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.

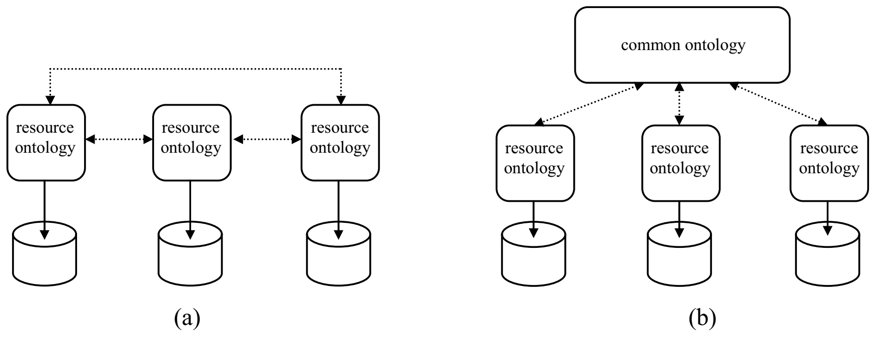


Figure 2.1 – Approaches for ontological mapping (Wache et al. 2001)

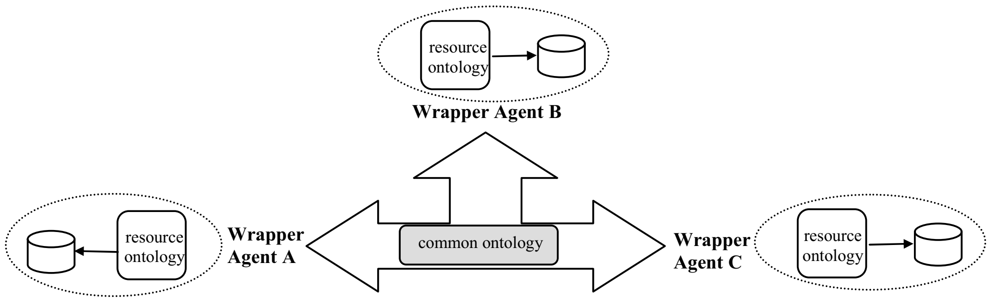


Figure 2.2 – Sharing of knowledge between wrapper agents

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).

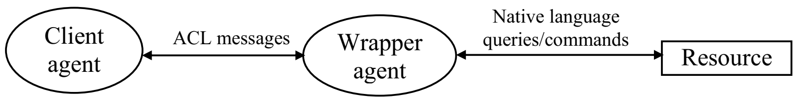


Figure 2.3 – Agent-resource communication

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).

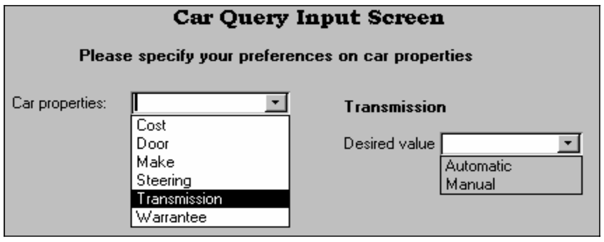


Figure 2.4 – User query formulation using concepts from ontology
Note: Concepts "Cost", "Door", "Make", "Steering", "Transmission" and "Warrantee" are defined in the Car Domain Ontology committed by the Car Seller agent. Concepts "Automatic" and "Manual" must have been defined as properties or sub-classes 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).

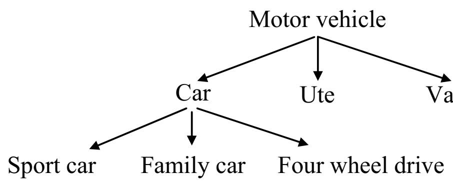


Figure 2.5 – Example fragment of Car Domain Ontology

– 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).

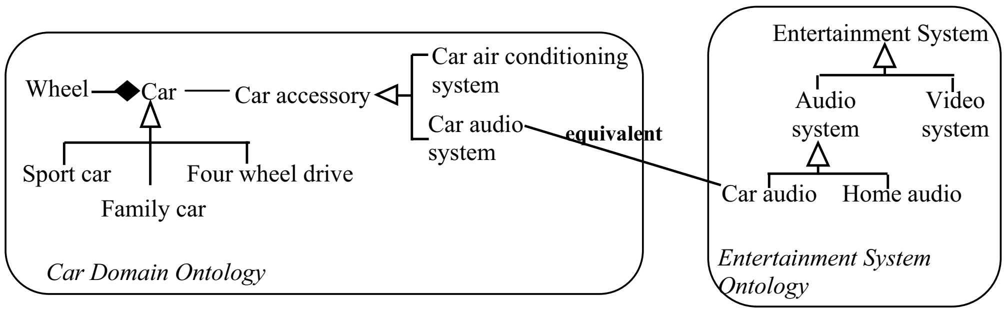


Figure 2.6 – Example ontological mappings between Car Domain Ontology and Entertainment System Ontology

– 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).

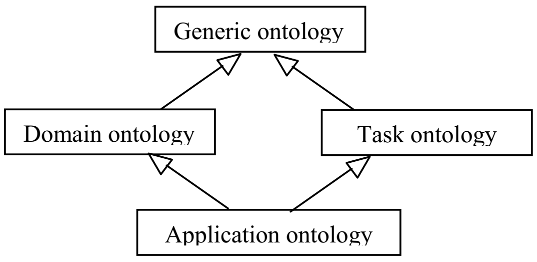


Figure 2.7 – Types of ontology (Guarino 1997)

- **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>
```

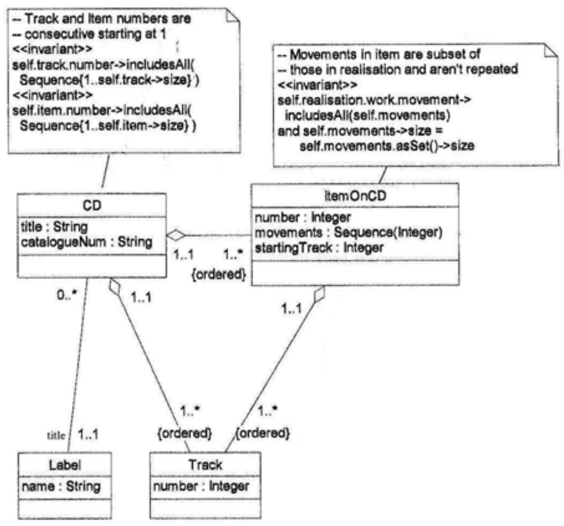
2.3.4.2. Graphical representation languages

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.



```
-- Track and Item numbers are
-- consecutive starting at 1
<<invariant>>
self.track.number->includesAll(
  Sequence{1..self.track->size} )
<<invariant>>
self.item.number->includesAll(
  Sequence{1..self.item->size} )
```

```
-- Movements in item are subset of
-- those in realisation and aren't repeated
<<invariant>>
self.realisation.work.movement->
  includesAll(self.movements)
and self.movements->size =
     self.movements.asSet()->size
```

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.

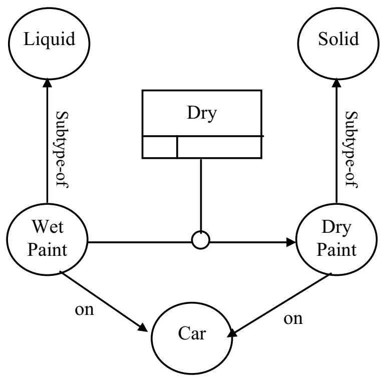


Figure 2.9 – Example of ontology representation in IDEF5 Schematic Language (Knowledge Based Systems Inc 1994)

- **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.

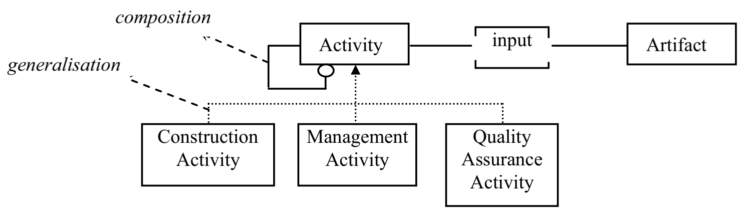


Figure 2.10 – Example of ontology representation in LINGO (Falbo et al. 1998)

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.

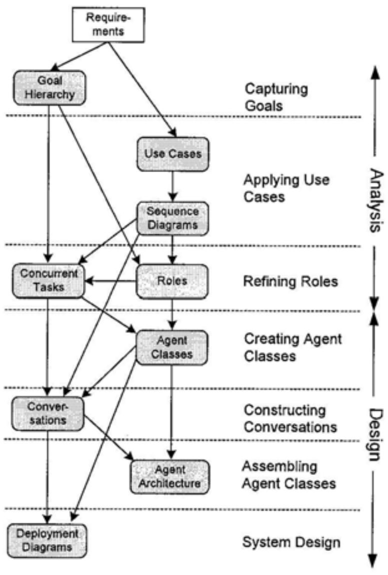


Figure 3.1 – Overview of MASE (Wood and DeLoach 2000a)

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.

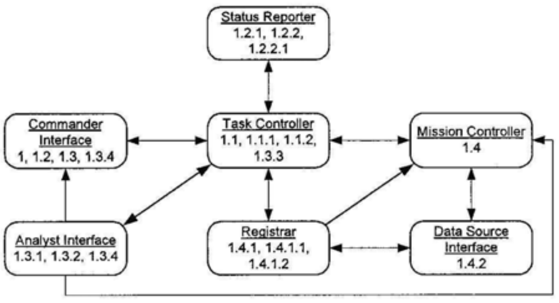


Figure 3.2 – MASE Role Model (Wood and DeLoach 2000a)

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).

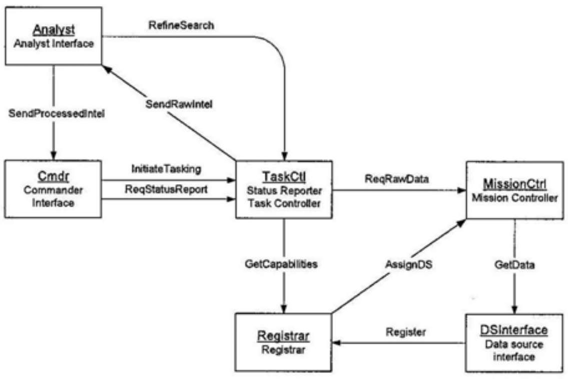


Figure 3.3 – MASE Agent Class Diagram (Wood and DeLoach 2000a)

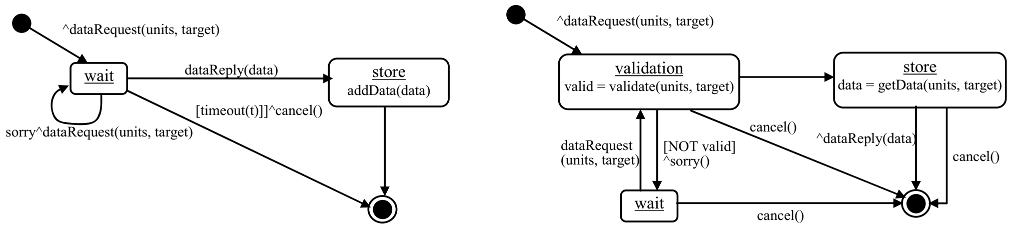


Figure 3.4 – MASE Communication Class Diagram for initiator (left) and responder (right) (Wood and DeLoach 2000a)

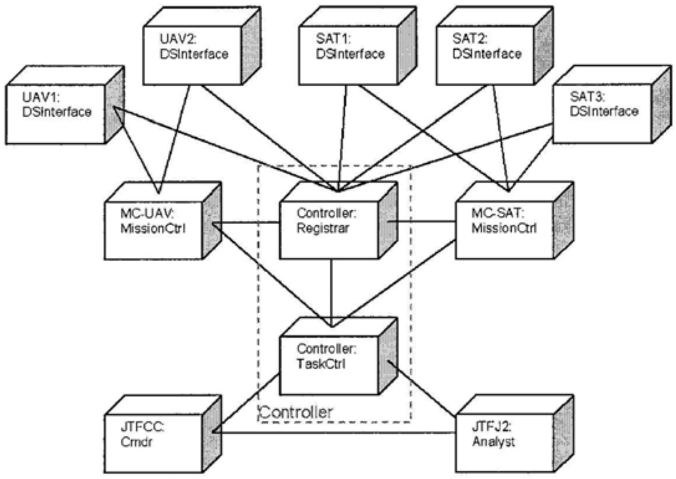


Figure 3.5 – MASE Deployment Diagram (Wood and DeLoach 2000a)

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.

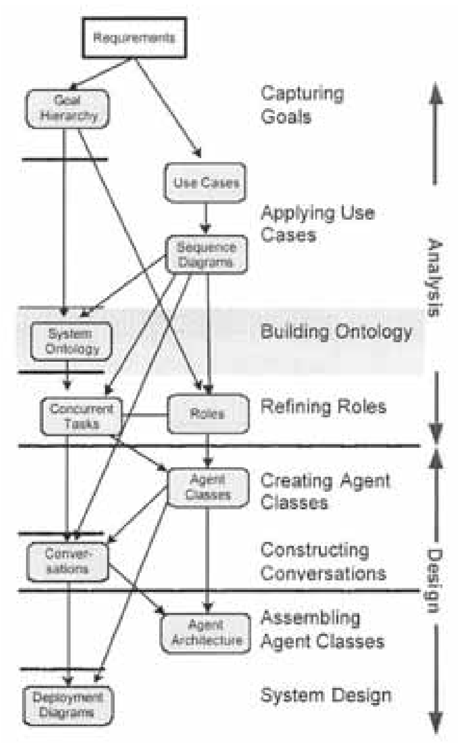


Figure 3.6 – Overview of extended version of MASE (DiLeo et al. 2002)

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.

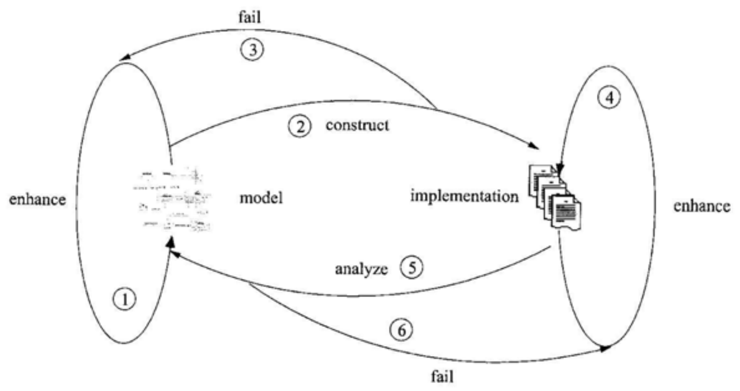


Figure 3.7 – MASSIVE Iterative View Engineering process (Lind 2000a)

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.

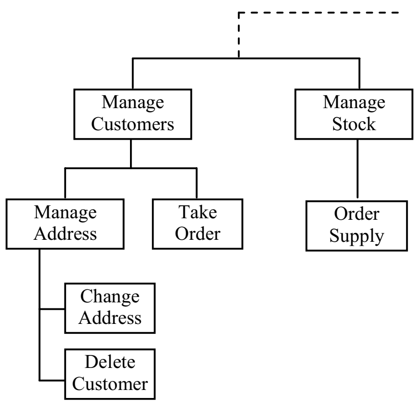


Figure 3.8 – MASSIVE Task View (Lind 1999)

- **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).

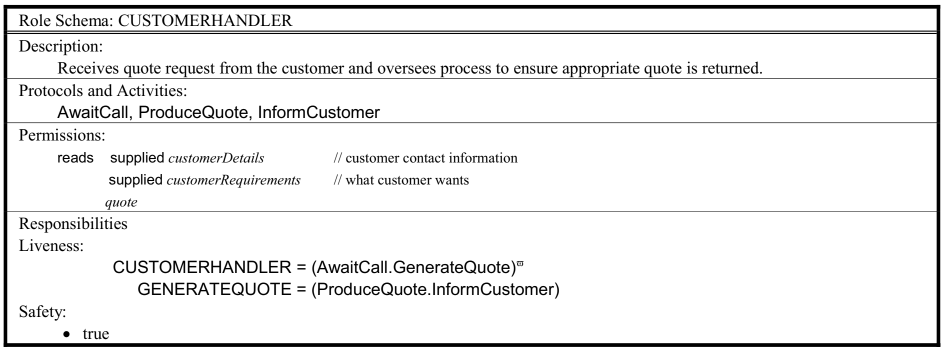
3.2.4. GAIA

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)$^{\varpi}$

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.

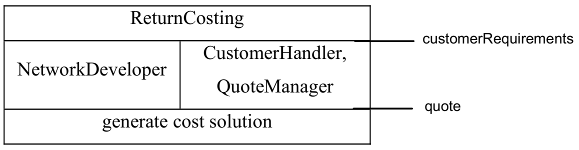


Figure 3.10 – GAIA Interaction Model (Wooldridge et al. 2000)
(Protocol *ReturnCosting* is initiated by role *NetworkDeveloper* and involves roles *CustomerHandler* and *QuoteManager*. The protocol takes as input *customerRequirements* and produces *quote*.)

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).

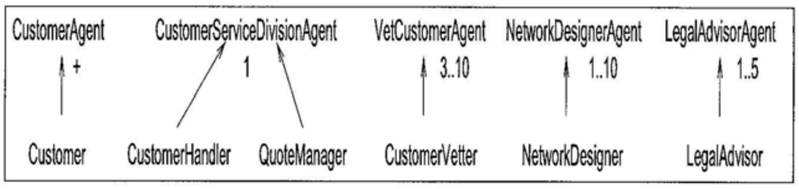


Figure 3.11 – GAIA Agent Model (Wooldridge et al. 2000)

- *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.

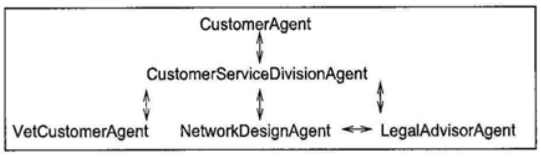


Figure 3.12 – GAIA Acquaintance Model (Wooldridge et al. 2000)

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.

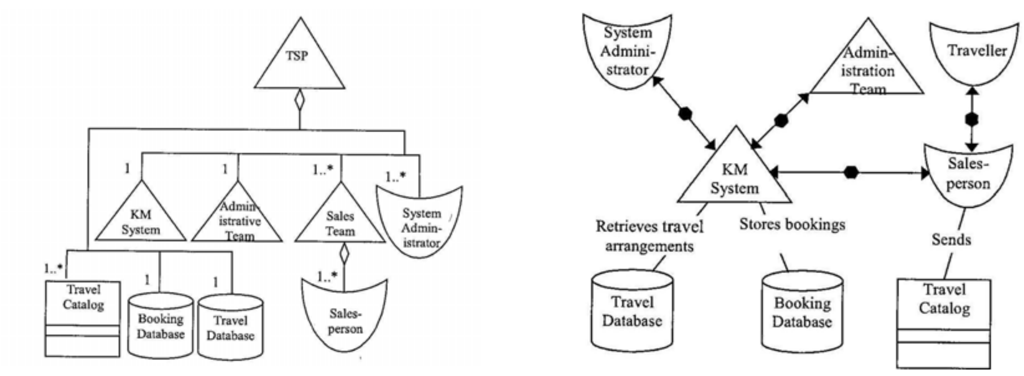


Figure 3.13 – MESSAGE Organisation Model – Structural Relationships (left) and Acquaintance Relationships (right) (Eurescom 2001b)

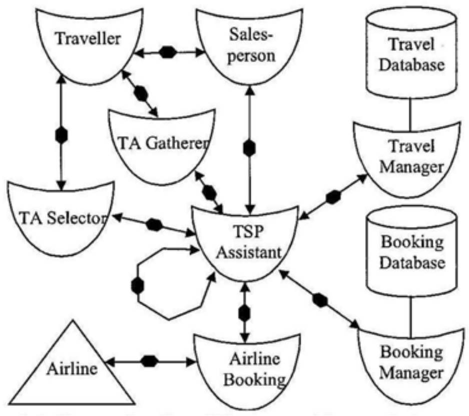


Figure 3.14 – MESSAGE Organisation Model - Agent/Role and Resources Acquaintance Relationships (Eurescom 2001b)

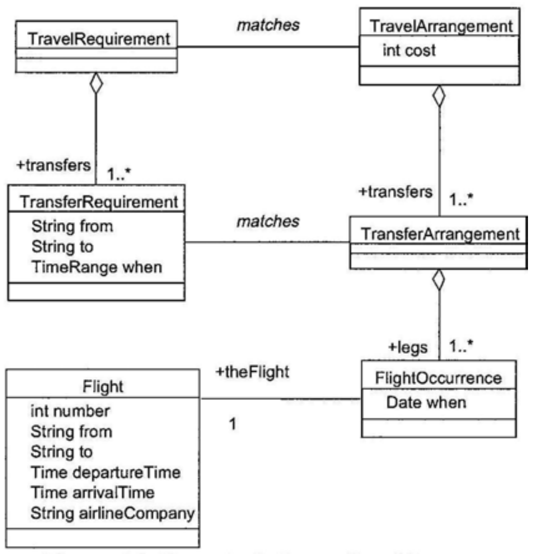


Figure 3.15 – MESSAGE Domain Model (Eurescom 2001b)

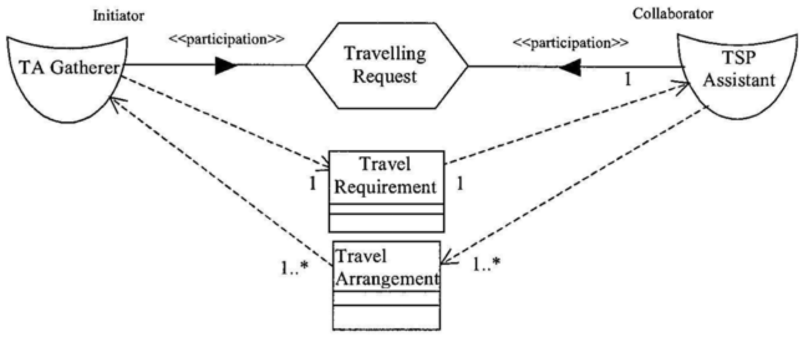


Figure 3.16 – MESSAGE Interaction Model (Eurescom 2001b)

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).

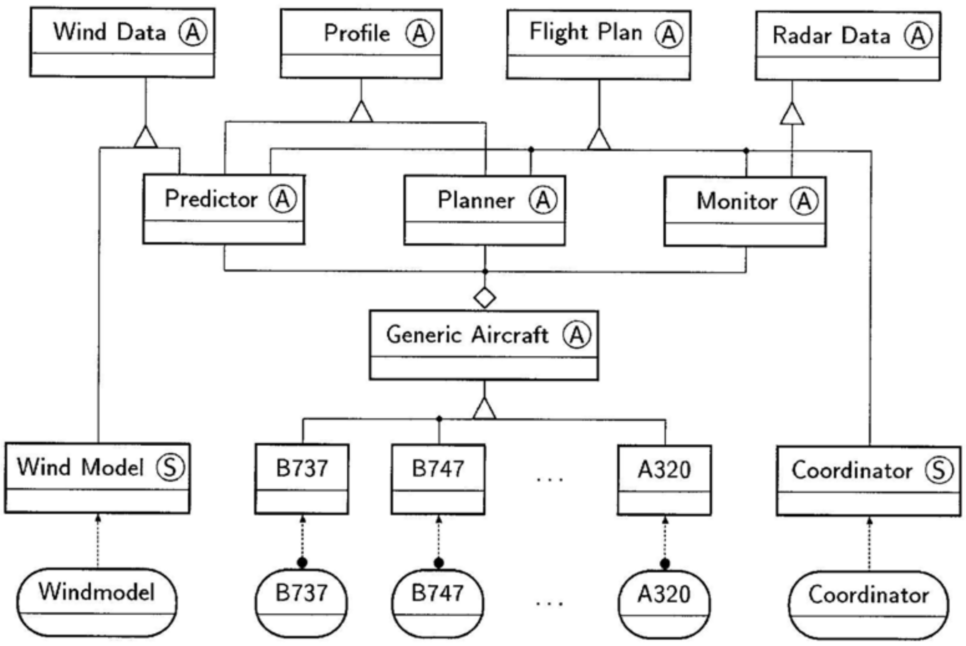


Figure 3.17 – BDIM Agent Model (Kinny et al. 1996)

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.

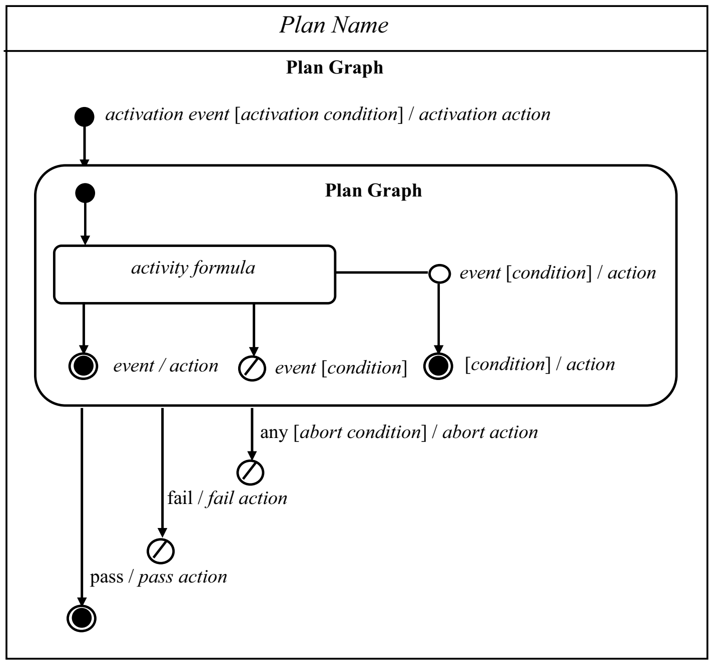
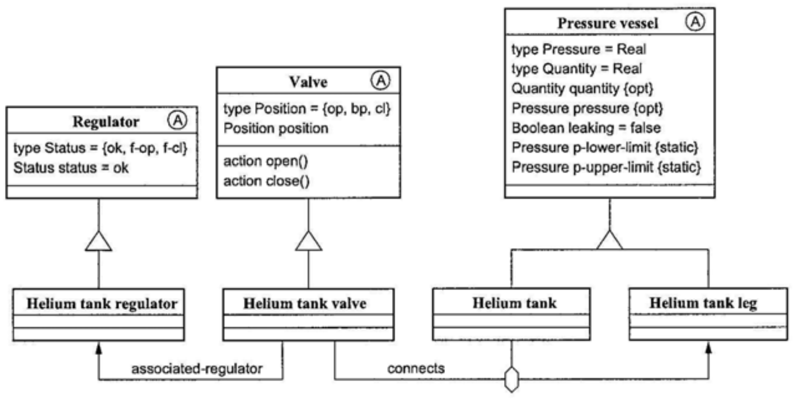


Figure 3.18 – BDIM Plan Diagram (Kinny et al. 1996)



Derived Predicates and Functions

status(Regulator, Status)	Status status(Regulator)
position(Valve, Position)	Position position(Valve)
quantity(Pressure-vessel, Quantity)	
pressure(Pressure-vessel, Pressure)	
leaking(Pressure-vessel, Boolean)	Boolean leaking(Pressure-vessel))
p-lower-limit(Pressure-vessel, Pressure)	Pressure p-lower-limit(Pressure-vessel)
p-upper-limit(Pressure-vessel, Pressure)	Pressure p-upper-limit(Pressure-vessel)
associated-regulator(Helium-tank-valve, Helium-tank-regulator)	
connects(Helium-tank-valve, Helium-tank, Helium-tank-leg)	

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.

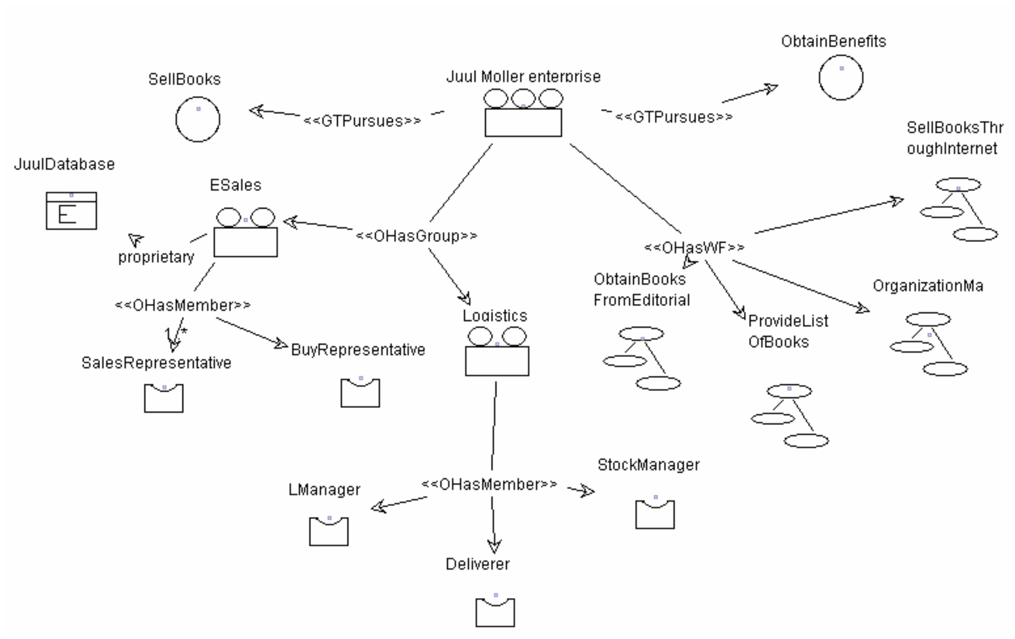


Figure 3.21 – INGENIAS Organisation Model (Pavon et al. 2005)

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.

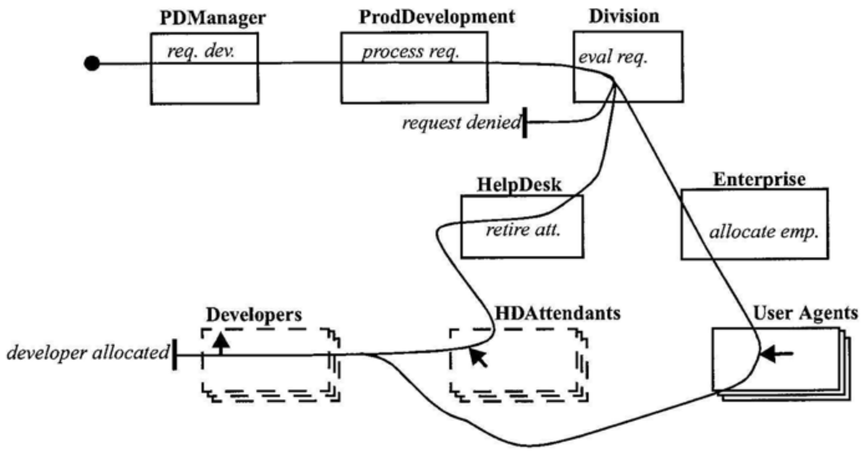
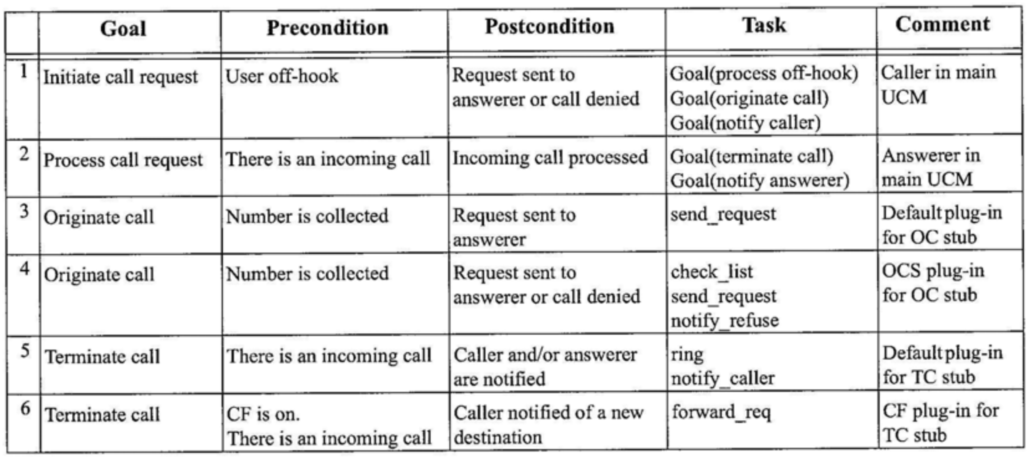


Figure 3.22 – HLIM Use Case Map (Elammari and Lalonde 1999)

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).

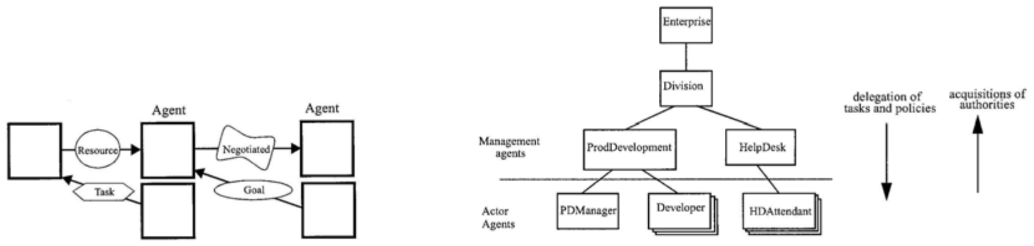
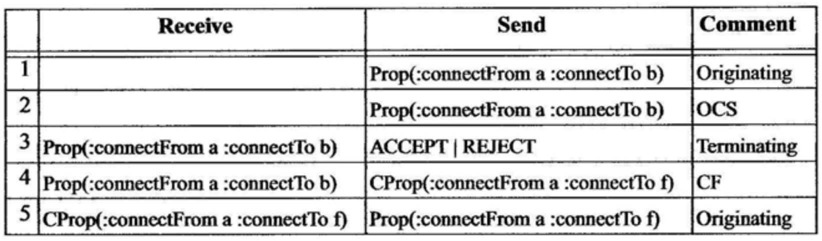


Figure 3.24 – HLIM Dependency Diagram (left) and Jurisdictional Diagram (Elammari and Lalonde 1999)

- *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.

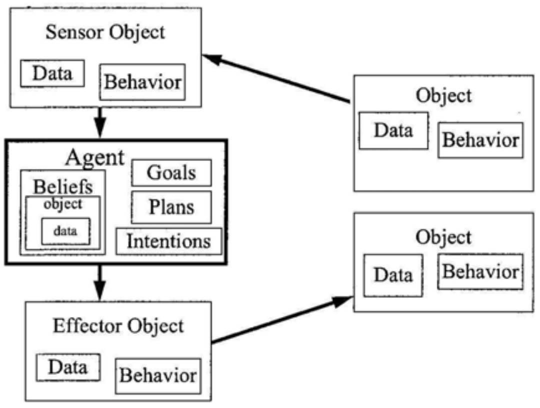


Figure 3.27 – MEI agent structure (Kendall et al. 1995)

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).

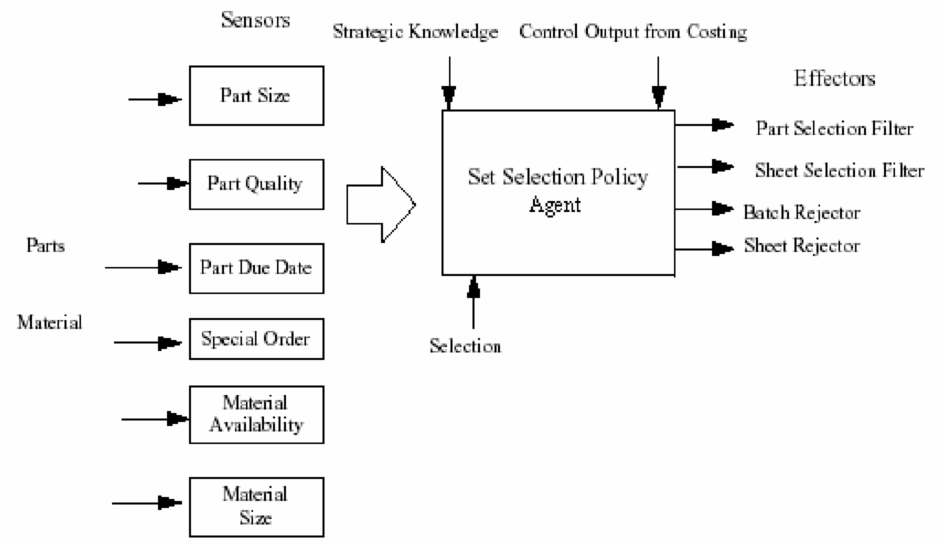


Figure 3.28 – MEI sensors and effectors specification (Kendall et al. 1995)

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).

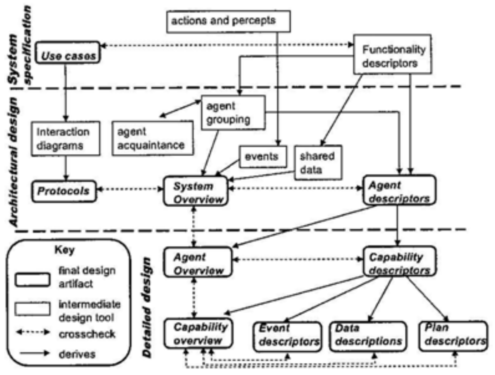


Figure 3.29 – Overview of PROMETHEUS (Padgham and Winikoff 2002a)

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).

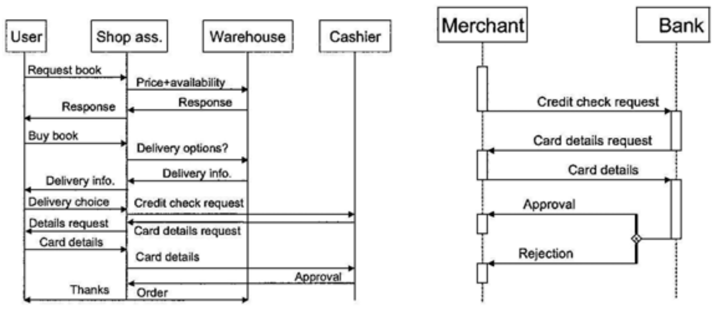


Figure 3.30 – PROMETHEUS Interaction Diagram (left) and Interaction Protocol (right) (Padgham and Winikoff 2002a)

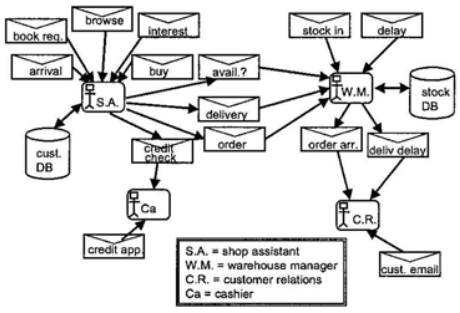


Figure 3.31 – PROMETHEUS System Overview Diagram (Padgham and Winikoff 2002a)

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.

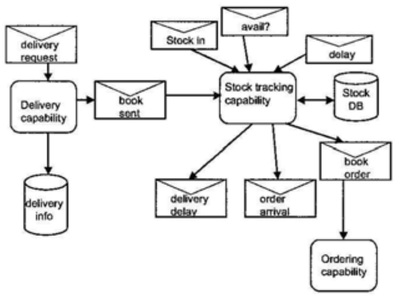


Figure 3.32 – PROMETHEUS Agent Overview Diagram (Padgham and Winikoff 2002a)

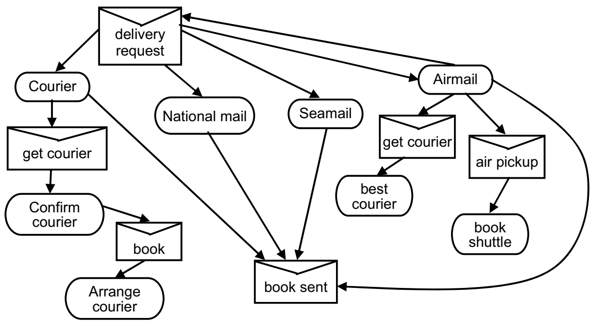


Figure 3.33 – PROMETHEUS Capability Diagram (Padgham and Winikoff 2002a)

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).

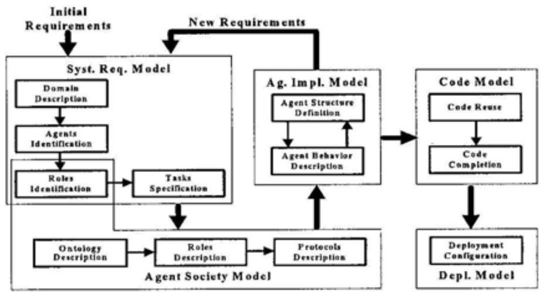


Figure 3.34 – Overview of PASSI (Burrafato and Cossentino 2002)

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.

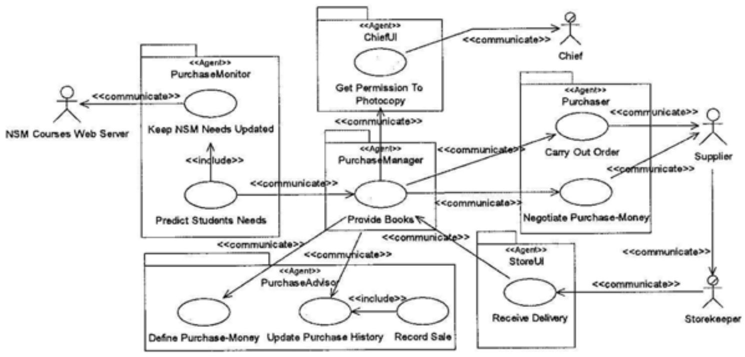


Figure 3.35 – PASSI Agent Identification Diagram (Burrafato and Cossentino 2002)

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.

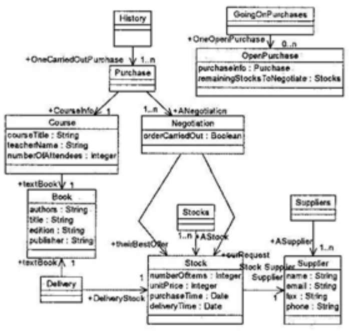


Figure 3.36 – PASSI Domain Ontology Diagram (Burrafato and Cossentino 2002)

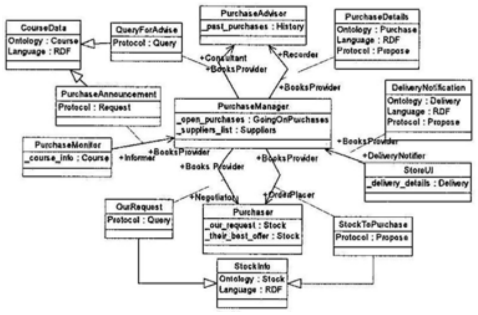


Figure 3.37 – PASSI Communication Ontology Diagram (Burrafato and Cossentino 2002)

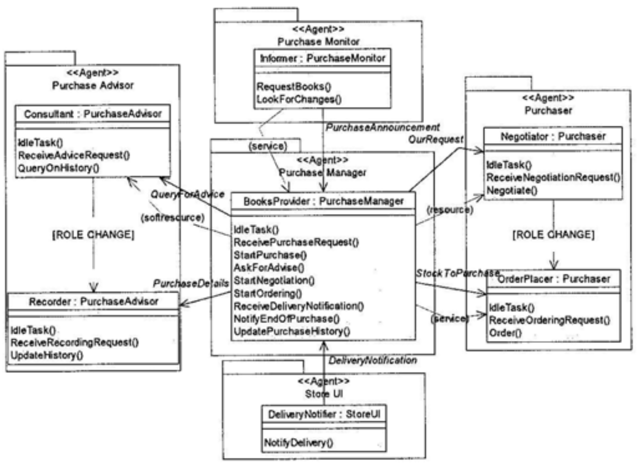


Figure 3.38 – PASSI Roles Description Diagram (Burrafato and Cossentino 2002)

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.

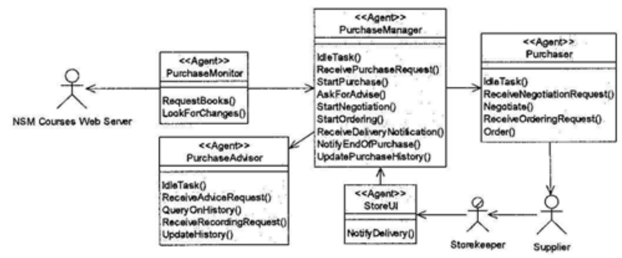


Figure 3.39 – PASSI MAS Structure Definition Diagram (Burrafato and Cossentino 2002)

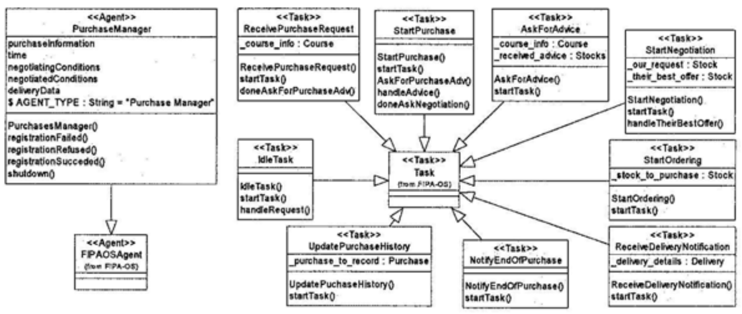


Figure 3.40 – PASSI Agent Structure Definition Diagram (Burrafato and Cossentino 2002)

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.

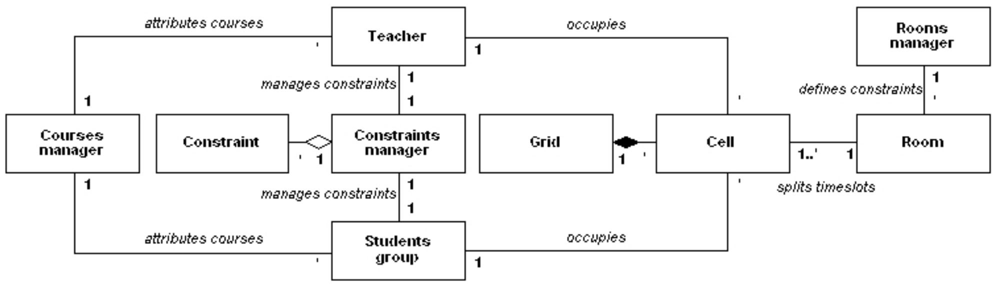


Figure 3.41 – ADELFE Preliminary Class Diagram (Institut de Recherche en Informatique de Toulouse n.d.)

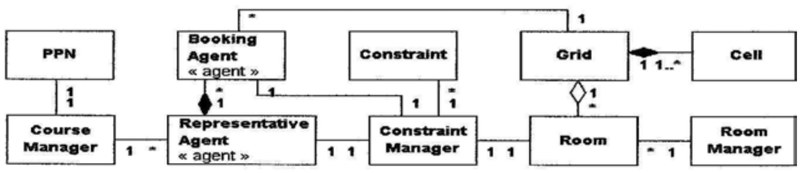
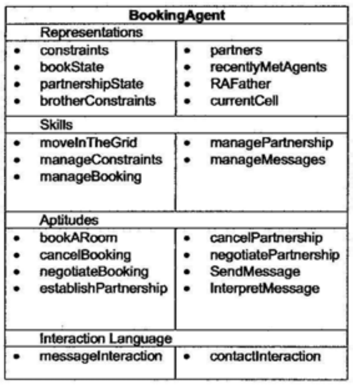


Figure 3.42 – ADELFE Refined Class Diagram (Institut de Recherche en Informatique de Toulouse n.d.)

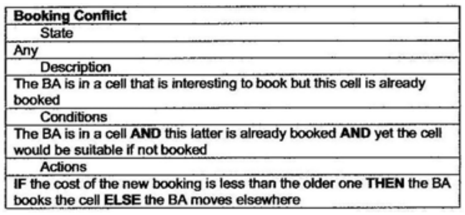
The **Design Phase** of ADELFE deals with the detailed design of system architecture, agent internal structure and non-cooperative situations. Overall architecture of the system is modelled in terms of packages, classes (of agents and objects) and relationships between them. The architecture of each 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).

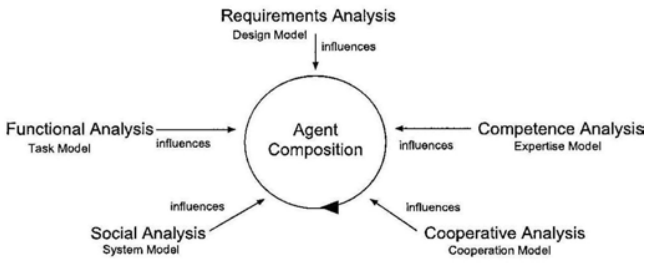
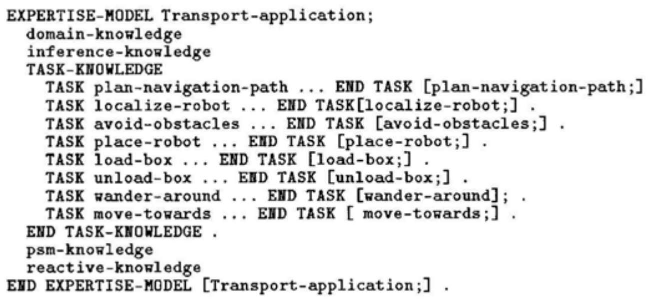


Figure 3.45 – COMOMAS steps and models (Glaser 1997a)

- "*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.

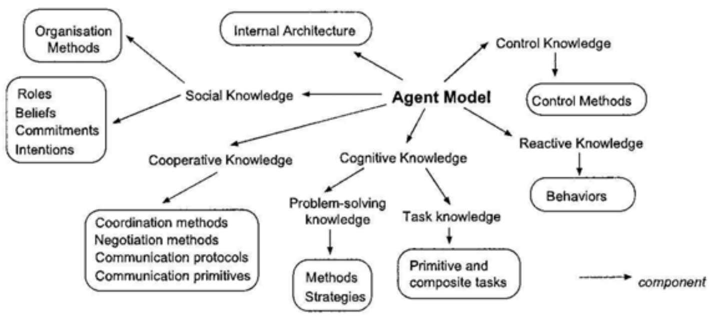


Figure 3.47 – COMOMAS Agent Model (Glaser 1997a)

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).

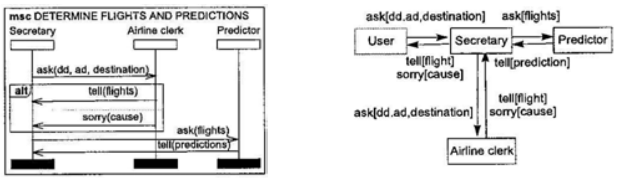


Figure 3.48 – MAS-CommonKADS Message Sequence Chart (left) and Event Flow Diagram (right) (Iglesias et al. 1998)

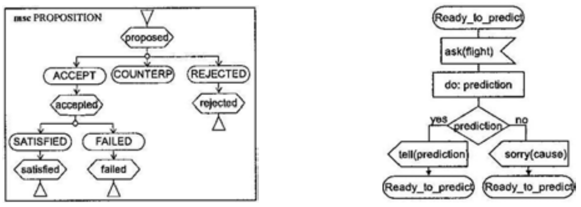


Figure 3.49 – MAS-CommonKADS High Level Message Sequence Chart (left) and State Transition Diagram (right) (Iglesias et al. 1998)

- *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.

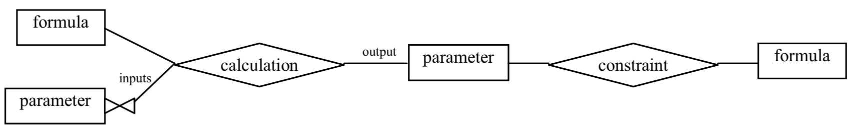


Figure 3.50 – MAS-CommonKADS Domain Knowledge Ontology (Schreiber et al. 1994)

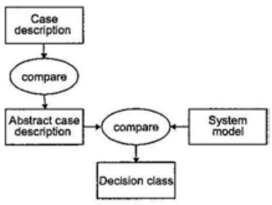


Figure 3.51 – MAS-CommonKADS Inferences Diagram (Iglesias et al. 1998)

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

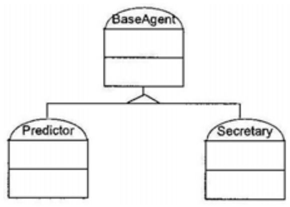


Figure 3.52 – MAS-CommonKADS Organisation Model (Iglesias et al. 1998)

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.

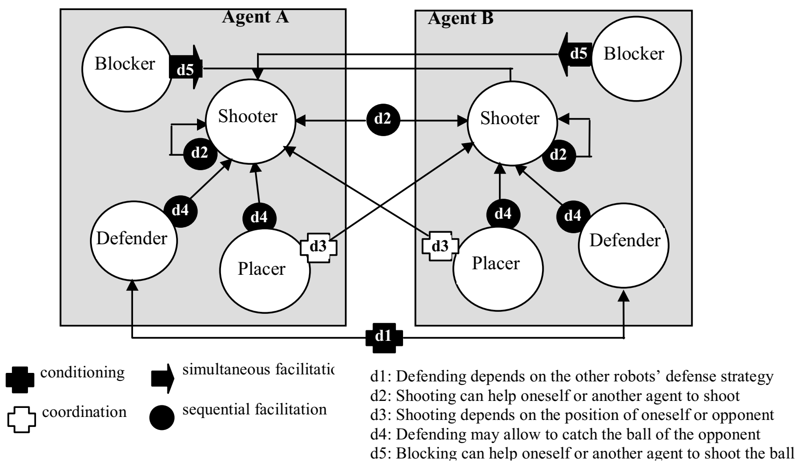

Agent A
Agent B
Blocker
d5
Shooter
d2
Defender
d4
Placer
d3
d1
conditioning
coordination
simultaneous facilitatic
sequential facilitation
d1: Defending depends on the other robots’ defense strategy
d2: Shooting can help oneself or another agent to shoot
d3: Shooting depends on the position of oneself or opponent
d4: Defending may allow to catch the ball of the opponent
d5: Blocking can help oneself or another agent to shoot the ball

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).

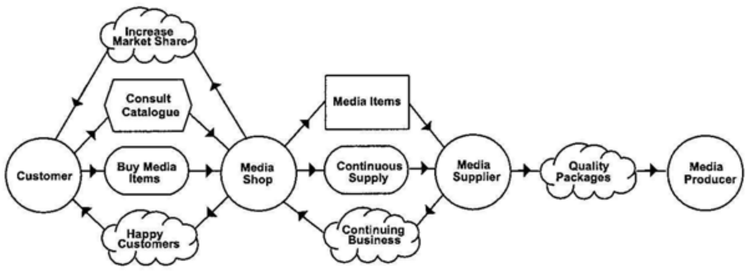


Figure 3.54 – TROPOS Strategic Dependency Model in Early Requirement phase (Castro et al. 2002)

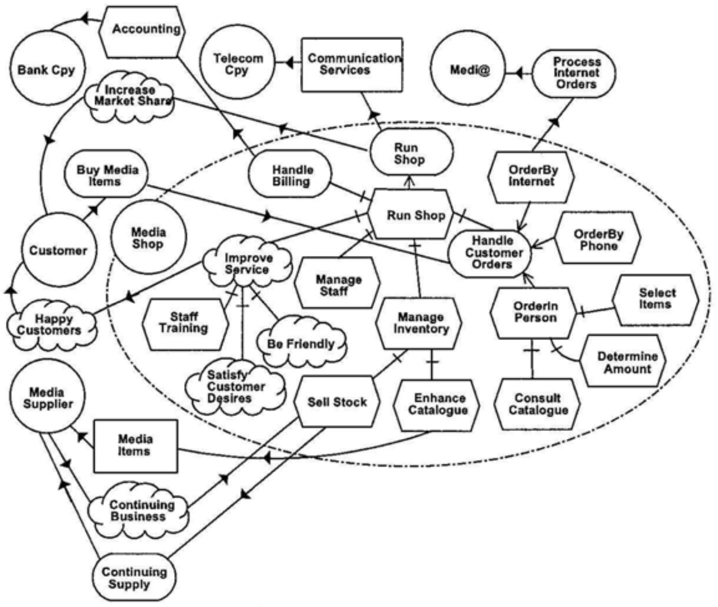


Figure 3.55 – TROPOS Strategic Rationale Model in Early Requirement phase (Castro et al. 2001)

- 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).

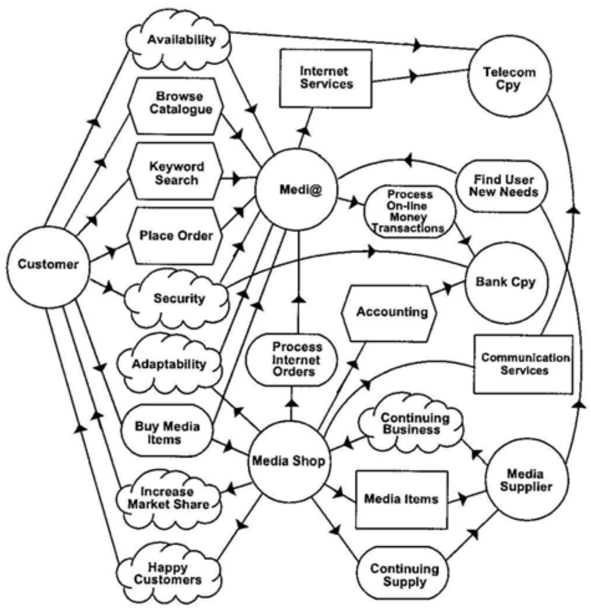


Figure 3.56 – TROPOS Strategic Dependency Model in Late Requirement phase (Castro et al. 2001)

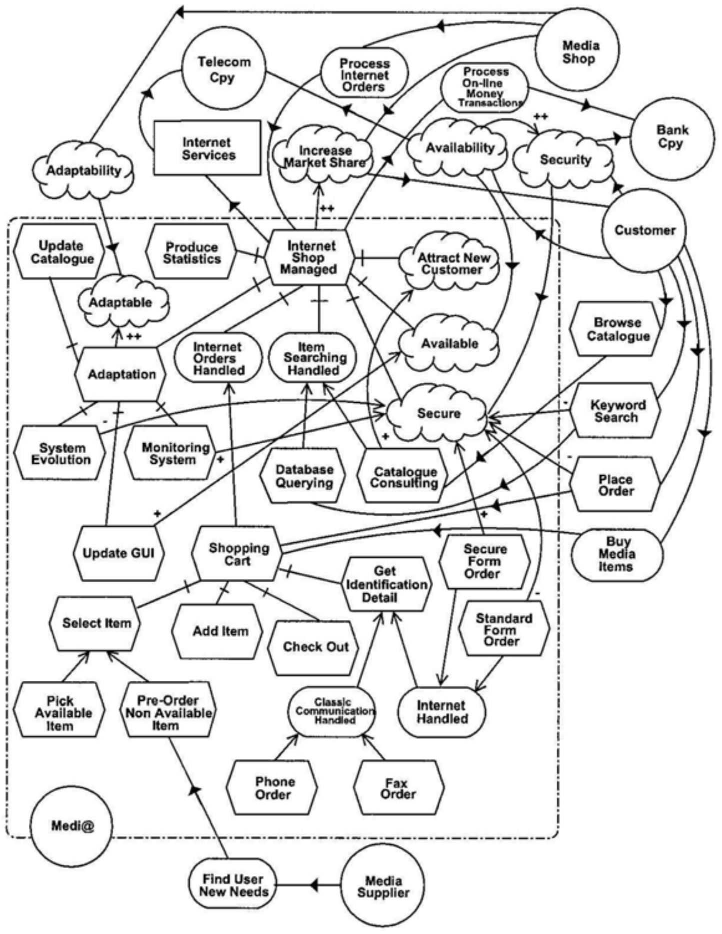


Figure 3.57 – TROPOS Strategic Rationale Model in Late Requirement phase (Castro et al. 2001)

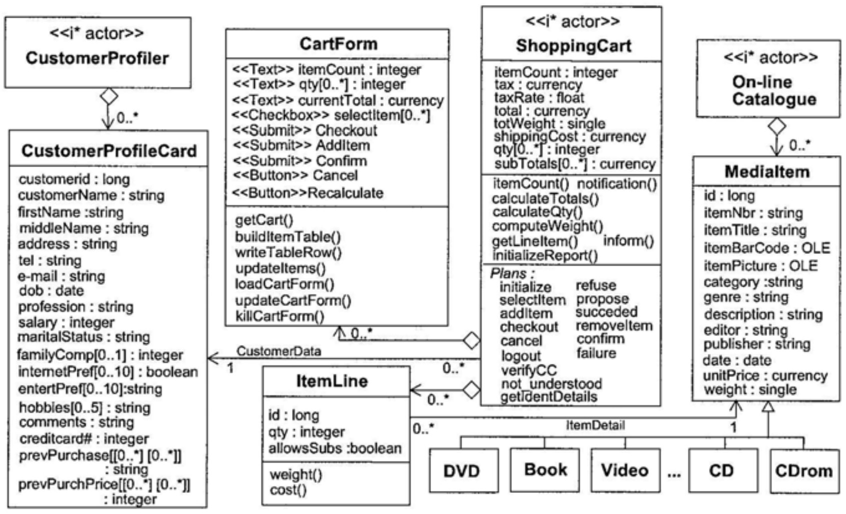
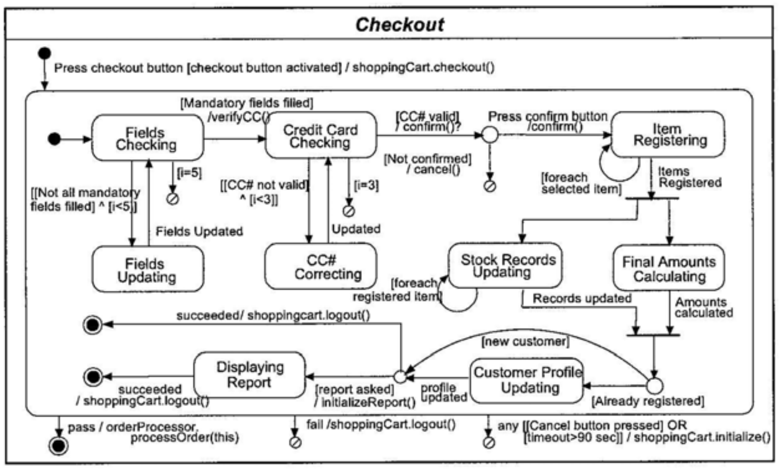


Figure 3.58 – TROPOS Agent Class Diagram (Castro et al. 2002)



Checkout
Press checkout button [checkout button activated] / shoppingCart.checkout()
[Mandatory fields filled] /verifyCC()
Fields Checking
Credit Card Checking
[CC# valid] / confirm()?
Press confirm button /confirm()
Item Registering
[Not confirmed] / cancel()
[foreach selected item]
Items Registered
[i=5]
[[Not all mandatory fields filled] ^ [i<5]]
Fields Updated
[[CC# not valid] ^ [i<3]]
[i=3]
Updated
Fields Updating
CC# Correcting
Stock Records Updating
Final Amounts Calculating
[foreach registered item]
Records updated
Amounts calculated
succeeded/ shoppingcart.logout()
[new customer]
Displaying Report
Customer Profile Updating
succeeded / shoppingCart.logout()
[report asked] / initializeReport()
profile updated
[Already registered]
pass / orderProcessor. processOrder(this)
fail /shoppingCart.logout()
any [[Cancel button pressed] OR [timeout>90 sec]] / shoppingCart.initialize()

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).

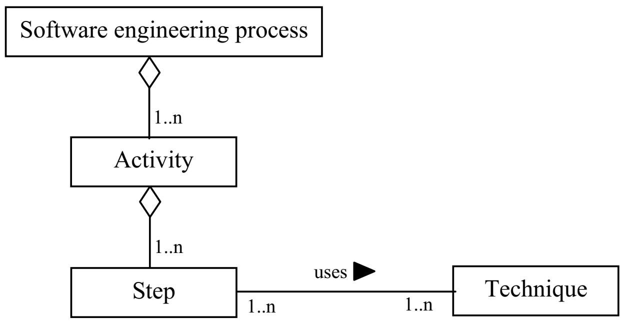


Figure 4.1 – Associations between "process", "activity", "step" and "technique" (represented in UML)

[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.

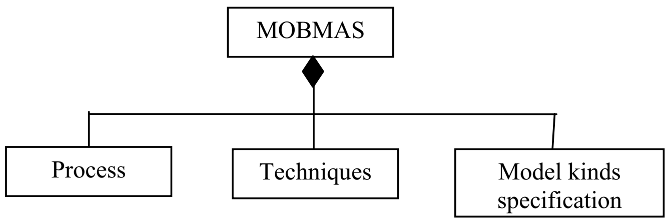


Figure 4.2 – Components of MOBMAS (represented in UML)

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).

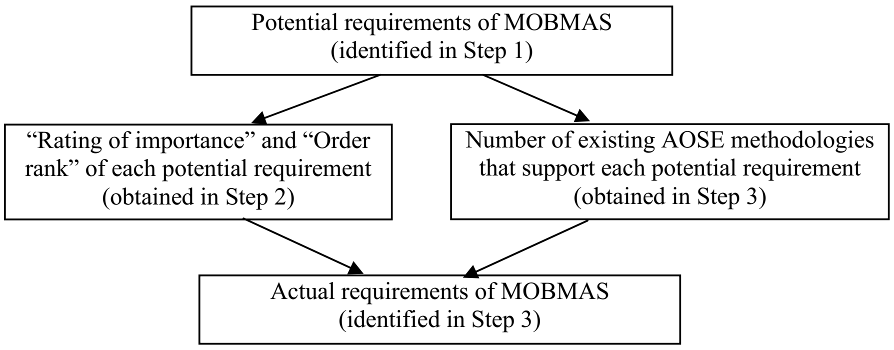

Potential requirements of MOBMAS (identified in Step 1)
"Rating of importance" and "Order rank" of each potential requirement (obtained in Step 2)
Number of existing AOSE methodologies that support each potential requirement (obtained in Step 3)
Actual requirements of MOBMAS (identified in Step 3)

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).

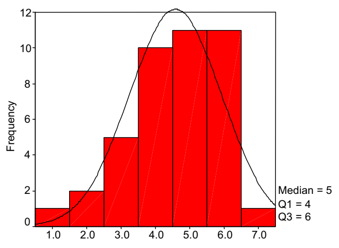
5.3.4.1. Part 1 – Demographic and professional characteristics of respondents

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

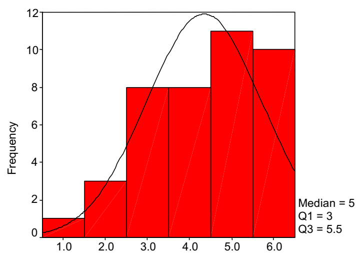
1. "*Field of work*"
2. "*Involvement in MAS development projects*" (Yes or No)
3. "*Size of past MAS projects*"
4. "*Level of complexity of past MAS projects*"
5. "*Application areas of past MAS projects*"
6. "*Adoption of AOSE methodologies in past MAS projects"*
7. "*Involvement in Ontology-Based 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.

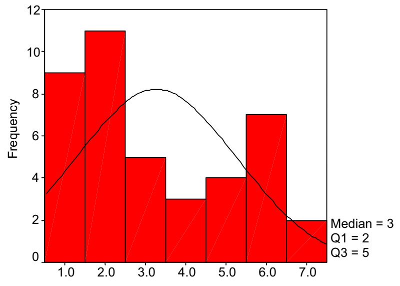
Response scores given to each expertise variable were obtained via a 7-point Likert scale, ranging from "1" (i.e. "Low") to "7" (i.e. "Extensive"). The distribution of the four variables showed that medians of both "*Theoretical knowledge of MAS*" and "*Theoretical knowledge of MAS development*" were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



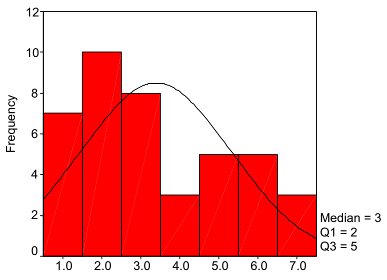
Theoretical knowledge of MAS



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.

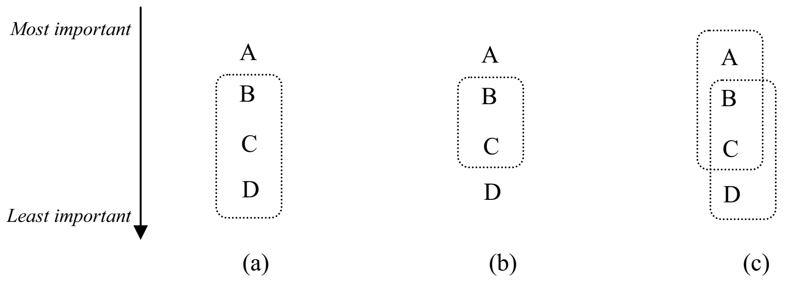


Figure 5.13 – Examples of ranking order results

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.

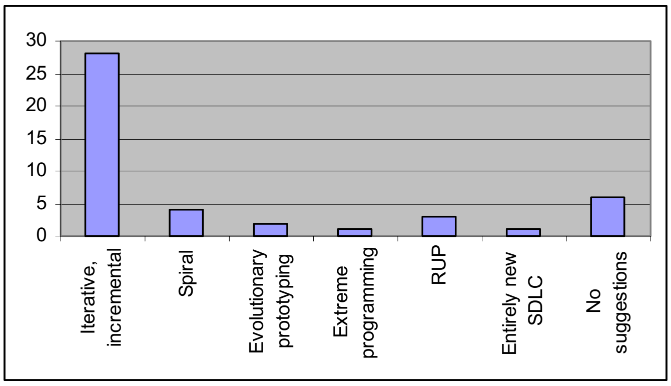


Figure 5.17 – Survey respondents' suggestions on MAS development SDLC

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).

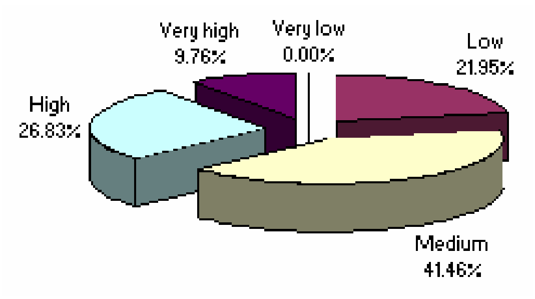


Figure 5.18 – Survey respondents' suggestions on the importance of a MAS development methodology to commit to an agent architecture

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).

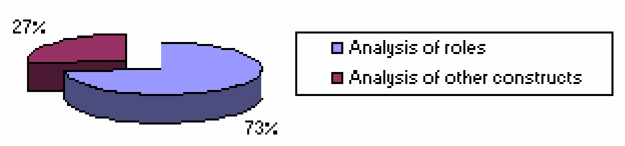


Figure 5.19 – Survey respondents' suggestions on the approaches to agent identification

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.

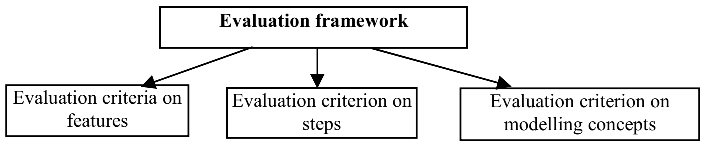

Evaluation framework
Evaluation criteria on features
Evaluation criterion on steps
Evaluation criterion on modelling concepts

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	MESS AGE	INGENIAS	BDIM	HLIM	MEI	PROME THEUS	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.