve as the completeness verification of the previous models. If developed by a different person, it may also serve as the validation of the previous two models.

Any suggestions for improvements on Analysis steps and techniques for performing these steps?
I wonder if there should be a spiral development between System Task Model and Role Model. From our experience in the Peer2Peer application with MOBMAS, identifying roles may lead to articulating some lower level system-tasks. For example, in P2P experience, identifying the role 'Portal agent' led to some deeper insights into refining the task 'Locating a Portal Agent'.

Any suggestions for improvements on Analysis models and modelling techniques?

The two models (System Task Model and Role Model) have some ontological units and relationships uncovered. Viewing the ontological view of the system, as a refinement of the view articulated by the two models, may allow the refinement and validation of the two models based on the ontological analysis. For example, a developed MAS Application Ontology can lead to the refinement of the System Task Model, and if another person undertakes the development of the ontology, this may serve as a validation. For example, if the notion of 'time' is captured in the ontology of the P2P system, then the task of 'Timeout recovery' may become relevant.

Evaluation of Architectural Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Specify agent classes		High	High
STEP 2. Specify MAS organisational structure	Hybrid structures where peers talk to each other and report to a mediator at the same time are not discussed. Maybe a new structure ('hybrid') is worth considering.	High	Medium
STEP 3. Specify resources	This is for heterogeneous systems only.	High	High
STEP 4. Develop System Overview Diagram	This is too small to be a step. Is it not combining steps 1 and 3? What does this step do in addition to 1 and 3?	High	High
STEP 5. Identify Resource Application	Should this not be merged with step 6?	High	High
STEP 6. Develop Resource Ontologies		High	High

Evaluation of models			
Models		**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)
Agent Class Model	**Agent Class Diagram**	Too many fields. I wonder whether it is possible to fit everything. With respect to 'ontologies': it is not clear what is meant by this.	Medium
	Agent Relationship Diagram	Association is very general. There are not any specified relationships between classes.	Medium
MAS Organisational Structure Model			High
Environment Model			Medium

Comments on the strengths of Architectural Design steps and techniques for performing steps:

- Considers a wide range of possibilities.
- Has a good focus on open heterogeneous systems.

Comments on the strengths of Architectural Design models and modelling techniques:

NA

Any suggestions for improvements on Architectural Design steps and techniques for performing these steps?

The number of steps should be reduced. Too many steps which are similar confuses. Suggestions to do this: note that the fundamental exercise is one of modelling. The result of this are models. Models can be represented in many ways, diagrams is one way to represent models. For instance, identifying the ontology is a modelling exercise. Why not combine 'identification' and construction of ontologies into one step: 'developing ontology'. An ontology is a model. Diagrammatic representation of the ontology is not a new modelling task. You can use the word 'diagram' instead of model, and combine similar steps into a single step which includes identification (I assume you mean by this identification of the basic units), development (you call this construction) and drawing.

In relation to the above point, you can present organisational structure first, then agent classes (e.g. authority can be mapped to classes relationships in the developed structure); or possibly you can combine these steps into one modelling task which is closely related to the Role Model. Some explicit spiral between the three (agent roles, classes and organisation structure) might be fruitful.

Much of the steps related to resources are related to a specific kind of MAS, heterogeneous systems. I recommend that this point is emphasised more.

Any suggestions for improvements on Architectural Design models and modelling techniques?

See above

Evaluation of Agent Internal Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Specify Agent Belief Set	If the initial belied set is to be modified, it is not clear what MOBMAS recommends (in terms of believed revision).	Medium	High
STEP 2. Specify Agent-goals and Events	It is not clear how an agent would handle conflicting goals. Commitment strategies are not also discussed.	High	Medium
STEP 3. Specify Agent Plans	It is not clear whether the plan is a sequence of sub-goals or a sequence of actions. It ought to be sequence of sub-goals if it is to be general.	High	Medium
Evaluation of models			
Model	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)	
Agent Behaviour Model	See earlier comment regarding agent plans	Medium	

Comments on the strengths of Agent Internal Design steps and techniques for performing steps:
It relates the earlier analysis of the system-tasks to the internal structure and behaviour of individual agents. It goes a long way towards this.

Comments on the strengths of Agent Internal Design models and modelling techniques:
The Agent Behaviour Model is a powerful modelling bridge to link the structure of the internals of agents to system-tasks from the analysis phase.

Any suggestions for improvements on Agent Internal Design steps and techniques for performing these steps?
In complex scenarios, the modelling units of a general plan model can not be the exact actions to be performed by the agents. One way to keep the plan model general is to express it in terms of sub-agent-goals that can be used later on by an off-the-shelf planning language (e.g. STRIPS or ADL). Specifically, these planners will select which actions to perform at run-time, and the sequence of these actions.

Other points which MOBMAS might want to comment on:

- Belief revision and how this or is not related to capabilities of an agent (e.g. learning)
- Commitment strategies (recognition of failed plans)
- Conflict between agent goals (preference and selection of goals)

It is not expected from a single methodology to handle all possible scenarios to deal with those issues. However, an awareness of those issues seems to be in place since the methodology discusses goal modelling to a great degree and learning to a lesser degree.

Any suggestions for improvements on Agent Internal Design models and modelling techniques?

It is not clear how far this phase wants to go into capturing details of an agent. It seems that it is going to very low level (e.g. in verification of ontologies against plans). This probably led to some of the observations above, with respect to how plans should be modelled.

Evaluation of Agent Interaction Design Phase

Evaluation of steps				
Steps		**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Select interaction mechanism			High	High
STEP 2. Develop Agent Interaction Model	**For direct interaction mechanism**	The structure of the organisation of the system can be checked here.	High	High
	For tuplespace/ tuple-centre interaction mechanism	The structure of the organisation of the system can be checked here.	High	High
Evaluation of models				
Model		**Comments on *weaknesses* of models and techniques to produce models**	**Ease of understanding** (High, Medium or Low)	
Agent Interaction Model			High	

Comments on the strengths of Agent Interaction Design steps and techniques for performing steps:

Integrating standard interaction mechanisms with the rest of the methodology makes the methodology much more useable.

Comments on the strengths of Agent Interaction Design models and modelling techniques:

Integrating standard interaction modelling notations (e.g. A-UML Interaction diagrams) with the rest of the methodology makes the methodology much more useable.

Any suggestions for improvements on Agent Interaction Design steps and techniques for performing these steps?

My only concern in this step is where the 'communicative actions' are coming from.

Any suggestions for improvements on Agent Interaction Design models and modelling techniques?

The Agent Interaction Model can be used to verify the system organisational structure as well as the Agent Class Model.

Evaluation of Deployment Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Identify agent-environment interface requirements	There is lots of effort on the distinction between physically embedded agents and software embedded agents.	Medium	Medium
STEP 2. Select agent architecture	Some mentioned agent architectures are not 'market' products (e.g. SOAR is not a market product). Some planning languages are out there in the market (e.g. STRIPS but the more commonly superseding language is ADL).	High	High
STEP 3. Specify infrastructure facilities	It assumes distributed processing. How about MAS built for simulations.	High	High
STEP 4. Instantiate agent classes		High	High
STEP 5. Specify deployment configuration	Agent mobility is represented in this phase. However, from earlier discussion on the architecture phase, mobility is excluded in MOBMAS otherwise.	High	Medium
Evaluation of models			
Model	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)	
Environment Model		High	
Architecture Model	Mobile agents are discussed only in this model (namely Deployment Diagram).	Medium	

Comments on the strengths of Deployment Design steps and techniques for performing steps:
They take into account important issues: number of agents deployed and organisation of external resources.

Comments on the strengths of Deployment Design models and modelling techniques:
It focuses on distributed environment issues, physically embedded agents and external resources planning.

Any suggestions for improvements on Deployment Design steps and techniques for performing these steps?
MAS built for simulations are not accommodated in this phase. In addition, mobile agents seem to appear only in this phase. I have been under the impression that MOBMAS does not deal with mobile agents.

Any suggestions for improvements on Deployment Design models and techniques modelling techniques
Explicitly exclude mobility or qualify and reconcile why it is accommodated here and is not accommodated elsewhere. If the notation for MAS Deployment Diagram accommodates mobile agents, but MOBMAS does not, I suggest that this is explicitly stated e.g. in a footnote. MOBMAS should also address the case of MASs developed for simulation.

Evaluation of Developer 2

DEVELOPER: DR. CESAR GONZALEZ-PEREZ

Overall ease of understanding of the development process: High

Overall usability of the development process: High

Overall ease of understanding of model definitions: High

Overall comments on MOBMAS

MOBMAS is described as being composed of a process, a collection of techniques and *models*. I imagine that you mean *model specifications*. As far as I understand, MOBMAS does not contain models; the models will be built by people using MOBMAS. MOBMAS contains the specifications (or definitions, if you want) of how to build these models. You can use the terms "model specifications", "model definitions" or, even better, "model kinds". Thus you could describe, for example, the Agent Interaction Model Kind, which specifies how developers can build Agent Interaction Models. But these models (which users will create) are not part of MOBMAS; they are an outcome of applying MOBMAS. I would suggest changing the wording wherever it is necessary to clarify this point, because in its current form it results confusing and detracts significantly from the otherwise excellent conceptualisations found in MOBMAS' documentation. You can have a look at the Australian Standard AS 4651 "Standard Metamodel for Software Development Methodologies", which defines concepts such as Model, Model Kind, Model Unit and Model Unit Kind (what you call "modelling concepts"). This could be useful to you.

MOBMAS is also described as being composed of "phases". What do you mean by "phase" here? From the text I imagine that you see "phase" as meaning the description of certain work that must be done. For example, the Architectural Design phase is different from the Agent Internal Design phase because it involves work of different a kind. If I am right, the correct term in software engineering is not "phase" but "activity" (OPEN) or "process" (ISO 15504, OOSPICE), or even "discipline" (SPEM). If you say "phase", software engineers will immediately think of temporal concerns. Phases mean giving temporal structure to the work defined by activities. Since you claim that MOBMAS is iterative and incremental (more on this later) and that the described phases do not necessarily have to be performed in order, then they are not phases because they do not address timing concerns. I suggest you drop the term "phase" in this context and adopt "activity" instead. After this, and only if you want to go beyond, you can organise your five activities into phases to give them temporal structure.

MOBMAS identifies system-tasks from system-goals. What if the target system does not have a single or overall goal? This is a criticism that has been made repeatedly to functional decomposition in traditional structured analysis and design approaches. Imagine an operating system. What is the System Goal? I can argue that there is no overall goal, and if you come up with one, I can show that it is a shoehorned example. If you really want to use the concept of System Goal, then you are constraining your methodology to systems that can be successfully described by an overall goal. You need to acknowledge this and accept that MOBMAS will not be usable for other kinds of systems.

You then define some terms such as System Task or Role. I have a couple of comments on the definitions. First, some of them (Role and Agent Class) are not definitions but comments and discussion. Regarding Role, you say that a role is "something similar to…" which is not really a definition. I would suggest that you try to find a proper definition as you have done with other concepts. Secondly, I would try to keep definitions concise and unambiguous. For example, you define System Task as "a concrete, low-level activity, or unit of work, that…". What is the meaning of "or" here? Is a System Task either a low-level activity *or* a unit of work? Is "low-level activity" the same as "unit of work"? You probably want to simplify this definition. Also, when defining Agent Class, you say that "agent class is abstract and…". What do

you mean by “abstract”? Finally, when defining Agent Plan Template, you use the concept of “sub-Agent Goal” that has not been introduced.

I also have some comments about the contents of the definitions. First, in Agent Goal, you say that the agent goal implies proactiveness, since agents need to take the initiative to satisfy their goals. I don’t agree with this. For example, I have a phone switch router agent that has the goal “route incoming calls to their destination using the optimal path”. It is a purely reactive agent, which uses a collection of lookup tables to decide how to do the routing. But it still has a goal. For this agent, having a goal does not imply that it is proactive or takes the initiative. I think that having a goal and being or not proactive are completely unrelated things. Secondly, you define Event as something to which an agent reacts. There is a subtle issue here. Imagine an event that happens and no agents react to it. It is still an event, right? Whether or not some agents react to an event is up to the agents, and is not dependent on the event. I suggest you change “to which an agent reacts” to “to which agents may react”. Thirdly, you define Reflexive Rules as “a sequence of actions that an agent performs to react to an event”. I would say “when reacting” rather than “to react”, since the reactive course is part of the reaction itself.

Figure 6.1 shows MOBMAS’ core concepts plus the relationships amongst them. You say that it is a meta-model. I would not use that term, since “meta-model” implies that this is a model of another model. This can be circumstantially true in this case, but is not the focus of the discussion. In my opinion, you are just presenting a *diagram*, not a *meta-model*. If you use the term “meta-model” I would expect a reason why this is so and not just a simple model or even a diagram. In addition, what is the notation being used for Figure 6.1? I mean, what is the meaning of the boxes, lines, arrowheads, diamonds, etc.? You need to include a legend or either make a reference to some well-known notation such as UML. Then you need to make sure that you stick to the standard. Secondly, the diagram does not show cardinalities, which would be very useful. For example, I can see in the diagram that role-tasks are derived from system-tasks, but how many of each? Furthermore, the diagram includes boxes labelled “Action” and “sub-Agent Goal”, which have not been defined or introduced previously. You probably need to check the consistency of this.

Section 6.1.3 describes the notational components for each model. You use the term “composed of” (or a synonym of this) to describe the relationship between a model kind (such as System Task Model) and the notational components that can be used to depict it (System Task Diagram in this case). I don’t think the relationship is of containment or composition. A model does not *contain* a diagram, but *can be depicted by* it. I would rather say that models can be depicted by notational components. This happens all over Section 6.1.3 at least. Moreover, I would expect the notational components to “depict” or “show” (or even “document”) things, but not “define” or “capture” or “specify” anything. Models define, capture and specify; notations show or depict. However, some of the notational components are described as defining, capturing or specifying their contents. I suggest you change these terms to “depict” or “show”. This also happens within other model kinds in this section. You also say that four models in MOBMAS reuse and extend UML notation. If they are models, how can they reuse a notation? You probably want to say that *the notational components* that depict MOBMAS models reuse and extend UML notation. I suggest clarifying this.

In the Organisational Context Model Kind, you use the term “real-world” quite often to refer to the organisational structure. Why “real-world”? From the explanation in the text, I know that you want to emphasise that the modelling is of the organisational context of the system rather than the system itself, but wouldn’t “organisational” be enough? If you add “real-world”, you are introducing two potential problems. First, I could ask “what do you mean by ‘real’?”, or “real to whom?”. This is a complex philosophical issue that I bet you don’t want to get into. Secondly, you are limiting your model to the “real” world (whatever it means) and discarding other “non-real” worlds. And software is plenty of non-real, virtual worlds! For example, if we take “real” as meaning simply the physical world in which we live, then a fictitious organisation that I find in a novel is not part of the real world. Can I still use MOBMAS to model it? I’m sure I can. But your Organisational Context Model is not depicting the real world. I would suggest getting rid of “real-world” in this context.

When describing Agent Behaviour Model, you introduce the notational components named “Agent Plans” and “Reflexive Rules”. I don’t think these are good names for notational components, because they are just the plural form of modelling concepts. “Agent Plans” really means a collection of agent plans; it does not convey the idea of a diagram or a document. But you want it to mean some other thing, namely a specific notational component that depicts a view of the Agent Behaviour Model. I think that you probably need to name it accordingly. For example, “Agent Plan Diagram” or “Agent Plan Description” or something like that. The same happens with “Reactive Rules”.

Evaluation of Analysis phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of step and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Develop System Task Model	- You claim (backed by Fan 2000) that users often have a clearer idea of system goals than of system tasks. In the experience of other authors (me included) this is quite the other way around. - You say that use cases can be used to identify system goals, but do not say how. - You talk about system goals producing and consuming resources. How can a goal produce or consume anything? Goals are states. You need some kind of process, activity or task in order to produce or consume resources. - I would suggest you have a look at “Software Requirements” by Karl Wiegers. Very different approach to the classical references. There is a new second edition now. - You make constant references to the “actors” or “performers” of system goals and tasks. But we haven’t determined them yet. We know nothing about them at this stage! - You talk about system tasks being “fulfilled”. I imagine you mean “executed”.	Medium	High
STEP 2. “Analyse Organisational Context” (optional)	- You start saying that the previous step elicits system functionality. I think it models system functionality. Elicitation is only a part of requirements engineering. - I would avoid using the term “real-world” here. Please	High	High
STEP 3. Develop Role Model	- One of the biggest problems of traditional structured methods is that they rely too much on functional decomposition. You are at risk of falling in the same trap here. You say that “System-tasks that share the same parent […] are typically related strongly in term of functionality, thus being good candidates for being combined into one role.”. I strongly disagree. Sharing a common parent is a functional property, completely unrelated to the structural property of who the responsible for such piece of functionality is. Sharing or not sharing a parent is irrelevant as far as role	Medium	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of step and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	assignment is concerned. Only this way you will avoid creating a system organised around a functional decomposition of its requirements. - Furthermore, you say that System Tasks should not be associated to the same role when they are expected to be executed on distributed physical locations or when they interface with distributed or different kinds of resources. Why? Some roles are precisely defined to coordinate tasks like this! - I don't think that the term "social task" is appropriate. The term "social" (systematically misused by the AI community) has strong implications such as the existence of a common set of rules and both subjective and inter-subjective spaces. I don't think you mean this. I would avoid this term and use "joint", "distributed" or "collective" instead. If this is what you mean, why use another word? - The cardinality of the relationship between System Tasks and Role Tasks is not clear. - You use guillemets to label the different sections in a box representing a role. This is extremely confusing since UML uses the same symbol for stereotypes. I would simply remove these symbols.		
STEP 4. Develop Ontology Model	- You state that the Conference Program Management system does not need a Domain ontology because the information to be handled is just "profiles" of the conference papers. First, I don't understand what you mean by "profiles". Secondly, I can see a clear Conference Program domain ontology containing the concepts Paper, Reviewer, etc. - I would say that an application always relates to a particular domain. Actually, I can argue that "application" and "domain" are almost synonyms. Therefore, you cannot say that an application only needs Task ontology without needing Domain ontology. If an ontology defines concepts and their relations (structure), this is orthogonal to any usage (behaviour) that you may do of such concepts. For example, you say that User Query, Keyword and Result List are domain independent and you build a "task ontology" for them because you happen to use them for a particular task. I would adopt a different approach. To me, these concepts belong to a Query Management domain that exists on its own and which	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of step and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	you happen to need for your particular task. - In connection with the previous point, it seems that an ontology that "specialises" from a more general one can do it by (a) adding subtypes of the concepts in the general ontology and (b) incorporating specific instances of such concepts. This is not said. For example, Diabetes is not "subsumes" by Disease, but is an instance of it. Similarly, Hyperglycaemia and Hypoglycaemia are instances of Symptom, and not subtypes. - You need to say what notation you are using for the diagrams. You say earlier that UML is used for ontology diagrams, but the reader will not know that when looking at diagrams before that. - You say that MOBMAS recommends a graphical language for ontology modelling but, if not powerful enough, a textual one can be used. Are you assuming that graphical languages are, in general, less powerful than textual ones? I would differentiate between power of expression and power of communication. The first relates to the abstract syntax and the semantics of the language, which are not related to graphical vs. textual whatsoever. The second relates to concrete syntax or notation, which does relate to graphical vs. textual. Some clarification would be helpful here. - You introduce the concept of generalisation but define specialisation! - You introduce the concepts of aggregation and composition. These concepts are poorly defined by UML, and they can get you in some trouble (see some papers by Henderson-Sellers on whole/part relationships). If you are taking these concepts straight from UML, I would recommend you just point to the definitions in UML and avoid including your own definition. If you want to enhance over UML, then you need to be careful with how you define these terms. See Brian's papers for details. - You say that relationships may be annotated with cardinality indicators. If this is only optional, how are the relationships supposed to be implemented? You need cardinality specifications for every single relationship before you can put them into a computer. - You consider three types of "ontological mapping"		

Evaluation of steps			
Steps	**Comments on *weaknesses* of step and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	relationships: equivalent, subsumes and intersects. I can see two problems with this. First, how is subsumption different from instantiation or specialisation? The definition is not clear; you say that "…one concept includes the other…", but the meaning of this can be either instantiation or specialisation. Secondly, and most importantly, an association in object-oriented modelling (e.g. UML) describes a potential relationship between instances of the involved classes. It does not describe a relationship between the classes themselves. However, from the definitions and description of your three stereotypes, you are trying to characterise relationships between classes. For example, if you say that Disease subsumes Diabetes, that means that instances of Disease may subsume instances of Diabetes. And this is not what you mean; you mean that the class Disease subsumes the class Diabetes. In summary: associations are not the appropriate mechanism to do this. - In Figure 6.13, how can a "Hit" be equivalent to a "Car"?		
STEP 5. Identify ontology management roles	To me, ontology management looks similar to database management in traditional systems. It is an infrastructural service provided to applications, and therefore application development usually does not deal with the modelling of such infrastructure. If this is not the case and ontology management is not an infrastructural service of MAS applications, this must be clearly stated to avoid confusion.	High	High

Evaluation of models		
Models	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)
System Task Model		High
Organisation Context		High
Role Model		High
Ontology Model		High

Comments on the strengths of Analysis steps and techniques for performing steps:

A lot of heuristics are given, which is unusual for a text on methodologies and extremely welcome.

Comments on the strengths of Analysis models and modelling techniques:

See above evaluation

Any suggestions for improvements on Analysis steps and techniques for performing these steps?
See above evaluation

Any suggestions for improvements on Analysis models and techniques modelling techniques
See above evaluation

Evaluation of Architectural Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Specify MAS organisational structure	- The phase is called "architectural design" but you talk about specifying the "organisational structure" of the system. What is the meaning of "organisational" here? If you mean architecture, just say "architecture" instead of "organisational structure". - You mention the term "control regimes" a few times. What is it? Control issues in a system are part of the system dynamics, which happen at run-time. On the other hand, architecture is a structural, design-time concept. Architecture cannot specify control, but support it. - All your discussion of organising roles in layers or federated groups seems to be based exclusively in performance reasons ("helps to reduce interaction traffic", "interaction costs are sufficiently low"). This is not the reason that most authors would give for architecting a system in layers or blocks. The main reasons are two: (a) modularity (most important) and (b) non-functional requirement support. Modularity is concerned with separation of concerns, technological isolation and, in general, encapsulation and containment. Non-functional requirement support is related to providing the necessary structural mechanisms so the necessary non-functional requirements (robustness, integrity and performance) are met. As you can see, performance is only a small bit. - You keep using keywords enclosed in guillemets, which closely resemble UML stereotypes. I don't think this is a good idea. - You use an arrow sign embedded in the «control→» keyword. This does not look very intuitive. Furthermore, it ties the model to the graphical representation, which is not a good idea. I would try to replace it by an adorned line, perhaps. - After Figure 11.6, you say that roles should be organised so they highlight the layers or groupings	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	adopted. Well, I think that this works the other way around. Once you organise your roles in a sensible manner, layers or groupings will arise. You cannot force your roles into groups just to make the model look neat.		
STEP 2. Develop Agent Class Model	- You talk about "grouping" roles into agent classes. I would say "associating" roles to agent classes. - The reasons that you give to associate multiple roles to a single agent class look very arguable to me. First, you say that if two roles interact intensively with each other, they should be associated to the same class. I don't think this is right. A client and a server (in whichever domain you want to think of) interact intensively but are, by definition, well separated entities. Second, you say that if two roles share a lot of common data or resources, then they again should be mapped to the same class. Again, I disagree. In my view, the most important issue you need to look at in order to decide whether or not to put two roles together into a single class is semantics. Look at their names, look at whether it makes sense, from a semantic perspective, that a single class plays both roles. You hint at this by talking about coherence and having a single class name with no conjunctions. This is OK; now you need to change your heuristics (currently based on interaction bandwidth and shared data) to match this. - You give some advice on the computational complexity of agent classes. I think this is not realistic at this point. When you develop a software system in the real world, you almost never know what kind of hardware is going to run it, and even if you know it, it will change every 12 or 18 months most probably. So any reasoning based on processor load, at this stage, seems inappropriate to me. - There is a great and ongoing ambiguity between "agent" and "agent class". You sometimes talk about assigning roles to agents, but you have just said that roles are assigned to agent classes. You probably want to revise this whole section and make sure that "agent" and "agent class" are used with rigour. - For the Agent Class Diagram, you say that when an	High	Medium

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	agent class dynamically plays a particular role, this role must be differentiated from other static roles by being annotated with an adornment. I don't think this is useful at all. Structural models show static properties of entities, i.e. properties that are observable at any point in time. When you represent an agent class in the Agent Class Diagram and enumerate its roles besides its name, you are saying that in the system there will be agents of that class and they will be able to play these roles. Whether or not a specific agent is playing a specific role cannot be guaranteed to be observable at any point in time. All roles look the same, no matter whether they are static or dynamic. That is a dynamic concern. You should remove the special annotation that dynamic roles are supposed to have in the current model definition, since it does not make sense in a structural model. - Some of the examples in the text are not very fortunate, in my view. For example, a Search Agent class is assigned roles User Interface and Searcher. The encapsulation of a computation (searching) and a user interaction (user interface) in the same entity violates the n-tier philosophy of separating persistent data from data access from computation (business logic) from user interaction. I am not sure whether this is a characteristic of agent modelling or just an oversight.		
STEP 3. Specify resources	- In your characterisation of resources, you use the term "knowledge source" to refer to databases and web servers. Well, a database can be seen as a data source, but only very arguably as a knowledge source. Why not use a more conventional, all-encompassing term such as "information sources"? Knowledge is a different thing. - You introduce an Environment Model which contains 3 notational components. One component (Resource Diagram) is developed in this step, while the remaining two will be dealt with in Deployment Design. I don't like this because, to me, a model describes its target at a certain level of abstraction and from a certain perspective. Phases as different as Architectural Design and Deployment Design almost certainly vary significantly in abstraction and	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	purpose, so how can they all be major contributors to the same model? - In the modelling of resources, you mention the "communication properties" dimension. This is very low level at this stage. Things like the IP address of a machine are probably irrelevant at the architectural design level (which is what you are doing here). I would remove this bit and move it to Deployment Design, perhaps. - You recommend considering the introduction of a Resource Broker role. This looks to me like something belonging to the infrastructural services of the run-time environment, not something that each application needs to design. It is similar to a name server or a middleware service in traditional services: they are already there for you to use. You don't design a new one for each application you write. - This step uses some not very good examples. For example, role task "Provide yellow page services" and "Display acknowledgement" are at very different levels of granularity. I would try to keep role tasks (or any other model units of the same kind) at the same level of granularity. Furthermore, the Feedback Manager role has been assigned the task "Display acknowledgement", which seems to belong more naturally to the User Interface role. Similarly, the User Interface role contains the task "Extract keywords from user query" which looks like a computation and not a user interface operation.		
STEP 4. Extend Ontology Model to include Resource Application ontology	- You say that for resources that are information sources, the resource ontology is the conceptual schema of the information stored in the resource. This is arguable. Consider levels of abstraction: the information source may store information at a very low level of abstraction, and therefore its conceptual schema would contain all sorts of details that are irrelevant (and even harmful) to you. You need a different ontology, one that maps to this conceptual schema but removes unnecessary detail. In my experience, this happens all the time in real-life projects with databases. So you cannot simply say that the conceptual schema of the information source is equivalent to the resource ontology.	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	- You say that, in some cases, a resource ontology can coincide with (a fragment of) the application ontology, and that, in this case, the application ontology will suffice. I don't agree, because the non-agent resource, by definition, is external to your system, and keeping a separate ontology for it, even if it repeats concepts in you application ontology, is recommendable for the sake of modularity. Imagine that you want to make a change in your application ontology but need to keep the resource ontology untouched: you cannot do it unless you have the two ontologies separate.		

Evaluation of models			
Models		**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)
Agent Class Model	**Agent Class Diagram**		High
	Agent Relationship Diagram		High
MAS Organisational Structure Model			High
Environment Model			Medium

Comments on the strengths of Architectural Design steps and techniques for performing steps:
NA.

Comments on the strengths of Architectural Design models and modelling techniques:
NA.

Any suggestions for improvements on Architectural Design steps and techniques for performing these steps?
See above evaluation

Any suggestions for improvements on Architectural Design models and techniques modelling techniques
See above evaluation

Evaluation of Agent Internal Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Specify Agent Belief Set	- The name "belief set" is not intuitive. "Belief set" means a set of beliefs, not a formal structure of beliefs. You may want to change this name. - Along the whole chapter, there is an ambiguity of "agent" vs. "agent class". This also happened in earlier chapters. For example, you talk about defining the belief set "for each agent", when you surely mean "for each agent class". - When a belief set changes (because the ontologies change), the belief state must also change accordingly, adjusting itself to the new structure. You don't discuss this. How are agents notified of ontology changes? - A belief set, as defined, is one or more ontologies to which the agent commits. Isn't this overkill? I can think of many cases in which an agent would only need a small bit of an ontology to do its work. Therefore it would be nice if agents could commit to a subset of a given ontology.	High	High
STEP 2. Specify Agent-goals and Events	- Across the whole chapter, it is stated (and assumed) that reactive agents exhibit simple behaviour while pro-active agents exhibit complex behaviour. For example, you give some rules to classify a Role Task of an agent class as pro-active if the task will need deliberation and complex processes. Also, you recommend a reactive architectural style if the agents hold a simple representation of the world, while a pro-active style is suggested if the knowledge involved is more complex. In my view, this is wrong. Two different concerns are being mixed here. The first is pro-activity vs. reactivity and the second is the complexity of behaviour. Pro-activity and reactivity are related only to the particular way in which a piece of behaviour (an action, a goal, whatever you want to call it) is triggered. The degree of complexity of this behaviour is completely orthogonal to how it is triggered. For example, I have a phone switch router agent that is completely reactive: only does something when a call comes in. The behaviour that this triggers, however, is highly complex, and a lot of deliberation with other agents is necessary.	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	- You classify the tasks "Extract keywords from user query" and "Get information from resources" as pro-active tasks. I disagree. You state that they are triggered by some stimuli, so they are a reaction to something. Therefore, they are reactive. You say that they are pro-active because they are highly complex, which is true, but not related to proactivity/reactivity at all. - In connection with the above two paragraphs, I think that the concepts of proactivity and reactivity are useful as abstract ideas to characterise agents, but they are not that useful at a detailed level. In software systems (and in most systems, actually), everything is reactive to a high degree. You keep talking about stimuli that trigger proactive tasks (see Figure 12.11 for example). This is an oxymoron. Only really complex entities (such as human beings) can be truly pro-active, i.e. initiate action without a stimulus. I am aware that this is a deep issue with multiple ramifications, but unfortunately I cannot defend your usage of proactivity/reactivity. - Regarding events, you define them as something significant that happens in the environment. Now, how do you define "environment"? Does the environment include other agents? Does it include all the agents, including self?		
STEP 3. Develop Agent Behaviour Model	- For each action, you define preconditions and effects. Why don't you use the term "postconditions" instead of "effects", for the sake of symmetry? - You mention that two (or more) agents can share the same agent-goal. However, from you previous chapters, I recall that each agent has its own agent-goal, but the definitions of these goals are the same. So, strictly speaking, they do not share a goal, but have equivalent (or identical) agent-goals. This may seem a bit of word play but it is not, as you can see in the next paragraph. - You say that when two (or more) agents "share" the same agent-goal, they will have to interact to achieve that goal and possibly do some distributed planning. I don't think this is necessarily true. Since agents do not actually share a goal but have goals that are identical (see previous paragraph), they may as well pursue their identical goals separately.	Medium	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for step**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	So, distributed planning is only an option. - The differences between blind (or fanatical), single-minded and open-minded strategies are hard to understand. I suggest you use the exact same words to define each one changing only the minimum. - Your usage of "vice versa" is a bit odd. This expression means that the same relationship that happens between A and B also happens between B and A. But in most of the cases where you use it, the relationship between the parties involved is not the same in one way and in the other. - You use attributes from the ontologies as datatypes. For example, you define carModel: Car.Model. This gives no room for using "basic" parameters such as a string or an integer that bear no relationship at all to the ontologies. For example, if you want to display a message to the user or wait for a specific number of seconds, you would need to pass a string or an integer, respectively. Forcing the developer to create a class in the ontology (and perhaps a whole new ontology!) because of this seems inappropriate.		
Evaluation of models			
Model	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)	
Agent Behaviour Model		High	

Comments on the strengths of Agent Internal Design steps and techniques for performing steps:
A lot of heuristics are given, which is unusual for a text on methodologies and extremely welcome.

Comments on the strengths of Agent Internal Design models and modelling techniques:
NA.

Any suggestions for improvements on Agent Internal Design steps and techniques for performing these steps?
See above evaluation

Any suggestions for improvements on Agent Internal Design models and techniques modelling techniques
See above evaluation

Evaluation of Agent Interaction Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Select interaction mechanism	- The capitalisation of "tuple centre" is random. You may want to homogenise it. - You frequently use the term "coordination" when you really mean "interaction". Many interactions are not related to coordination at all. I think you should use "interaction" where you say "coordination", except for the (perhaps) few places where you really mean "coordination". - You say that using a tuple centre and using direct messages between agents are the two common agent interaction mechanisms. However you do not say which others exist, or where to find them. In this step, things are explained as if these two mechanisms where the only ones. - You say that "…the ACL interaction mechanism […] infers a strong…". I am not sure what you mean with this sentence, but "to infer" means to deduce, to conclude. Is that what you mean? - Also in the same section, you sometimes use the term "agent" when you really mean "agent class". This is extremely confusing. For example, you say that embedding the constraints that govern agent interaction into the agents themselves can be difficult if the number of agents is large. You mean agent classes, I bet. Otherwise, the sentence does not make any sense. - In a footnote in the same section, you say that if an agent goes down, then another agent of the same class can replace it. I don't think this is the rule. As you well know, agents obtain knowledge during their lives, so two agents of the same class that have lived for a while will probably have different beliefs and intentions. How can then one replace the other? - You say that the ACL Interaction Mechanism is not as "efficient" (I would suggest "appropriate" instead) as using the tuple centre mechanism when the MAS is open and dynamic, contains heterogeneous agents, and when the agents have many shared agent-goals. You say that this happens because there is a strong coupling between "the agents' behaviour and the management of the coordination process". I think you should change the wording here, because the coordination process is part of	Medium	High

Evaluation of steps				
Steps		**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
		agents' behaviour. How an agent interacts with other agents, what for and when, is all part of the agent's behaviour. In addition, I don't think that agents have "shared agent-goals". They may have the same agent-goals, but not shared. I already discussed this in a previous chapter. - You say several times that the tuple centre is capable of some "reasoning", or even "low-level reasoning". Do you mean "processing"? If so, I would use "processing", which has the right technical meaning. "Reasoning" is what humans do.		
STEP 2. Develop Agent Interaction Model	**For direct interaction mechanism**	- In "Specify agent synchronisation" you supposedly describe two synchronisation methods: synchronous and asynchronous. Well, these are not synchronisation methods, especially the latter. If a MAS adopts an asynchronous approach, the agents in it are not synchronised at all, by definition. They will need to use mutexes or some other synchronisation objects to actually synchronise with one another. This would be "synchronisation methods". What you are describing are "synchronisation approaches" or "modes" but not "methods". What's the meaning of "method", by the way? - You define a guard condition as the condition "by which" the message is sent. I can't understand this. Is it the condition that sends the message, that makes sensing the message possible, that triggers the message being sent…? You need a better definition here. - You try to define "sequence-expression" but the words are not a definition. It is only a comparison with UML. - When describing the content of messages, you use datatypes that look very much like coming from an ontology. However, you don't say that. I think it would be good, at this stage, to say explicitly that datatypes map to the some ontology. - You use some "built-in" datatypes such as "Integer". What do you mean by "built-in"? Where are they built-in? Who or what provides these datatypes?	High	High

Evaluation of steps				
Steps		**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
	For tuplespace/ tuple-centre interaction mechanism	- Again, in the message content, you use datatypes that look like coming from an ontology but you say nothing about this. This is puzzling for the reader. - Again, you mention the existence of "built-in" datatypes. Where are they built in? - You describe "synchronisation methods" which are really approaches or modes, not methods. - Figure E.6 shows an "out" arrow coming out of the tuple centre into an agent. I think that is wrong since "out" arrows always go into the tuple centre. - What does an interaction diagram represent? You don't say that anywhere. Is it a specific conversation, or is it a specification of the conversations that may take place?	High	High

Evaluation of models		
Model	**Comments on *weaknesses* of models and techniques to produce models**	**Ease of understanding** (High, Medium or Low)
Agent Interaction Model		High

Comments on the strengths of Agent Interaction Design steps and techniques for performing steps:
A lot of heuristics are given, which is unusual for a text on methodologies and extremely welcome.

Comments on the strengths of Agent Interaction Design models and modelling techniques:
NA.

Any suggestions for improvements on Agent Interaction Design steps and techniques for performing these steps?
See above evaluation

Any suggestions for improvements on Agent Interaction Design models and techniques modelling techniques
See above evaluation

Evaluation of Deployment Design Phase

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 1. Identify agent-environment interface requirements	- Again, you sometimes use the term "coordination" meaning "interaction", like in the previous chapter. - You keep mentioning the implementation phase but it is not discussed in MOBMAS. Since you refer to it in specific terms, I wonder how you conceptualise it. What kind of work is expected to be done? What products are generated? Can you add some information on this to your work? - I would think that designing the agent's interface with its environment is not related to deployment at all but architecture or detailed design. It is an integral part of the agent, not related to the specifics of the run-time infrastructure. - You mention differences between sensors/effectors in hardware and software. In real life situations, software developers do not interact with hardware; hardware is virtualised or wrapped by driver components so other software components interface directly with them. For example, if your agent has an effector to move a robotic arm and you want to make it move the arm, you will talk to a software component that virtualises the physical arm. You don't have to worry about the hardware. So, for the software developer, there is no difference at all between interfacing with software or hardware. - As part of the identification of agent-environment interaction requirements (you use "interaction" here rather than "coordination", which is good), you include issues that have already been dealt with in previous phases, such as deciding whether to use a tuples centre or direct messages. - You explain that the Environment Model contains 3 notational elements, 1 of which is created by the Architectural Design phases. I have already commented on this. But here you can see very clearly how the different notational elements of this model have really nothing to with each other. They are not view on the same model but completely unrelated models. I suggest you decompose the Environment Model into smaller models which will make much more sense.	High	High

Evaluation of steps			
Steps	**Comments on *weaknesses* of steps and techniques for steps**	**Ease of understanding** (High, Medium or Low)	**Ease of following** (High, Medium or Low)
STEP 2. Select agent architecture	- You talk about the Implementation phase, but you have not mentioned it before. Is there one? - You say that there are a number of agent architectures available on the market. What do you mean? Are agent architectures products that you can buy? - You mix the concerns of proactivity/reactivity with the complexity of behaviour, as I said some paragraphs earlier. You probably want to revise this. - Two of the criteria that you mention are "Size of knowledge base" and "Support for scalability". The differences between these two are not clear. - Figure 6.51 shows some little solid black circles that are not defined.	High	High
STEP 3. Specify infrastructure facilities	In my view, everything in this step is actually architecture design, not deployment design. Specifying how a system will interact with other systems, including its infrastructure, is part of architecture. What do you understand by "deployment"?	High	High
STEP 4. Instantiate agent classes	You use a rounded rectangle to represent an agent instance in the Agent Instantiation Diagram. What do you need this for? Agent instance icons bear no information and add no value to the diagram. Can't you get rid of them altogether?	High	High
STEP 5. Specify deployment configuration	You use the word "physical" quite often. For example, you say that agent platforms are the *physical* infrastructure in which agents are deployed. And nodes are *physical* hosts. And they are linked by *physical* connections. I am not sure what you mean by "physical", but if you delete the word from these sentences everything makes sense and, in addition, you are not limited to "physical" entities. For example, nodes do not have to be real computers; they can be virtual machines or other non-physical processors. The same for connections between nodes. Network connections are physical at a very low level, but most application deployment activities take place at high levels where the physical topology of the network is not important, only its logical topology. I would suggest deleting this word unless you have a good reason to keep it.	High	High

Evaluation of models		
Model	**Comments on *weaknesses* of models and modelling techniques**	**Ease of understanding** (High, Medium or Low)
Environment Model		Medium
Architecture Model		High

Comments on the strengths of Deployment Design steps and techniques for performing steps:

NA

Comments on the strengths of Deployment Design models and modelling techniques:

NA.

Any suggestions for improvements on Deployment Design steps and techniques for performing these steps?

See above evaluation

Any suggestions for improvements on Deployment Design models and techniques modelling techniques

See above evaluation

APPENDIX H

APPLICATION OF MOBMAS

This appendix documents the "Peer-to-Peer Information Sharing" application on which MOBMAS was used by the two developers, Dr. Ghassan Beydoun and Dr. Cesar Gonzalez-Perez. The appendix also presents the major models produced by each developer to illustrate the design of MAS for the application as a result using MOBMAS.

Problem Description – "Peer-to-Peer Information Sharing"

In recent years, the Peer-to-Peer (P2P) networking paradigm has become one of the most rapidly developing areas of modern computing (Klampanos and Jose 2003). P2P contrasts with the well-known Client-Server networking model in that all nodes in the network are capable of acting as both server and client, that is, each node can serve as both provider and user of services (Klampanos et al. 2003; Mine et al. 2004). The P2P networking model helps to avoid the problems of bottle-neck and heavy traffic that is commonly witnessed in the Client-Server architecture.

In this application, the P2P model is employed for *information sharing*. Information to be shared is files such as HTML, pdf and multimedia (e.g. music or video). The users of the system are distributed "peers" in the information sharing network. Their knowledge bases (i.e. stored files) are enlisted, and the users can communicate directly with each other to exchange this knowledge.

System requirements

Each user possesses a knowledge base containing files that he/she is willing to distribute to other peer users. Each file is identified by its title and type (e.g. HTML, pdf, music or video).

Any user in the network can enter a query to request for files that satisfy his/her query. Each query contains a set of keywords. The system is responsible for identifying those candidate users who may have files that satisfy the query, and sending the query to these users. The answer from each candidate user may either be:

- the titles and types of the files that satisfy the query; or
- a refusal of service (either because no appropriate files are found, or because the user is unwilling to supply the files at the time of request).

When the answers are received from all candidate provider users, the system will combine and refine the results to compose a list of files' titles and types, which is then presented to the user. The user can then select which files he/she wants to download. The system then contacts the respective provider user to carry out the file transfer process. After a successful transfer, the user's knowledge base is updated to contain the new received file(s).

Each user keeps a record of his/her history of information sharing. The history contains:

- a list that records the queries made by the user and their responders; and
- a list that records the queries received by the user and their senders.

The former needs to be updated every time the user receives a result list from the system, while the latter requires update every time the user replies to a query sent by the system. This history lists help the system to produce *short lists of candidate providers* for future queries, by calculating the similarity between the user's query and a past query (Mine et al. 2004). If no candidate providers can be identified this way, or if all candidate users do not provide the service required, the system will need to broadcast the query to all users in the community, so as to identify new candidate providers. The new providers are eventually added to the history of the user, thereby expanding the user's contact circle.

Although the above schema of information sharing can be applied to any application domain, this research illustrates the use of MOBMAS on the *Movies* domain. An ontology for this domain (written in DAML) is currently available from http://www.cse.dmu.ac.uk/~monika/Pages/Ontologies/CinemaAndMovies.daml.

Major models produced by Developer 1

DEVELOPER: DR. GHASSAN BEYDOUN

System Task Model

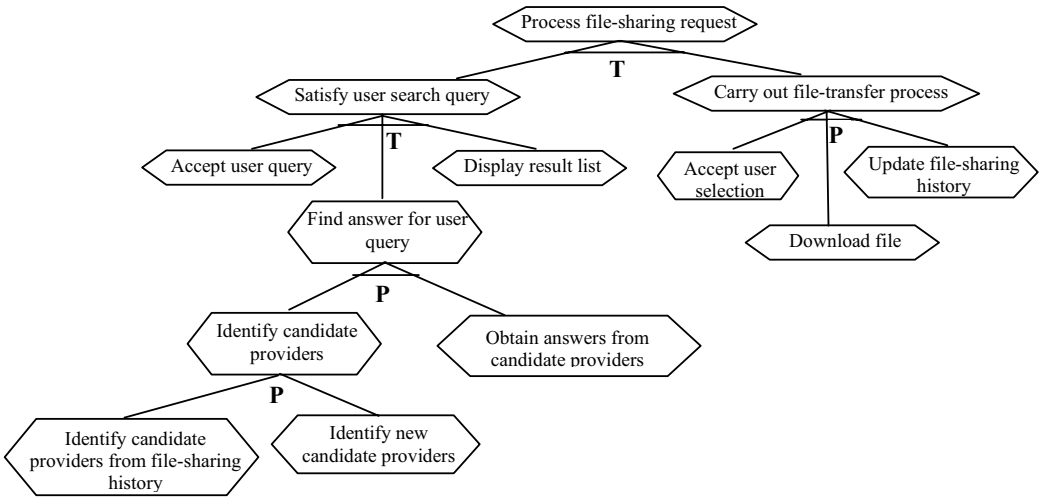


Figure AppendixH.1 – System Task Diagram by Developer 1

Ontology Model

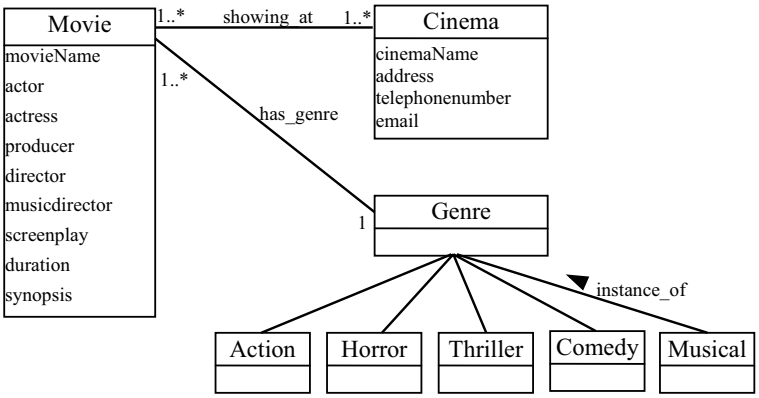


Figure AppendixH.2 – Ontology Diagram for *Movie Ontology* by Developer 1

(based upon DAML ontology at

http://www.cse.dmu.ac.uk/~monika/Pages/Ontologies/CinemaAndMovies.daml)

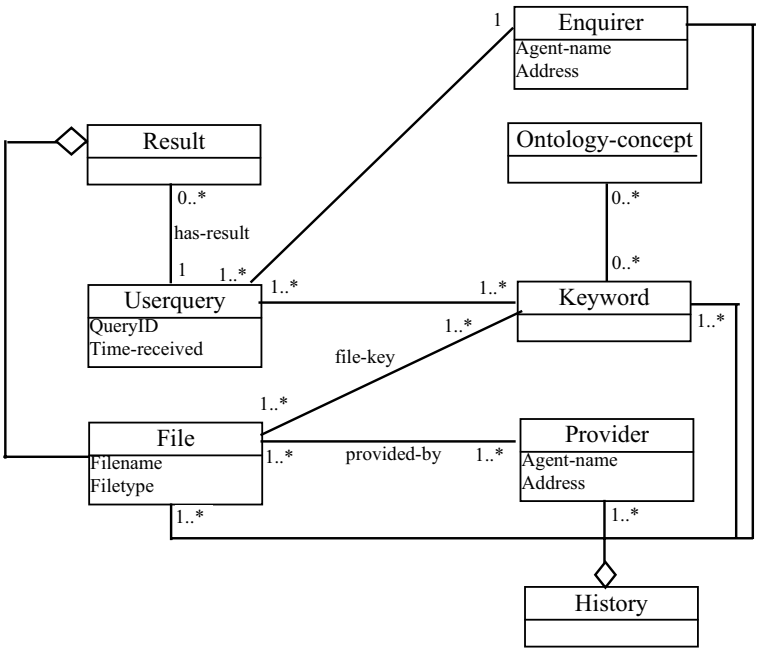


Figure AppendixH.3 – Ontology Diagram for *File Retrieval Ontology* by Developer 1

Role Model

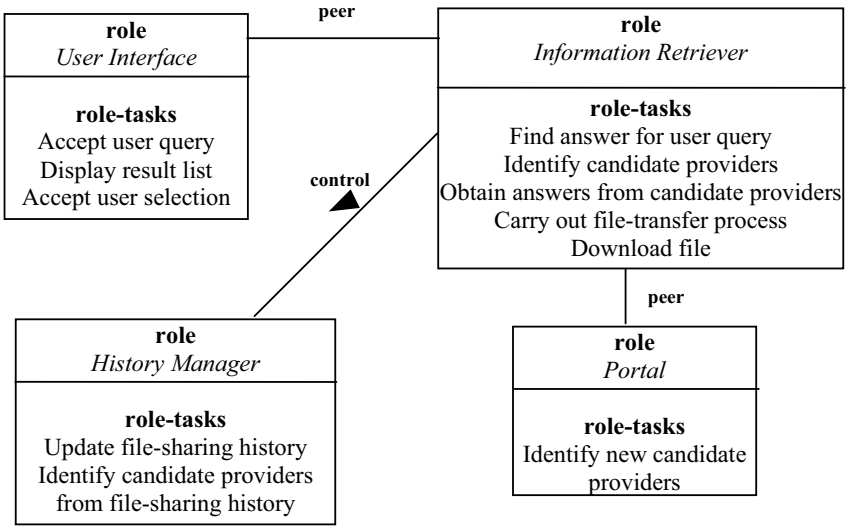


Figure AppendixH.4 – Role Diagram by Developer 1

Agent Class Model

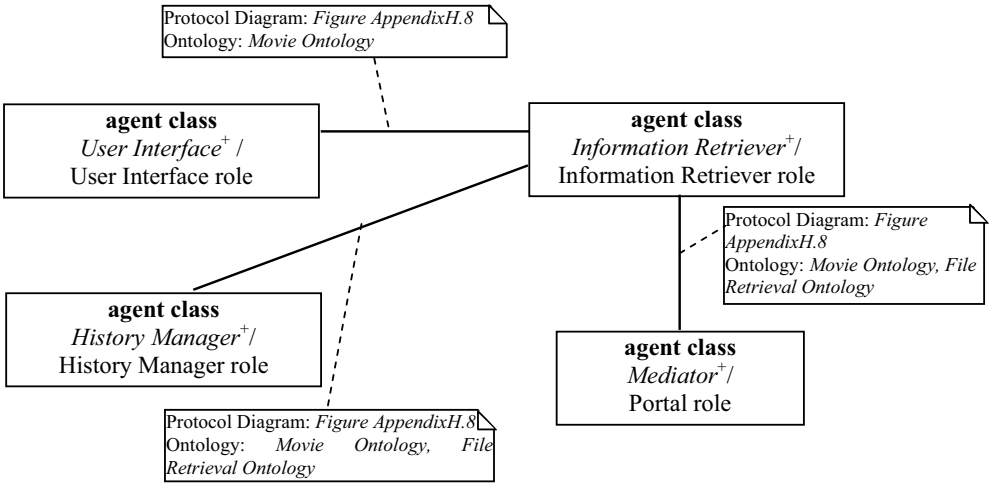


Figure AppendixH.5 – Agent Relationship Diagram by Developer 1

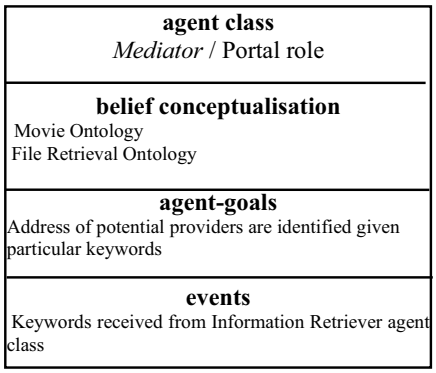


Figure AppendixH.6 – Agent Class Diagram by Developer 1 (for *Mediator* agent class)

Agent Behaviour Model

Initial state: *kw*: Keyword is received from Information Retriever agent
Target agent-goal: *filepointer*: File and *p*:*Provider* are identified and sent to Information Retriever agent
Commitment strategy: single-minded
List of sub-agent-goals: OntologyConcept_Identified, FileLocated, HistoryUpdated
List of actions:
Action 1: MatchKeyword (*kw:* Keyword, *h:* History)
Pre-condition: *kw* is received from Information Retriever agent
Post-condition: *oc*: Ontology-concept is identified

Action 2: RetrieveFile (*oc*: Ontology-concept, *h:* History)
Pre-condition: *oc* is identified
Post-condition: *filepointer*:File and corresponding *p*:*Provider* are located

Action 3: UpdateHistory (*oc*: Ontology-concept, *agent_ID*:Enquirer.Agent-name)
Pre-condition: *oc* is identified
Post-condition: History is updated with *agent_ID*

Events: message(*kw*: Keyword) arrives from Information Retriever agent

Figure AppendixH.7 – Agent Plan Template Diagram by Developer 1 (for *History Manager* agent class)

Agent Interaction Model

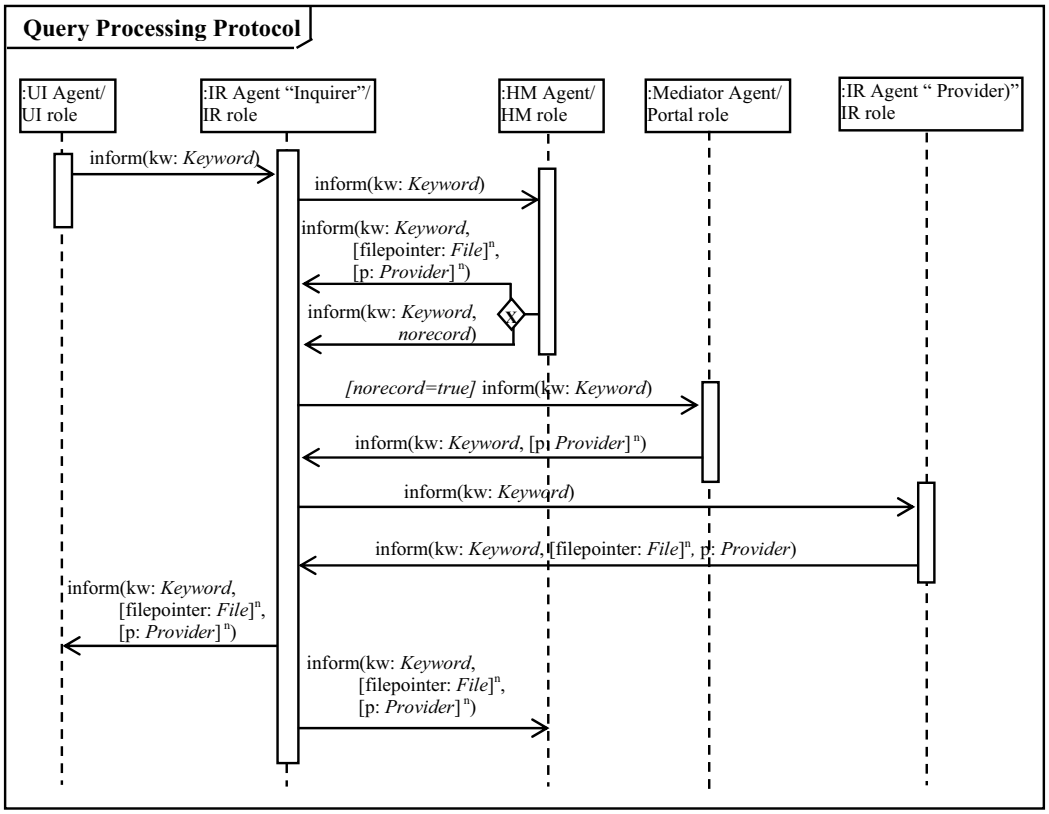


Figure AppendixH.8 – Interaction Protocol Diagram by Developer 1

Major models produced by Developer 2

DEVELOPER: DR. CESAR GONZALEZ-PEREZ

System Task Model

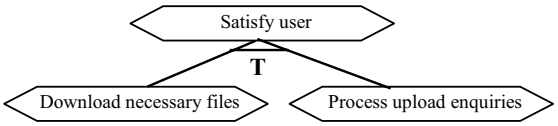


Figure AppendixH.9 – System Task Diagram 1 by Developer 2

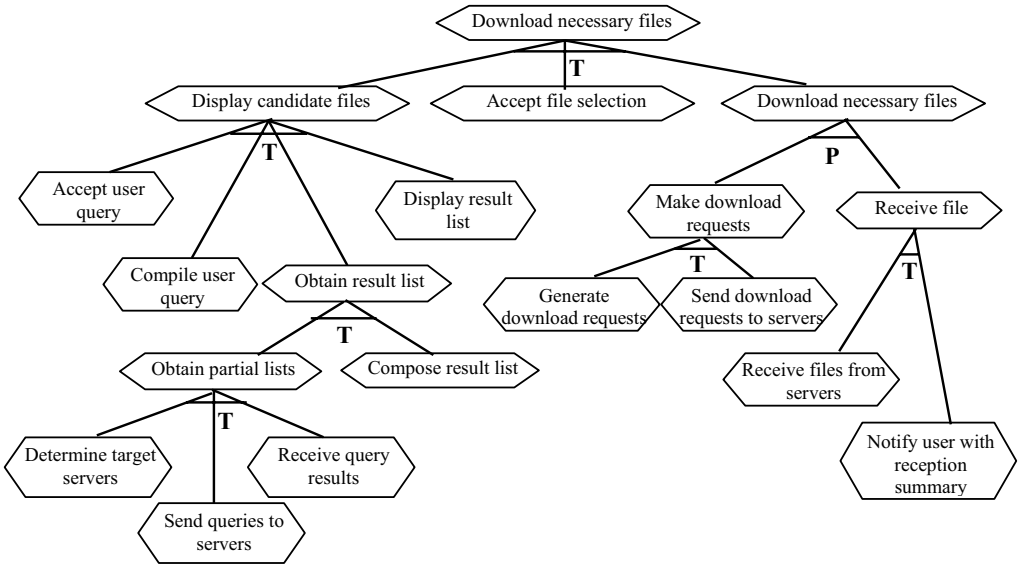


Figure AppendixH.10 – System Task Diagram 2 by Developer 2

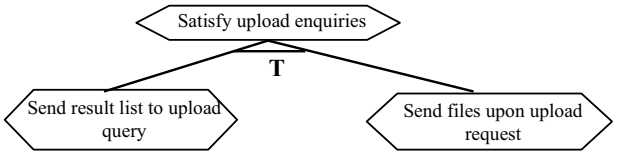


Figure AppendixH.11 – System Task Diagram 3 by Developer 2

Ontology Model

Note that Ontology Diagram for *Movie Ontology* is reused from that developed by Developer 1 (Figure AppendixH.2).

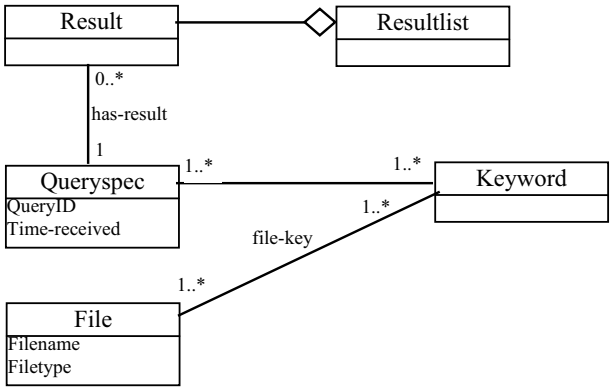


Figure AppendixH.12 – Ontology Diagram for *File Sharing Ontology* by Developer 2

Role Model

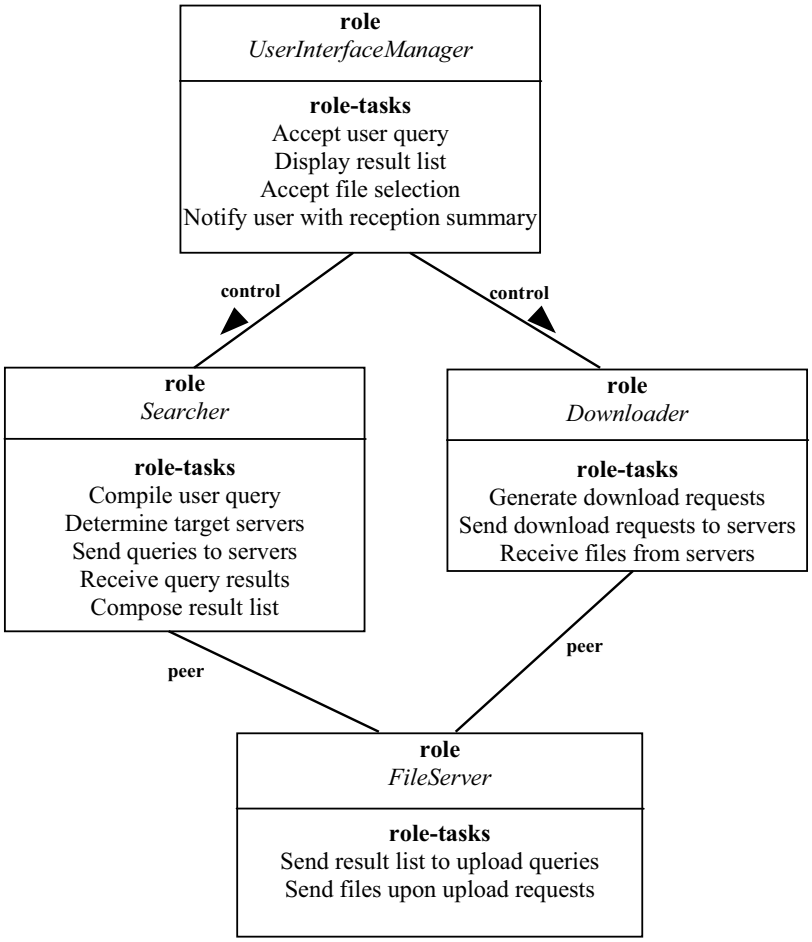


Figure AppendixH.13 – Role Diagram by Developer 2

Agent Class Model

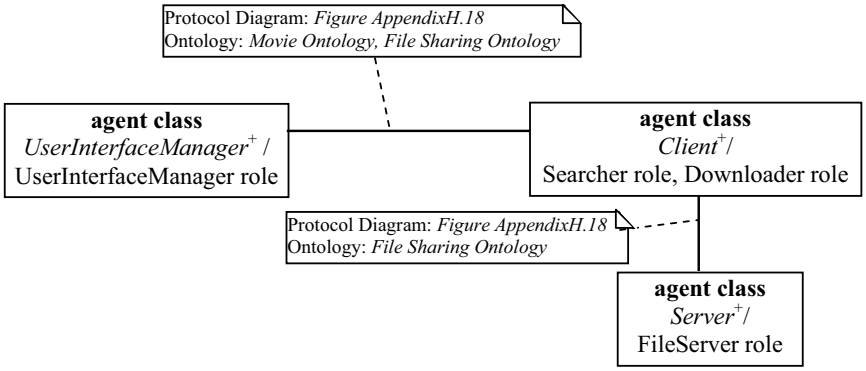


Figure AppendixH.14 – Agent Relationship Diagram by Developer 2

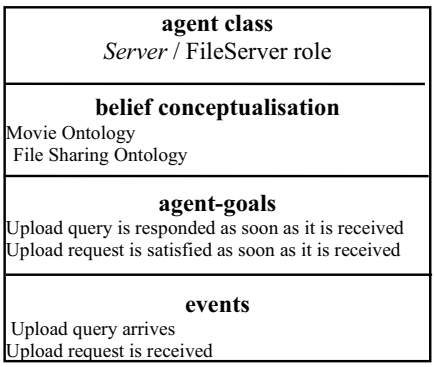


Figure AppendixH.15 – Agent Class Diagram by Developer 2 (for *Server* agent class)

Agent Behaviour Model

Initial State: any
Agent Goal: Upload query is responded as soon as it is received
Commitment Strategy: single-minded
Action 1: ValidateQuerySyntax(*q*: QuerySpec)
Pre-condition: true
Post-condition: Query *q* is valid OR refusal message has been replied
Action 2: ExecuteQuery(*q*: QuerySpec)
Pre-condition: *q* is valid
Post-condition: Query *q* has been executed and result list is known
Action 3: ReplyToQuery(*rl*: ResultList)
Pre-condition: true
Post-condition: Result list *rl* has been sent back to remote server *s*
Event 1: Query *q* arrives from remote server *s*

Figure AppendixH.16 – Agent Plan Template by Developer 2 (for *Server* agent class)

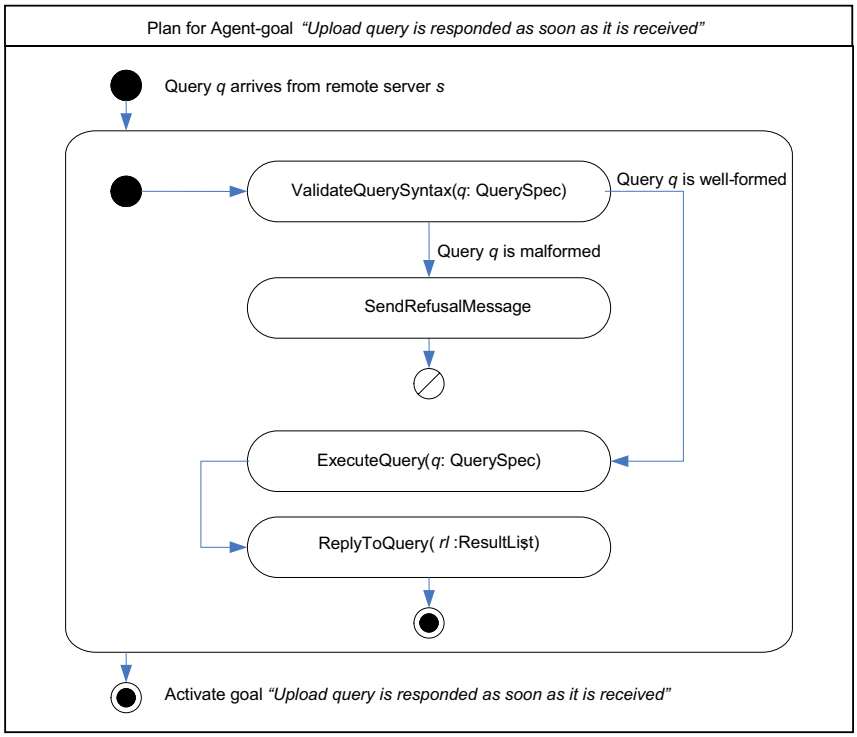


Figure AppendixH.17 – Agent Plan Diagram by Developer 2 (for *Server* agent class)

Agent Interaction Model

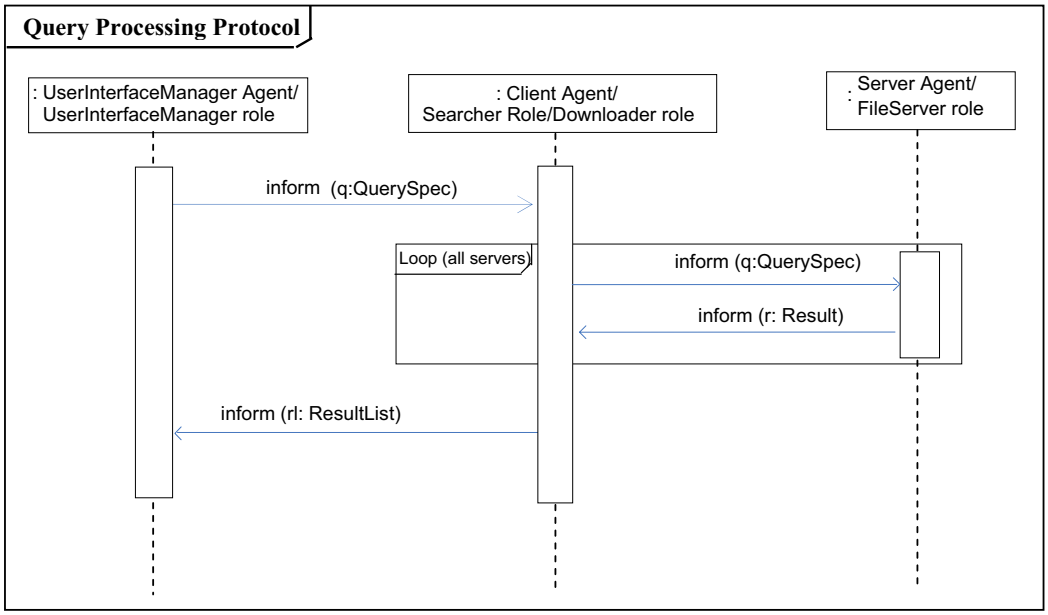


Figure AppendixH.18 – Interaction Protocol Diagram by Developer 2