# CHAPTER 6
# DOCUMENTATION OF MOBMAS

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

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

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

## 6.1. OVERVIEW OF MOBMAS

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

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

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

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

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

## 6.1.1. MOBMAS Conceptual Framework

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

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

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

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

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

---

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

The relationships between MOBMAS abstractions are shown in Figure 6.1.

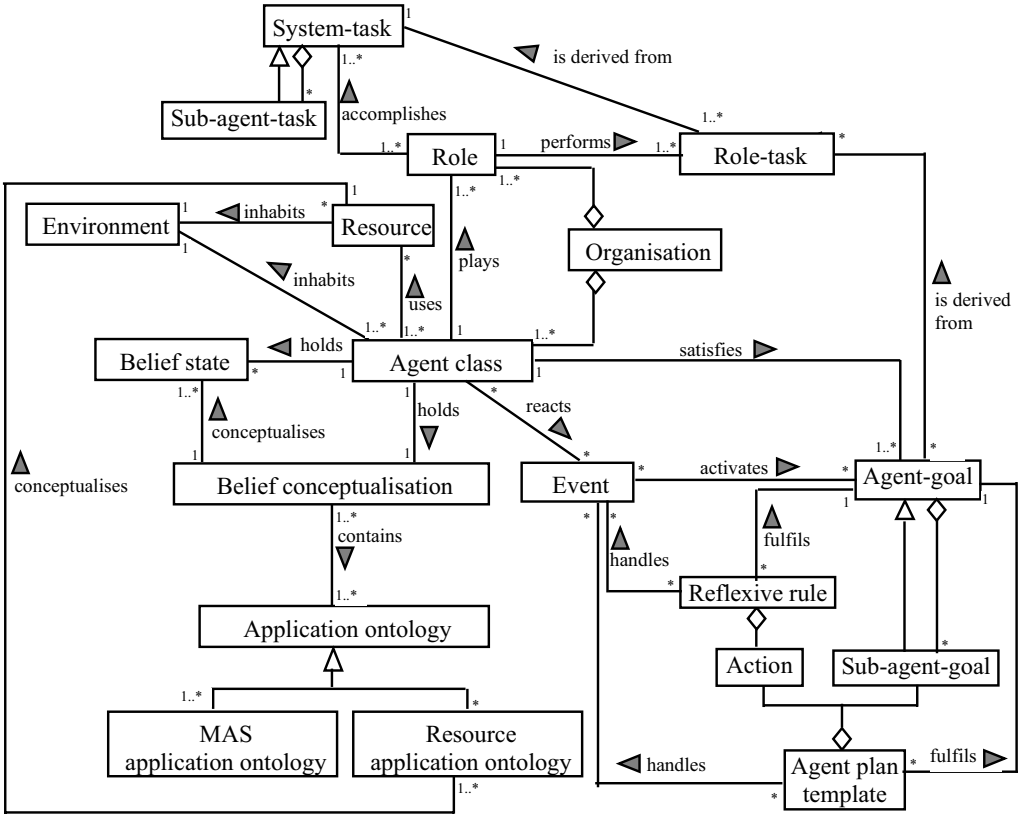



Figure 6.1 – MOBMAS abstractions and their relationships (represented in UML)

## 6.1.2. MOBMAS Development Process

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

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

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

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

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

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

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

---

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

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

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

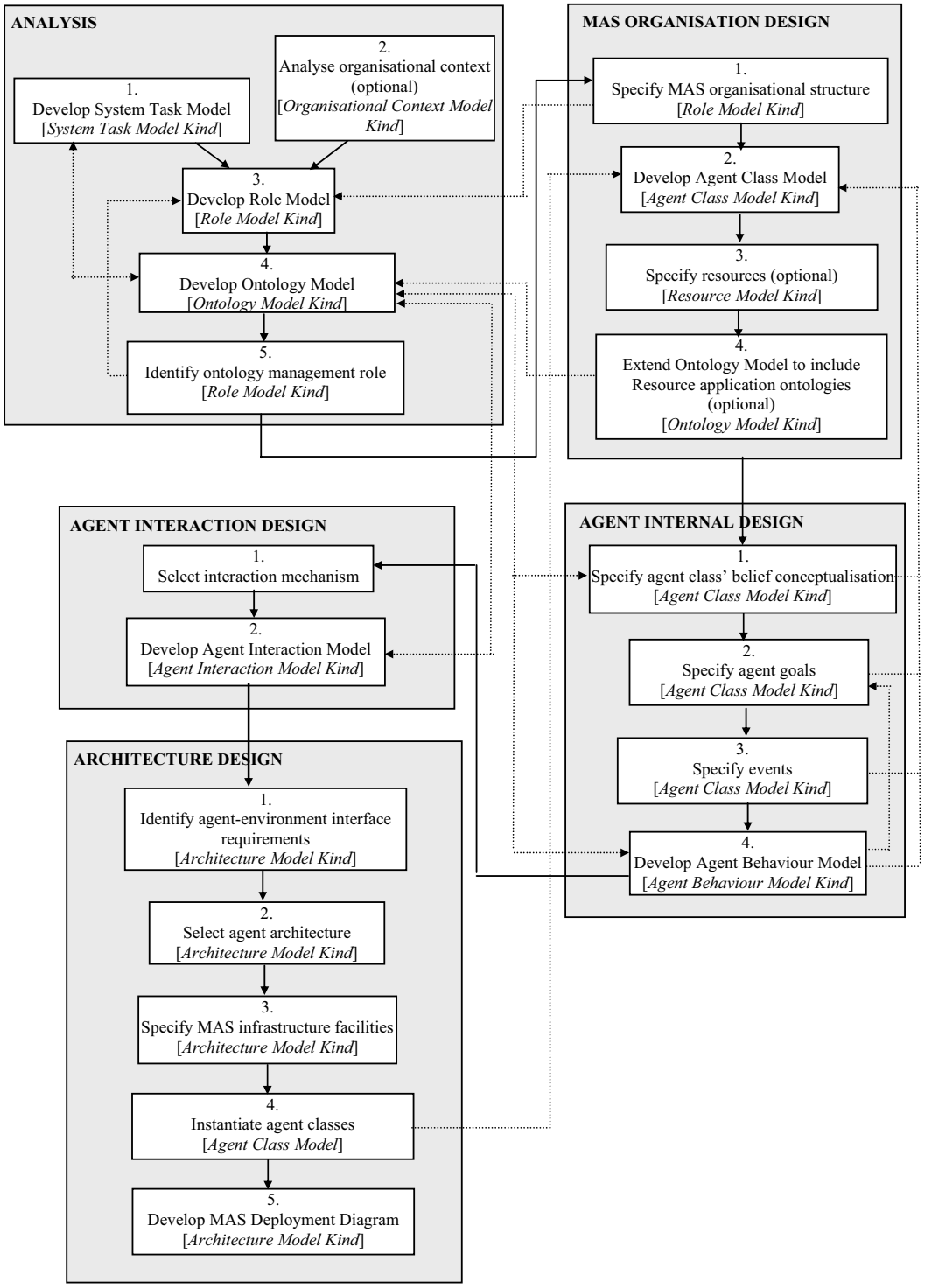



Figure 6.2 – MOBMAS development process

## 6.1.3. MOBMAS Model Kinds

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

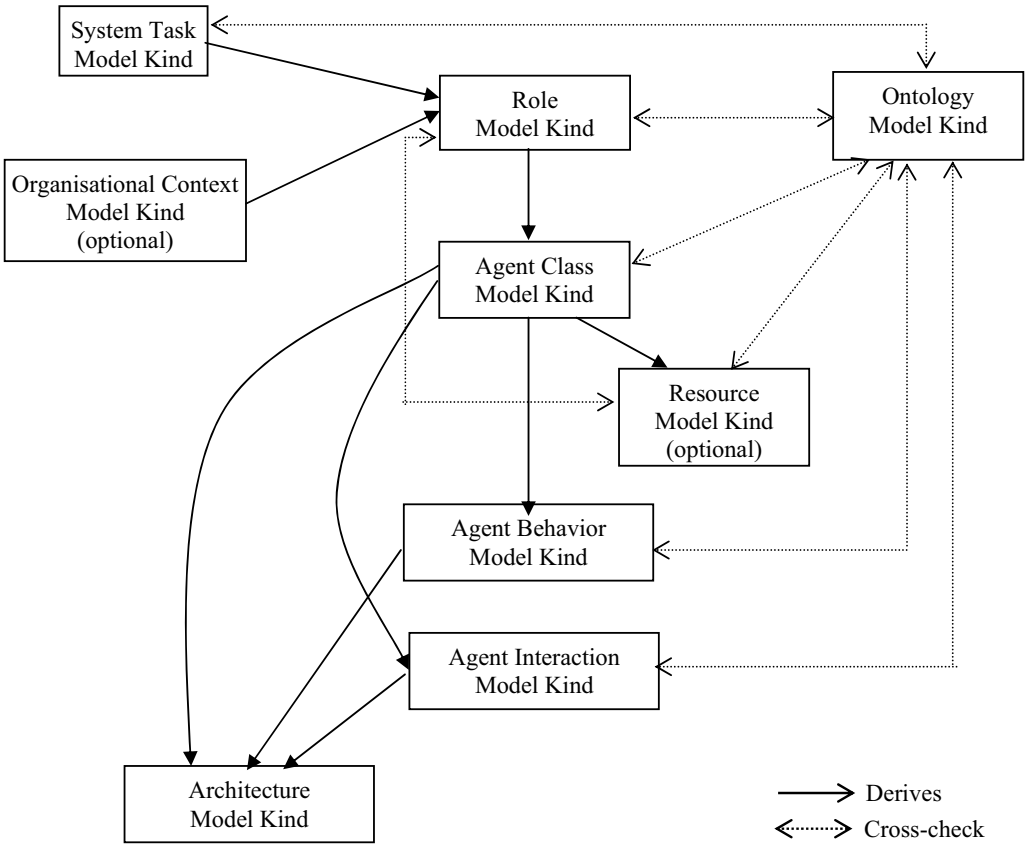



Figure 6.3 – MOBMAS Model Kinds

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

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

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

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

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

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

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

[41] Acquaintance refers to interaction pathway.

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

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

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

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

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

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

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

## 6.1.4. Illustrative Applications

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

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

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

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

### *6.1.4.1. Product search application*

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

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

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

### *6.1.4.2. Conference program management application*

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

# 6.2. ANALYSIS ACTIVITY

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

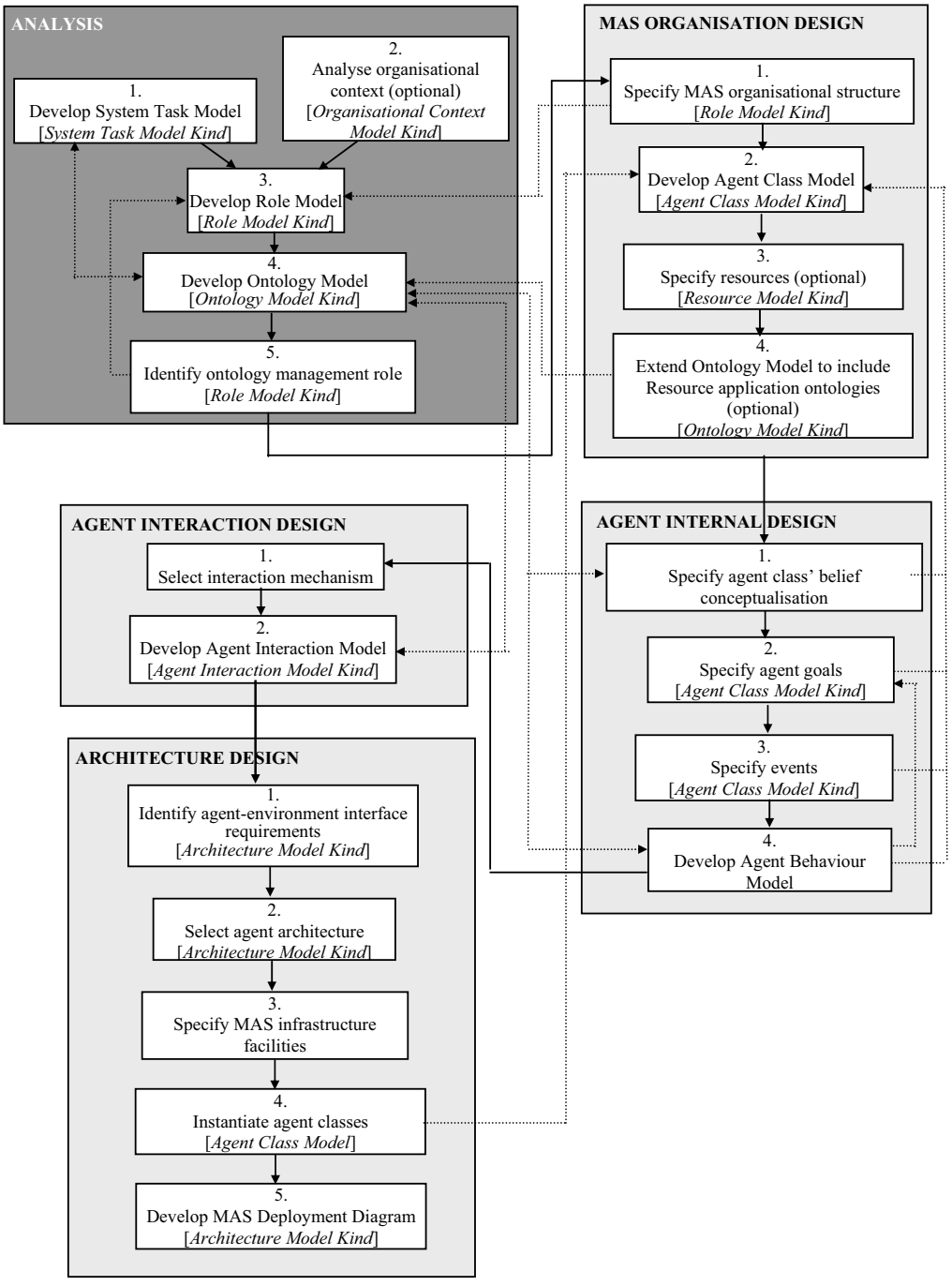



Figure 6.4 – MOBMAS development process

## 6.2.1. Step 1 – Develop System Task Model

The term "*system-task*" is used to mean anything that the target system should or will do. It represents the required functionality of the system. For example, system-tasks of the illustrative Product Search application (cf. Section 6.1.4.1) are "Satisfy user query" and "Process user feedback" (Figure 6.5).

The identification of system-tasks is not part of MOBMAS. It is presumed to be conducted by a separate Requirements Engineering effort. Hence, MOBMAS refers the developer to the vast amount of existing work on Requirements Engineering for more techniques on system-tasks elicitation, e.g. Kotonya and Sommerville (1998), Macaulay (1996), Haumer et al. (1998), Duursma (1993), Dardenne et al (1993), Yourdon (1989), DeMarco (1978), Potts (1999) and Wiegers 2003. The development process of MOBMAS starts from the identified set of system-tasks to produce a **System Task Model Kind**. This model kind aims to capture the following information.

- Identity of system-tasks.
- Conflicts amongst system-tasks (if any): Conflicts exist when different system-tasks cannot be accomplished together without being compromised (Dardenne et al. 1993). In the illustrative Product Search application, no conflicting system-tasks are found. However in many other applications such as MASs for library service management, system-tasks such as "Maintain long borrowing period" and "Maintain regular availability" are in conflict with each other (Dardenne et al. 1993).
- Functional decomposition of system-tasks (if required): A particular system-task may be decomposed into smaller-scale, constituent system-tasks which are referred to as "*sub-system-tasks*". Each decomposition may either be:
  - **AND-decomposition**: that is, when the accomplishment of a system-task requires the execution of *all* of its sub-system-tasks. For example, system-task "Find answer to user query" can be AND-decomposed into sub-system-tasks "Extract keywords from user query*"* and "Gather information from resources" (Figure 6.5); or
  - **OR-decomposition**: that is, when the accomplishment of a system-task involves the execution of *any* of its sub-system-tasks. For example, system-task "Identify appropriate resources*"* can be OR-decomposed into sub-system-tasks "Identify appropriate databases*"* and "Identify appropriate web-servers*"* (Figure 6.5).

For each decomposition, MOBMAS recommends the developer to specify whether the decomposition is full or partial.

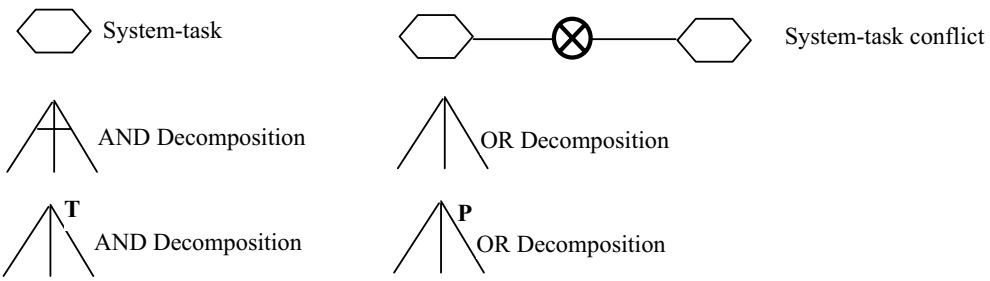
- **Full decomposition** applies when the accomplishment of a system-task is *totally equivalent* to the execution of its sub-system-tasks (either all or any of the sub-system-tasks, depending on whether the decomposition is an AND or OR decomposition). For example, system-task "Satisfy user query" is fully decomposed into sub-system-tasks "Accept user query", "Find answer to user query" and "Display result for query" (Figure 6.5) because the successful execution of these three sub-system-tasks automatically results in the accomplishment of the system-task "Satisfy user query".
- **Partial decomposition** applies when the accomplishment of a system-task is *not totally equivalent* to the execution of its sub-system-tasks. In other words, there are certain actions that need to be performed by the system-task but which are not accounted for by its sub-system-tasks. For example, system-task "Process user feedback" is partially decomposed into sub-system-tasks "Receive user feedback" and "Display acknowledge" (Figure 6.5) because, apart from receiving feedback and displaying acknowledgement, system-task "Process user feedback" also needs to compute feedback rating and refine search algorithms to improve the system's future performance.

The identification of full versus partial decomposition will assist the developer in identifying roles and roles' tasks later on.

After system-tasks are identified and a (preliminary) System Task Model is constructed, the developer is strongly recommended to validate the model against the Ontology Model, whose development is discussed in Step 4 of the Analysis activity (c.f. Section 6.2.4). The MAS Application ontologies specified in the Ontology Model may help to reveal new system-tasks that have not yet been uncovered, thus assisting in the refinement of the System Task Model. More discussion on this validation is presented in Section 6.2.4.1.c.

### *6.2.1.1. Notation of System Task Diagram*

System Task Model Kind of MOBMAS is depicted by one or more **System Task Diagrams**. The notation for the System Task Diagram is as follows.




A System-Task Diagram for the illustrative Product Search MAS is presented in Figure 6.5. It should be noted that the functional decomposition of system-tasks does not always have a simple tree structure as in Figure 6.5. Two or more system-tasks may share the same sub-system-task(s).

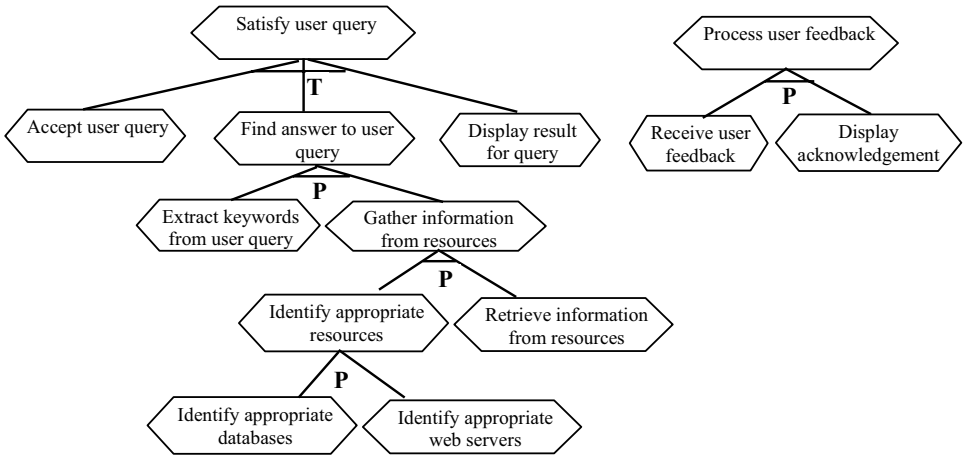



Figure 6.5 – System Task Diagram for Product Search MAS

## 6.2.2. Step 2 – Analyse Organisational Context (Optional)

Even though the analysis of system-tasks generally provide adequate inputs to the identification of roles later on, an investigation of the *existing structure of the MAS' organisational context* (i.e. the structure of the organisation which MAS supports, automates or monitors) can further assist in the process of role identification. This is so

because the organisational structure of MAS may be directly derived from the existing structure of the MAS' organisational context (e.g. consider enterprise information systems and workflow management systems) (Zambonelli et al. 2003).

MOBMAS suggests investigating the structure of the MAS' organisational context if this structure satisfies **all** of the following conditions.

- It is known and clearly defined.
- It is well-established, not likely to change, and has proven or been accepted to be an effective way to function. Accordingly, it is desirable for the future MAS to mimic this existing structure.

For example, consider the illustrative Conference Program Management application (cf. Section 6.1.4.2). The existing organisational structure of the human conference committee is composed of a "Program Committee (PC) Chair", several "PC Members" and many "Reviewers" (Ciancarini et al. 1998; Zambonelli et al. 2001a; Zambonelli et al. 2003; Ciancarini et al. 1999). Assuming that this structure has always been adopted by the human conference organisers, and that the organisers do not wish to change the way their conferences are managed, the developer should investigate this structure for the development of a software Conference Program Management MAS.

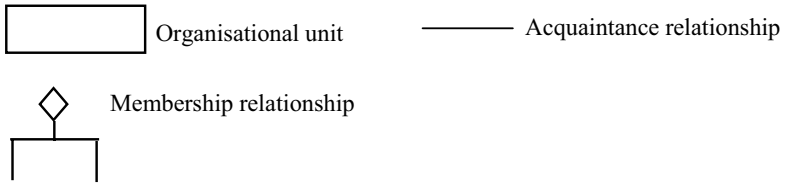
### *6.2.2.1. Develop Organisational Context Model Kind*

The existing structure of the organisational context is captured in an **Organisational Context Model Kind**, which contains one notational component, **Organisational Context Chart**. Since this MOBMAS step is only recommended to applications where the structure of the organisational context is known, clear and well-established, it is generally a straightforward task to develop this model kind. The developer needs to specify:

- the *organisational units*: i.e. the positions or individuals or departments that exist in the organisational context; and
- the *relationships* between these units, namely "*acquaintance*" relationships (where one organisational unit interacts with another) and "*membership*" relationships (where one organisational unit is part of another).

The developer may identify these elements from various sources, including the business organisational chart, business process documentation, interviews, questionnaires, investigation of employee manuals, orientation pamphlets, memos and annual company reports (Awad 1985).

MOBMAS borrows UML notation for the Organisational Context Chart.




An Organisational Context Chart for the illustrative Conference Program Management MAS is presented in Figure 6.6.

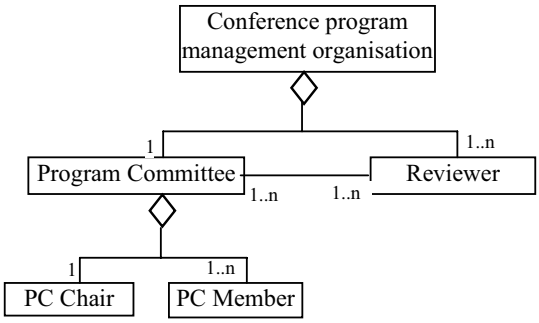



Figure 6.6 – Organisation Context Chart for the Conference Program Management MAS

## 6.2.3. Step 3 – Develop Role Model

The notion of "role" in agent-oriented development is analogous to the notion of "role" in a play or members in a typical company (Wood 2000; Kendall 1999). It refers to the position of an entity in an organisation and defines what the entity is expected to do in the organisation (Ferber and Gutknecht 1998; Demazeau and Costa 1996). In MOBMAS, roles serve as the building blocks for defining agent classes. Each agent class is associated with one or more roles, which establish the agent class' expected behaviour and position in the MAS organisational structure.

MOBMAS specifies all roles in the **Role Model Kind**, which consists of one notational component, **Role Diagram**. Sections 6.2.3.1 and 6.2.3.2 discuss the identification of

roles and role-tasks respectively, while Section 6.2.3.3 presents the notation for Role Diagram.

### *6.2.3.1. Identify roles*

MOBMAS identifies roles from system-tasks and, if Step 2 – "*Analyse Organisational Context*" has been carried out, from the existing structure of the MAS' organisational context. Section 6.2.3.1.a presents techniques for identifying roles from system-tasks, while Section 6.2.3.1.b recommends how roles can be identified with the analysis of MAS's organisational context.

#### 6.2.3.1.a. Identify roles from system tasks

Generally, each system-task specified in the System Task Model should be assigned to one role. However, each role can be delegated multiple system-tasks for the sake of efficiency.

The grouping of multiple system-tasks into one role should be guided by the principle of *strong internal coherence and loose coupling* in term of functionality. Each role should represent a functionally coherent cluster of system-tasks that is sufficiently different from other clusters (Lind 1999; Padgham and Winikoff 2002). This principle helps to promote modularity in system design.

Some other heuristics that may indicate the need to delegate multiple system-tasks into one role are:

- when the system-tasks are likely to interact significantly with each other (e.g. for inputs/outputs exchanges). Grouping these system-tasks into one role will help to reduce the amount of interactions amongst roles and ultimately amongst agents; or
- when the system-tasks require the same data (e.g. input information, domain knowledge). Assigning these system-tasks to one role means that only this one role needs to acquire, or be equipped with, the required data; or
- when the system-tasks need to access to the same resource (e.g. information sources or legacy systems). Delegating these system-tasks to one role means that only this one role needs to implement an interface with the resource.

On the other hand, some system-tasks may *not* be appropriate to be grouped into one single role, particularly when:

- the system-tasks are executed at different locations at the same point in time. For example, system-task "Accept user query" and "Retrieve information from resources" are executed at the user's site and the remote resources' site respectively, thus they should not be performed by the same role; or
- the system-tasks need to satisfy certain security and privacy requirements. For example, data associated with one system-task should not be available to another.

In the case when a system-task has been *fully decomposed* into sub-system-tasks (cf. Section 6.2.1), it does not need to be assigned to any role, because it is presumed to be accomplished via the execution of the sub-system-tasks. Accordingly, only the sub-system-tasks need to be assigned to roles. However if a system-task has been *partially decomposed* into sub-system-tasks, it must be assigned to a particular role because its accomplishment is not equivalent to the accomplishment of the sub-system-tasks.

In some cases, a single system-task needs to be assigned to multiple roles. This occurs when the system-task requires the *collective* effort of different roles. This type of task is referred to as a "**joint task**" (or "social task" in Omicini 2000 and Ciancarini et al. 2000). For example, in the illustrative Conference Program Management application, one system-task is to "Distribute papers among members" in such a way that each paper is allocated to a required number of PC members, and each member is assigned a required number of papers. This system-task should be modelled as a joint task because it is the shared responsibility of both "PC Chair" and "PC Member" roles (cf. Section 6.1.4.2; Figure 6.9).

Nevertheless, just because a system-task requires inputs from multiple roles does not mean that it should be modelled as a joint task. Such a system-task can be assigned to one single role, which acts as the primary accountable and controlling party for the task execution. This role will then interact with other roles when necessary for input/output information. For example, system-task "Find answer to user query" can be assigned to one role "Searcher", which interacts with "InfoSource Wrapper" role for information to fulfil the system-task.

Therefore, as a generic guideline, MOBMAS suggests classifying a system-task as a joint task (thus assigning it to multiple roles) only if the *control* of the system-task needs to be *equally* spread among the participating roles, or equivalently, if all the participating roles are *equally* accountable for the accomplishment of the system-task, such as in the case of system-task "Distribute papers among members" (note that for this system-task, "PC Members" are given the authority to choose for themselves the papers they want to review. "PC Chair" does not impose the papers on them[44]).

Figure 6.7 shows the identification of roles for the illustrative Product Search MAS.

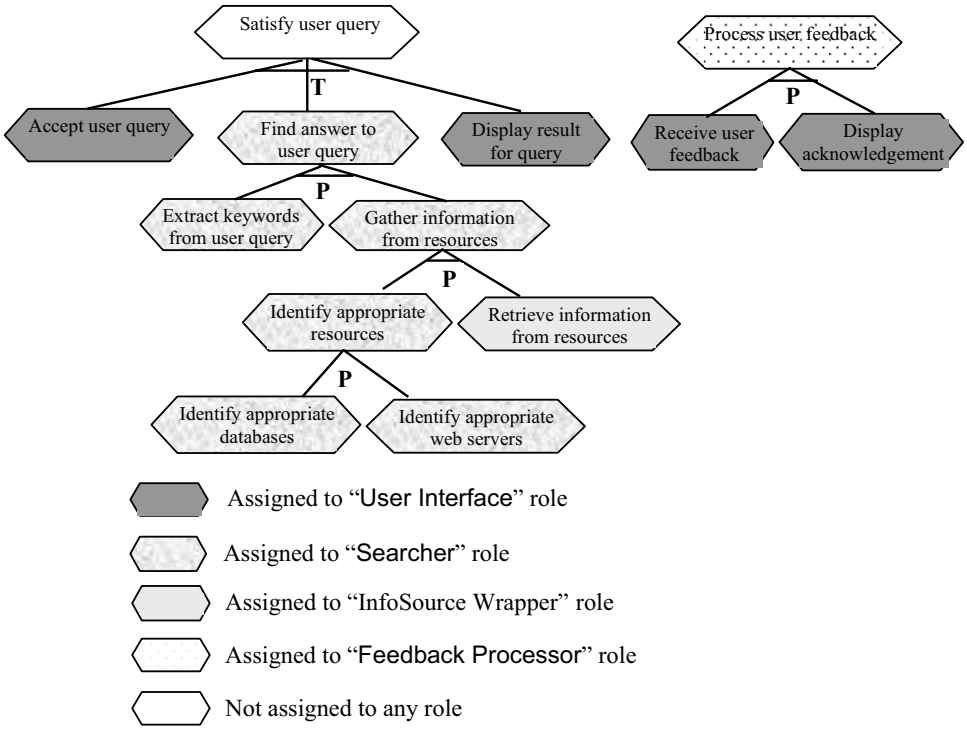



Figure 6.7 – Final roles for Product Search MAS

### 6.2.3.1.b. Identify roles from the structure of MAS' organisational context (optional)

If Step 2 – "*Analyse Organisational Context*" has been performed, the investigation of the existing structure of MAS' organisational context can greatly assist in the process of role identification. Specifically, some preliminary roles can be identified from the

[44] Even though this assumption does not always hold in real-life conferences, it is made in this thesis for the purpose of illustration (cf. Section 6.1.3.2).

organisational context's structure, thereafter being verified with the analysis of the system-tasks as discussed in Section 6.2.3.1.a.

In general, each *organisational unit* in the MAS' organisational context can be mapped onto one role, since the functionality of each organisational unit is often internally coherent and loosely coupled from the other units (Wooldridge et al. 2000). In cases when the MAS' organisational context exhibits a hierarchical structure (i.e. when there are "membership" relations among the units), the developer can either:

- map each *leaf-node* organisational unit to a role; or
- map the whole *upper-level unit* to a role.

The former is recommended if the leaf-node organisational units are loosely coupled in term of functionality, while the latter is appropriate if the leaf-node units are strongly coupled. For example, units "PC Chair", "PC Member" and "Reviewer" in Figure 6.6 can each be mapped to a different role since their functionality is loosely coupled from each other.

It should be noted that the direct mapping between existing organisational units and the software roles does not necessarily result in an efficient MAS design. This is so because, firstly, the reasons that may have driven the existing organisation to adopt a particular structure may not necessarily apply to the MAS organisation, and secondly, the mere presence of MAS may introduce changes to the existing organisation (Zambonelli et al. 2003). The developer therefore should *always* analyse the system-tasks to validate the set of identified roles (cf. Section 6.2.3.1.a). The existing structure of the MAS' organisational context should only serve as an additional resource for the identification of software roles.

### *6.2.3.2. Specify role-tasks*

"Role-tasks" are tasks that a particular role is responsible for fulfilling. In MOBMAS, role-tasks can be *directly* mapped from the *system-tasks* that the developer delegates to roles during the process of role identification (cf. Section 6.2.3.1.a). This mapping is generally one-to-one. For example, role-tasks of "User Interface" role are "Accept user

query", "Display result for query", "Receive user feedback" and "Display acknowledgement" (cf. Figure 6.7).

Given the close inter-connection between roles, role-tasks and system-tasks, the development of *Role Model* and *System Task Model* should be performed in a highly iterative and spiral manner. In practice, the modelling of a particular role may discover some role-tasks that have not been identified in the System Task Model, leading to the revision of the System Task Model.

### *6.2.3.3. Notation of Role Diagram*

The Role Model Kind of MOBMAS is depicted by a Role Diagram, which shows the *specification of each role*, *acquaintances* between roles and *authority relationships* that govern inter-role acquaintances.

The specification of each role involves the specification of role name and role-tasks, both of which have been defined during the process of role identification (cf. Section 6.2.3.1.a).

Acquaintances between roles represent inter-role interaction pathways. Preliminary role acquaintances can be identified from:

- *the relationships between system-tasks that the roles are responsible for*: If two roles are responsible for a system-task and a sub-system-task respectively, or if the two roles are responsible for sibling system-tasks[45], they are likely to interact with each other; or
- *the acquaintance relationships between the existing organisational units of the MAS' organisational context*, if Step 2 – "*Analyse Organisational Context*" has been performed: If two roles are responsible for two acquainted organisational units, they are likely to interact with each other.

The specification of roles and inter-role acquaintances will be validated/refined in later steps of MOBMAS development process. The specification of authority relationships between roles is discussed in the "*MAS Organisation Design*" activity (Section 6.3).

---

[45] That is, sub-system-tasks that share the same parent.

MOBMAS notation for the Role Diagram is as follows.

- Each role is depicted as a rectangle, which contains two compartments.
  - The top compartment, marked with keyword **role**, specifies the role name.
  - The bottom compartment, marked with keyword **role-tasks**, lists the role-tasks.
- Each inter-role acquaintance is depicted as an undirected line between roles.

For roles that share a common *joint task* (cf. Section 6.2.3.1.a), the task should be labelled with an **adornment *(J)*** to distinguish itself from other role-tasks. This highlight will help the developer to identify the existence of joint tasks in the target MAS during the subsequent steps of MOBMAS.

Notation for inter-role authority relationships is presented in the "*MAS Organisation Design*" activity.

Figure 6.8 shows the Role Diagram for the illustrative Product Search MAS, while Figure 6.9 illustrates a Role Diagram for the Conference Program Management MAS.

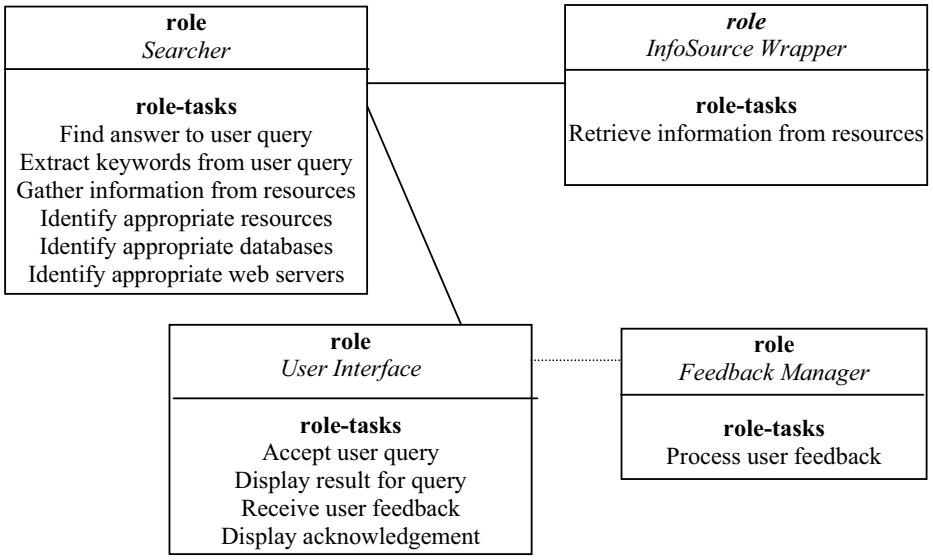



Figure 6.8 – Role Diagram for Product Search MAS (cf. Figure 6.7)

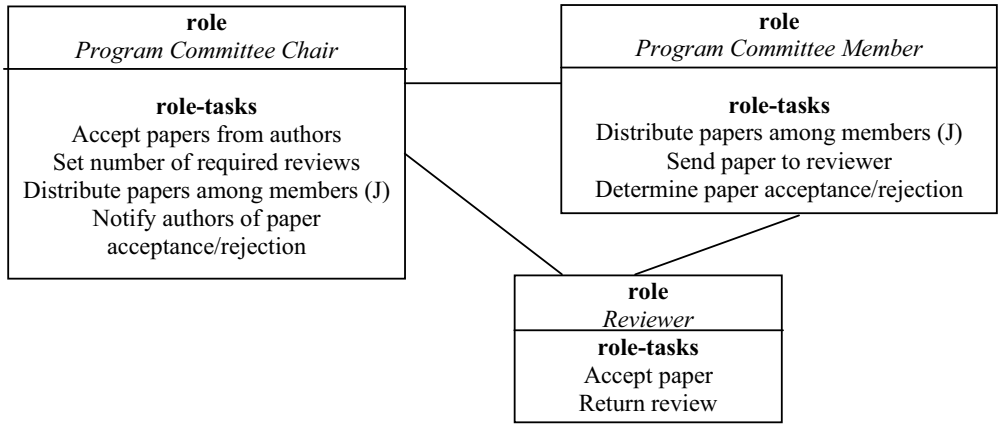



Figure 6.9 – Role Diagram for Conference Program Management MAS

## 6.2.4. Step 4 – Develop Ontology Model

MOBMAS captures all ontologies required by the target MAS in an **Ontology Model Kind**. This model kind is subject to ongoing refinement and verification throughout the development process of MOBMAS.

Previously in Section 2.3.3, a taxonomy of ontology has been presented. This taxonomy defines four types of ontology based on their level of generality: Top-level ontologies, Domain ontologies, Task ontologies and Application ontologies. The former three types are generic across applications since they are independent of the tasks at hand[46] and/or the domains at hand[47]. Meanwhile, Application ontologies are specific to a particular application because concepts in these ontologies are related to both domains and tasks at hand (Guarino 1998; van Heijst et al. 1997; Gennari et al. 1994).

In the context of software system development, only ***Application ontologies*** need to be defined and used. This is so because a software system only needs to concern with those concepts that are specific to the application at hand (Guarino 1998).

[46] Domain ontologies such as Medicine Domain Ontology, Car Domain Ontology and Automobile Domain Ontology are independent of the tasks at hand.
[47] Task ontologies such as Negotiation Task Ontology, Diagnosis Task Ontology, "Ranking by Weight" Task Ontology and Propose-and-Revise Task Ontology are independent of the domains at hand.

In the development of MAS in particular, MOBMAS recommends classifying Application ontologies into two categories: *MAS Application ontologies* and *Resource Application ontologies*.

- **MAS Application ontologies:** conceptualise the application provided by *the target MAS*. In particular, they define all the concepts and relations that the agents need to know and share about the MAS application (e.g. about the target domains and tasks). By committing to and sharing these MAS Application ontologies, agents in the system are equipped with the (same) conceptual knowledge of their application, thereby being able to operate and communicate with each other.

- **Resource Application ontologies**: conceptualise the application provided by a *resource* in the MAS system. In particular:
  - if the resource is a *processing application system* (e.g. a legacy system), the corresponding Resource Application ontology captures all the concepts and relations that conceptualise the domains and tasks/services of the resource; and
  - if the resource is an *information source* (e.g. a database), the corresponding Resource Application Ontology captures all the concepts and relations that conceptualise the information stored inside the resource. It may be derived from the information source's conceptual schema (Hwang 1999; Guarino 1997).

Resource Application ontologies are only necessary for heterogeneous MASs that contain non-agent software resources apart from agents. In these systems, only agents that directly interface with the resources will need to hold knowledge of the Resource Application ontologies, since only these agents are required to know about the conceptualisation of the resources' applications.

Figure 6.10 shows the difference between MAS Application ontologies and Resource Application ontologies in their scope of usage.

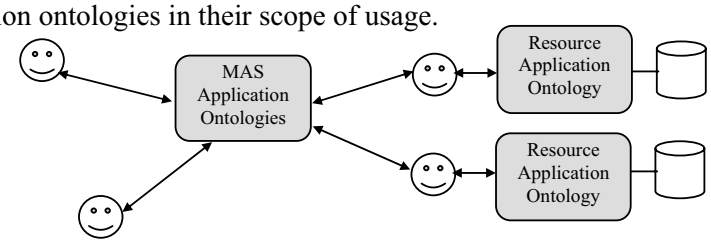



Figure 6.10 – MAS Application ontologies and Resource Application ontologies

The development of MAS Application ontologies is discussed in Section 6.2.4.1, while Resource Application ontologies are examined during the "*MAS Organisation Design*" activity (Section 6.3).

### *6.2.4.1. Develop MAS Application Ontologies*

MAS Application ontologies can be built by selecting concepts from either or both of the relevant Domain ontologies and Task ontologies, thereafter specialising these concepts to suit the MAS application (van Heijst et al. 1997) (Figure 6.11). As recommended by Guarino (1998), a concept in the MAS Application ontology can be a specialisation (or an instantiation) of a concept in a relevant Domain ontology and/or a concept in a Task ontology. For example, the MAS Application ontology of a Car E-business MAS may define concept "Car-price-offer", which is the specialisation of concept "Price" from a Car Domain Ontology and concept "Offer" from a Negotiation Task Ontology.

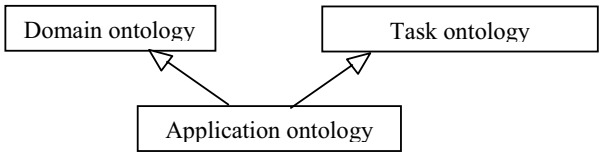



Figure 6.11 – Application ontology as a specialization of Domain ontology and Task ontology, represented in UML (Guarino 1998)

MOBMAS does *not* provide support for the process of ontology development. It assumes that all required MAS Application ontologies for the target system are developed by a separate ontology engineering effort, conducted by domain experts, ontology engineers or the MAS developer himself. MOBMAS however provides techniques for the identification of relevant input Domain ontologies and Task ontologies for the construction of MAS Application ontologies.

#### 6.2.4.1.a. Identify input Domain ontologies and Task ontologies for the construction of MAS Application ontologies

MAS Application Ontologies should obtain and/or specialise concepts from those Domain ontologies that conceptualise the target domains of the MAS. For example, in the illustrative Product Search MAS, since the system deals with search queries on *cars*, its MAS Application ontology should specialise concepts defined in a Car Domain

Ontology, such as "Make", "Model", "Steering" and "Cost". These car-related concepts allow the agents to "understand" and process user queries, as well as communicate the car-related information with each other. Similarly, for the Conference Program Management MAS (Figure 6.9), the developer should reuse and/or specialise the concepts from a Conference Domain Ontology, such as "Paper", "Paper author" and "Reviewer".

In order to identify the target domains of MAS, the developer can investigate the *System Task Model* and *Role Model*. The identified system-tasks and role-tasks provide an overview of the MAS' purpose, scope and behaviour, thereby revealing whether the MAS is related to any specific domains. For example, role-tasks "Accept user query", "Extract keywords from user query" and "Display result for query" indicate the need to know about the Information Retrieval domain, which involves concepts such as "User Query", "Keyword" and "Hit".

MAS Application Ontologies may also need to reuse and/or specialise concepts from Task ontologies. This need arises when the knowledge required to fulfil MAS' tasks do not match the domain knowledge provided by Domain ontologies (Gennari et al. 1994). This mismatch may be a *semantic* gap (i.e. when the knowledge needed for the tasks is missing from the Domain ontology), or a *syntactic* mismatch (i.e. when the knowledge required for the tasks can be satisfied by the domain knowledge, but the domain knowledge needs to be rearranged or at least renamed) (Gennari et al. 1994). If the mismatch is syntactic, the MAS Application ontology can be derived solely from the Domain ontologies. However if the mismatch is semantic, the developer should build the MAS Application ontology by augmenting the Domain ontologies with those concepts obtained from Task ontologies.

For example, consider a Car E-business MAS which involves a "car price negotiation" task. Apart from the need to know about the price of cars (which is a domain-specific concept), the task also needs to deal with negotiation-specific concepts such as offer of car price, utility rating of offer and price settlement. Thus, the system should specialise concepts "Offer" and "Settlement" of the Negotiation Task Ontology into "Car-price-offer" and "Car-price-settlement" for the MAS Application ontology. These specialised

concepts represent the roles that the domain-specific concept "Car-price" plays in the negotiation task (Guarino 1996).

Note that all concepts defined in the MAS Application ontologies are no longer solely domain-specific or task-specific. They are essentially application-specific, that is, related to both domains and tasks at hand.

The developer should investigate the **Role Model**, particularly role-tasks, to identify the need for concepts from Task ontologies. As stated earlier, only role-tasks that require knowledge *semantically* mismatched from domain knowledge will need to be investigated. For example, in the illustrative Product Search MAS, role-task "Find answer to user query" may employ a "ranking by weight" problem-solving method to rank the results found for user query. Thus, it may require task-specific concepts such as "Hypothesis", "Weight" and "Rank". The developer should therefore consult the Ranking –by-Weight Task Ontology to obtain and specialise these concepts. On the other hand, role-tasks "Accept user query" and "Extract keywords from user query" deal directly with the domain concepts "User query" and "Keywords", thus do not require concepts from any Task ontology.

Note that the knowledge requirements of MAS' tasks may not be apparent until the "*Agent Internal Design*" activity (Section 6.4), thus indicating the need for iterative refinement of MAS Application ontologies.

Generally, the developer should consult the existing, reusable Domain ontologies and Task ontologies when building MAS Application ontologies for the target system. However, if no reusable ontologies can be found for the target domains and/or tasks, the developer can either:

- develop the Domain ontologies and Task ontologies from scratch, thereupon building the MAS Application ontologies by specialising these ontologies; or
- directly develop the MAS Application ontologies from scratch, without consulting any Domain ontologies or Task ontologies.

The former approach is time-consuming but facilitates the use of MAS Application ontologies by future applications, while the latter approach is time saving, but not capable of supporting reuse.

The development of MAS Application Ontologies can be supported by numerous specialised ontology-engineering methodologies and guidelines, e.g. IDEF5 (Knowledge Based Systems Inc 1994), METHONTOLOGY (Fernandez et al. 1997), Grüninger and Fox (1995), Uschold and King (1995), Noy and McGuinness (2001), Boicu et al. (1999), Gruber (1993a) and Wache et al. (2001).

#### 6.2.4.1.b. Specify ontological mappings between MAS Application ontologies

When modelling MAS Application ontologies, the developer should pay particular attention to the specification of "*ontological mappings*" between these ontologies where necessary. As defined in Section 2.3.2.1, an ontological mapping is a semantic correspondence between two concepts of two different ontologies (Madhavan 2002). Research in linguistics, logics and psychology has proposed many potential semantic correspondences between concepts (Winston et al. 1987). Chaffin and Herrmann (1988), for example, provides a list of 31 possible semantic correspondences. Winston et al. (1987) presents a taxonomy of semantic correspondences that pertain to part-whole relationships. Storey (1993) suggested seven major semantic correspondences between concepts: "inclusion", "possession", "attribution", "attachment", "synonym", "homonym" and "case". The developer is therefore allowed to adopt whichever semantic correspondences that suit the mapping of the target MAS Application ontologies. However, MOBMAS suggests the developer consider the following three "basic" semantic correspondences, which cover most (if not all) of the possible semantic associations between concepts (Parent and Spaccapietra 1998):

- **equivalent**: i.e. where two concepts are semantically equivalent. For example, concept "Car audio system" in the Car MAS Application Ontology is equivalent to concept "Car audio" in the Entertainment Systems Domain Ontology (Figure 6.14);
- **subsumes**: i.e. where one concept semantically includes another concept (either in term of whole-part, specialisation or instantiation). For example, concept "Type" in

the CarInfo Resource Ontology subsumes concept "Sport car" in the Car MAS Application Ontology (Figure 6.32); and

- **intersects**: i.e. where one concept overlaps partially in semantics with another concept. For example, concept "Car air conditioning system" in the Car MAS Application Ontology intersects with concept "Heater" in the Electronic Product Domain Ontology (Figure 6.32).

The related MAS Application ontologies can either be mapped against each other, or against a common ontology (c.f. Section 2.3.2.1). Normally, when there are more than two ontologies to be mapped amongst themselves, the second approach should be favoured over the first, given the reasons listed in Section 2.3.2.1. The common ontology to be used in the second approach may be one of the existing MAS Application ontologies itself, or built from scratch as an inter-lingua of the existing MAS Application ontologies.

Generating ontological mappings is a labour-intensive and error prone task. The developer is referred to other research work for more information on this activity, e.g. Ehrig and Sure (2004), Kalfoglou and Schorlemmer (2003), Stumme and Maedche (2001), Calvanese et al. (2001) and Madhavan et al. (2002).

### 6.2.4.1.c. Validate System Task Model and Role Model against Ontology Model

As mentioned previously, ontologies used by MAS are constructed by a separate stream of development effort, conducted by either domain experts, ontology engineers or the MAS developer himself. This ontology development effort, while modelling the target system from the ontological point of view, is closely related and supplementary to the MAS development effort, since both involve a detailed investigation of the target application's requirements.

Accordingly, while the System Task Model and Role Model assist in the development of MAS Application ontologies, the contents of the developed MAS Application ontologies can be used to verify and validate the completeness of the two MAS analysis models. In particular, the concepts defined in the MAS Application ontologies may

correspond to, or indicate, some system-tasks or roles. For example, if concept "Keyword" has been defined in the MAS Application ontology of the Product Search MAS, the developer may uncover system-task "Extract keywords from user query" and add it to the System Task Model if not already included. Similarly, concept "Review" in the MAS Application ontology of the Conference Program Management MAS will clearly indicates a role "Reviewer" for the Role Model. Thus, the examination of ontological concepts may help to identify new system-tasks or roles and thereby help to refine the System Task Model and Role Model.

### *6.2.4.2. Language for Ontology Model Kind*

Since there are currently many representation languages for ontologies (cf. Section 2.3.4), MOBMAS does not restrict itself to any particular modelling language. However, MOBMAS recommends using a graphical language for ontology modelling to facilitate the communication with users during the analysis and design of MAS (e.g. UML, IDEF5 Schematic Language and LINGO). Nevertheless, if a graphical language is not powerful enough in term of power of expression for the ontologies at hand, textual languages can be used (e.g. CycL, KIF, KL-ONE and DAML+OIL).

For the purpose of illustration, MOBMAS adopts *UML* and *OCL* notation for ontology modelling. With this notation, the Ontology Model Kind of MOBMAS consists of multiple **Ontology Diagrams**, each of which represents an ontology as an *UML class diagram*. Ontological concepts are represented as UML ***classes*** or ***attributes*** of classes, while relations between concepts are represented as ***relationships*** between classes (Cranefield and Purvis 1999; Cranefield et al. 2001). Operations/methods of classes are not modelled because ontologies only capture the structure of concepts, not their behaviour (Bergenti and Poggi 2002).

UML allows for the representation of the following types of relationships between concepts (Object Management Group 2003).

- **Specialisation**: that is, when concept A is a type of concept B. For example, "Sport car" is a type of "Car" (Figure 6.14).
- **Aggregation**: that is, when concept A is a part of concept B. For example, "Keyword" is a part of "User query" (Figure 6.15). **Composition** can be used to

model a stronger type of aggregation, when concept A belongs to only one whole concept B and lives and dies with the whole (e.g. concept "Wheel" and "Car" in Figure 6.14) (Object Management Group 2003).

- **Association**: that is, when concept A is related to concept B. An association relationship may be described by a *predicate*, which is basically an ontological concept itself (Bergenti and Poggi 2002). For example, "Car" is related to "Car accessory" via an "accessory" association (Figure 6.14). If an association relationship embraces attributes, the association can be modelled as an "association class" (Figure 6.12).

Each relationship between concepts should be annotated with "cardinality indicators", which indicate the number of potential instances of each concept that can be involved in the relationship.

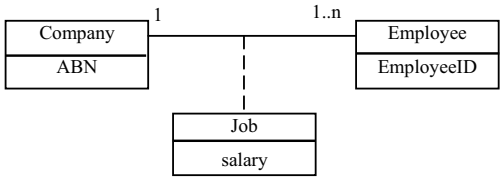



Figure 6.12 – Association Class in an ontology

With regard to the representation of *ontological mappings*, MOBMAS suggests extending the "dependency relationship" of UML (Figure 6.13). Each mapping relationship should be marked with a keyword stating the semantic correspondence, e.g. **equivalent**, **subsumes** or **intersects** (Parent and Spaccapietra 1998). If the semantic relationship is bi-directional (e.g. equivalent or synonym), the arrow can be double-headed.

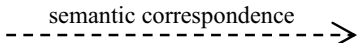



Figure 6.13 – Notation for ontology mapping

If there are axioms, rules or other assertions that specify constraints on ontological classes, attributes and relationships, these can be modelled by OCL. OCL constraints are represented as *notes* in Ontology Diagrams.

Figures 6.14 and 6.15 illustrate the Car MAS Application Ontology and Query MAS Application Ontology of the Product Search MAS.

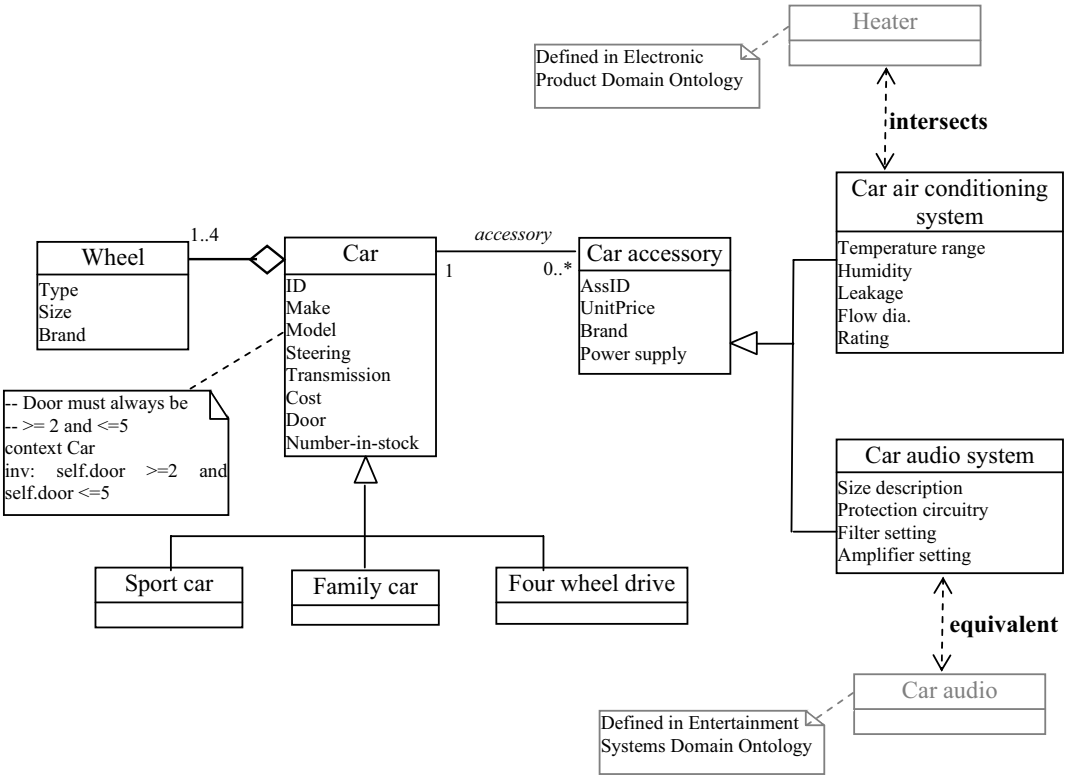



Figure 6.14 – Car MAS Application Ontology

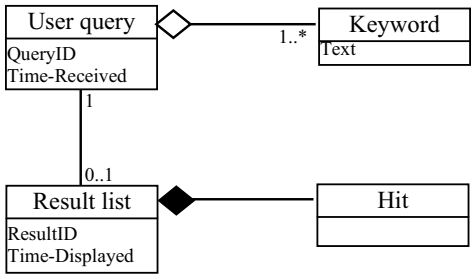



Figure 6.15 – Query MAS Application Ontology

## 6.2.5. Step 5 – Identify Ontology-Management Role

As recommended by FIPA (2001b), a MAS may store ontologies at an ontology server(s), which is exclusively controlled by an "Ontology Manager" agent[48]. Other agents in the system which wish to obtain, access or update ontologies have to communicate with the Ontology Manager (Figure 6.16).

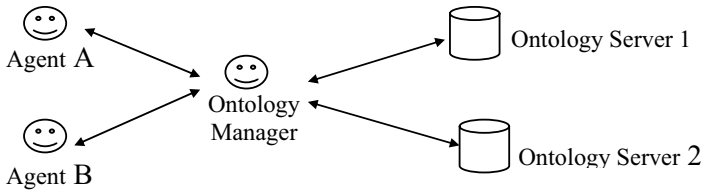



Figure 6.16 – Ontology Manager role

[48] The agent is referred to as Ontology Agent in FIPA (2001b).

Potential tasks of the "Ontology Manager" agent are:

- to perform all necessary reasoning, inferences or ontology-mapping activities to answer ontology-related queries posed by other agents;
- to distribute copies of ontologies to authorised agents;
- to control the update of ontologies (e.g. when suggestions of updates are sent by other agents); and/or
- to inform other agents (which are holding copies of ontologies) of changes in the ontologies.

The use of a specialised "Ontology Manager" agent is useful in that it helps to relieve the workload from other agents by taking care of all ontology-related reasoning and mapping activities. It also helps to ensure security by checking whether a particular agent is authorised to obtain a requested ontology or to update an ontology. Note that low-level, simple reasoning and/or maintenance activities of ontologies can be provided by the underlying ontology servers (e.g. Ontolingua Server; Farquhar 1996). What an Ontology Manager agent offers is a higher-level layer of services that may be performed on the ontologies (e.g. authorisation or complex ontology-related query processing).

Note that "Ontology Manager" agent is an application-independent component that is generally provided by the implementation framework (e.g. FIPA-based platforms such as JACK, JADE, FIPA-OS and ZEUS). The developer therefore does not have to design one from scratch, but can fine-tune the provided specification of the "Ontology Manager" agent to suit the application at hand. However, the *Role Model* of the target MAS needs to be updated to add an "Ontology Manager" role.

Nevertheless, the developer can let the agents to have direct access to the ontologies, without using any specialised "Ontology Manager" agent (Cheikes 1995; Figure 6.17). The advantage of this design is its simplicity. However, its drawbacks are that any agent can access or change the ontologies (unless the ontologies are set to "read only" mode) and the agents have to perform all the reasoning and mapping activities on the ontologies to satisfy their ontology-related queries.

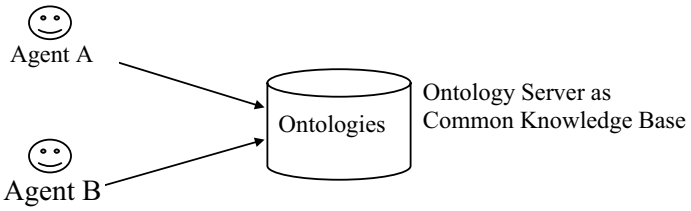



Figure 6.17 – Ontology servers without Ontology Manager role

Figure 6.18 presents the updated Role Diagram for the Product Search MAS (cf. Figure 6.8).

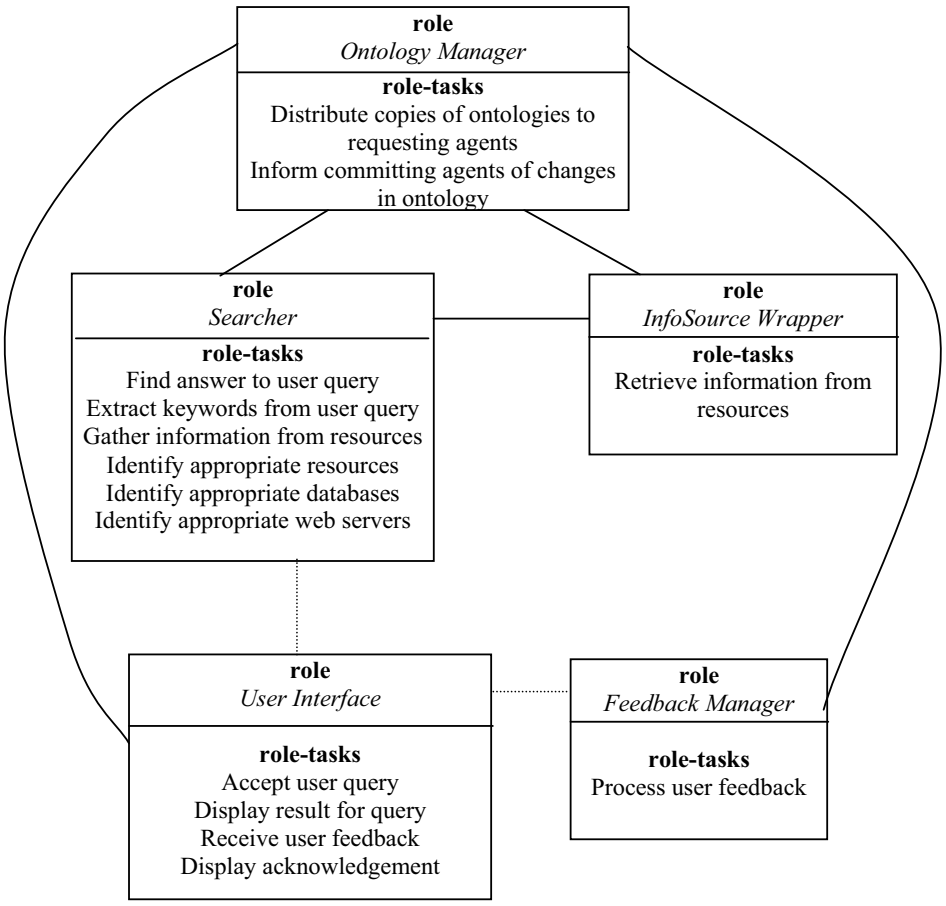



Figure 6.18 – Updated Role Diagram for Product Search MAS

# 6.3. MAS ORGANISATION DESIGN ACTIVITY

This activity of MOBMAS is concerned with specifying the organisational structure for the target MAS and a set of agent classes composing the MAS. If the MAS is a heterogeneous system that contains non-agent resources, these resources need to be identified and their applications conceptualised.

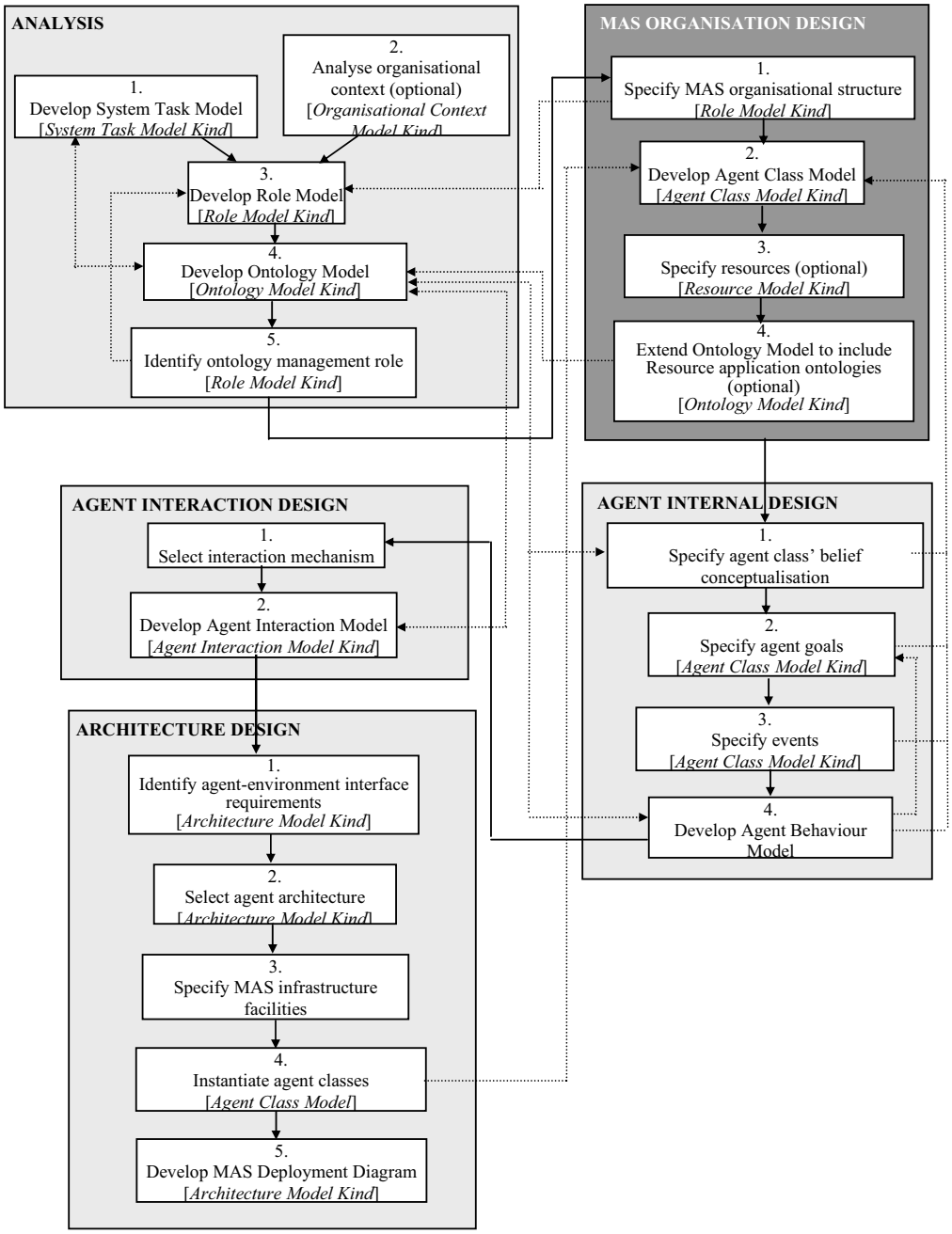



Figure 6.19 – MOBMAS development process

## 6.3.1. Step 1 – Specify MAS Organisational Structure

In MOBMAS, MAS organisational structure is modelled via *roles*, *acquaintances* between roles and *authority relationships* that govern these acquaintances (e.g. peer-to-peer or superior-subordinate relationship). This inter-role organisational structure will determine the run-time organisational structure between agents, since each agent will play a particular role(s) in the MAS organisation. It should be noted that, while the inter-role organisational structure is static and can be defined at design time, the inter-agent organisational structure may be dynamic, since each agent may change from one role to another at run-time.

In MOBMAS, a preliminary organisational structure of MAS has been defined via the identification of roles and acquaintances between roles in the "*Analysis*" activity (cf. Section 6.2.3). This current step further investigates and confirms the organisational structure of MAS, by examining the organisational structure style, specifying the authority relationships between roles, and if necessary, revising the acquaintances amongst roles. Even if the developer has performed Step 2 of the "*Analysis*" activity – "*Analyse Organisational Context*" (cf. Section 6.2.2), this step should still be performed, since the software organisational structure of MAS should *not* always mimic the structure of the MAS' organisational context. The result of this step is an updated Role Model, which is initially developed in the "*Analysis*" activity (cf. Section 6.2.3).

### *6.3.1.1. Determine MAS organisational structure*

The organisational structure of MAS can be based upon any of the following four common, basic organisational styles (Fox 1981; Lind 1999; Shen and Norrie 1999; Parrott et al. 2003):

- **Peer-to-peer**: In this structure, all roles in MAS work together as peers, with each role assuming an equal authority status compared to other roles (Figure 6.20a). Coordination between roles is based upon mutually agreed decisions. The topology of interactions is a fully connected one, where each role is allowed to interact with any other roles in the system without having to go through a mediator party.

- **Hierarchical structure:** This organisational structure organises roles into a *hierarchy of layers*, where roles in the higher layer assume a "superior" status over those in the lower layer, which assume a "subordinate" status (Figure 6.20b). A superior role exercises its authority over the subordinate roles by delegating work to the latter and coordinating the latter's efforts. A subordinate role is obliged to perform the delegated tasks and should not reject a request from its superiors (thus, the autonomy of the subordinate roles is restrained by the superior roles). Interaction pathways across layers are limited, since roles within each layer are endorsed to only interact with its immediate superiors or subordinates.

- **Federation structure**: This structure organises roles into *peer-to-peer groups*, where roles in each group are mediated by a superior role within the group (Figure 6.20c). Roles within each group can directly interact with each other, but need to interact with roles in other groups via the superior role.

- **Hybrid structure**: This structure is one that integrates any of the above styles. For example, some roles may directly coordinate as peers with each other during the fulfilment of some particular tasks, but need to be controlled by a superior role during some other tasks (Figure 6.20d).

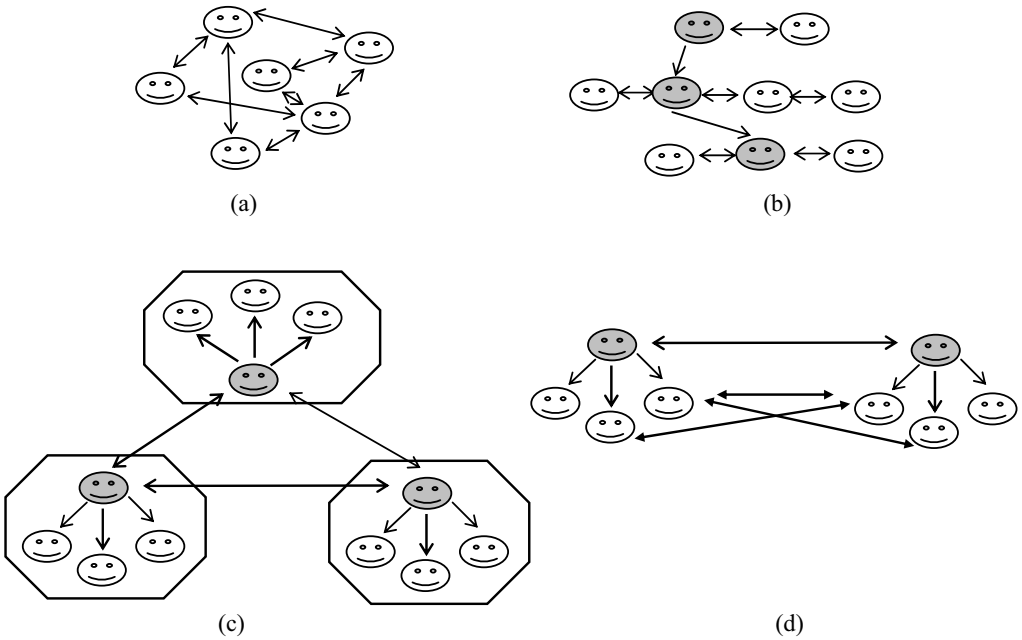



Figure 6.20 – Styles of organisational structure

The determination of which organisational style to adopt for the target MAS should take into account the following factors.

- *The existing structure of the MAS' organisational context* (if Step 2 of the "*Analysis*" activity – "*Analyse Organisational Context*" – has been performed; cf. Section 6.2.2): Generally, there is a natural tendency to mimic/reflect the existing structure of the MAS' organisation context into the software MAS system. For example, MASs that support e-business are likely to mimic the organisational structure of the human commercial transactions. This imitation is not only desirable due to the sake of conceptual simplicity, but in some cases may also come as a requirement (Zambonelli et al. 2003). Nevertheless, the developer should determine whether the existing structure is indeed efficient for the target MAS. For example, a strictly hierarchical structure found in a human organisation may be better replaced by a hybrid software organisation, where certain roles are allowed to autonomously interact as peers with each other to fulfil some tasks. This refined organisational structure promotes the autonomous capability of software roles/agents.

- *Modularity*: The developer should consider organising roles into layers or groups to promote modularity for the target system. Generally, this layering or grouping is applicable if there exist different sets of roles that are concerned with disjoint sets of responsibilities, interact loosely with each other, and/or require competencies that are not required by other sets of roles (Zambonelli et al. 2003).

- *Support for non-functional requirements*: If the target MAS has specific non-functional requirements such as security, scalability or flexibility, the adopted organisational structure should allow the MAS to meet these requirements. For example, if some information available to a group of roles should not be available to another group of roles, the developer should consider organising MAS into layers or groups. Similarly, if scalability needs to be supported (e.g. when new roles are added), federation and hierarchical structures may be more efficient than a peer-to-peer structure, since only the superior role in the respective group or layer needs to know about the existence of the new roles.

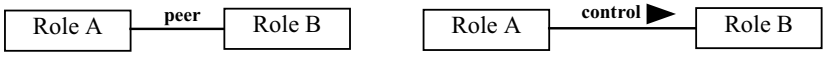
- *The number of roles in the system*: If the number of roles is small, the interaction overheads between roles may be sufficiently low for a peer-to-peer structure to be efficient. However when the number of roles is large, a hierarchical or federation structure may be more appropriate, given their ability to limit the interaction pathways amongst roles.

To date, there have been a number of attempts to catalogue the different organisational structures for software systems (based on the above four basic styles), e.g. Kolp et al. (2001), Tahara et al. (1999), Kendall (1999) and Kendall (2000). The developer can reuse and/or adapt these catalogued structures to the target MAS.

### *6.3.1.2. Update Role Model*

Once the organisational structure of MAS has been determined, the Role Model previously developed in the "*Analysis*" activity should be revised to:

- include any *new roles* that have not been identified. For example, a "Mediator", "Coordinator" or "Broker" role may be uncovered if a hierarchical or federation structure is adopted;
- show *new acquaintances* between roles (if any); and
- show the *authority relationship* governing each acquaintance.
  - Keyword **peer** should be used to represent a *peer-to-peer* relationship where two roles have equal status (Figure 6.21a).
  - Keyword **control** should be used to represent a *superior-subordinate* relationship where one role has authority over another (Figure 6.21b). The keyword should be adorned with an arrow pointing from the superior role to the subordinate role.




(a) (b)

Figure 6.21 – Notation for authority relationships between roles in Role Diagram

Figures 6.22 and 6.23 show the updated Role Diagrams for the Product Search MAS and Conference Program Management MAS respectively. The former adopts a primarily peer-to-peer structure, although it contains a superior-subordinate relationship between "Searcher" and "InfoSource Wrapper" roles. The Conference Program Management MAS adopts a hierarchical structure.

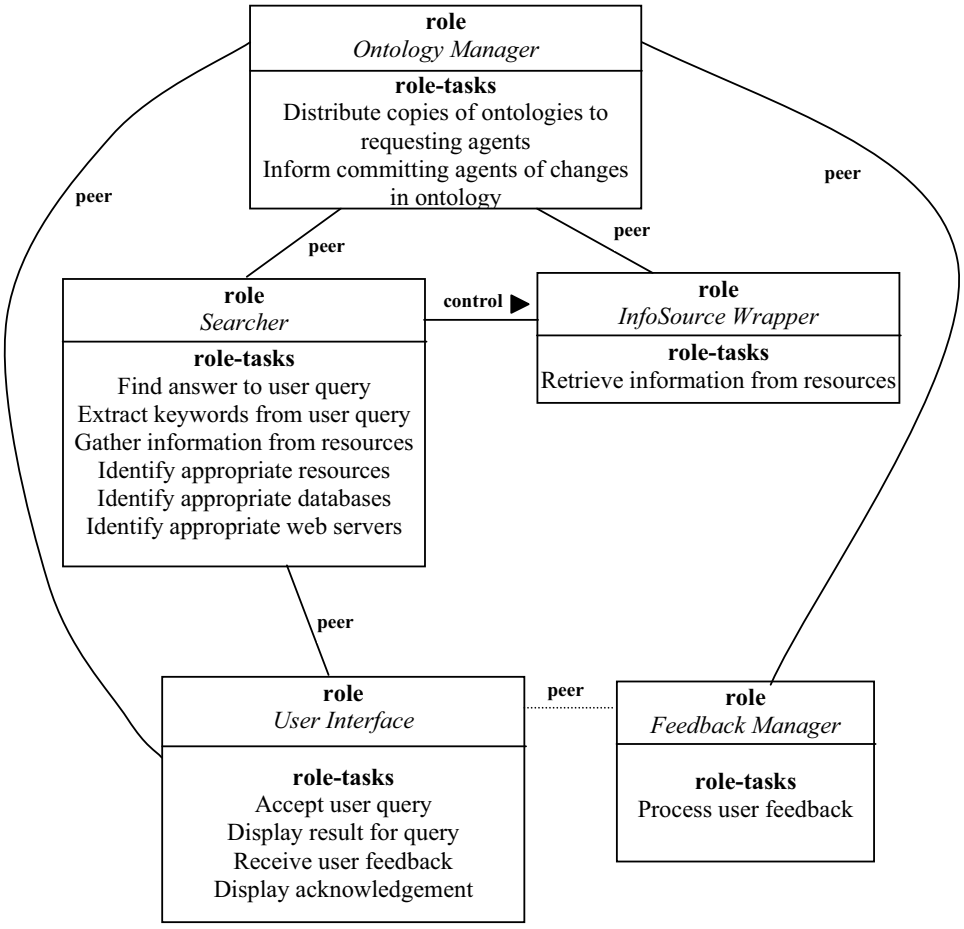



Figure 6.22 – Updated Role Model for Product Search MAS (cf. Figure 6.18)

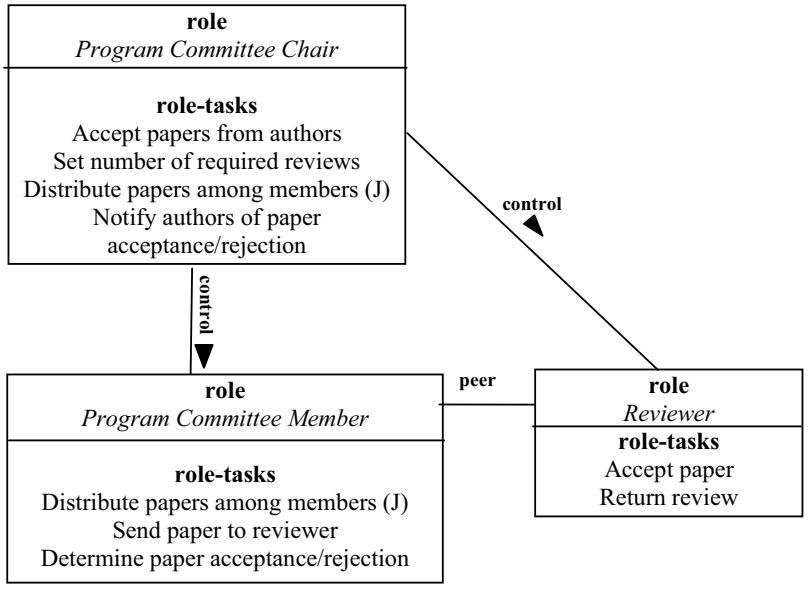



Figure 6.23 – Updated Role Model for Conference Program Management MAS (cf. Figure 6.9)

## 6.3.2. Step 2 – Develop Agent Class Model

The notion of "agent class" is analogous to the notion of "class" in OO modelling (Wood 2000). Each agent class is a template descriptor of a set of agents with similar characterisation. MOBMAS captures all agent classes in the **Agent Class Model Kind**. Techniques for identifying agent classes are presented in Section 6.3.2.1, while the notation for the Agent Class Model Kind is documented in Section 6.3.2.2.

### *6.3.2.1. Identify agent classes*

In MOBMAS, agent classes are built upon roles, with each agent class being assigned one or more roles. At run-time, an agent from an agent class may dynamically change amongst its assigned roles, thereby exhibiting dynamic behaviour and occupying dynamic positions in the MAS organisation.

Generally, roles can be associated to agent classes via *one-to-one mappings*. This one-to-one correspondence from roles to agent classes can be justified by the modularity or functional coherence of roles – a characteristic resulted from the way roles are identified[49].

Nevertheless, multiple roles may be assigned to one single agent class for the purpose of convenience. The decision of whether, and how, to map multiple roles to one agent class should be driven by the following factors.

- *Modularity*: The grouping of roles must *not* compromise the modularity of the MAS design. In other words, each agent class should represent a *coherent* software entity that does not have disparate functionality, and the overall functionality of the agent class should be easy to understand. The following simple heuristic can be applied to evaluate the functional coherence of an agent class: to find a suitable name for the agent class that encompasses all of its functionality. A coherent agent class should be easily described by a single name without any conjunctions. For example, consider an Online Shop application. An agent class that plays both roles "Client

[49] Recall that in Step 3 of the "*Analysis*" activity – "*Develop Role Model*" (cf. Section 6.2.3.1.a), roles are identified from the system-tasks in such a way that promotes strong internal coherence and loose coupling in term of functionality.

welcomer" and "Seller" can still have a simple descriptive name "Sale assistant" (Padgham and Winikoff, 2002a). Although this heuristic is not always applicable, it offers a useful and quick way for assessing modularity in MAS design.

- *Efficiency considerations*: Associating multiple roles to one single agent class may result in various efficiency improvements. For example, having one agent class playing multiple roles will mean a smaller number of agents in the MAS system than if each agent class plays a single role. Likewise, if some roles interact intensively with each other, the assignment of these roles to one single agent class will mean less interaction between agent classes. Nevertheless, the grouping of roles may also result in lower system efficiency. For instance, mapping both roles "Searcher" and "InfoSource Wrapper" in the Product Search MAS to a single agent class, say "Information Gatherer", will mean that this "Information Gatherer" agent needs to *sequentially* interact with multiple information sources to find answers to a user query. However, if each role "Searcher" and "InfoSource Wrapper" is assigned to a separate agent class "Searcher" and "Wrapper" respectively, the "Searcher" agent can *simultaneously* dispatches the query to all relevant "Wrapper" agents, which then *simultaneously* access the information sources for answers. The response time is therefore greatly improved.

- *Other non-functional requirements considerations*: The binding of multiple roles to one agent class should *not* compromise any fault-tolerant, security or privacy requirements.

### 6.3.2.1.a. Characterise agent class' dynamics

If a particular agent class has been assigned multiple roles, the developer should characterise its dynamics, that is, whether the agent class is *static* or *dynamic* regarding its role-playing behaviour.

- A *static* agent class is one whose instances are required to play all of the assigned roles throughout their lifetime[50]. For example, a "Shop assistant" agent is active in both of its assigned roles, "Client welcomer" and "Seller", during its lifetime.

[50] MOBMAS' definition of static versus dynamic property of agent classes is based upon the definition of "dynamic activation" proposed by Odell et al. (2003b).

- A *dynamic* agent class, on the other hand, is one whose instances may change their active roles from one time to another. This dynamic change occurs when (Odell at al. 2003b):
  - an agent is initially active in one role but becomes also active in some other roles. For example, in a Human Resource Management MAS, a "Staff" agent constantly plays the role "Employee", but may also become active in role "Manager" as a result of a promotion (thus taking on additional managerial responsibilities apart from normal employee responsibilities); or
  - the agent has been active in one role but becomes inactive in that role. For instance, the "Staff" agent in the above illustration may become inactive in the role "Manager" after a demotion, thus retaining only one role "Employee"; or
  - the agent switches from one active role to another. For example, a "Soccer Player" agent often switches between role "Striker" and role "Defender".

The determination of each agent class' dynamics in term of its role-playing behaviour helps to predict the dynamics of the MAS system at run-time, since the roles that each agent plays at a point in time determine its behaviour, its interactions with other agents and the overall inter-agent organisational structure.

### *6.3.2.2. Notation of Agent Class Model Kind*

The Agent Class Model Kind of MOBMAS is depicted by two notational components.

- **Agent Class Diagram** captures the specification of each agent class, including the listing of its roles, belief conceptualisation, goals and events.
- **Agent Relationship Diagram** shows the interaction pathways between agent classes, interaction pathways between agent classes and their wrapped resources (if exist), and instantiation cardinality of each agent class.

These two diagrams need to be developed in an ongoing manner throughout the MOBMAS development process. MOBMAS notation for Agent Class Diagram and Agent Relationship Diagram are presented in Figures 6.24 and 6.25 respectively.

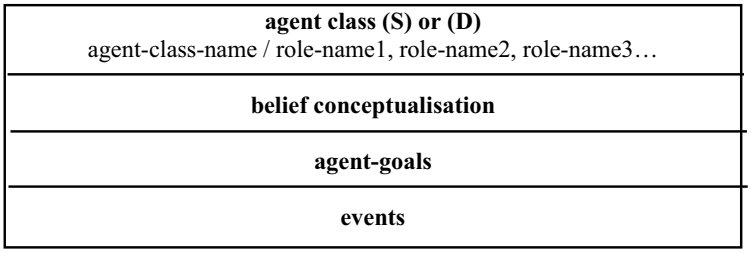



Figure 6.24 – Agent Class Diagram

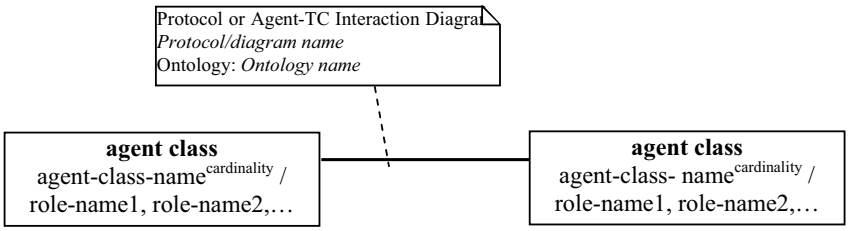



Figure 6.25 – Agent Relationship Diagram

In the **Agent Class Diagram** (Figure 6.24), each agent class is depicted as a rectangle with several compartments.

- The top compartment - marked with keyword **agent class** – specifies the agent class' name, roles and dynamics characteristic. The dynamics characteristic should be annotated as **(S)** for static agent classes, or **(D)** for dynamic agent classes. Note that if an agent class is assigned one single role, it is presumed to be static.
- The remaining compartments specify the agent class' belief conceptualisation, agent-goals and events. These constructs will be defined during the "*Agent Internal Design*" activity. At this stage, these compartments can be left empty.

In the **Agent Relationship Diagram**,

- each agent class is depicted as a rectangle marked with keyword **agent class**. The agent class' name and its roles should be restated. For simplicity, this diagram does not show the dynamics characteristic of each agent class. This information has been captured in the Agent Class Diagram;
- each acquaintance between agent classes is depicted as an undirected line connecting the agent classes. Inter-agent acquaintances can be derived from the acquaintances amongst roles in the Role Model; and
- descriptive information about each acquaintance between agent classes (e.g. ontology and interaction protocol governing the acquaintance) is attached to the

acquaintance as an UML note. This descriptive information will be determined during the "*Agent Interaction Design*" activity.

If the target MAS is a heterogeneous system which contains non-agent resources, the Agent Relationship Diagram should also show these resources and their connections with the wrapper agent classes. This issue will be discussed in Step 3 of this activity – "*Specify resources*" (Section 6.3.3).

Figures 6.26 and 6.27 present the preliminary Agent Class Diagram and Agent Relationship Diagram for the illustrative Product Search MAS (cf. Figure 6.22). Both diagrams will be refined in later steps of MOBMAS.

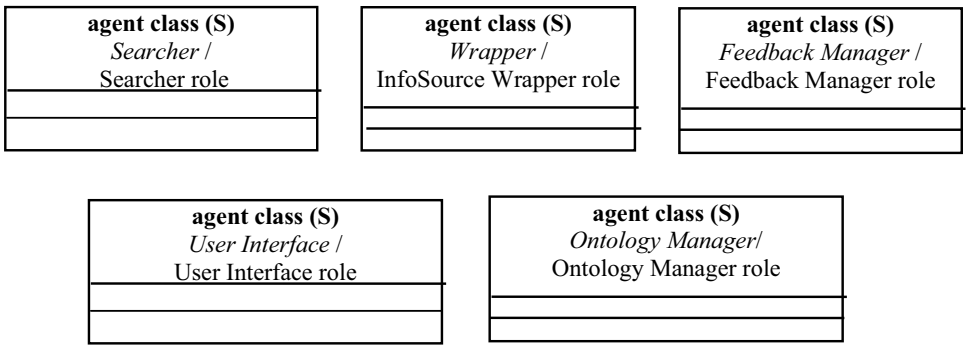



Figure 6.26 – Preliminary Agent Class Diagram for Product Search MAS

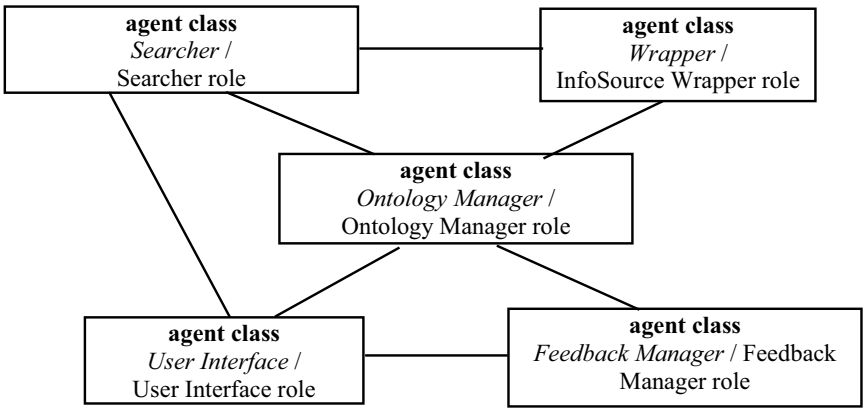



Figure 6.27 – Preliminary Agent Relationship Diagram for Product Search MAS

## 6.3.3. Step 3 – Specify Resources (Optional)

This step is only needed if the target MAS is a heterogeneous system which, apart from agents, contains non-agent resources that provide information and/or services to the agents. All of these resources are specified in the **Resource Diagram** of MOBMAS **Resource Model Kind**. Section 6.3.3.1 discusses the identification of resources for the target MAS, while Section 6.3.3.2 presents notation for the Resource Diagram.

### *6.3.3.1. Identify resources*

A resource is a non-agent software system that provides application-specific[51] information and/or services to the agents in MAS. Resources in a MAS may include (FIPA 2001a; Jennings and Wooldridge 1995):

- *information sources*, e.g. databases or web servers; and
- *processing application systems*, e.g. legacy systems, language translation programs or web services programs.

Note that the resources do not need to belong *internally* to the system and owned/used exclusively by the system (e.g. legacy system) (Figure 6.28a). They may exist externally and are available to agents in other systems (e.g. web servers) (Figure 6.28b).

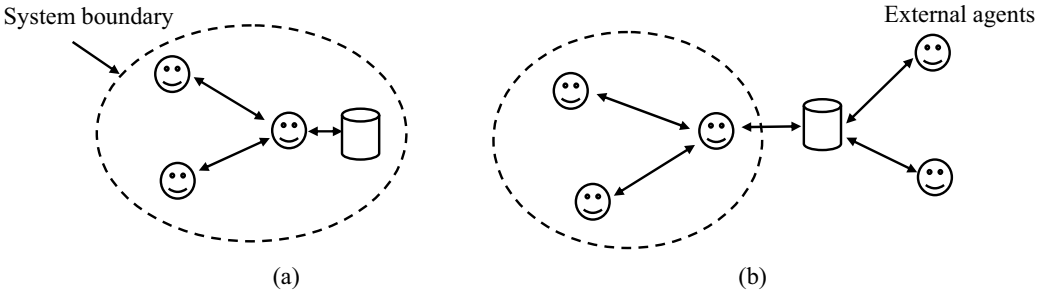



Figure 6.28 – Internal resources (a) and external resources (b)

Resources in a MAS can be identified by investigating the System Task Model or Role Model. These models specify the functionality of the target application, thereby revealing the major resources that accompany with, or are required by, the target

[51] Resources are to be distinguished from infrastructure facilities which provide system-specific services such as naming service or message transport service. The specification of infrastructure facilities is discussed in Step 3 of "*Architecture Design*" activity – "*Specify MAS Infrastructure Facilities*" (Section 6.6.3).

system. For example, the potential resources for the illustrative Product Search MAS are various external databases, web servers and search engines on cars (cf. Section 6.1.4.1). However, more resources may be uncovered during the detailed design of agents' behaviour (i.e. "*Agent Internal Design*" activity – Section 6.4). Thus, the Resource Diagram needs to be iteratively revised.

### *6.3.3.2. Notation of Resource Diagram*

A Resource Diagram should display the following notational elements.

- **Resources**: Each resource is depicted as a rectangle with multiple compartments.
  - The top compartment, marked with keyword **resource**, specifies the resource's name.
  - Each remaining compartment describes the resource from a different dimension. The keyword in each compartment indicates its respective dimension. Some potential dimensions are:
    - ***Resource type***: which defines the category of the resource, e.g. "database", "web server" or "processing application system"; and
    - ***Resource Application Ontology***: which states the name(s) of the Resource Application ontology(ies) that conceptualises the application provided by the resource. The construction of these ontologies is discussed in Step 4 of this phase – "*Extend Ontology Model to include Resource Application ontologies*" (cf. Section 6.3.4).

  MOBMAS does *not* impose the above set of dimensions as a fixed template for resource modelling. These dimensions may vary largely from one type of resource to another, and/or from one development project to another. Therefore, the developer is free to adapt the Resource Diagram to fit the project at hand.

- **Agent classes that wrap around resources** (i.e. "wrapper" agent classes): A resource is typically accessed by agents via a dedicated "wrapper" agent (Jennings and Wooldridge 1995; FIPA 2001a). In the Resource Diagram, a wrapper agent class is represented as a rectangle with keyword **agent class**, followed by its name and roles.

- **Connections between agent classes and resources**: Each resource is linked with its wrapper agent class via an undirected line marked with keyword **wrap**.

An example Resource Diagram for the Product Search MAS is presented in Figure 6.29.

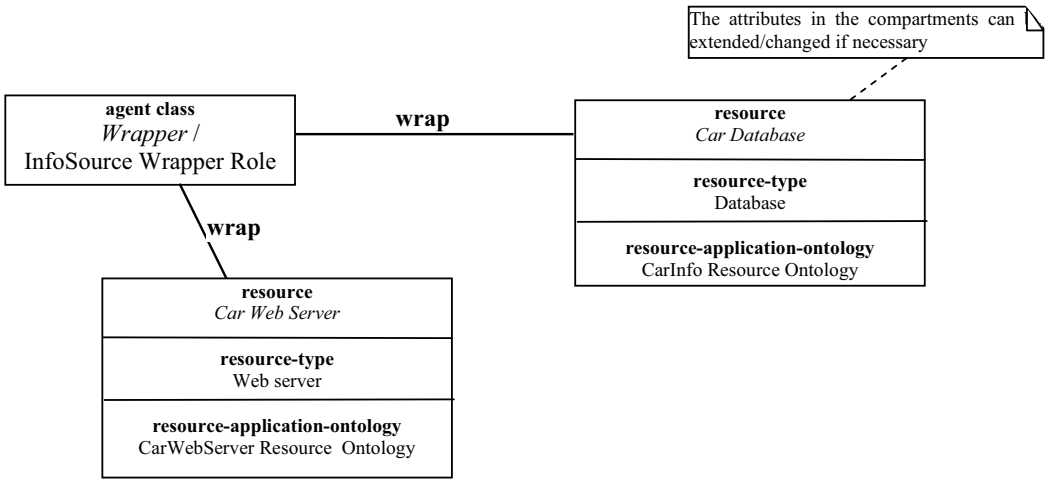



Figure 6.29 – Resource Diagram of Product Search MAS

## *6.3.3.3. Revise Role Model*

Since each resource requires an associated wrapper agent class, the developer should revise the *Role Model* to check if all necessary wrapper role(s) has been identified. Each different type of resource may require a different wrapper role if the responsibilities involved in assessing each resource type are largely different from each other, e.g. "Database Wrapper" role or "Expert System Wrapper" role.

In addition, if the target MAS is open and frequently adds new resources and agents, the developer should consider introducing a "Resource Broker" role. Agents playing this role are responsible for brokering the available resources to interested agents (FIPA 2001a, O'Brien and Nicol 1998). Any newly added wrapper agents (as a result of the addition of new resources) can register its services with the "Resource Broker" agent. A client agent (who may be newly added to the system) can contact the "Resource Broker" to find out which wrapper agents can satisfy its requests. The client agent can then directly interact with the identified wrapper agents for services. The "Resource Broker" agent may also assume the responsibilities of negotiating over the terms and conditions of resource usage, or authorizing the client agents before giving them details of wrapper agents (FIPA 2001a). Note that, as with "Ontology Manager" agent, "Resource Broker"

agent is an application-independent component that is generally provided by the implementation framework (e.g. FIPA-based platforms such as JACK, JADE, FIPA-OS and ZEUS). The developer therefore does not have to design one from scratch, but can fine-tune the provided specification of the "Resource Broker" agent to suit the application at hand. Nevertheless, the Role Model should be extended to include the new "Resource Broker" role.

Figure 6.30 presents the updated Role Diagram for the Product Search MAS (cf. Figure 6.18).

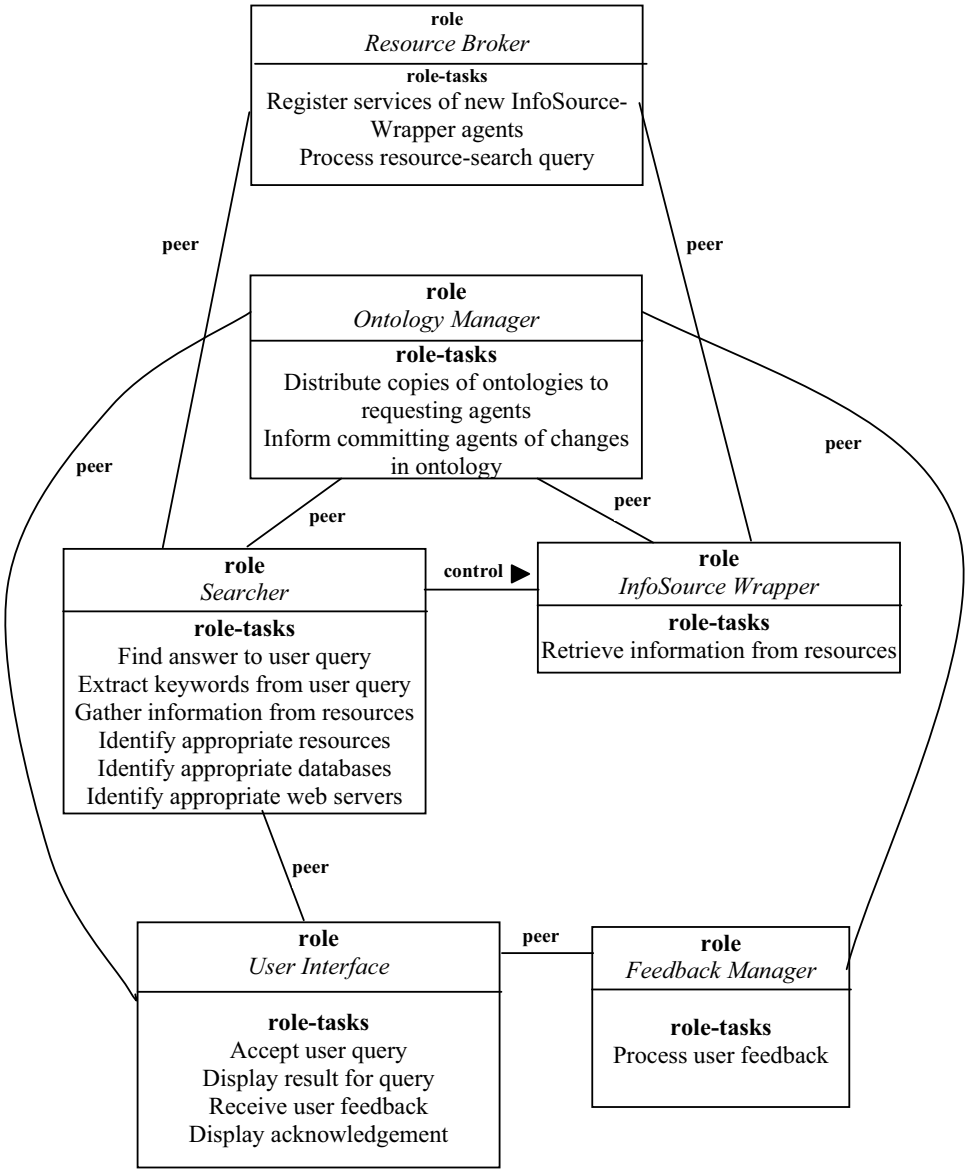



Figure 6.30 – Updated Role Diagram for Product Search MAS (cf. Figure 6.18)

### *6.3.3.4. Update Agent Class Model*

*Agent Relationship Diagram* of MOBMAS' Agent Class Model Kind should be extended to show newly identified resources and their connections with wrapper agent classes.

- Each resource is depicted as a rectangle marked with keyword **«resource»**. For simplicity, the Agent Relationship Diagram only shows the names of the resources without specifying their internal configuration. This information has been captured in the Resource Diagram of Resource Model Kind.
- The connection between resources and their wrapper agent classes is represented as an undirected line marked with keyword **wrap**.

In addition, if the Role Model has been changed, both *Agent Class Diagram* and *Agent Relationship Diagram* need to be updated to show new agent classes and/or new role assignments to existing agent classes.

Figure 6.31 presents the updated Agent Relationship Diagram for the Product Search MAS (cf. Figure 6.27).

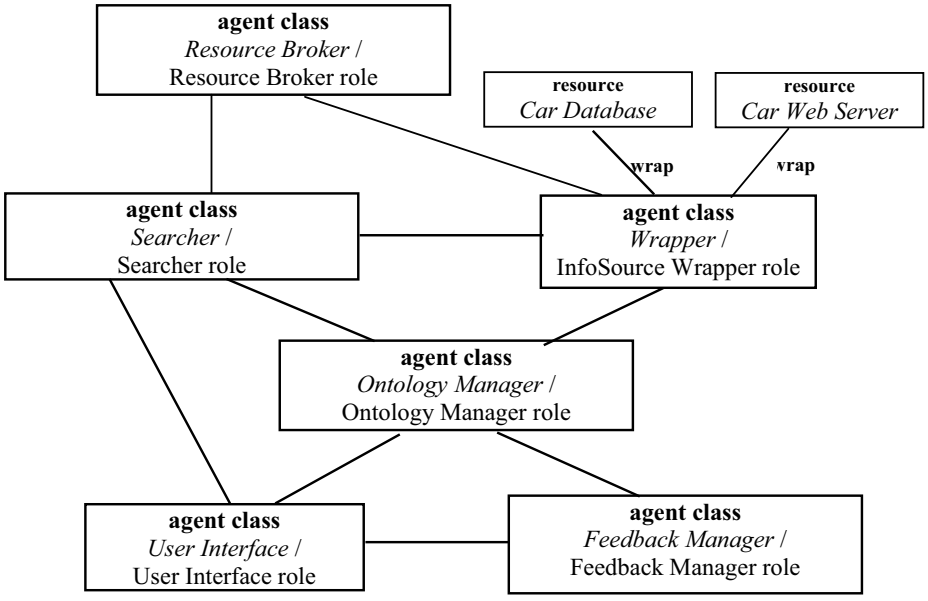



Figure 6.31 – Updated Agent Relationship Diagram for Product Search MAS

## 6.3.4. Step 4 – Extend Ontology Model to Include Resource Application Ontologies (Optional)

If the target MAS contains resources, the developer needs to extend the **Ontology Model** to include Resource Application ontologies that conceptualise the applications provided by these resources. As stated in Section 6.2.4:

- if the resource is a *processing application system* (e.g. a legacy system), its Resource Application ontology should capture all the concepts and relations that conceptualise the domains and tasks/services provided by the resource; and
- if the resource is an *information source* (e.g. a database), its Resource Application ontology should capture all the concepts and relations that conceptualise the information stored in the resource. This Resource Application ontology can be derived from the conceptual schema of the resource, e.g. database schema.

Generally, each resource in a MAS should be conceptualised by a separate Resource Application ontology. The development of Resource Application ontologies is not part of MOBMAS. The developer is referred to other research work on Resource Application ontology development, e.g. Hwang (1999), Pazzaglia and Embury (1998), Mars et al. (1994), Decker et al. (1999) and FIPA (2001b).

The following section discusses the issue of ontology mapping between Resource Application ontologies and MAS Application ontologies.

### *6.3.4.1. Specify ontological mappings between Resource Application ontologies And MAS Application ontologies*

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

- they enable wrapper agents to translate ACL messages (formulated in MAS Application ontologies' vocabulary) into resource-level queries (formulated in Resource Application ontologies' vocabulary), and from resource-level information back to ACL messages; and

- they allow the interoperability between heterogeneous resources. For example, information retrieved from different resources can be integrated using MAS Application ontology as an inter-lingua (cf. Section 2.3.2.1).

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

Figure 6.32 presents an example Resource Application ontology for the Car Database used by the Product Search MAS, named "CarInfo Resource Ontology" (Figure 6.29). The figure also shows the ontological mappings between the CarInfo Resource Ontology and Car MAS Application Ontology (cf. Figure 6.14).

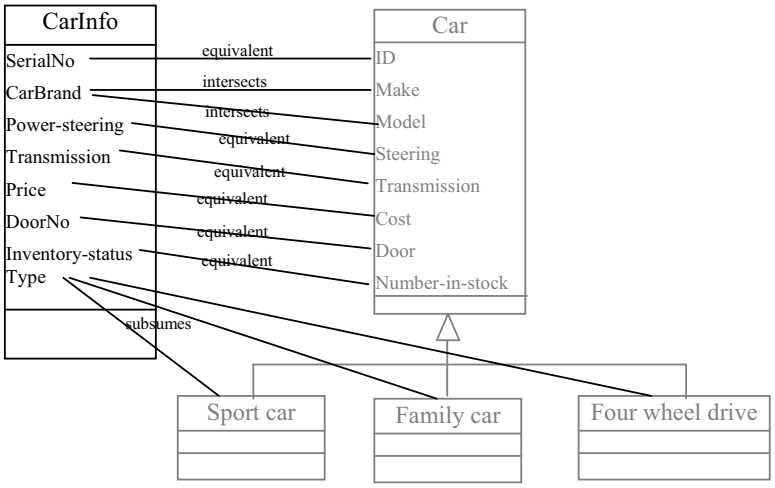



Figure 6.32 – CarInfo Resource Ontology and its mappings to Car MAS Application Ontology

# 6.4. AGENT INTERNAL DESIGN ACTIVITY

This activity of MOBMAS deals with the internal design of each agent class, namely the specification of each agent class' belief conceptualisation, agent goals, events, plan templates and reflexive rules.

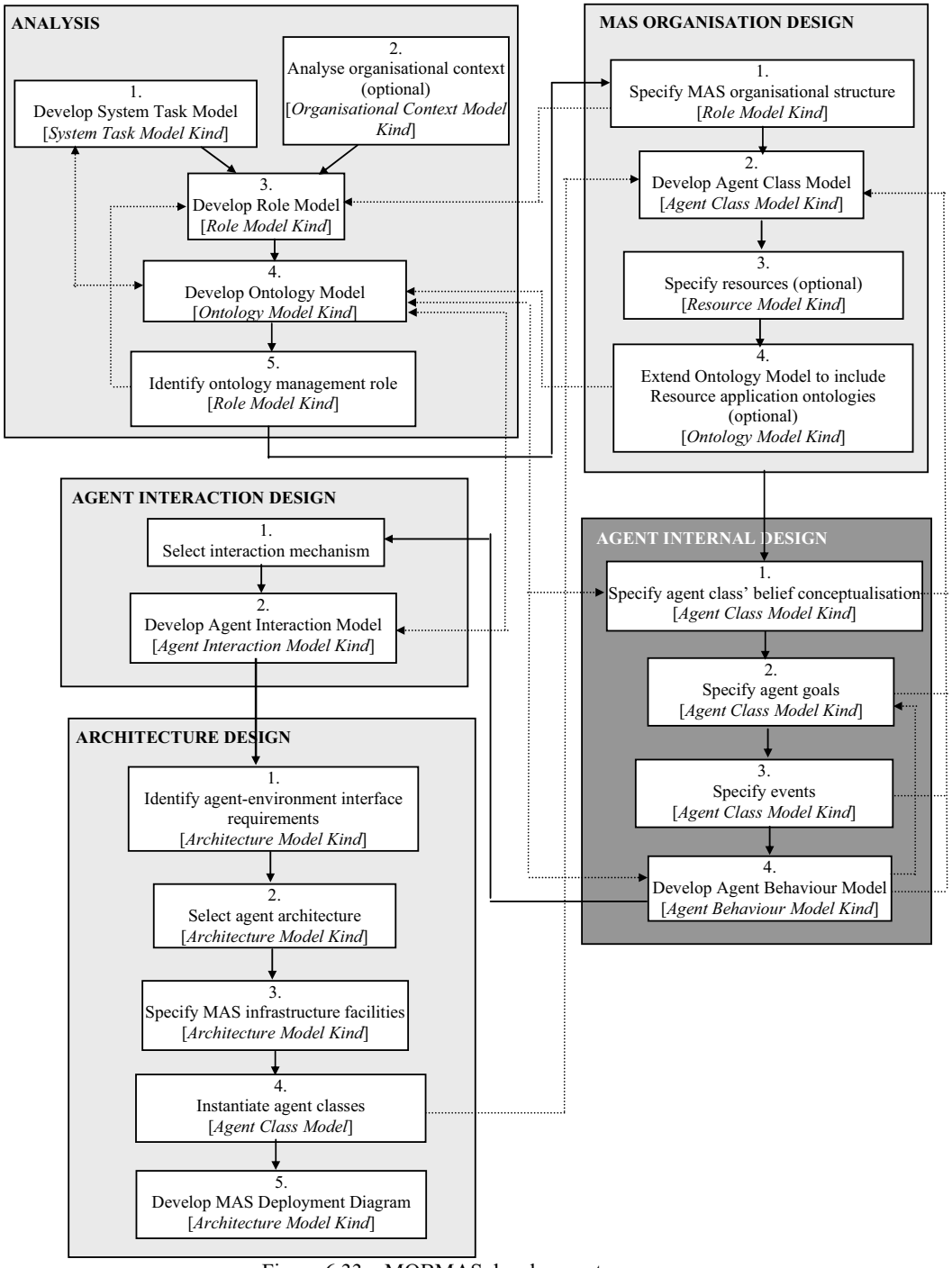



Figure 6.33 – MOBMAS development process

## 6.4.1. Step 1 – Specify Agent Class' Belief Conceptualisation

Agent beliefs refer to the information that an agent holds about the world (Shoham and Cousins 1994; Rao and Georgeff 1995). Agent beliefs exist at two levels of abstraction: *Belief State* and *Belief Conceptualisation* (Kinny and Georgeff 1996; Agent Oriented Software 2004).

- **Belief State**: corresponds to an agent's knowledge about a *particular state* of the world (in the past, present or future) (Shoham 1993) (Figure 6.34). It captures the *run-time facts* about the state of entities that exist in the agent's application (i.e. domains and tasks) and the environment (i.e. resources and other agents).
- **Belief Conceptualisation**: corresponds to the agent's knowledge about the *conceptualisation* of the world, particularly the conceptualisation of the entities referred to in the Belief State (Figure 6.35).

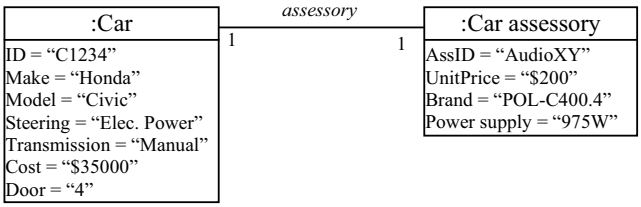



Figure 6.34 – Agent Belief State

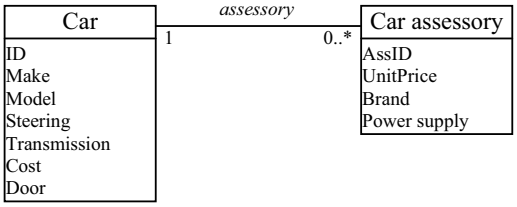



Figure 6.35 – Agent Belief Conceptualisation

At design time, it is only feasible to define the Belief Conceptualisation for each agent class, or more exactly, the *initial start-up* Belief Conceptualisation (because agents in each class may dynamically update their conceptualisation of the world during their lifetime). Belief States are run-time knowledge and therefore can only be populated when the agents of each class interact with the environment to obtain factual information about the world.

### *6.4.1.1. Specify belief conceptualisation of agent classes*

Since an agent class' Belief Conceptualisation stores *conceptual* knowledge of the agent class' world, it should be composed of those **ontologies** which conceptualise the agent class' knowledge of its application (i.e. domains and tasks) and/or wrapped resources' applications[52]. Even though at run-time, agents also maintain beliefs about other agents in the environment, the conceptualisation of the "agent" component is normally imposed by the agent implementation platform and implicitly embedded into the agent coding. For example, JACK (Agent Oriented Software 2004) conceptualises the "agent" component in terms of "name", "capability", "event" and "database", while JADE (Fabio et al. 2004) defines each agent in terms of "name", "address" and "resolver". MOBMAS therefore only investigates the conceptualisation of "application", namely MAS application and resources' applications. As such, the specification of an agent class' Belief Conceptualisation comes down to the determination of **which (part of[53]) MAS Application ontologies and/or Resource Application ontologies the agent class should commit**.

#### 6.4.1.1.a. Identify ontology commitments of agent classes

In general, an agent class needs to commit to a particular (part of) ontology if the agent class' functionality is related to the domain, task or resource that this (part of) ontology conceptualises. In MOBMAS, an agent class' functionality is reflected via its *roles* and *role-tasks*. For example, the "Searcher" agent class in the Product Search MAS plays the "Searcher" role, thereby being responsible for processing car-search queries (Figure 6.30). Accordingly, the "Searcher" agent class should commit to the Car MAS Application Ontology and Query MAS Application Ontology in order to know about car-related concepts (e.g. "Make", "Model" and "Transmission"; Figure 6.14) and querying-related concepts (e.g. "Keyword", "Result list" and "Hit"; Figure 6.15).

---

[52] Only agent classes that directly wrap around the resources need to commit to the corresponding Resource Application ontologies. Other agent classes in the system which wish to use the resources can interact with the wrapper agent classes using ACL messages formulated in MAS Application ontologies (Jennings and Wooldridge 1995; FIPA 2001a).

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

In addition, an agent class' functionality and its required ontologies may also be identified by investigating:

- *resources* wrapped by the agent class (cf. Agent Relationship Diagram of Agent Class Model or Resource Diagram of Resource Model Kind); and/or
- the *acquaintances* of the agent class and other agent classes in the system (cf. Agent Relationship Diagram of Agent Class Model Kind).

For example, the "Wrapper*"* agent class of the illustrative Product Search MAS needs to commit to the CarInfo Resource Ontology and CarWebServer Resource Ontology, since it wraps around the Car Database and Car Web Server resources (Figure 6.29). Besides, the "Wrapper*"* agent class also needs to commit to the Car MAS Application Ontology, because it needs to communicate car-related messages with the "Searcher" agent class (Figure 6.31).

It should be noted that not all ontological commitments of an agent class are apparent at this stage. The developer should proceed to the specification of agent classes' behaviour (i.e. Step 4 of this activity – "*Develop Agent Behaviour Model*"; Section 6.4.4) and agent classes' interactions (i.e. "*Agent Interaction Design*" activity; Section 6.5) in order to get more insight into the knowledge requirements of each agent class. Consequently, the development of each agent class' Belief Conceptualisation is an ongoing process.

At run-time, the initial start-up Belief Conceptualisation specified at design time for each agent class may be dynamically modified. Agents of each class may *extend* their Belief Conceptualisations to include the conceptualisation of new domains, tasks or resources, and/or *update* their Belief Conceptualisations with a new conceptualisation of their existing domains, tasks or resources. The extension of a Belief Conceptualisation normally involves the addition of new (parts of) MAS Application ontologies or Resource Application ontologies into the Belief Conceptualisation, while the update of a Belief Conceptualisation typically requires the modification of the existing MAS Application ontologies or Resource Application ontologies. When an ontology has been modified, all other agents in the MAS that commit to the same ontology should be notified of this modification, so that they can accordingly update their Belief

Conceptualisations. The mechanism of how the modification of ontologies can be propagated across agents at run-time is largely dependent on how ontologies are managed in the MAS. In particular, if an "Ontology Manager" agent class is used to take care of the distribution and maintenance of the ontologies (cf. Section 6.2.5), any agent which wants to modify an ontology can send the modification (or request to modify) to the "Ontology Manager". The "Ontology Manager" then multicasts this modification to all other agents that commit to the modified ontology. Otherwise, if no "Ontology Manager" exists, agents in the MAS will have to communicate the modifications directly to each other, probably in a serial manner. It should be noted that when an existing ontology has been modified, the *Belief State* of an agent committing to that ontology must also be modified to adjust the recorded run-time facts to the new conceptual structure. This modification requires the mapping/revision of knowledge that is not part of MOBMAS.

#### *6.4.1.2. Update Agent Class Model to show belief conceptualisation*

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to specify the ontologies that each agent class commits. The developer only needs to show the *names* of the ontologies in the **belief conceptualisation** compartment (Figure 6.36). Ontologies themselves are modelled in the Ontology Model Kind.

Figure 6.36 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown.

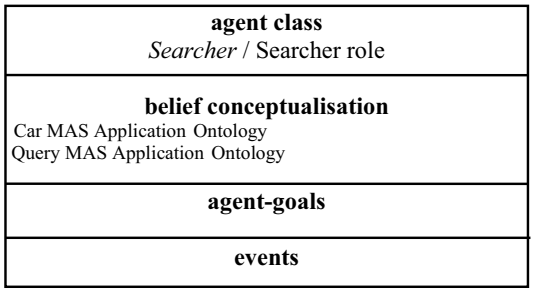



Figure 6.36 – Updated Agent Class Diagram for Product Search MAS ("Searcher" agent class)

## 6.4.2. Step 2 – Specify Agent Goals

An **agent-goal** is a state of the world that an agent class would like to achieve or satisfy (Silva and Lucena 2004; Wooldridge 1999). It signifies the purpose of existence of an

agent class. In MOBMAS, agent-goals are derived directly from role-tasks, since role-tasks describe what the agent class is responsible for fulfilling when playing its roles. The state of the world that each role-task seeks to achieve, satisfy or maintain indicates an agent-goal. For example, role-tasks "Accept user query" and "Display result for query" of "User Interface" agent class (Figure 6.30) indicate two agent-goals, "Incoming user query is accepted" and "Available result for query is displayed" respectively. Meanwhile, two role-tasks "Extract keywords from user query" and "Find answer to user query" of "Searcher" agent class (Figure 6.30) result in two agent-goals "Keywords are extracted from user query" and "Answer is found for user query" respectively.

Note that two different agent classes may have an identical agent-goal if they are mutually in charge of a "*joint task*". Recall that a joint task is one that requires the collective effort of more than one role (cf. Section 6.2.3.1.a). For example, in the illustrative Conference Program Management application, two roles "PC Chair" and "PC Member" are mutually in charge of a joint task "Distribute papers among members" (Figure 6.9). This joint task needs to be mapped to an agent-goal "Papers are distributed among members" in *each* of the two agent classes "PC Chair" and "PC Member". As such, the two agent classes aim to achieve an identical agent-goal. Since all joint tasks have been highlighted with the adornment ***(J)*** in the Role Diagram of Role Model Kind (cf. Section 6.2.3.3), it should be easy at this stage to identify the existence of all the joint tasks in the target system.

### *6.4.2.1. Update Agent Class Model to show agent-goals*

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to show the agent-goals of each agent class in the **agent-goals** compartment. At design time, agent-goals can be specified in an informal natural language. Since agent-goals represent states, they should be defined in the form "*something is achieved/satisfied*", not as a phrase starting with an imperative as in system-tasks or role-tasks.

Figure 6.37 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown (cf. Figure 6.30).

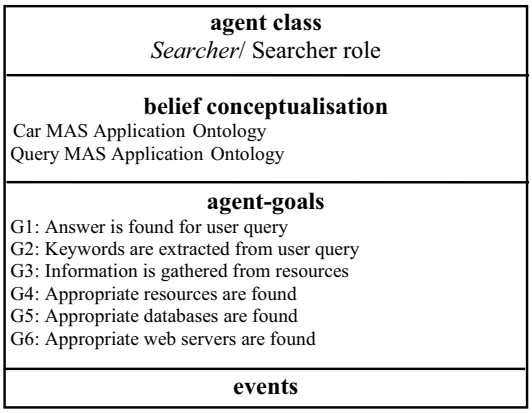



Figure 6.37 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

## *6.4.2.2. Develop Agent Goal Diagram (Optional)*

If a particular agent class is found to pursue multiple agent-goals and these agent-goals are related, an **Agent Goal Diagram** can be developed to capture the relationships among the agent-goals. This diagram is an optional notational component of the **Agent Behaviour Model Kind**, which is examined in Step 3 of this activity – "*Develop Agent Behaviour Model*" (Section 6.4.4).

The Agent Goal Diagram of each agent class may capture the following types of agent-goal relationships.

- *Decomposition relationship*: an agent-goal of an agent class may be decomposed into sub-agent-goals if the role-task from which it is derived has sub-role-tasks. For example, in Figure 6.38, agent-goal "G1-Answer is found for user query" is decomposed into sub-agent-goals "G2-Keywords are extracted from user query" and "G3-Information is gathered from resources", because agent-goal "G1" is derived from role-task "Find answer for user query", which has sub-role-tasks "Extract keywords from user query" and "Gather information from resources" that indicate sub-agent-goals "G2" and "G3" respectively (Figure 6.30). The decomposition of an agent-goal can be an AND-decomposition or an OR-decomposition, depending on the nature of the decomposition of the role-task from which the agent-goal is derived.

- *Goal conflict relationship*: agent-goals of a particular agent class may be in conflict with each other (i.e. intra-agent conflicts[54]). For example, regarding a MAS for library management, agent-goals "User's book extension request is satisfied" and "Reserved book is recalled" of a "Librarian" agent class is in conflict with each other when the book requested for extension is also a reserved book.

  Conflicts between agent-goals may be resulted from the conflicts between system-tasks that these agent-goals aim to achieve[55]. For example, the two conflicting agent-goals "User's book extension request is satisfied" and "Reserved book is recalled" of the library management MAS aim to achieve two conflicting system-tasks "Maintain long borrowing period" and "Maintain regular availability" (cf. Section 6.2.1). The developer should trace through the *Role Diagram* of Role Model Kind and *System Task Diagram* of System Task Model Kind to identify any potential conflicts among agent-goals.

The notation of Agent Goal Diagram is presented below. Many notational elements are reused from the System Task Diagram (cf. Section 6.2.1.1). Each Agent Goal Diagram should be labelled with the name of the respective agent class.

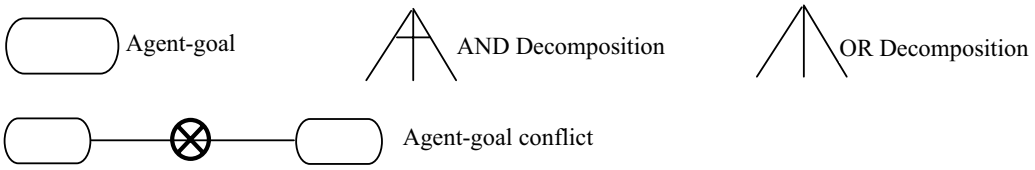



Figure 6.38 illustrates the Agent Goal Diagram for the "Searcher" agent class in the Product Search MAS (cf. Figure 6.37).

[54] The issue of inter-agent conflicts (i.e. conflicts between agent-goals of different agent classes) will be discussed in the "*Agent Interaction Design*" activity (Section 6.5).
[55] Recall that agent-goals are derived from role-tasks, which are in turn mapped from system-tasks (cf. Section 6.2.3.2).

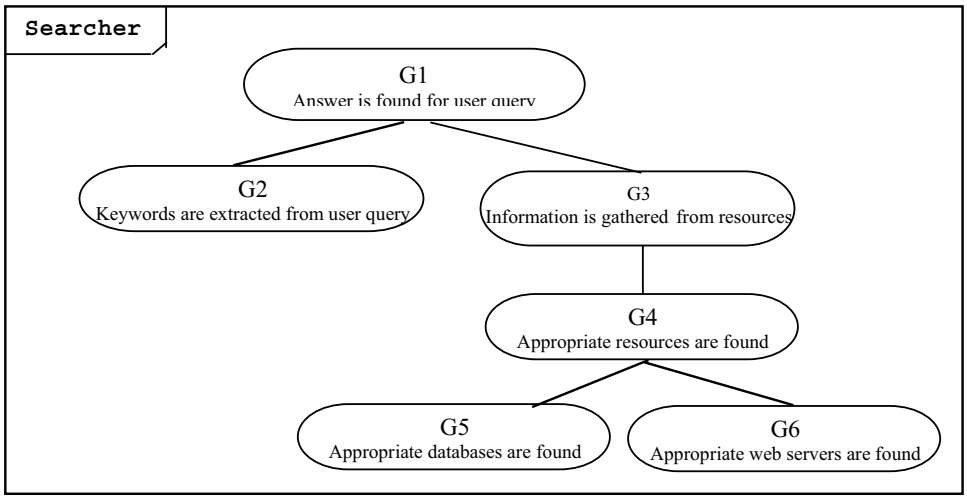



Figure 6.38 – Agent Goal Diagram of "Searcher" agent class of Product Search MAS

## 6.4.3. Step 3 – Specify Events

At run-time, even though agent-goals of a particular agent may be activated[56] proactively by the agent itself (i.e. the agent takes the initiative to pursue the agent-goal[57]), an agent-goal may also be activated by an event coming from the environment. For example, agent-goal "Answer is found for user query" of a "Searcher" agent is activated by a communication message sent by an "User Interface" agent. In addition, an agent's course of actions to achieve a particular agent-goal may also be affected by certain changes in the environment, for example, changes that cause an agent-goal to be "deactivated"[58], or that cause the current course of actions to be no longer applicable. As a result, an agent in a MAS is required to constantly perceive the environment and respond to relevant events within it (Wooldridge 1999).

An **event** is defined as a significant occurrence in the environment that an agent may respond (Winikoff et al. 2001). It may be generated in various ways: by agents via the execution of their actions, by human users via their inputs into the system, or by resources via the execution of their services (Silva and Lucena 2004).

---

[56] The term "activate" is used to mean that the agent starts carrying out some processing to satisfy an agent-goal. Accordingly, an active agent-goal is one that is being actively pursued or satisfied.
[57] For example, agent-goals "Keywords are extracted from user query" and "Appropriate resources are identified" of the "Searcher" agent class are proactive because they can be activated by the "Searcher" agent itself.
[58] The term "deactivate" is used to mean that the agent stops its processing to pursue or satisfy an active agent-goal.

For each agent class, the developer should identify those events that agents of that class need to respond at run-time. These events can typically be derived from stimuli in the environment[59] which:

- *activate agent-goals of the agent class*: For example, agent-goal "User query is accepted" of the "User Interface" agent class is activated by the event "Input of user query". Meanwhile, agent-goal "Information is retrieved from resource" of the "Wrapper" agent class is activated by the event "Incoming message from Searcher agent"; or
- affect the agents' course of actions to fulfil the agent-goals. For example, agent-goal "Answer is found for user query" of the "Searcher" agent class is cancelled if a cancel request is received from the human user (i.e. an event "Input of cancel message from user").

### *6.4.3.1. Update Agent Class Model to show events*

The **Agent Class Diagram** of the **Agent Class Model Kind** should be updated to show the identified events for each agent class in the **events** compartment. Figure 6.37 shows the updated Agent Class Diagram for the illustrative Product Search MAS. Only the specification of the "Searcher" agent class is shown.

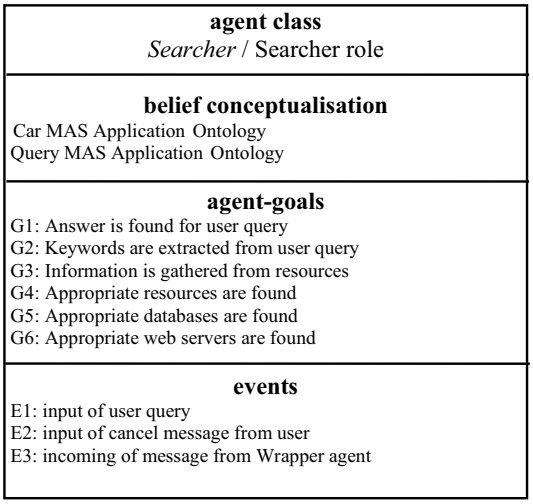



Figure 6.39 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

[59] Only stimuli from the environment are modelled as events because events are meant to reflect agent reactivity, which in turn is defined as the ability to perceive the environment and respond accordingly (Wooldridge 1999). Stimuli that occur from within the agent are not classified as events but as internal processing triggers.

## 6.4.4. Step 4 – Develop Agent Behaviour Model

Agent behaviour refers to the way an agent behaves in order to achieve/satisfy its agent-goals. Two major styles of behaviour have been commonly implemented for agents: "*planning*" and "*reflexive acting*"[60] (Wooldridge and Jennings 1994; Chelberg et al. 2001; Stone and Veloso 2000; Vidal et al. 2001). Planning requires an agent to carry out logical (or at least pseudo-logical) symbolic reasoning to dynamically choose among potential courses of actions for achieving an agent-goal, taking into account the current state of the environment, events occurring during the process of agent-goal achievement, and the failure/success of past actions (Russell and Norvig 2003; Wooldridge and Jennings 1994). Reflexive acting, on the other hand, frees the agent from complex symbolic reasoning by allowing it to behave in a hard-wired situation-action manner, similar to reflexes (Wooldridge 1999; Nareyek 2001). The agent simply follows pre-defined situation-action rules to determine which actions it should execute for achieving/satisfying an agent-goal. These rules, referred to as "reflexive rules" in MOBMAS, can be represented in if-then logic.

Each style of agent behaviour has its strengths and weaknesses (Nareyek 2001; Chelberg et al. 2001). The strength of planning is that it allows an agent to deal with unforeseen situations via reasoning. The developer does not have to predict at design time all the possible situations that the agent may encounter at run-time, or the actions to be executed in these situations. Nevertheless, the weakness with planning is its lack of speed. Every time the situation at hand is different from that anticipated (e.g. when an event occurs), a new plan must be formed, resulting in delays in the fulfilment of the respective agent-goal. Reflexive acting, on the other hand, allows agents to act fast, as the actions to be executed are already defined via reflexive rules. Nevertheless, the problem of reflexive behaviour is that every possible situation at run-time must be known and considered in advance. If the developer fails to foresee a particular event, the reflexive rules dealing with that event will not be defined, resulting in the agents not knowing how to act, or acting in an undesirable manner.

---

[60] Planning and reflexive acting are often referred to as "deliberative" and "reactive" behaviour respectively in the AOSE literature (e.g. Wooldridge and Jennings 1994; Chelberg et al. 2001; Stone and Veloso 2000; Vidal et al. 2001). However, MOBMAS avoids using these terms because "deliberative" may indicate the need for collaboration (which is not unique to planning agents, since reflexive behaviour may also involve collaboration), and "reactive" may refer to the mode of being triggered (which is not unique to reflexive behaviour, since planning agents may also be triggered).

Thus, for *each* agent-goal of each agent class, the developer should determine the style of agent behaviour to be adopted for the agent-goal. Each agent class may adopt different styles of behaviour for different agent-goals. Considering the strengths and weaknesses of each behavioural style, MOBMAS recommends the developer to consider the following characteristics of an agent-goal before making the decision.

- *Complexity of reasoning required by the agent-goal*: If an agent-goal can be satisfied by executing a simple, straightforward, pre-definable sequence of actions (i.e. no complex reasoning is required), the agent class can adopt reflexive behaviour to achieve the agent-goal. Otherwise, if the achievement of the agent-goal requires a complicated, dynamic set of actions where various alternative courses of actions are available, and logical reasoning is needed to decide which course of actions to follow (or which alternative course of actions to switch to when the chosen course of actions fails or when the situation changes), planning would be necessary.
- *Real-time requirement of the agent-goal*: Since planning agents cannot react well in real time, planning behaviour may not be appropriate to agent-goals which need to be achieved in a timely, immediate manner.
- *Predictability of environment situations*: If the developer can foresee each and every situation that may apply during the agent-goal achievement process, and can pre-define situation-action rules for the agent class to achieve the agent-goal, reflexive behaviour is applicable. Otherwise, planning is required to deal with unforeseen situations via logical reasoning.

In the illustrative Product Search MAS, agent-goal "User query is accepted" of the "User Interface" agent class can be achieved with reflexive behaviour, because it pertains to a simple, pre-definable course of actions. This agent-goal also needs to be fulfilled in a timely manner. On the other hand, agent-goal "Answer is found for user query" of the "Searcher" agent class calls for planning behaviour, because the developer cannot pre-define all of the potential courses of actions for achieving the agent-goal. This agent-goal also does not need to be achieved in an immediate manner, thus allowing for planning to take place.

It should be noted that an agent-goal may call for *both* planning and reflexive behaviour. Planning takes care of high-level, long-term reasoning for the agent-goal, while reflexive rules handle decisions about minor plan steps (Nareyek 2001).

MOBMAS models the behaviour of all agent classes in the **Agent Behaviour Model Kind**. This model kind is represented by three notational components:

- **Agent Goal Diagram**: which has been mentioned in Section 6.4.2.2;
- **Agent Plan Template**: which models the planning behaviour of a particular agent class for a particular agent-goal; and
- **Reflexive Rule Specification**: which models the reflexive behaviour of a particular agent class for a particular agent-goal.

Section 6.4.4.1 discusses the development of Agent Plan Templates, while Section 6.4.4.2 deals with Reflexive Rule Specifications.

### *6.4.4.1. Develop Agent Plan Templates*

Normally, agent architectures and implementation platforms that support planning behaviour will offer a "planner" (or a "reasoner" or a "means-end analyser") which takes care of the formation of plans at run-time for agents, e.g. STRIPS (Fikes and Nilsson 1971), IPEM (Ambros-Ingerson and Steel 1988), AUTODRIVE (Wood 1993) and IRMA (Bratman et al. 1988). MOBMAS therefore does not address the issue of plan formation during run-time. Rather, it supports planning by *specifying the pieces of information that are used by planners to formulate plans* (Figure 6.40). This information is captured in the **Agent Plan Template**.

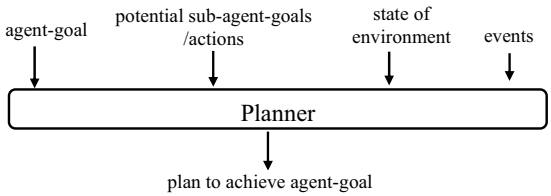



Figure 6.40 – Formation of plans by planner (Wooldridge 2002)

Any agent-goal that requires planning should be associated with an Agent Plan Template. Each Agent Plan Template should specify the following elements.

❖ **Target agent-goal**

This is the agent-goal that a plan derived from the Agent Plan Template aims to achieve. The agent-goal must have been listed in the **agent-goals** compartment of the respective agent class in the Agent Class Diagram of Agent Class Model Kind.

❖ **Triggering event (optional)**

This is the event that activates the target agent-goal, thereby triggering the planning process. This event must have been listed in the **events** compartment of the respective agent class in the Agent Class Diagram of Agent Class Model Kind. Note that an agent-goal may be proactively activated by the agent, in which case no "triggering event" exists.

❖ **A set of sub-agent-goals and/or actions**

To achieve the target agent-goal, the agent may pursue sub-agent-goals and/or actions. Accordingly, the Agent Plan Template should specify a set of sub-agent-goals, or a set of sub-agent-goals and actions, or a set of actions only (Figure 6.41). Each sub-agent-goal (if exist) should be accompanied by its own Agent Plan Template.

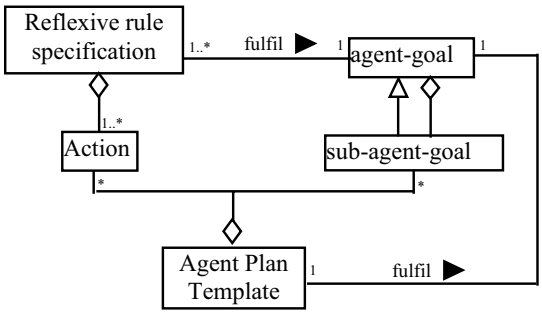



Figure 6.41 – Agent Plan Template and Reflexive Rule Specification (represented in UML)

The need for *sub-agent-goals* in an Agent Plan Template can be identified by investigating the Agent Goal Diagram of the corresponding agent class. This diagram shows the decomposition structure of the agent-goals (cf. Section 6.4.2.2). For example, Agent Plan Template for the agent-goal "Information is gathered from resources" of the "Searcher" agent class (Figure 6.38) should specify a sub-agent-goal "Appropriate resources are found". The Agent Plan Template for this sub-agent-goal should in turn specify two sub-agent-goals "Appropriate databases are found" and "Appropriate web servers are found".

It should be noted that, during the process of Agent Plan Template development, new sub-agent-goals may be discovered which have not been identified in the *Agent Class Diagram* of Agent Class Model Kind and *Agent Goal Diagram* of Agent

Behaviour Model Kind. This indicates the need for iterative development of these three notational components.

An *action* in an Agent Plan Template is an atomic unit of work that an agent class can perform, for example, carrying out some calculation or reasoning, changing the state of an entity in the environment, activating another agent-goal or sending a message to another agent class (Shoham 1993). MOBMAS defines each action in terms of:

- *pre-condition*: which specifies a state that must be true before an action can be executed;
- *post-condition*: which specifies a state resulted from the execution of the action; and
- *action name and parameter list*.

Note that "pre-condition" and "post-condition" constructs are necessary for the selection and sequencing of actions by planners at run-time (Russell and Norvig 2003).

If an action of an agent class involves the sending of an ACL message to another agent class, it is referred to as a "*communicative action*" in MOBMAS. Communicative actions are needed if the achievement of the target agent-goal requires inputs from, and/or provides outputs to, the other agent classes. The developer can refer to the *Agent Relationship Diagram* of Agent Class Model Kind and *Role Diagram* of Role Model Kind to obtain an overview of the agent acquaintances and dependencies[61]. In the case when two or more agent classes aim to achieve an identical agent-goal (cf. Section 6.4.2), they are likely to engage in "distributed planning" to achieve the agent-goal in a joint manner. In such case, the Agent Plan Template of each agent class should contain many communicative actions to allow for distributed planning[62]. Note that the specification of communicative actions may uncover acquaintances that have not been captured in

[61] Inter-agent dependencies are reflected via inter-role authority relationships in Role Diagram of Role Model Kind.

[62] Distributed planning is a complicate research issue by itself. MOBMAS refers the developer to other research work on distributed planning for more techniques, such as desJardins et al. (1999), Conry et al. (1988) and Durfee (1999).

the Agent Relationship Diagram or Role Diagram, thus resulting in a refinement of these two notational components.

The identification of actions for an agent class can be assisted by the investigation of those *MAS Application ontologies* that the agent class commits. These ontologies may define concepts that correspond directly to actions. For example, MAS Application ontology committed by a "Soccer player" agent class defines concepts such as "move", "kick", "turn" and "search-ball", which signify the basic actions of the agent class.

❖ **Events that affect the agent's course of actions**

As mentioned in Section 6.4.3, during the process of agent-goal achievement, certain events may occur that affect an agent's course of actions. For example, a cancel request from a human user will result in the "Searcher" agent class in the Product Search MAS forfeiting its agent-goal "Answer is found for user query".

At design time, it is not always feasible to determine the new course of actions for the agent given the occurrence of these events. This task is delegated to the built-in planners, which use complicated planning algorithms to determine (on the fly) the next best alternative course of actions for the agent (Russell and Norvig 2003). Nevertheless, to facilitate this run-time replanning, MOBMAS recommends the developer to identify the *potential events* that may affect the agent's run-time course of actions.

All events identified here should be listed in the **events** compartment of the corresponding agent class in the Agent Class Diagram of the Agent Class Model Kind. Likewise, all events listed in the **events** compartment of the Agent Class Diagram should be considered in the Agent Plan Template.

❖ **Commitment strategy**

At run-time, the planning process of agents may be largely affected by the way agents are committed to achieving the target agent-goal. For example, agents may persist on pursuing the agent-goal until it is satisfied, or are willing to forfeit the agent-goal after some time. At design time, MOBMAS recommends the developer

to identify the desirable commitment strategy for each agent class with respect to each agent-goal, so as to facilitate the agent class' planning process at run-time. Some example commitment strategies are proposed by Rao and Georgeff (1991).

- "blind or fanatical commitment": the agent will continue pursuing an agent-goal until it believes the agent-goal has actually been achieved.
- "single-minded commitment": the agent will continue pursuing an agent-goal until it believes that either the agent-goal has been achieved, or else that it is no longer possible to achieve the agent-goal.
- "open-minded commitment": the agent will pursue an agent-goal as long as it is still believed possible.

MOBMAS suggests the developer to consider the following factors when selecting the commitment strategy for a particular agent-goal.

- The importance of the agent-goal: For example, agent-goal "Appropriate resources are found" in Figure 6.38 needs to be achieved in order for agent-goal "Information is gathered from resources" to be achieved.
- The existence of events that make the achievement of the agent-goal impossible (e.g. input of user's cancel request): The existence of these events means that the adopted commitment strategy should *not* be blind or fanatical.

❖ **Conflict resolution strategy (optional)**

If certain agent-goals of an agent class are in conflict with each other (i.e. intra-agent conflict), the agent's actions at run-time must be selected in such a way as to minimise or resolve these conflicts. At design time, MOBMAS recommends the developer to identify the desirable *conflict resolution strategy* for the agent class in order to facilitate the agent class' planning process at run-time. There exists a vast amount of work in the area of conflict resolution. Some example strategies for intra-agent conflict resolution[63] are priority conventions (Ioannidis and Sellis 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981) and evidential reasoning (Carver and Lesser 1995).

[63] The issue of inter-agent conflicts (i.e. conflicts between agent-goals of different agent classes) will be discussed in the "*Agent Interaction Design*" activity (Section 6.5).

### 6.4.4.1.a. Notation of Agent Plan Template

Any representation languages for classical planners can be adopted for Agent Plan Template, e.g. STRIPS (Fikes and Nilsson 1971) and ADL (Pednault 1989). However at design time, it is acceptable to represent Agent Plan Templates in an informal natural language. MOBMAS suggests the following schema for Agent Plan Template.

**Initial state**: *state definition*
**Target agent-goal**: *state definition*
**Commitment strategy**: e.g. *blind, single-minded* or *open-minded*
**List of sub-agent-goals** (if any): *state definition and name of the Agent Plan Template that achieves the sub-agent-goal*
**List of actions** (if any): *action name and parameter list*
    **Pre-condition**: *state definition*
    **Post-condition**: *state definition*
**Events:** *list of events*
**Conflict resolution strategy** (if applicable): *strategy name for each agent-goal*

Figure 6.42 – Agent Plan Template

The definition of *states* in Agent Plan Templates (namely the initial states, agent-goals, sub-agent-goals, pre-conditions and post-conditions of actions) may contain variables. Datatypes of these variables should be defined. For example, the initial state of Agent Plan Template for the agent-goal "Information is gathered from resources" is

"keywords**: User query.Keyword** are known",

with "keywords" being a variable and "User query.Keyword" being the datatype of this variable (interpreted as "Keyword" of "User query"). "User query" and "Keyword" are application-specific concepts that are defined in a MAS Application ontology, namely, the Query MAS Application Ontology (Figure 6.15). Note that a datatype may be a "basic" concept that is known to every agent class without being defined in a MAS Application ontology, e.g. Integer or String datatypes.

*Parameters of actions* may be constants or variables. Again, datatypes of variable parameters should be defined. For example, an action in the Agent Plan Template for the agent-goal "Information is gathered from resources" is

"recordResultFromResource(carID**:Car.ID**, carModel**:** **Car.Model**, carStock**:** **Car.Number-in-stock**)".

"CarID", "carModel" and "carStock" are variables while "Car.ID", "Car.Model" and "Car.Number-in-stock" are datatypes of these variables respectively. Note that "Car", "ID" "Model" and "Number-in-stock" are application-specific concepts defined in the Car MAS Application Ontology (Figure 6.14).

Figure 6.43 shows the Agent Plan Template for achieving agent-goal "Information is gathered from resources" of "Searcher" agent class (cf. Figure 6.38).

**Initial state**: keywords:User query.Keyword are known
**Target agent-goal**: Information is gathered from resources
**Commitment Strategy**: single-minded
**Sub-agent-goal**: "Appropriate resources are found", cf. sub-plan X
**Action 1**: sendQueryToWrapper (keywords:User query.Keyword)
**Pre-condition**: Sub-agent-goal "Appropriate resources are found" is achieved successfully and resourceNo: Integer > 0
**Effect**: Event 2 incurs
**Action 2**: recordResultFromResource(carID:Car.ID, carModel: Car.Model, carStock: Car.Number-in-stock)
**Pre-condition**: message:Result list is received from Wrapper agents and carResultArray: [Car]* is empty
**Effect**: carResultArray: [Car]* is populated
**Action 3**: cancelSearch()
**Pre-condition**: Event 1 incurs
**Effect**: agent-goal is forfeited
**Event 1**: input of user's cancel request
**Event 2**: incoming message from Wrapper agents

Figure 6.43 – Agent Plan Template for agent-goal "Information is gathered from resources" of "Searcher" agent class in Product Search MAS

Even though the selection and sequencing of sub-agent-goals and actions for agents at run-time is delegated to built-in planners, if there exists a *tentative* course of sub-agent-goals/actions for achieving a particular agent-goal, this sequence can be captured in an **Agent Plan Diagram**. The notation of Agent Plan Diagram is borrowed from Kinny et al. (1996). It is an extended UML Statechart diagram where each state represents a sub-agent-goal or an action (Figure 6.44). Transition from one state to another occurs when an event happens and/or a condition applies.

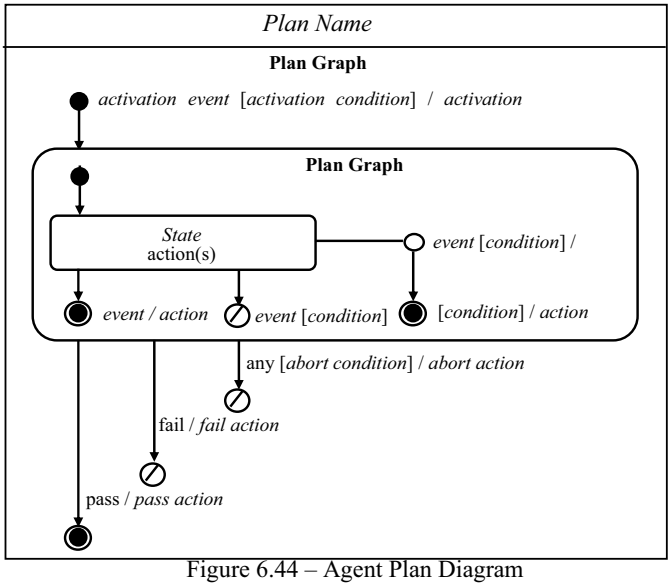



Figure 6.44 – Agent Plan Diagram

Figure 6.45 illustrates the Agent Plan Diagram for the plan that achieves agent-goal "Information is obtained from resources" (cf. Figure 6.43).

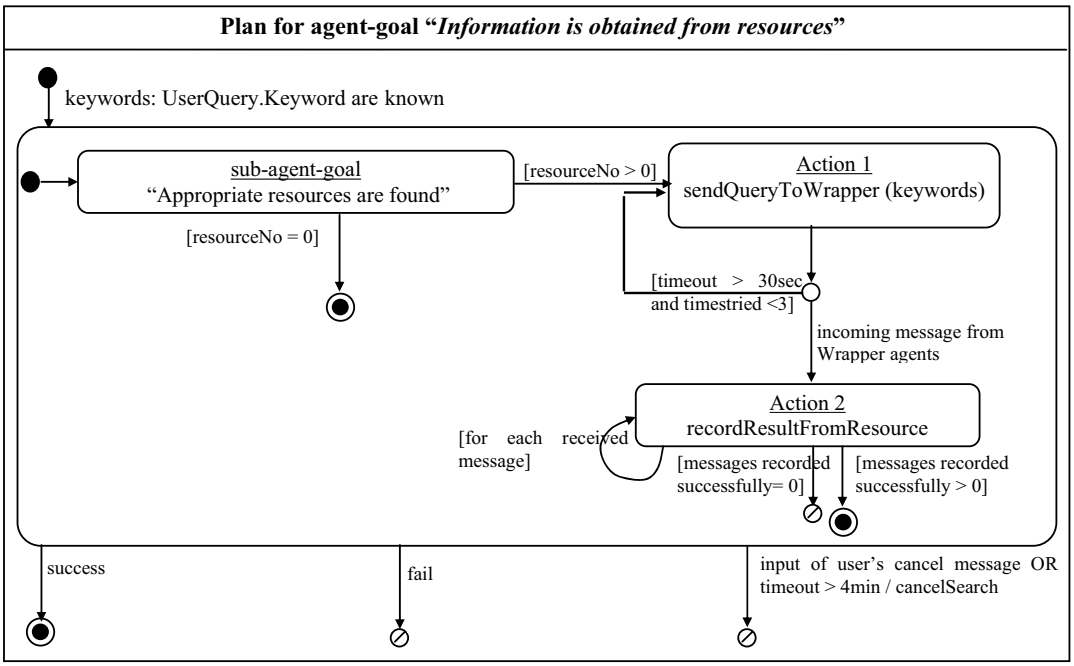



Figure 6.45 – Agent Plan Diagram for agent-goal "Information is gathered from resources" of "Searcher" agent class in Product Search MAS

## *6.4.4.2. Develop Reflexive Rule Specifications*

Reflexive rules are basically (sequences of) "if-then" rules that couple stimuli and/or states of the environment with actions to be executed by an agent class. Each reflexive rule specifies either a complete course of actions to achieve an agent-goal, or a set of actions that works towards the achievement of the agent-goal. Each agent-goal requires one or more Reflexive Rule Specifications, each of which documents the following information.

❖ **Target agent-goal**

This is the agent-goal that the reflexive rule aims to achieve or satisfy. The goal must have been listed in the **agent goals** compartment of the corresponding agent class in the Agent Class Diagram of Agent Class Model Kind.

❖ **Course of actions**

Actions in a reflexive rule are analogous to actions in an Agent Plan Template (cf. Section 6.4.4.1). The only difference is that the sequence of actions is known at design time. The developer is referred to Section 6.4.4.1 for more discussion on actions.

❖ **Events and/or internal processing triggers**[64]

These are the events and/or internal processing triggers that initiate an action in the reflexive rule. The events must have been listed in the **events** compartment of the corresponding agent class in the Agent Class Diagram of the Agent Class Model Kind.

❖ **Guard conditions**

These are the states that make certain actions applicable for execution.

### 6.4.4.2.a. Notation of Reflexive Rule Specification

MOBMAS borrows the notation from *UML Activity diagrams* for Reflexive Rules Specification. Each action is depicted as an UML activity, while events, internal processing triggers and guard conditions are specified alongside the transition flows between activities just as events[65] and guard-conditions in UML Activity diagrams (Object Management Group 2003). However, each Reflexive Rule Specification should the Reflexive Rule Specification of "User Interface" agent class to satisfy its agent-goal "User query is accepted".

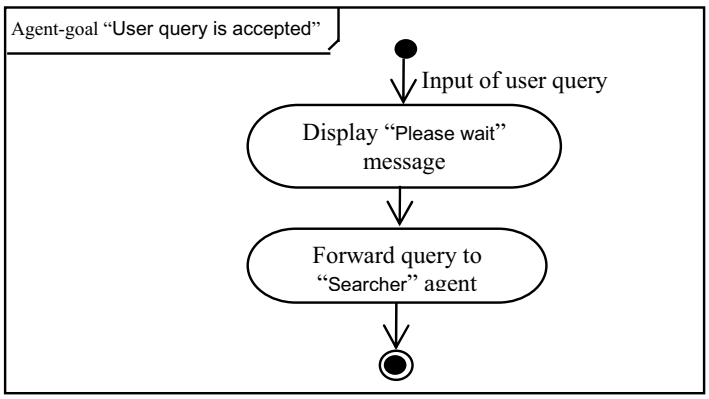



Figure 6.46 – Reactive Rule Specification

[64] Internal processing triggers are stimuli generated from within the agent itself.
[65] For representation simplicity, internal processing triggers are represented in the same way as events.

### *6.4.4.3. Verify Agent Behaviour Model against Ontology Model*

Both Agent Plan Templates and Reflexive Rule Specifications of Agent Behaviour Model Kind contain the definition of *states* and *actions*. The *states* (namely, initial states of Agent Plan Templates, agent-goals, sub-agent-goals, pre-conditions, post-conditions and guard conditions of actions) normally refer to the states of entities/concepts that exist in the agent class' application and wrapped resources' applications. Likewise, parameters of actions in an Agent Plan Template or Reflexive Rule Specification often involve entities/concepts that exist in the agent class' application and wrapped resources' applications. Thus, Agent Plan Templates and Reflexive Rule Specifications can be used to verify the completeness of the content of MAS Application ontologies and Resource Application ontologies. In the other way around, MAS Application ontologies and Resource Application ontologies can be used to verify Agent Plan Templates and Reflexive Rule Specifications.

Specifically, the *datatypes* of all variables specified in Agent Plan Templates and Reflexive Rule Specifications (particularly, in the *states* and *actions' parameters*) should have been defined in a particular MAS Application ontology or Resource Application ontology, with the exception of basic datatypes such as Integer or String. Meanwhile, only concepts defined in the MAS Application ontologies and Resource Application ontologies should be used to define the datatypes of variables in Agent Plan Templates and Reflexive Rule Specifications.

For example, in the following action of the "Searcher" agent class in the Product Search MAS:

"recordResultFromResource(carID**:Car.ID**, carModel**: Car.Model**,
carStock**: Car.Number-in-stock**)",

concepts "Car", "ID", "Model" and "Number-in-stock" should have been defined in the Car MAS Application Ontology (Figure 6.14).

In another example, consider the following action of the "Wrapper" agent class:

"getPrice(carID**: CarProduct.SerialNo**)
*Pre-condition*: productPrice: **CarProduct.Price** = unknown
*Post-condition*: productPrice: **CarProduct.Price** = known".

The action aims to retrieve the price of a particular car product from the Car Database resource of the Product Search MAS (Figure 6.29). Concepts "CarProduct", "SerialNo" and "Price" should have been defined in the CarInfo Resource Ontology (Figure 6.15).

The above guidelines imply the need to *reciprocally* and *iteratively* develop the Agent Behaviour Model and Ontology Model. More specifically, the developer should:

- use the developed ontologies as inputs to define states and actions' parameters for the Agent Plan Templates and Reflexive Rule Specifications; and
- examine the states and actions' parameters of Agent Plan Templates and Reflexive Rule Specifications to determine if any concepts have *not* been defined in the developed ontologies, thereby verifying the content of these ontologies.

### *6.4.4.4. Verify Agent Behaviour Model against Agent Class Model*

Since an agent needs to know about the entities/concepts that are mentioned in its Agent Plan Templates and Reflexive Rule Specifications, the conceptualisation of these entities/concepts should have been defined in its Belief Conceptualisation. Accordingly, the developer should check each agent class' Belief Conceptualisation to confirm that it contains all those *ontologies* which conceptualise the entities/concepts in the Agent Plan Templates and Reflexive Rule Specifications (namely those specified in the *states* and *actions' parameters*).

# 6.5. AGENT INTERACTION DESIGN ACTIVITY

This activity of MOBMAS models the interactions between agent instances by selecting a suitable interaction mechanism for the target MAS, thereafter specifying the patterns of data exchanges between agents given the chosen interaction mechanism.

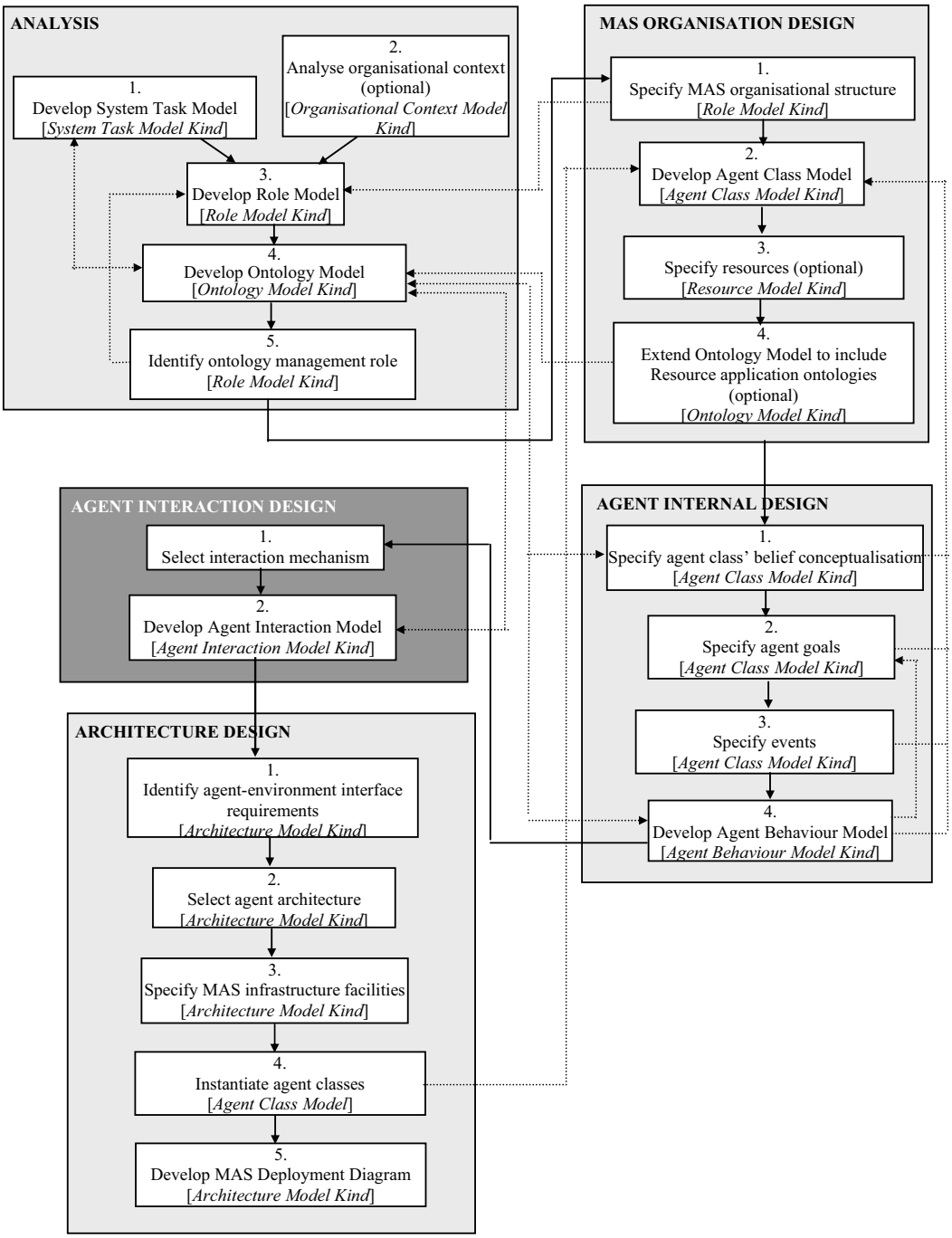



Figure 6.47 – MOBMAS development process

## 6.5.1. Step 1 – Select Interaction Mechanism

### *6.5.1.1. Overview of interaction mechanisms*

"Interaction" refers to the exchange of data amongst agents, either two-way or multi-way (Goldin and Keil 2004). This exchange can be conducted using either of the two interaction mechanisms: direct interaction and indirect interaction (Weyns et al. 2004; Bandini et al. 2004; Goldin and Keil 2004).

- In **direct interaction**, agents exchange data by sending communication messages directly to each other. These messages are typically expressed in *ACL*, such as KQML or FIPA-ACL. The specification of message contents can be made using content languages such as KIF, FIPA-SL, LOOM and Prolog. Generally, the exchanges of ACL messages between agents need to conform to "*interaction protocols*", which are *allowed* communication patterns between the interacting agents. They specify the possible sequences of exchanged messages and the constraints on the content of these messages (Odell et al. 2000b). Some examples of interaction protocols are FIPA Contract Net, Simulated Trading, Request, Query and Subscribe Protocols (FIPA 2002).

- In **indirect interaction**, agents exchange data indirectly through some kind of communication abstraction. A well-known indirect interaction mechanism is *tuplespace interaction*, where agents interact by inserting "*tuples*"[66] into, and removing them from, a shared tuplespace in an associative way. Recently, this mechanism has progressed into a more advanced form, *tuple-centre interaction*. A tuple-centre is no longer a mere communication channel like a tuplespace, but a *programmable reactive interaction medium*, which is equipped with computational capacity to react to events (Omicini and Denti 2001; Ciancarini et al. 1999). Example middleware systems or models built upon the tuple-centre interaction mechanism are TuCSoN (Cremonini et al. 1999), LuCe (Denti and Omicini. 2001), $\mathcal{ACLT}$ (Omicini et al. 1995), LIME (Picco et al. 1999), Berlinda (Tolkdorf 1997) and MARS-X (Cabri et al. 2000). Other indirect interaction mechanisms for agents are

[66] Each tuple is an ordered collection of heterogeneous information chunks.

*stigmergy* and *spatially founded* interaction. The term "stigmergy" is used by biologists to refer to the coordination of insects through "pheromone" – a chemical substance deposited into the environment and sensed by the individual insects. Agents adopt this indirect interaction mechanism by generating and detecting "artificial pheromone objects" in an artificial dissipation environment (Valckenaers et al. 2002; Klugl 2001). Spatially founded interaction mechanism, meanwhile, is related strongly to the spatial structure of the environment. Co-Fields (Mamei and Zambonelli 2004) is an example interaction model that employs this mechanism. It aims to support agents' "motion" coordination by representing the agents' operational environment into "computational fields". These fields are a sort of spatial data structures that can be propagated across the environment by some network infrastructure. An agent can make its movement decisions by examining the shape of the computational fields, just as a physical mass moves in accord to the gravitational field. The movement of this agent may induce changes to the shape of some specific fields, which in turn affect the movement of other agents.

### *6.5.1.2. Select interaction mechanism*

The developer should decide which interaction mechanism is best suited to the target MAS. MOBMAS supports this decision by presenting a comparison of the different mechanisms, thereafter providing recommendations on when to use which mechanism.

The direct interaction mechanism is different from the indirect mechanism in terms of the following key aspects (Goldin and Keil 2004; Weyns et al. 2004).

- *Early binding of recipient*: The direct interaction mechanism requires an agent to know its target interaction partner before the interaction can take place. Meanwhile, the indirect mechanism allows the identity of the target partner to be determined after the sending of tuples/pheromones/fields.
- *Name and location coupling*: With the direct interaction mechanism, the interacting agents have to know about each other's configuration and location before the interaction can take place, while with the indirect mechanism, the interacting agents do not have to hold this knowledge.
- *Time coupling*: The direct exchange of messages between agents in the direct interaction mechanism requires the interacting agents to exist and be available to

communicate at the same time. On the other hand, in the indirect mechanism, there can be a delay between the sending of a tuple/pheromone/field and its observation.

Among the various mechanisms of indirect interaction listed in Section 6.5.1.1, stigmergy and spatially-founded mechanisms are very limited in their applicability. Stigmergy coordination is mainly suited to those domains that involve some kind of attraction to specific locations, or attraction to move in a specific direction (e.g. synthetic ecosystem, network routing) (Biegel 2002; Bonabeau et al. 1998; Brueckner 2000). Likewise, spatially-founded mechanism should only be considered when space is an essential factor in agent interactions (e.g. motion coordination). Thus, MOBMAS focuses only on the *tuplespace*/*tuple-centre* indirect interaction mechanism. In the following section, a comparison between the direct interaction mechanism via ACL and the indirect mechanism via tuplespace/tuple-centre is presented.

#### 6.5.1.2.a. Comparison between direct interaction mechanism and tuplespace/tuple-centre indirect interaction mechanism

Most of the existing tuplespace and tuple-centre frameworks are based upon the LINDA model (Papadopoulos 2001). They therefore use Linda-like communication primitives such as out, in, rd, inp and rdp to specify communication messages (Omicini and Zambonelli 1999). These primitives offer a very low level of semantics for the communication language. Meanwhile, the direct interaction mechanism employs complicated ACLs that offer a large range of expressive *speech-act performatives*, e.g. inform, tell, query-if, ask-if, ask-all, advertise, achieve, refuse and failure (FIPA. n.d.a).

With regard to popularity, the direct interaction mechanism is far more commonly used by the existing MAS development projects than the tuplespace/tuple-centre mechanism (Bergenti and Ricci 2002). The main reasons for its popularity are that:

- interaction protocols have been widely adopted in the OO paradigm. The developer can therefore borrow techniques from OO modelling to specify agent interaction protocols; and
- numerous interaction protocol patterns have been catalogued for reuse, e.g. those developed by FIPA (FIPA 2002).

Nevertheless, with its programmable behaviour, the tuple-centre mechanism offers various advantages over the direct interaction mechanism.

- *Decoupling of computation and coordination concerns*: The behaviour of the tuple-centre can be programmed in such a way as to embody any rules that govern the agent coordination[67] (referred to as "coordination rules"). Agents can thus be freed of the load of coordination and focus on their individual computation during the interaction process (Cremonini et al. 1999; Omicini and Denti 2001; Bergenti and Ricci 2002; Ciancarini et al. 2000). In particular, the interacting agents can simply be concerned with providing inputs to, and obtaining outputs from, the interaction process. The tuple-centre can be programmed to:
  - ensure that all the coordination rules governing the interaction process are satisfied; and
  - carry out some (low level) processing to assist in the fulfilment of the target coordinating task, thereby taking some processing load off the interacting agents.

  On the other hand, in the direct interaction mechanism, computation and coordination concerns are merged into the design of agents (Bergenti and Ricci 2002; Ciancarini et al. 2000). The interacting agents cannot abstract away from the coordination concerns, but have to embed the coordination rules into their interaction protocols, which are in turn embedded in their codes (Omicini and Denti 2001). This may be very difficult to implement if the number of interacting agent classes is large or the coordination rules are complex.

- *Support for modification*: With the tuple-centre interaction mechanism, any changes to the coordination rules may only lead to changes in the behaviour of the tuple-centre. The direct interaction mechanism, on the other hand, may require an update of the design of all interacting agents (Ciancarini et al. 2000; Omicini and Denti 2001).

- *Security control*: In MAS, security-related concerns include authentication (i.e. how agents are identified) and authorisation (i.e. what are agents allowed to do) (Cremonini et al. 1999). With the tuple-centre interaction mechanism, authentication

[67] "Coordination" refers to the management of interactions (Nwana et al. 1996; Wegner 1996)

and authorisation activities can be delegated to the tuple-centre. Meanwhile, with the direct interaction mechanism, the direct exchange of messages between agents will mean that the individual agents need to implement their own authentication and authorisation activities.

Nevertheless, the tuple-centre interaction mechanism exhibits a strong centralised design due to the tuple-centre's essential role in the interaction process (Bergenti and Ricci 2002). This centralisation may seriously compromise the robustness of the system, e.g. when the tuple-centre experiences downtime. The direct interaction mechanism, in contrast, spreads the locus of control over the interacting agents, hence avoiding the robustness problem if a particular agent goes down[68].

In summary, considering its popularity and reusability, MOBMAS recommends the direct interaction mechanism to most MASs. However, in various situations, the tuplespace or tuple-centre interaction mechanism is perceived more appropriate than the direct interaction mechanism, namely,

- *when the MAS environment is open and dynamic*: With its support for late binding of recipient, name decoupling, location decoupling and time decoupling, the tuplespace/tuple-centre interaction mechanism is able to facilitate flexible and robust interaction in open and dynamic systems (Zambonelli et al. 2001b; Bergenti and Ricci 2002). In addition, by embedding the coordination rules into the tuple-centre, the tuple-centre interaction mechanism can help preventing illegitimate or self-interested behaviour in newly added agents; or

- *when many agent classes aim to achieve identical agent-goals* (cf. Section 6.4.2): The tuple-centre interaction mechanism is particularly suited to the interactions amongst agents of these classes because:
  - the joint achievement of the identical agent-goals often requires many coordination rules to be enforced on the interacting agents. The tuple-centre can take charge of enforcing these rules; and

[68] In this case, other instances of the same agent class may serve as an substitute for the problematic agent.

- the tuple-centre can carry out some processing to assist in the achievement of the joint agent-goals.

For example, consider the agent-goal "Papers are distributed among members" jointly achieved by "PC Chair" and "PC Member" agent classes (cf. Figure 6.7). Some example coordination rules governing the interactions between agents of these classes are:

- "Each paper must be distributed to a required number of members";
- "Each member must collect a required number of papers";
- "Each member must not collect the same paper twice"; and
- "Members can only start collecting papers after all other members have viewed the Title List of all papers"[69].

All of these coordination rules can be enforced by the tuple-centre, which is programmed in such a way as to check and control the tuples sent from the "PC Chair" and "PC Member" agents (cf. Figure 6.57). The tuple-centre can also carry out some processing to identify which "PC Member" agent has not collected the required number of papers, thereby posting reminder tuples to these agents. This processing helps to enforce the rule "Each member must collect a required number of papers".

## 6.5.2. Step 2 – Develop Agent Interaction Model

The Agent Interaction Model Kind of MOBMAS captures the patterns of data exchanges between agent instances when they interact using the chosen interaction mechanism. Section 6.5.2.1 discusses the specification of Agent Interaction Model Kind for the direct interaction mechanism, while Section 6.5.3.2 examines the Agent Interaction Model Kind for the tuplespace/tuple-centre mechanism.

### *6.5.2.1. Develop Agent Interaction Model for Direct Interaction Mechanism*

With the direct interaction mechanism, the exchanges of ACL messages between agents need to be governed by *interaction protocols*, each of which defines an allowed communication pattern between the interacting agents during a particular conversation

[69] This rule is needed to ensure fairness in paper selection by members.

(Odell et al. 2000b). Accordingly, the task of developing Agent Interaction Model for the direct interaction mechanism includes the task of defining interaction protocols for the agent conversations. Agent Interaction Model Kind of this mechanism is represented by a set of ***Interaction Protocol Diagrams***, each graphically describing an interaction protocol for an inter-agent conversation.

### 6.5.2.1.a. Define interaction protocols

To define interaction protocols for the target MAS, the developer should examine the *Agent Behaviour Model*, namely the Agent Plan Templates and Reflexive Rule Specifications of each agent class in the system.

- Each *communicative action* in the Agent Plan Templates and Reflexive Rule Specifications indicates a message being sent from one agent to another; and
- Any *event* that is specified in an Agent Plan Template or that triggers a Reflexive Rule Specification may itself be a message sent from one agent to another.

As recommended by FIPA (2001c), each ACL message should be defined in terms of:

- **Predecessor**: which defines the order by which a message is sent in relation to its concurrent messages (if ordering is important). For example, "1/inform(…)" denotes an *inform* message that has to be sent first, before any other concurrent messages are sent;
- **Guard-condition**: which defines the condition in which a message is applicable to be sent, e.g. [CarMake = unknown];
- **Sequence-expression**: which specifies the constraint of message sending. For example, "n..m" denotes that a message is sent *n* up to *m* times, while "broadcast" denotes a broadcast sending of a message.
- **Performative**: which defines the type of *speech-act* that the sender wishes to perform, e.g. query-if, inform, refuse, failure; and
- **Arguments**: which are pieces of information that a message conveys. Each argument can be a constant or a variable. For variable arguments, their datatypes must be defined. For example, in the following ACL messages

  "query-if (carCost:**Car.Cost** < custPrice: **UserQuery.Price**)" and

  "inform (carStockNo: **Car.Number-in-stock**)",

variables "carCost", "custPrice" and "carStockNo" are of types "Car.Cost", "UserQuery.Price" and "Car.Number-in-stock" respectively. Note that concepts "Car", "Cost", "Number-in-stock", "User query" and "Price" are application-specific concepts that are defined in the MAS Application ontologies which are shared between the communicating agents (in this case, Car MAS Application Ontology and Query MAS Application Ontology).

MOBMAS recommends the developer to *reuse* and *customize* the patterns of interaction protocols provided by the various libraries and catalogues. FIPA (2002), for example, offers a large range of interaction protocol patterns, supporting both cooperative-style interaction (e.g. Contract Net, Query, Request and Brokering) and negotiation-style interaction (e.g. English/Dutch Auction). If necessary, a complex interaction protocol can be built from a few basic pre-defined protocol definitions.

It should be noted that when developing interaction protocols for agents that aim to achieve an identical agent-goal (cf. Section 6.4.2), the developer should ensure that all the *coordination rules* governing the successful achievement of the agent-goal are embedded in either the *interaction protocols*, or in the *agent class' individual behaviour*, which is modelled in the Agent Behaviour Model Kind.

For example, consider the agent-goal "Papers are distributed among members" jointly achieved by the "PC Chair" and "PC Member" agent classes (cf. Figure 6.7). The rule "Members can only start collecting papers after all other members have viewed the Title List of all papers" (cf. Section 6.5.1.2) can be embedded in the definition of the interaction protocol between the "PC Chair" and "PC Member" agents. Specifically, the protocol can specify the pattern of exchanged messages in such a way that the "PC Member*"* agent is not allowed to send a "Paper-Request" message to a "PC Chair" agent until the "PC Chair" agent has sent a "Title-List" message to all "PC Member*"* agents. In other words, the sending of "Paper-Request" messages is sequenced after the sending of "Title-List" messages (Figure 6.56).

On the other hand, the rule "Each member must not collect the same paper twice" can be enforced by defining the behaviour of the "PC Chair" agent. Specifically, a "PC Chair" agent should not approve the paper request from a "PC Member" agent if that agent has

previously requested the same paper. This behaviour can be defined in the "PC Chair" agent's Agent Plan Template or Reflexive Rule Specification. As a result, when defining interaction protocols for agent classes that achieve identical agent-goals, the developer may have to revise the Agent Behaviour Model.

In addition, the developer should also identify the potential conflicts between the interacting agent classes and define the interaction protocols in such a way as to deal with these conflicts (note that the issue of intra-agent conflicts has been discussed in the "*Agent Internal Design*" activity; Section 6.4). In the context of MOBMAS, two agent classes may be in conflict if their agent-goals have been derived from conflicting system-tasks[70]. The developer should therefore trace through the *Agent Class Model*, *Role Model* and *System Task Model* to identify any potential conflicts between agent classes.

To date, there is a vast amount of research work in the area of conflict resolution. Some example conflict resolution strategies are negotiation (Sycara 1988), voting (Ephrati and Rosenschein 1991), priority conventions (Ioannidis and Sellis 1989), assumption surfacing (Mason and Johnson 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981), evidential reasoning (Carver and Lesser 1995), and standardization and social rules (Shoham and Tennenholtax 1992).

Moreover, the developer should specify the synchronisation mode engaged by each agent in each interaction protocol. Two common modes of agent synchronisation are (Mishra and Xie 2003; Weyns and Holvoet, 2003):

- *synchronous interaction*: i.e. when an agent yields a thread of control after sending a message (i.e. wait semantics). In other words, the agent will wait for a reply ACL message from its interaction partner after it sends a message to that agent; or
- *asynchronous interaction*: i.e. when an agent sends messages without yielding any control. The agent basically continues with its processing right after the sending of messages.

Generally, asynchronous interaction is the most common mode of synchronisation used in agent interaction (Odell et al. 2000a).

---

[70] Recall that agent-goals are derived from role-tasks, which are in turn derived from system-tasks (cf. Section 6.2.3.2).

### 6.5.2.1.b. Notation of Interaction Protocol Diagrams

The Agent Interaction Model Kind for the direct interaction mechanism is represented by a set of Each ***Interaction Protocol Diagrams*****,** of the Agent Interaction Model Kind each graphically describesing an interaction protocol for an inter-agent conversation. MOBMAS reuses the notation of AUML Sequence Diagram for the Interaction Protocol Diagram (Odell and Huget 2003; Odell et al. 2000a). Major notational rules of the AUML Interaction Protocol Diagram are presented below. The developer can refer to Odell and Huget (2003) and Odell et al. (2000) for a more extensive documentation[71].

- Each lifeline represents an agent and the role(s) that the agent plays during the conversation. The rectangle at the top of each lifeline should be specified in the format

  **:**agent-class-name /
  role-name1, role-name2…

  The colon in front of the agent-class-name signifies an instance of the agent class.

- Arrows represent the sending of messages between agents. An "asynchronous" message is drawn as ⟶ , while a "synchronous" message is shown as ⟶ . If there is a delay between the time a message is sent and the time it is received (e.g. "mobile communication"), the message arrow is drawn as ↘.
- If an agent belongs to a class that plays multiple roles (cf. Section 6.3.2.1.a), the dynamics of the agent's role-playing behaviour during the conversation should be modelled.
  - If the agent's role-playing behaviour is *static* (i.e. if the agent plays some particular roles statically throughout its lifetime), the rectangular box at the top of the lifeline should specify *all* of the agent's roles (Bauer 2001b) (Figure 6.48a).
  - If the agent's role-playing behaviour is *dynamic* (i.e. if the agent dynamically changes its active role(s) from one time to another), the rectangular box should *only* specify the name of the active role. If a change of role occurs during the

[71] At the time of this research, the notation of AUML Sequence Diagram is in its preliminary version. The developer should check for updated versions (if exist) at http://www. auml.org.

conversation, this change can be represented as a **«role change»** stereotyped arrow (Bauer, B. 2001b) (Figure 6.48b).

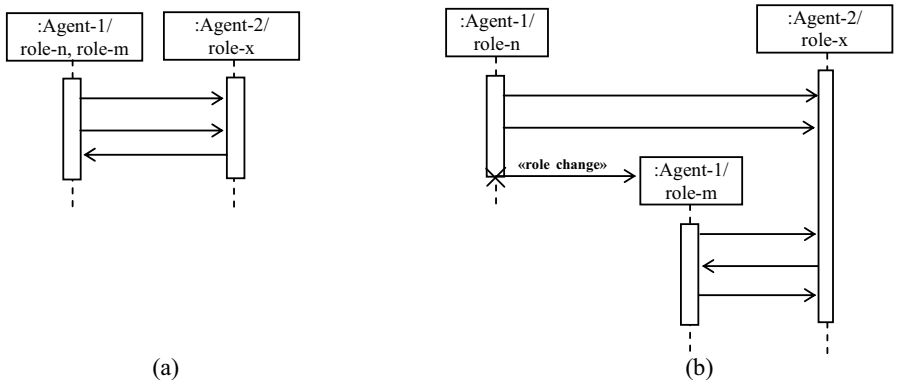



Figure 6.48 – AUML notation for the dynamics of agents' role-playing behaviour (Bauer 2001b)

- Concurrent threads of communication are modelled as shown in Figure 6.49. Figure 6.49a indicates that all messages are sent concurrently. Figure 6.49b includes a decision box to indicate that a decision will be made regarding which messages (zero or more) will be sent (i.e. *inclusive OR*). Figure 6.49c describes an *exclusive OR* (i.e. exactly one message will be sent).

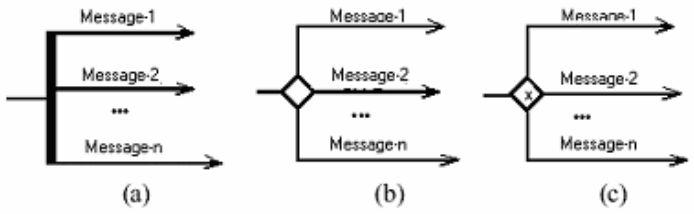



Figure 6.49 – AUML notation for concurrent threads of interaction

- Concurrent threads of processing in the recipient agent are modelled as either a *split of lifeline* (Figure 6.50a) or *activation bars* appearing on top of each other (Figure 6.50b).

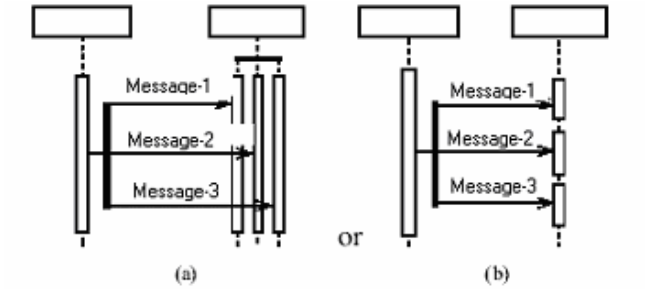



Figure 6.50 – AUML notation for concurrent threads of processing

Figure 6.51 presents an example Interaction Protocol Diagram for the illustrative Product Search MAS. The diagram describes a conversation between a "Searcher" agent and a "Wrapper" agent (cf. Figure 6.33 and Figure 6.41).

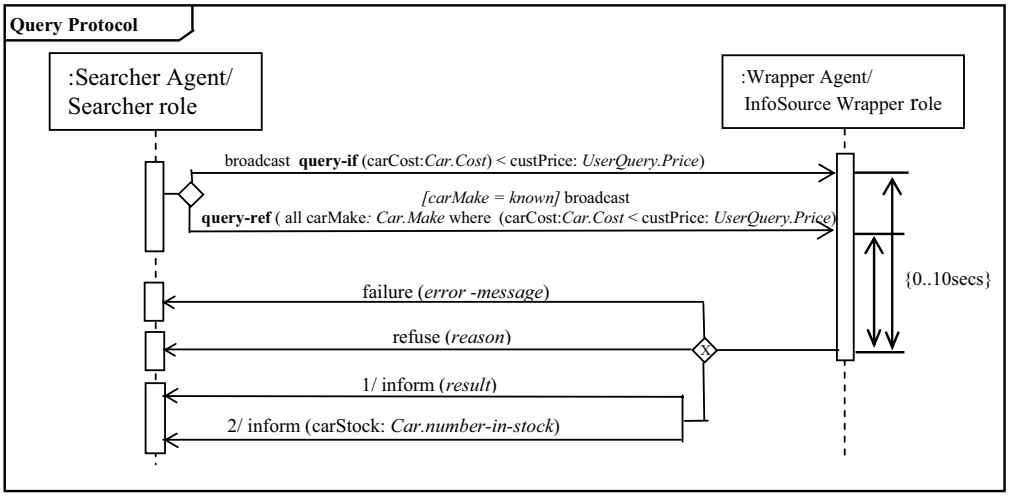



Figure 6.51 – Interaction Protocol Diagram for Product Search MAS

## 6.5.2.1.c. Update Agent Class Model and Role Model

The *Agent Relationship Diagram* of the *Agent Class Model Kind* should be updated to show:

- any new acquaintances between agent classes that have not been identified; and
- various descriptive information about *each* acquaintance, namely:
  - the *identity of the Interaction Protocol Diagrams* that govern the conversations between the acquainted agents; and
  - the *MAS Application ontology(ies)* which governs the semantics of the messages exchanged during the conversation and which must be shared between the communicating agents[72].

  This descriptive information is modelled as *UML notes* attached to each acquaintance (cf. Figure 6.22).

Figure 6.52 presents the updated Agent Relationship Diagram for the Product Search MAS (cf. Figure 6.28).

[72] That is, this MAS Application ontology must exist in both agents' Belief Conceptualisation.

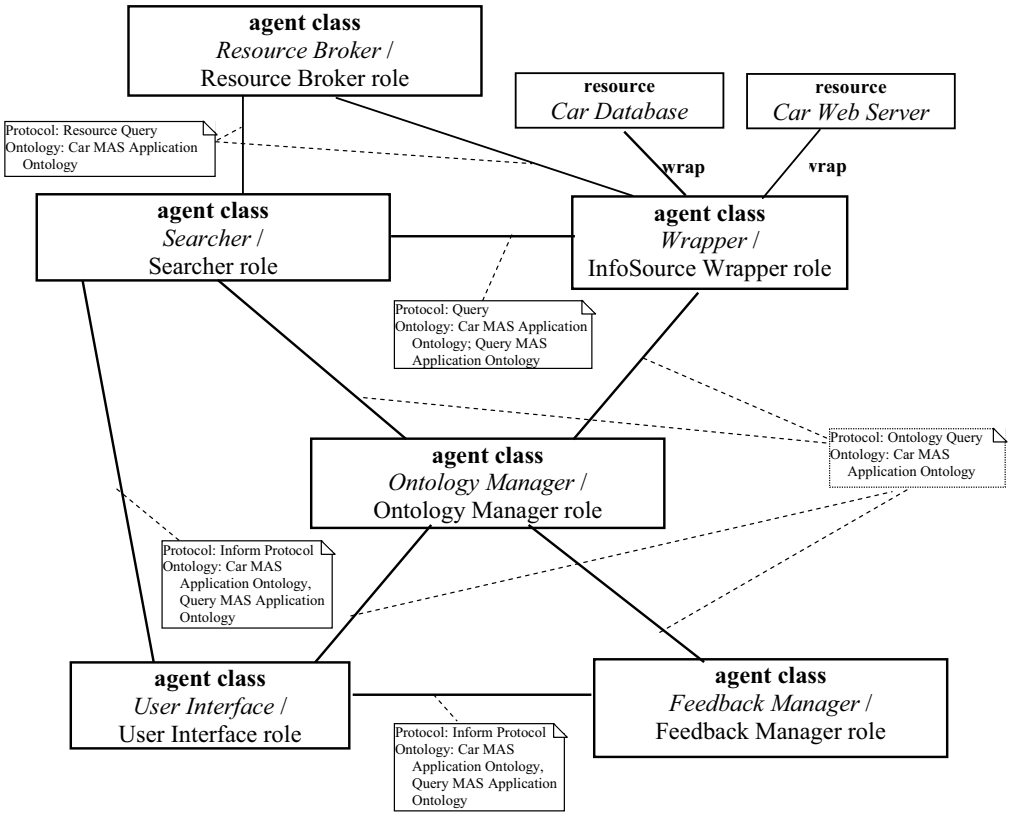



Figure 6.52 – Updated Agent Relationship Diagram for Product Search MAS

The *Role Diagram* of the *Role Model Kind* should also be checked to ensure that all the acquaintances between roles have been captured, and all the authority relationships are still valid. For example, two roles that are initially thought to be in a peer-to-peer relationship may turn out be in a superior-subordinate relationship after an in-depth investigation of the agents' interactions.

## 6.5.2.1.d. Conceptualise interation protocols with ontology (Optional)

So far, the interaction protocols governing the potential agent conversations in the target MAS have been identified and defined, particularly by step "*6.5.2.1.a. Define interaction protocols*". Traditionally, these protocols will be directly hard-coded into the agents at the implementation time, allowing any agents embedding the appropriate protocols to be able to participate in the respective conversations. Such an implementation mechanism, however, is only suitable to a closed or semi-open MAS environment, where the agents taking part in the interactions are known in advance and can be controlled. In an open environment, on the other hand, this mechanism is not sufficient. An open system would allow new agents to frequently enter the system and

join any existing conversations (such as an auction or an open marketplace). If the interaction protocols are hard-coded into the agents, joining a conversation whose protocol an agent does not know would mean that the agent has to either use one of the protocols it already knows, or else go off-line to be re-programmed. Similarly, in a dynamic environment where the interaction protocols can change over time, the hard-coding of protocols into agents will imply the need for re-coding of all affected agents when the protocols change.

Thus, MOBMAS proposes this optional design step, "*Conceptualise interaction protocols with ontology*", to avoid the above issues at run-time. This step is recommended to be performed if:

- the conversations in the future MAS are expected to be open; that is, any agents are allowed to join thethe pre-existing conversations; and/or
- the interactions in the future MAS are expected to be dynamic; that is, the interaction protocols governing the conversations can change overtime at run-time.

The techniques for this step are based upon Tamma et al.'s approach to ontology-based agent negotiation (Tamma et al. 2005; Tamma et al. 2002a; Tamma et al. 2002b). This approach suggests that agents do not have to hold priori knowledge about the interaction protocols. Instead, when an agent joins a conversation, it will then be provided with the protocol's definition, which is expressed in terms of an *ontology* shared between the agents participating in the conversation. Accordingly, the only priori knowledge that an agent needs to hold is this shared ontology. The definition of the protocol itself would be owned by some agents in-charge (such as auctioneer agents in an auction MAS) and distributed to the requesting agents at run-time. By committing to Ththe protocol'se ontology, the requesting agents would be able will provide the basic vocabulary for the agent to acquireunderstand the acquired and understand the definition of the protocols at run-time. . If the protocol changes, a new protocol definition could be sent to all participating agents.

For example, considering the "Query Protocol" governing the conversation between "Searcher" agents and "Wrapper" agents in the illustrative Product Search MAS (Figure 6.51). The traditional approach would have the protocol's definition directly hard-coded into the "Searcher" agents. Meanwhile, the ontology-based approach by Tamma et al.

(2005) suggests that only the *ontology* conceptualising the protocol should be hard-coded. The protocol's definition itself, which is built upon the ontology, would be given to the "Searcher" agents at run-time (for example, by the "Wrapper" agents).

This approach implies the need for the following design tasks:

- firstly, to define the *ontology* that conceptualises the protocols. MOBMAS refers to this ontology as the "***Protocol Ontology***";
- secondly, to develop *ontology-based definitions of the interaction protocols*; that is, to describe the protocols in terms of the Protocol Ontology.

These design tasks are sub-steps of the step "*Conceptualise interaction protocols with ontology*". The following sections describe these sub-steps.

It should be noted that not all interaction protocols in the MAS need to be conceptualized. Some conversations may be open and dynamic, but some others are closed, static and fixed. Protocols of the latter can be directly hard-coded into the agents without being conceptualised via any ontology.

**Define Protocol Ontology**

The Protocol Ontology should provide a set of generic concepts and relationships that can be used as the vocabulary for describing interaction protocols, for example, concepts "Protocol", "Participating party", "Message" and "Rule". Even though the different protocols can be largely different in term of their specifications, the vocabulary underlying these specifications should often be the same. Accordingly, it is desirable to have only one Protocol Ontology for the whole MAS. In that way, all agents in the MAS can commit to this single Protocol Ontology and can still obtain definitions of many different protocols at run-time.

The designer can either define the Protocol Ontology from scratch, or adopt/adapt an existing ontological work. In either case, the Protocol Ontology should contain all (and only) the concepts needed to define the relevant interaction protocols. Tamma et al. (2005; 2002a,b) have made a pioneering effort in developing an ontology for negotiation protocols. The ontology, called "Negotiation Ontology", provides the basic vocabulary for describing any negotiation protocols (as claimed by the authors).

The Protocol Ontology can be graphically represented by an Ontology Diagram like other types of ontologies. The Protocol Ontology Diagram should be added to the Ontology Model. As a result, the Ontology Model for a particularly MAS can embrace 3 types of ontologies in total: MAS Application ontologies, Resource Application ontologies and Protocol ontology.

Figure 6.53 presents a simple Protocol Ontology for the illustrative Product Search MAS. The ontology is based upon Tamma et al.'s (2005) Negotiation Ontology.

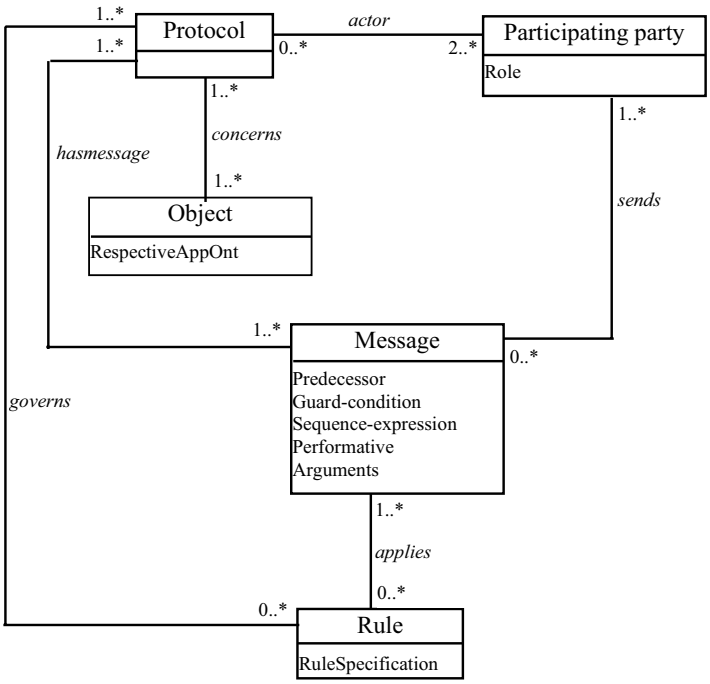



Figure 6.53 – Protocol Ontology

**Specify ontology-based definitions of interaction protocols**

Previously, step "*6.5.2.1.a. Define interaction protocols*" has produced the definitions of potential interaction protocols for the target MAS. In this sub-step, these protocol definitions are re-expressed in terms of the Protocol Ontology, thereby generating "ontology-based definitions" for the protocols. In order to do this, the designer should *instantiate* the concepts of the Protocol Ontology with specific values, so as to describe a particular protocol. For example, concept "Protocol" in the Protocol Ontology can be instantiated with "Query Protocol", and concept "Participating party" with "Requester" and "Informant", so as to define a query protocol. The underlying aim is to reproduce the protocol definition specified by step "*6.5.2.1.a. Define interaction protocols*" by using the vocabulary provided by the Protocol Ontology. At the implementation time, these

ontology-based definitions of protocols would be coded into some selected agents, who are in charge of distributing the definitions to other agents at run-time, for example, auctioneer agents in an auction MAS, or the "Wrapper" agents in the illustrative Product Search MAS.

Figure 6.54 illustrates the ontology-based definition of the "Query Protocol" in the illuastrative Product Search MAS. This protocol is previously defined in Figure 6.51. Ontology-based definitions of protocols can be represented as Object Diagrams, which is built upon Protocol Ontology's Class Diagram.

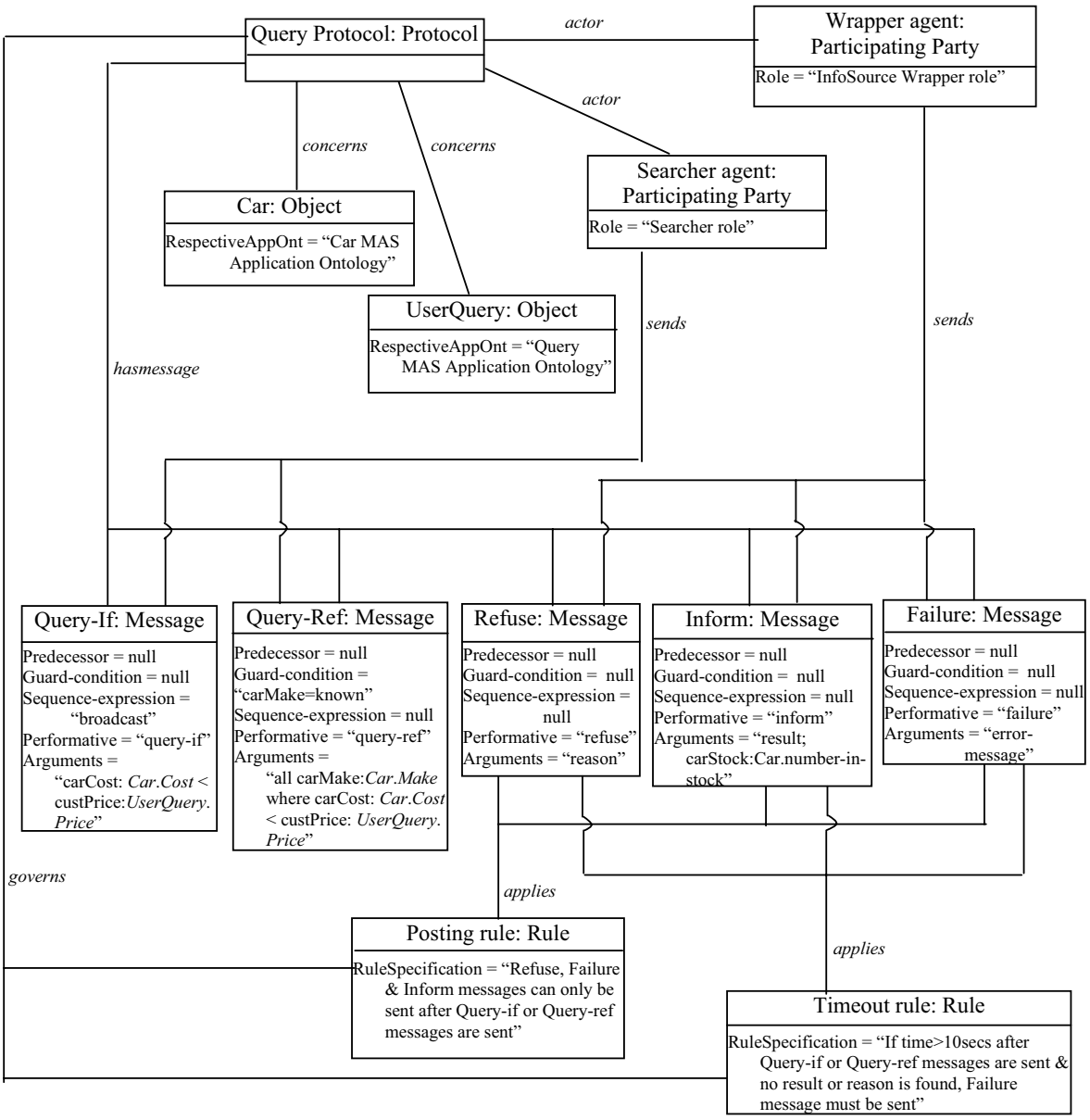



Figure 6.54 – Ontology-based definition of "Query Protocol" (c.f. Figure 6.51)

**Update Agent Class Model**

Any agent that potentially joins a conversation whose protocol is conceptualised should hold knowledge of the Protocol Ontology that conceptualises this protocol. This means that the Protocol Ontology should be part of the Belief Conceptualisation of the agent.

Figure 6.55 shows the updated Agent Class Diagram of "Searcher" agent class, where the Protocol Ontology (c.f. Figure 6.53) has been added to the Belief Conceptualisation compartment of the agent class. This will allow "Searcher" agents to acquire and understand the ontology-based definition of the "Query Protocol" at run-time.

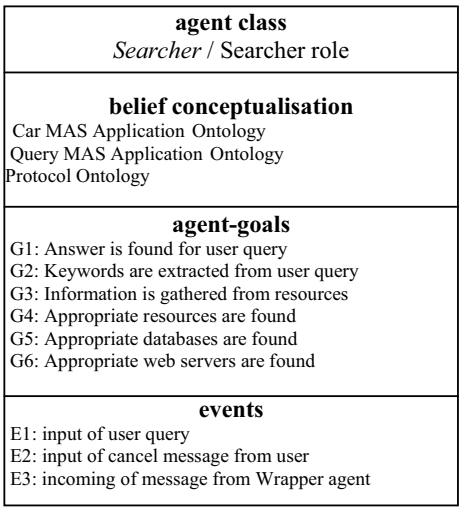



Figure 6.55 – Updated Agent Class Diagram (for "Searcher" agent class) of Product Search MAS

**Introduce new interactions to Agent Interaction Model**

Since new agents would need to acquire an appropriate ontology-based protocol definition from an agent in-charge before joining a conversation, there should be some initial interactions between the new agents and the agent in-charge so as to allow for this acquisition. These initial interactions, while simple, should be specified in the Agent Interaction Model. Naturally, protocols governing these interactions should be simple and fixed, hence can be directly hard-coded into the agents.

## *6.5.2.2. Develop Agent Interaction Model for Tuplespace/Tuple-Centre Interaction Mechanism*

In the tuplespace/tuple-centre interaction mechanism, agents interact by inserting, inspecting and removing tuples from a shared tuplespace or tuple-centre. Accordingly,

the Agent Interaction Model Kind of this mechanism is represented by a set of ***Agent-TC[73] Interaction Diagrams***, each of which models the interactions between particular agents and the tuplespace/tuple-centre during a conversation. In addition, for the *tuple-centre* interaction mechanism, since the tuple-centre exhibits programmable reactive behaviour, the Agent Interaction Model Kind should also contain ***Tuple-Centre Behaviour Diagrams***, which model the behaviour of the tuple-centre. Section 6.5.2.2.a discusses the development of Agent-TC Interaction Diagrams, while Section 6.5.2.2.b examines TC Behaviour Diagrams.

### 6.5.2.2.a. Develop Agent-TC Interaction Diagrams

To identify the potential interactions between the agents in MAS and the share tuplespace/tuple-centre, the developer should examine the *Agent Behaviour Model* of the MAS, namely the Agent Plan Templates and Reflexive Rule Specifications of each agent class.

- Each *communicative action* in the Agent Plan Templates and Reflexive Rule Specifications indicates a tuple being sent from an agent to the tuplespace/tuple-centre.
- Any *event* that is specified in an Agent Plan Template or that triggers a Reflexive Rule Specification may be resulted from a tuple being sent from an agent to the tuplespace/tuple-centre.

Each exchanged tuple should be defined in terms of:

- a communication primitive; and
- definition of the tuple's content.

Regarding the *communication primitives*, most of the existing tuplespace/tuple-centre middleware/models are built upon the LINDA framework (Papadopoulos 2001). As such, they adopt Linda-like communicative primitives such as ***out***, ***in***, ***rd***, ***inp*** and ***rdp*** (Omicini and Zambonelli 1999).

- out writes a tuple to the tuplespace/tuple-centre.

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

- in and rd send a tuple template to the tuplespace/tuple-centre and expect the tuplespace/tuple-centre to return a tuple that match the template, either deleting it or not from the tuplespace/tuple-centre, respectively.
- inp and rdp work analogously to in and rd, however while the latter wait until a matching tuple becomes available, inp and rdp fail if no such tuple is found.

Regarding *the content of tuples*, MOBMAS describes each tuple's content in terms of:

- ***a descriptive name***: e.g. "paperTuple", "all-paper-title-listTuple", "req-num-of-distributionTuple"[74] and "req-num-of-paperTuple"[75]; and
- ***arguments***: which represent the pieces of information that the tuple conveys. An argument can be a *constant* or a *variable*. For variable arguments, their datatypes must be defined. For example, in the following tuples of the illustrative Conference Program Management MAS,

  "req-num-of-distributionTuple(numberDist**: Integer**)" and

  "paperTuple(paperID**: Paper.ID**, paperTitle**: Paper.Title**, paperContent**: Paper.Content**)",

  "numberDist" is an "Integer" variable, while "paperID", "paperTitle" and "paperContent" are variables of datatypes "Paper.ID", "Paper.Title" and "Paper.Content" respectively[76]. "Paper", "ID", "Title" and "Content" are application-specific concepts that are defined in the MAS Application ontology which is shared between the communicating agents to govern the communication's semantics. Note that basic datatypes such as Integer or String are assumed known to every agent in the MAS, without having to be defined in a MAS Application ontology.

Given the above conventions, an example tuple sent by a "PC Chair" agent in the Conference Program Management MAS to the shared tuplespace/tuple-centre is

"**out**(*paperTuple*(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**))",

while a tuple sent by a "PC Member" agent to the tuplespace/tuple-centre is

"**inp**(*paperTuple*(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**)).

---

[74] That is, the required number of members to whom each paper must be distributed.
[75] That is, the required number of papers that each member must collect.
[76] These datatypes should be interpreted as, "ID" of "Paper", "Title" of "Paper" and "Content" of "Paper" respectively.

In the tuplespace/tuple-centre interaction mechanism, all interactions are asynchronous, due to the decoupling in agent identity, location and time during interactions (cf. Section 6.5.1.2).

***Notation of Agent-TC Interaction Diagrams***

Each Agent-TC Interaction Diagram graphically specifies an allowed pattern of tuple exchanges between agents and a shared tuplespace/tuple-centre during a conversation. MOBMAS reuses the notation of AUML Sequence Diagram for Agent-TC Interaction Diagram. The only difference is that the exchanged elements are tuples instead of ACL messages. Figure 6.56 presents an Agent-TC Interaction Diagram for a conversation between a "PC Chair" agent, a "PC Member" agent and the shared tuplespace/tuple-centre in the Conference Program Management MAS.

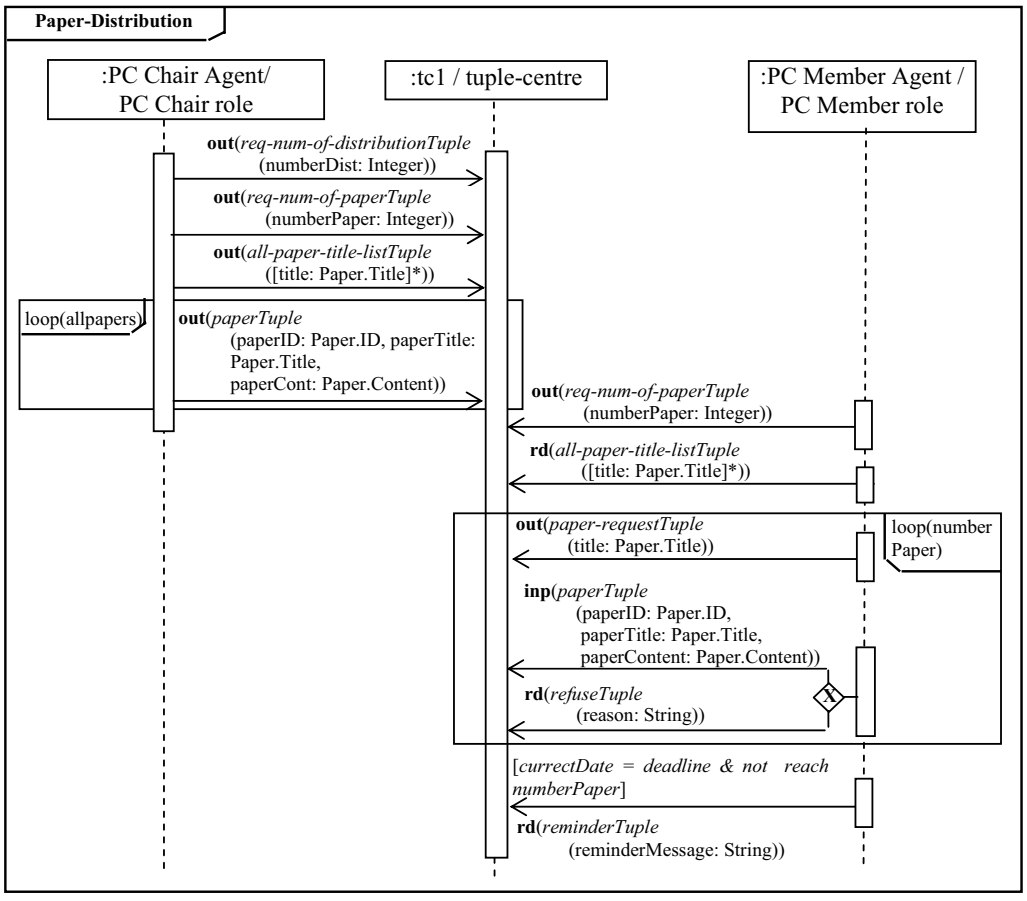



Figure 6.56 – Agent-TC Interaction Diagram for Conference Program Management MAS

### 6.5.2.2.b. Develop Tuple-Centre Behaviour Diagram (Optional)

This step is applicable only to the *tuple-centre interaction mechanism*. A tuple-centre behaves by *reacting* to the incoming tuples from agents (Omicini and Zambonelli 1999). Each "reaction" is a set of non-blocking actions which, if successfully executed, will change the state of the tuple-centre from one state to another. Otherwise the reaction will yield no transition in the tuple-centre's state at all (Dente et al. 1998).

MOBMAS recommends the following guidelines for the definition of the tuple-centre's reactions.

- *The reactions should allow the tuple-centre to enforce all the necessary coordination rules*: For example, consider agent-goal "Papers are distributed among members" jointly achieved by the "PC Chair" and "PC Member" agent classes in the Conference Program Management MAS (cf. Figure 6.7). One of the coordination rules governing the achievement of the agent-goal is that "Each PC member must not collect the same paper twice" (cf. Section 6.5.1.2). The tuple-centre can enforce this rule by defining a reaction that checks whether a particular "PC Member" agent is eligible to obtain a particular "Paper" tuple (Figure 6.57).

  Similarly, to enforce another coordination rule "Members can only start collecting papers after all other members have viewed the Title List of all papers" (cf. Section 6.5.1.2), the tuple-centre can make *n* copies of the "Title List" tuple, where n is the total number of the "PC Member" agents in the system. Then, the tuple-centre will not allow any "PC Member" agents to consume a "Paper" tuple until all copies of the "Title List" tuples have been consumed by all "PC Member" agents (Figure 6.57).

- *The reactions should allow the tuple-centre to carry out some (low level) processing to help fulfilling the target coordinating task, thereby taking some processing load off the interacting agents*: For example, to assist in the achievement of the agent-goal "Papers are distributed among members", the tuple-centre can try to determine which "PC Member" agent has *not* collected the required number of papers on the due date, thereby posting "Reminder" tuples to these agents.

- *The reactions should help the tuple-centre to deal with* "*inter-agent conflicts*": The issue of inter-agent conflicts have been discussed in Section 6.5.2.1.a.

***Notation of Tuple-Centre Behaviour Diagram***

MOBMAS adopts UML Statechart Diagram for the Tuple-Centre Behaviour Diagram.

- Each state of the diagram represents a reaction of the tuple-centre. A state can either be passive (i.e. idle, denoted as ⬭) or active. If active, it should contain one or more actions to be executed sequentially by the tuple-centre.
- Transitions between states occur when an event happens, e.g. when a tuple is sent by an agent.

At design time, the developer can specify reactions and state transitions using natural language and descriptive method names. These reactions and transitions can be formally coded using a "behaviour specification language" at implementation, e.g. ReSpecT (Denti et al. 1998).

Figure 6.57 presents the Tuple-Centre Behaviour Diagram for the tuple-centre of the Conference Program Management MAS during a conversation between the "PC Chair" and "PC Member" agents (cf. Figure 6.56).

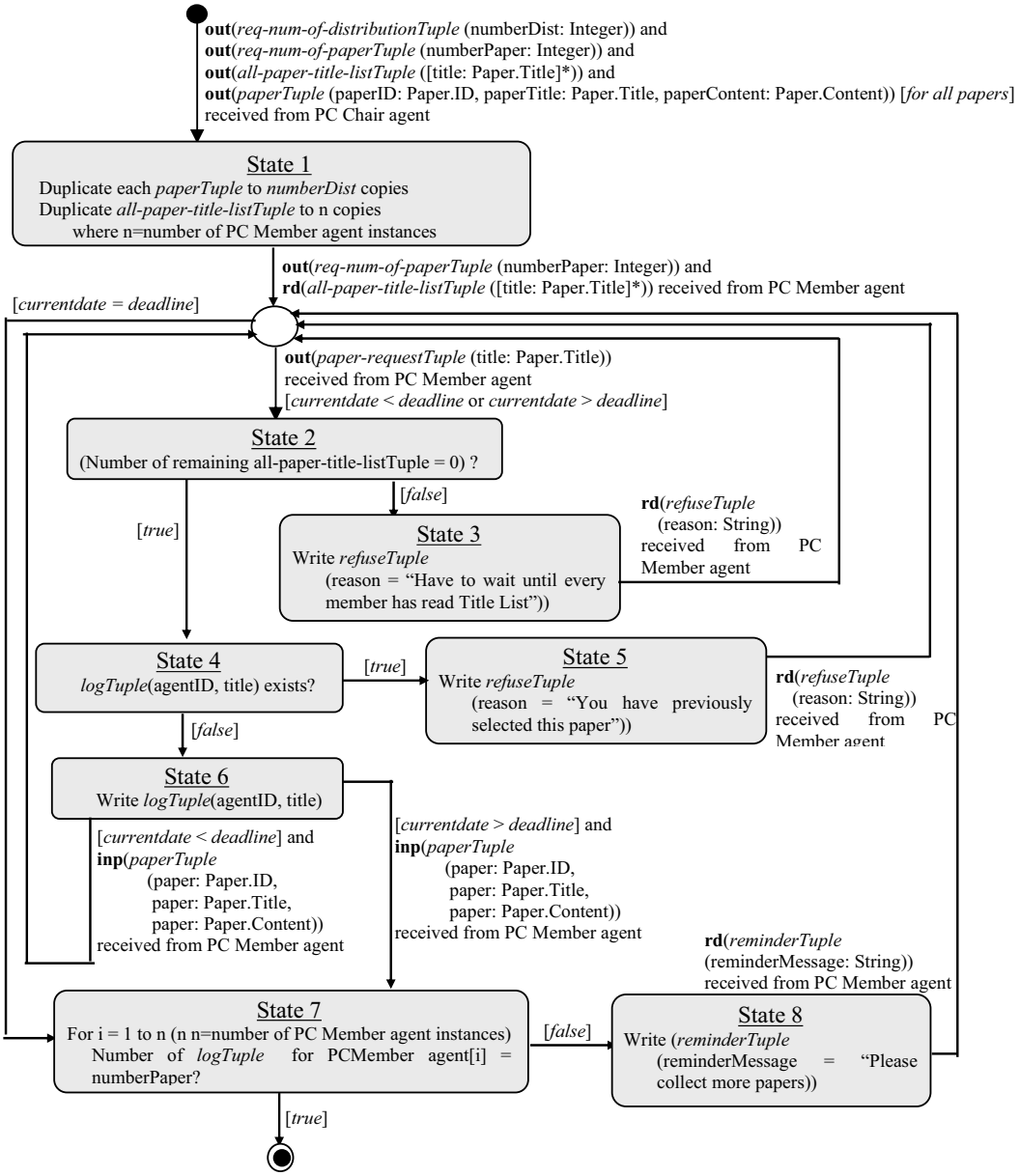



Figure 6.57 – Tuple-Centre Behaviour Diagram for Conference Program Management MAS

## 6.5.2.2.c. Update Agent Class Model and Role Model

The *Agent Relationship Diagram* of the *Agent Class Model Kind* should be updated to show:

- any new acquaintances between agent classes that have not been identified; and
- various descriptive information about *each* acquaintance, namely:

- the *identity of the Agent-TC Interaction Diagrams* that govern the conversations between the acquainted agents and the shared tuplespace/ tuple-centre; and
- the *MAS Application ontology(ies)* that governs the semantics of the tuples exchanged during the conversation and which must be shared between the communication agents[77].

This descriptive information is modelled as *UML notes* attached to each acquaintance (cf. Figure 6.22).

Figure 6.50 presents the Agent Class Diagram for the Conference Program Management MAS (cf. Figure 6.7).

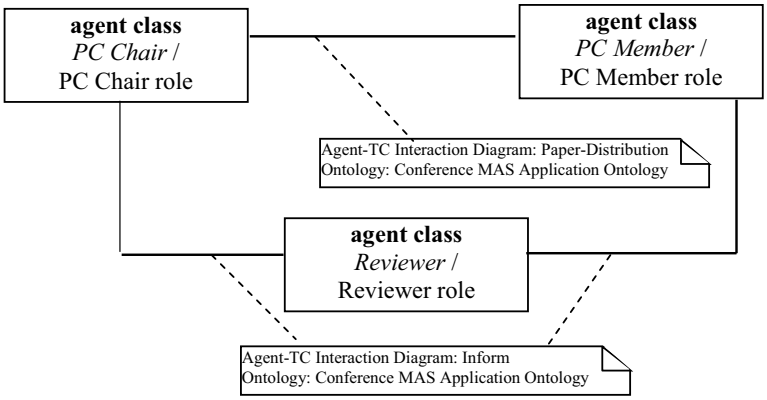



Figure 6.58 – Updated Agent Class Diagram of Conference Program Management MAS

The *Role Diagram* of *Role Model Kind* should also be checked to ensure that all the acquaintances between roles have been captured, and all the authority relationships are still valid. For example, two roles which are initially thought to be in a peer-to-peer relationship may turn out be in a superior-subordinate relationship after an in-depth investigation of the agents' interactions.

### *6.5.2.3. Verify Agent Interaction Model against Ontology Model and Agent Internal Model*

In both the direct interaction mechanism via ACL and indirect interaction mechanism via tuplespace/tuple-centre, the *semantics* of the information conveyed in the ACL messages and tuples must be *consistently* interpreted by the interacting agents. To ensure this consistency, the ***datatypes*** of all variable arguments in the exchanged ACL

[77] That is, this MAS Application ontology must exist in both agents' Belief Conceptualisation.

messages or tuples *must* be defined using the concepts obtained from the MAS Application ontology *shared* between the communicating agents. Being "shared" means that this ontology must exist in the Belief Conceptualisation of each communicating agent. Vice versa, only concepts defined in the *shared* MAS Application ontology can be used to define the datatypes of the variable arguments in the ACL messages or tuples. Basic datatypes such as Integer or String are assumed known to every agent and thus do not need to be defined in an ontology.

For example, in the ACL message sent from a "Searcher" agent to a "Wrapper" agent,

"query-if (carCost:**Car.Cost** < custPrice: **User query.Price**)",

concepts "Car", "Cost", "User query" and "Price" should have been defined in the Car MAS Application Ontology and Query MAS Application Ontology that are shared between the "Searcher" and "Wrapper" agents (cf. Figures 6.13 and 6.14).

Similarly, in the tuple sent from a "PC Chair" agent to a "PC Member" agent via a shared tuplespace/tuple center,

"paperTuple(paperID**: Paper.ID**, paperTitle**: Paper.Title**,
paperContent**: Paper.Content**)"

concepts "Paper", "ID", "Title" and "Content" must be defined in a MAS Application Ontology which is shared between the "PC Chair" and "PC Member" agents.

The above guidelines highlight the need for *reciprocal* and *iterative* development of the Agent Interaction Model and Ontology Model. More specifically, the developer should:

- use the concepts defined in the MAS Application ontologies to formulate the content of the exchanged ACL messages and tuples; and
- examine the content of the ACL messages and tuples to determine if any concepts have *not* been defined in the developed MAS Application ontologies, thereby verifying the content of these ontologies.

The developer should also verify the Agent Interaction Model against the Agent Internal Model, to ensure that the ontology(ies) governing the semantics of communication between each pair (or group) of agent classes is indeed listed in the Belief Conceptualisation of each agent class (that is, the ontology is *shared* by the

communicating agent classes). If the communicating agents do not yet share a common ontology, such an ontology should be built and added to each communicating agent's Belief Conceptualisation. This ontology should contain concepts that serve as the inter-lingua between the agents' local (heterogeneous) ontological concepts.

# 6.6. ARCHITECTURE DESIGN ACTIVITY

This activity of MOBMAS deals with various design issues relating to the architecture of agents and MAS, namely the identification of agent-environment interface requirements, the selection of agent architecture, the identification of required infrastructure facilities, the instantiation of agent classes and the deployment configuration of the agent instances. The product of this activity is an **Architecture Model Kind**, which is represented by the following notational components.

- **Agent-Environment Interface Requirements Specification**: documents any special requirements of agents' sensor, effector and communication modules.
- **Agent Architecture Diagram**: provides a schematic view of the agent architecture(s).
- **Infrastructure Facilities Specification**: documents the specifications of the infrastructure facilities needed to support the target MAS' operation.
- **MAS Deployment Diagram**: shows the deployment configuration of the target MAS, including the allocation of agents to nodes and the connections between nodes.

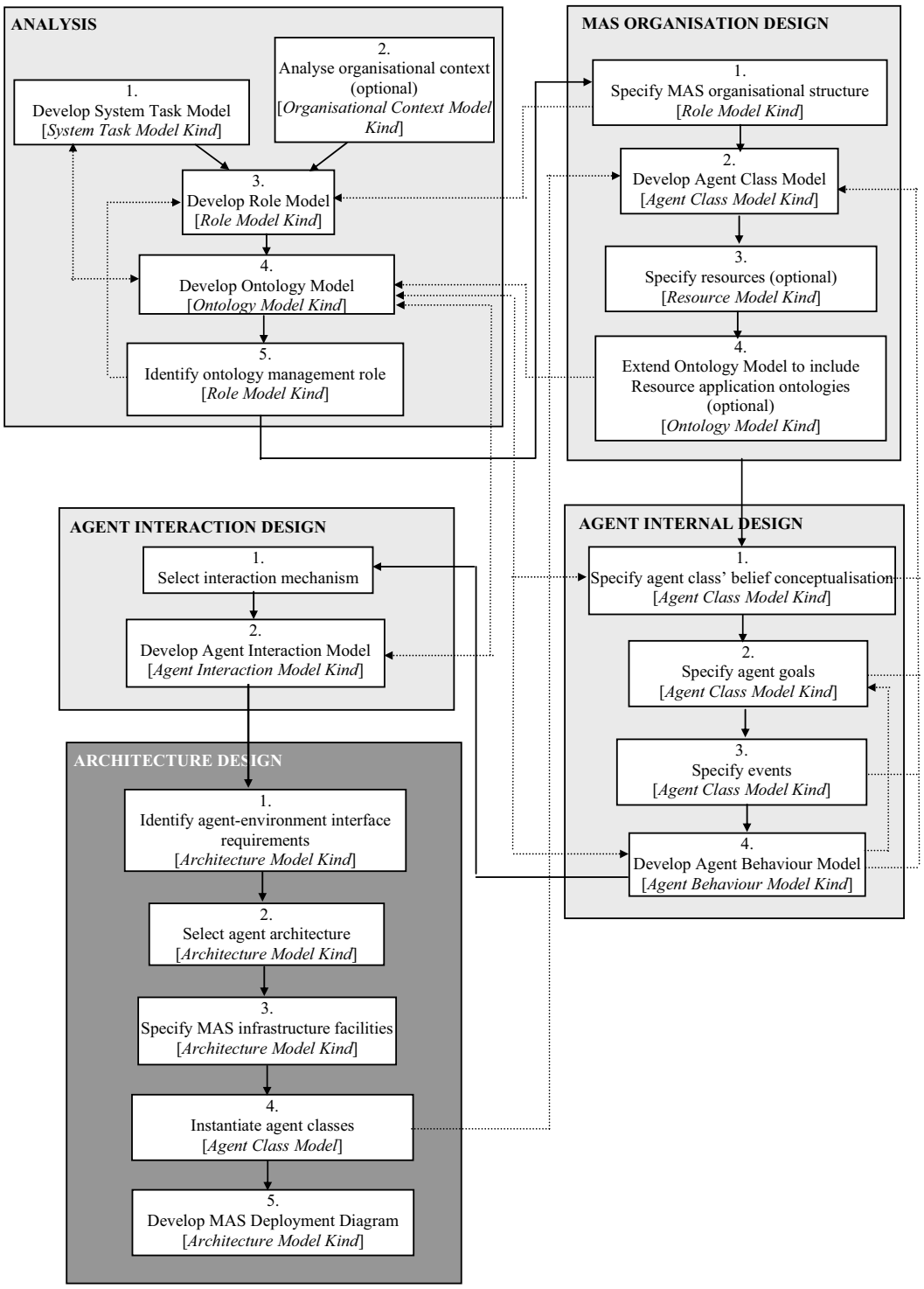



Figure 6.59 – MOBMAS development process

### 6.6.1. Step 1 – Identify Agent-Environment Interface Requirements

An agent may interact with its environment via (van Breemen 2002):

- *Perception*: which is an activity of observing or sensing the state of the environment;
- *Effect*[78]: which is an activity of changing the state of the environment; and
- *Communication*: which is an activity of exchanging ACL messages or tuples with other agents.

MOBMAS recommends the developer to investigate the characteristics of the perception, effect and communication activities to be performed by each agent class, so as to facilitate the selection of sensors, effectors, agent architectures and/or implementation platform.

Specifically, with regard to **perception** and **effect**, the developer should consider:

- whether they are related to the *physical world* (e.g. observing and changing the location of a soccer ball) or the *virtual world* (e.g. observing and changing the price of a car product) (Russell and Norvig 2003; Weyns et al. 2004). Perception and effect on the physical world typically require hardware components such as thermostats, infra-red sensor, wheels and grippers. In such cases, the sensor and effector of the agents must be able to connect and control these hardware components (probably by communication with the hardware's driver software). Meanwhile, for virtual perception and effect, the sensor and effector of the agents can directly perceive and impact on the environment without the use of hardware components;

- the *degree of complexity* of the perceptual inputs and/or effect outputs. If the agents operate in an open, dynamic and fast changing environment overloaded with sensor information, they may require robust perception mechanisms and efficient perception strategies (e.g. filtering) (Wray et al. n.d.). The developer should also consider employing *sensor/effector objects* to take care of the perception and effect activities for the agents, thereby relieving the agents from some workload. For

[78] "Effect" is referred to as "action" in van Breemen (2002). However, the term "action" is not used here because it may be confused with "actions" in Agent Plan Templates and Reflexive Rule Specification. The latter may or may not result in a change in state of the environment.

example, a sensor object can monitor percepts from the environment, thereupon alerting the agent if certain conditions exist. The sensor object may also serve as a "historical sensor" that watches for trends and patterns in the incoming percepts (Kendall et al. 1995); and

- *interaction with human user*: If an agent is required to engage in intensive interaction with human users, its sensor needs to be connected to an elaborate user-interface component that provides efficient means for inputs from, and outputs to, the users.

Regarding **inter-agent communication**, most existing agent architectures and implementation platforms provide built-in support for basic communication operations, e.g. message dispatching/receiving, tuple inserting/reading/removing, and message/tuple transport services. However, if the target MAS has some special communication requirements, for example, encryption of exchanged messages, mobile and ubiquitous communication, or support for binary data exchanged (such as rich multimedia objects), these requirements should be documented. Note that the interaction mechanism adopted by the target MAS may also impose certain requirements that are not commonly supported by existing agent architectures and implementation platforms. For example, spatially-founded interaction mechanism requires an implementation platform that can provide a spatially structured environment in which agents can be placed and communicate, e.g. SWARM (Minar et al. 1996).

At the end of this step, the *Agent-Environment Interface Requirement Specification* of the *Architecture Model Kind* should be developed to document the followings:

- any special requirements of the agents' sensor and effector (e.g. the need for connecting to hardware components, or the need for using sensor/effector objects); and
- any special requirements of the agents' communication activities.

All of these specifications can be documented in an informal natural language.

## 6.6.2. Step 2 – Select Agent Architecture

An agent architecture is a structural model of the *modules* that constitute an agent and the *interconnections* between these modules (Lind 1999). Abstract constructs of each agent class (namely, belief conceptualisation, agent-goals, plans and reflexive rules) will be mapped onto these concrete modules during implementation, which is not part of MOBMAS.

Given the availability of a large number of agent architectures, MOBMAS does not address the issue of agent architecture design, but instead presents guidelines on how to select the most appropriate agent architecture(s) for the target MAS.

### *6.6.2.1. Select agent architecture*

Agents in a MAS may adopt the same agent architecture or require different architectures to support their different functional requirements. To select the most appropriate architectures for agents, the developer should consider the following factors.

- *Style of agent behaviour*: The style of agent behaviour is reflected in the Agent Behaviour Model Kind. If an agent class adopts *planning behaviour* to fulfil its agent-goals, the chosen agent architecture must be able to support planning activities via the use of planners, reasoners, analysers, or the like. Some well-known planning architectures are STRIPS (Fikes and Nilsson 1971), IPEM (Ambros-Ingerson and Steel 1988), IRMA (Bratman et al. 1988), Homer (Vere and Bickmore 1990) and SOAR (Newell 1990). If otherwise the agent class adopts solely reflexive behaviour, the chosen agent architecture does not need to support complex symbolic reasoning. Several well-known architectures for reflexive agents are subsumption architecture (Brook 1986), PENGI (Chapman and Agre 1986), AuRA architecture (Arkin and Balch 1997) and situated automata (Kaelbling 1991). In cases when an agent class exhibits hybrid behaviour (e.g. planning behaviour for some agent-goals and reflexive behaviour for some other agent-goals), a hybrid agent architecture which employs a *layered structure* to support both planning and reflexive behaviour is required. Example hybrid architectures are RAPs (Firby 1989), ATLANTIS (Gat 1991), TouringMachines (Ferguson 1992), INTERRAP (Muller and Pischel 1993),

Prodigy (Carbonell et al. 1991), PRS (Myers 1997; Georgeff and Lansky 1987) and dMARS (d'Inverno et al 1997).

- *Required agent behavioural capabilities*: The selected agent architecture should allow an agent to implement its desirable behavioural capabilities. For example, if an agent class is required to learn, the selected agent architecture should be able to support learning capability. Or, if the agent class needs to be time-persistent, its architecture must allow the knowledge structures to be maintained over time and over unavoidable downtimes (Wray n.d.).

- *Style of Control*: An agent class' desirable style of control can help to determine the required agent architecture. Some example styles of control are (Wray n.d.; Lind 1999):
  - asynchronous versus synchronous: The former requires an agent architecture that supports asynchronous processing threads while the latter does not; and
  - static versus dynamic: The former can be supported by an agent architecture that implicitly hardcodes the control flow, while the latter requires an architecture that allows the control flow to be explicitly specified (e.g. in plan scripts).

- *Knowledge representation mechanism*: The desirable mechanism of agent knowledge representation should be matched against the mechanism provided by the agent architecture. For example, an agent architecture may allow the agent knowledge to be explicitly stored in a knowledge base, or requires the knowledge to be implicitly embedded in the agent coding (Lind 1999).

- *Complexity of sensor input*: Agents that operate in a dynamic, open and fast-changing environment often faces overloads in sensor information. They should therefore adopt an architecture that supports robust perception mechanisms and efficient sensing strategies (e.g. sensor inputs filtering).

- *Support for scalability*: Some agent architectures such as HOMER experience a processing slow-down when its episodic knowledge base increases in size (Vere and Bickmore 1990). Thus, the developer should select an agent architecture that can

accommodate the agents' expansion of knowledge (e.g. via the support for easy-upsizing of the knowledge base) (Wray n.d.).

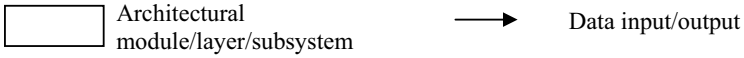
- *Agent-environment interaction requirements*: The sensor, effector and communication modules of the selected agent architecture should be able to support the requirements specified in the Agent-Environment Interface Requirements Specification (cf. Section 6.6.1).

### *6.6.2.2. Develop Agent Architecture Diagram*

As mentioned previously, *Agent Architecture Diagram* is one of the four notational components of the *Architecture Model Kind*. It aims to provide a schematic view of an agent architecture. If agents in the target MAS are homogeneous in architecture, only one Agent Architecture Diagram would be needed. Otherwise, multiple diagrams are required.

An Agent Architecture Diagram should specify:

- *the* modules or layers or subsystems of the architecture. These modules/layers/subsystems should be represented as boxes; and
- the potential flows of data between these modules/layers/subsystems. These flows are represented as arrows.




Figures 6.60 and 6.61 show the Agent Architecture Diagrams for the TouringMachines and INTERRAP architectures.

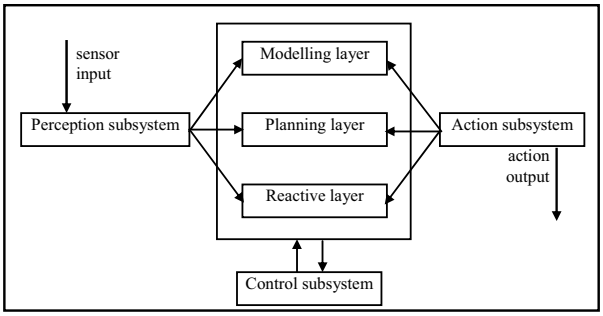



Figure 6.60 – Agent Architecture Diagram for TouringMachines architecture (Ferguson 1992)

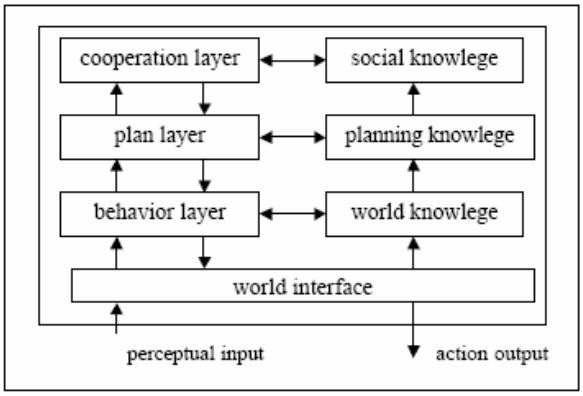



Figure 6.61 – Agent Architecture Diagram for INTERRAP architecture (Wooldridge 1999)

## 6.6.3. Step 3 – Specify MAS Infrastructure Facilities

As in other computing systems, MAS needs to be supported by various infrastructure facilities in order to operate. The developer should thus identify these infrastructure facilities and determine how these facilities can be provided and managed in the target MAS.

Various potential infrastructure facilities for a MAS system are (Iglesias et al. 1998; Shen et al. 1999):

- *Network facilities*: e.g. agent naming service, agent creation/deletion service, agent migration service, security service and accounting service;
- *Coordination facilities*: e.g. agent directory/yellow-page/white-page service, message transport service, protocol servers and "police" facilities to detect misbehaviours in agents or misusage of common resources; and
- *Knowledge facilities*: e.g. ontology servers, problem-solving methods servers and language translation service.

Some of these facilities are only necessary if the MAS is open and/or deployed in a distributed environment, e.g. agent migration services, security service and agent directory services. For many applications such as simulation, the MAS can be closed and run on a single platform, thus they do not require network-related infrastructure facilities.

MOBMAS suggests two common mechanisms for providing and managing infrastructure facilities.

- To employ the "built-in" infrastructure facilities provided by the MAS implementation platform, without managing them by any dedicated agents. For example, the kernel of MADKIT platform provides built-in "message transport services" (MADKIT 2002), while the kernel of AgentTcl offers built-in "agent migration services" (Gray 1995).

- To provide and manage customized infrastructure facilities via the use of some dedicated agents. For example, "directory services" may be offered by a "Facilitator/Broker" agent, while "security services" can be handled by a specialised "Police" agent. These agent classes may have already been defined in the *Agent Class Model*, or are now introduced. In both cases, the developer should refine the Agent Class Model to include new agent classes or to update the specification of the existing agent classes. All related models, including *Agent Behaviour Model* and *Agent Interaction Model*, should be accordingly revised.

An *Infrastructure Facility Specification* should be developed and included in the *Architecture Model Kind* to document the specifications of all the identified infrastructure facilities. These specifications can be documented in an informal natural language.

## 6.6.4. Step 4 – Instantiate Agent Classes

Each agent class should be instantiated into concrete agent instances. Common types of cardinality for agent instantiation are (Wooldridge et al. 2000):

- *n* cardinality: i.e. where each agent class is instantiated with exactly *n* agents;
- *m..n* cardinality: i.e. where each agent class is instantiated with no less than *m* and no more than *n* agents; and
- + cardinality: i.e. where each agent class is instantiated with one or more agents.

The instantiation cardinality of each agent class is represented as an annotation next to the agent class name in the **Agent Relationship Diagram** of the **Agent Class Model**

**Kind** (cf. Figure 6.25). Figure 6.62 presents the updated Agent Relationship Diagram for the illustrative Product Search MAS.

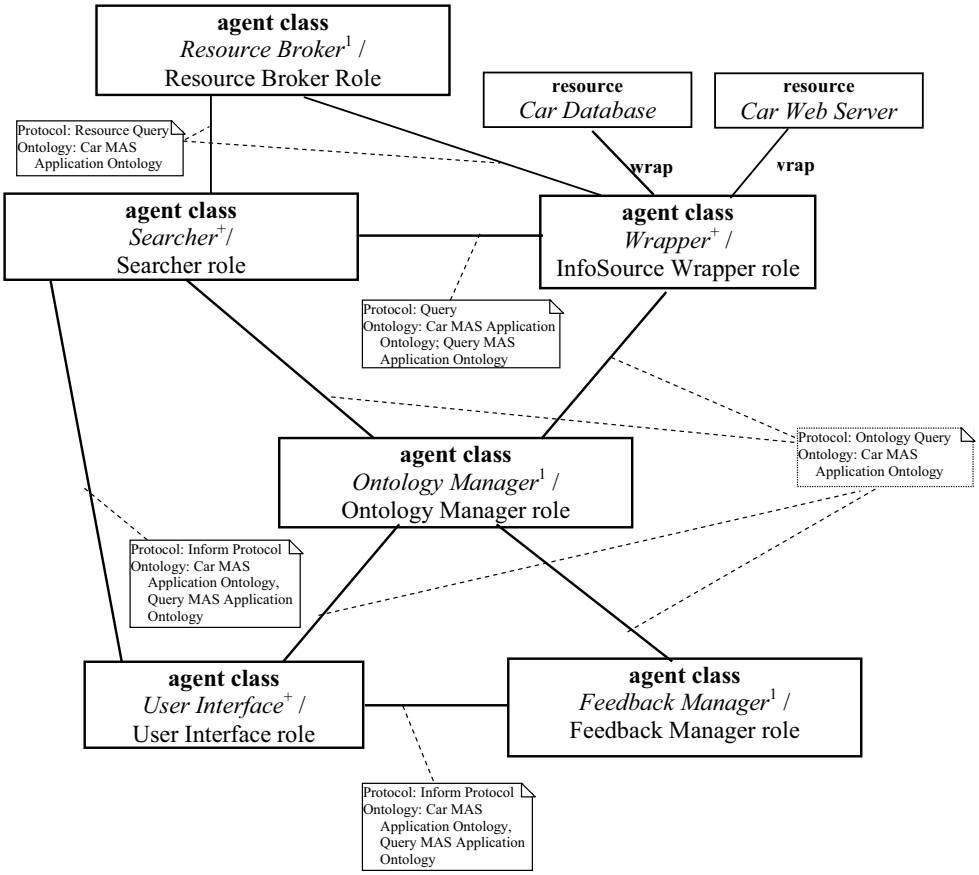



Figure 6.62 – Updated Agent Relationship Diagram of Product Search MAS

## 6.6.5. Step 5 – Develop MAS Deployment Diagram

In MOBMAS, the logical architecture of the target MAS is revealed in the *Agent Relationship Diagram* of the *Agent Class Model Kind*. This diagram shows all the agent classes composing the MAS system, the resources existing in the system, the acquaintances between agent classes, the connections between agent classes and resources, and the instantiation of agent classes. In this step, a *MAS Deployment Diagram* should be developed to describe how this logical MAS architecture can be actuated in the operational environment (i.e. how the MAS components can be located, distributed and connected). This diagram is particularly necessary for highly distributed MASs where it is important to visualise the system's physical topology. The MAS

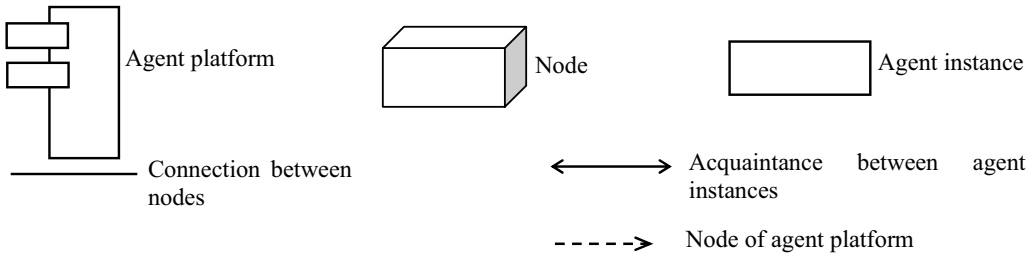
Deployment Diagram is one of the four notational components of the Architecture Model Kind as mentioned previously.

Major configuration details to be specified in the MAS Deployment Diagram include:

- *agent platforms*: i.e. the infrastructure on which agents are deployed. Agent platforms can be single processes containing lightweight agent threads, or fully built platforms around proprietary or open middleware standards (FIPA 2003);
- *nodes*: i.e. hosts on each agent platform;
- *agent instances located at each node*;
- *connections between nodes*; and
- *acquaintances between agent instances*.

The deployment configuration of MAS can be determined by investigating various factors, including the message traffic between nodes (estimated from *Agent Interaction Model*) and the required processing power of each node to accommodate the behaviour of agents (estimated from the *Agent Behaviour Model*).

MOBMAS reuses the notation of AUML Deployment Diagram (FIPA 2003) for its MAS Deployment Diagram[79].




The naming of agent platforms, nodes and agent instances should be specified in the format

instance-name : class-name

[79] AUML Deployment Diagram contains notation for agent mobility. However, since agent mobility is not discussed in MOBMAS, this notation is not presented.

Figure 6.63 presents an example MAS Deployment Diagram for the Product Search MAS.

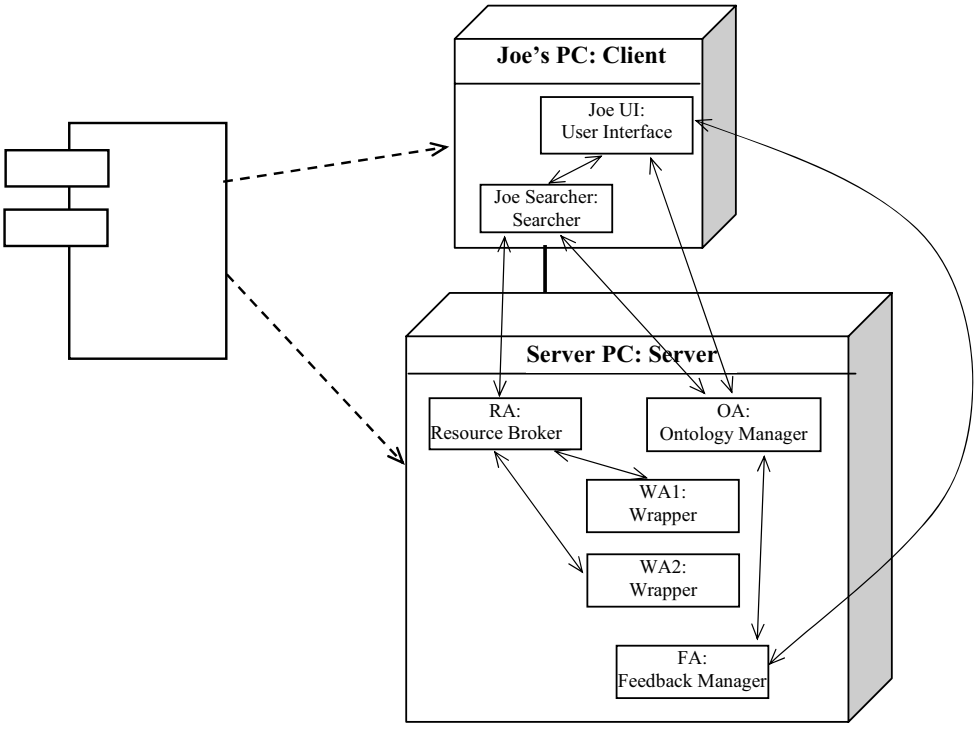



Figure 6.63 – MAS Deployment Diagram for Product Search MAS

# 6.7. SUMMARY

This chapter has presented the full documentation of MOBMAS – a "***M***ethodology for ***O***ntology-***B***ased ***MAS*** development". In total, MOBMAS consists of five activities, twenty steps and nine model kinds. An overview of MOBMAS' conceptual framework, development process and model kinds can be found in Section 6.1. Figure 6.2, in particular, depicts the five activities of MOBMAS, their associated steps, the iterative flow between these steps, and the model kinds produced or refined by each step. A detailed description of each MOBMAS' activity is provided in Sections 6.2 to 6.6.

In the next chapter, Chapter 7, the evaluation and refinement of MOBMAS are documented. These evaluation and refinements progressively led to the final version of MOBMAS as presented in this chapter.

# CHAPTER 7

# EVALUATION AND REFINEMENT OF MOBMAS

## 7.1. INTRODUCTION

This chapter documents the evaluation and refinement of MOBMAS as resulted from the execution of Research Activity 3 – "*Evaluate and Refine MOBMAS*" (cf. Section 4.3). The process of evaluating and refining MOBMAS consisted of three sequential steps (cf. Section 4.6):

- *Step 1* - Obtaining expert reviews;
- *Step 2* - Using MOBMAS on a test application; and
- *Step 3* - Performing a feature analysis on MOBMAS.

The first two steps resulted in the most refined version of MOBMAS which is presented in Chapter 6, while the third step evaluated this final version. Sections 7.2, 7.3 and 7.4 respectively report on the performance and outcome of each step.

## 7.2. EXPERT REVIEWS

The objective of expert reviews was to gather experts' non-empirical evaluation of the initial version of MOBMAS, specifically their opinions on the strengths, areas for improvement of the methodology, and how to improve these areas. The obtained feedback was used to refine MOBMAS before the methodology was empirically used on a test application by external developers (Section 7.3).

Section 7.2.1 firstly describes the procedures of the expert reviews, followed by Section 7.2.2 which provides brief biographic information about each expert. Section 7.2.3 then documents the refinements of MOBMAS as a consequence of the expert reviews.

### 7.2.1. Expert Review Procedures

Each of the two experts was provided with a full documentation of MOBMAS and requested to:

- identify the strengths of the methodology (optional).
- identify the areas for improvement of the methodology; and
- recommend how to improve these areas (optional).

The evaluation was based on the experts' non-empirical investigation of the methodology's documentation, and documented informally as "comment notes" on the methodology's documentation.

The two expert reviews were collected *independently* and in a *sequential* order. Comments from the first expert were used to refine the initial version of MOBMAS before the second expert was asked to evaluate the refined version. This sequential and independent procedure

- prevented the possibility of two experts identifying the same areas for improvement; and
- helped to identify new areas of improvement that potentially arose from the refinement of the methodology after the first review. Modifying the methodology according to the first expert review might introduce new problematic areas, which would be undetected if the second review was not subsequently conducted.

A face-to-face meeting was then organised with each expert to get a walkthrough of his/her feedback. This face-to-face communication helped to get an in-depth understanding of, and avoid any possible misunderstanding of, the experts' comments.

After MOBMAS was refined in accordance with each expert's feedback, follow-up communication via email was carried out with each expert to ensure that the experts were satisfied with all the refinements made to the methodology.

## 7.2.2. Experts' Biography

The two experts who participted in the study were Prof. Brian Henderson-Sellers (first reviewer) and Prof. Mary-Anne Williams (second reviewer). Both experts have active research interests in agent technology in general and AO methodologies in particular. They both have made major research contributions to the area.

- Prof. Brian Henderson-Sellers is the Director of the Centre for Object Technology Applications and Research at the University of Technology, Sydney. His research interest in AO methodologies arose from his long-term involvement in OO methodologies. He co-organised a number of international workshops on AO methodologies (including Annual ACM Conference on Object-Oriented Programming, Systems, Languages & Applications – OOPSLA, and International Bi-Conference Workshop on Agent-Oriented Information Systems). He is currently co-leading a research project funded by the Australian Research Council on "Metamodel-based Methodology for Developing Agent-Oriented Systems". The project investigates how to extend the OPEN framework, which was initially designed for OO development and of which he was a co-founder, to offer support for AO system development.

- Prof. Mary-Anne Williams is the Director of the Innovation and Technology Research Laboratory at the University of Technology, Sydney. She has been actively involved in research on both agent technology (particularly in the context of e-business, interactive marketing and artificial intelligence) and ontology. She has also supervised PhD students whose research interests were agent technology and/or ontology. Two of her current research projects are "Agent-Oriented Concept Management" and "Information and Knowledge Integration", both of which focused on agents and ontologies. She has also been the co-organiser of the International Conference on Principles of Knowledge Representation and Reasoning (KR), which covers the research topic of ontologies. With regard to this PhD research project, prior to getting involved as an expert reviewer of the MOBMAS methodology, Prof. Mary-Anne Williams was involved in the research in two aspects: she helped to shape the broad topic area of this research by sharing her background knowledge of the agent literature, and she assisted in recruiting participants for the research's

survey (Section 5.3) by helping to post the survey's recruiment advertisement on the UMBC Agents-Digest mailing list and UMBC AgentNews newsletter. At no stage was Prof. Mary-Anne Williams involved in the design of the MOBMAS methodology itself. Thus, Prof. Mary-Anne Williams was a totally independent reviewer of the methodology.

## 7.2.3. Refinements of MOBMAS

This section documents the refinements made to MOBMAS as a consequence of each expert's review. Only the refinements made to the steps and model definitions of MOBMAS are presented. Refinements made to the wording to improve MOBMAS' comprehensibility, coherence and expressiveness are not listed. The experts' comments are presented in Appendix F.

### *7.2.3.1. Refinements of MOBMAS as a result of the first expert review*

1. Re-define the origin of MOBMAS modelling notation. If the notation of a particular MOBMAS notational component is not applicable to be an extension of UML and AUML (i.e. when all valid UML and AUML extensibility mechanisms[80] are inapplicable), that notation should be documented as MOBMAS' *own* notation rather than UML's or AUML's extension. (The initial version of MOBMAS states that some notational components are an extension of UML/AUML while they are not). In the refined version of MOBMAS, the notation for the following notational components is re-defined as MOBMAS' own notation:
   - Role Diagram;
   - Agent Class Diagram[81];
   - Agent Relationship Diagram;
   - Agent Plan Template; and
   - Resource Diagram.

[80] Namely, "stereotype", "tagged value" and "constraint" (Object Management Group 2003).
[81] For the Agent Class Diagram, the notation of AUML Agent Class Diagram (Huget et al. 2003; Bauer et al. 2000; Bauer, 2001a; Bauer 2001b) had been considered. However, since much of the syntax and semantics of the AUML Agent Class Diagram notation has not yet been determined and finalized at the time of this research, it was not adopted.

2. Provide elaborate definition of the syntax and semantics of MOBMAS notation. Whenever UML or AUML notation was reused or extended (namely in Ontology Diagram, Agent Plan Diagram, Reflexive Rule Specification, Interaction Protocol Diagram, Agent-TC Interaction Diagram, Tuple-Centre Behaviour Diagram and MAS Deployment Diagram), the reused or extended notational elements were highlighted, followed by the clarification of how the syntax and semantics of UML/AUML notation have been reused or extended.

3. Document MOBMAS' conceptual framework, which defines the semantics of the main abstractions that underlie MOBMAS development process and model kinds. A diagram that shows the relationships between these abstractions was also added.

4. Extend the modelling of ontological relationships in the Ontology Model Kind to include the modelling of "composition" relationship. The difference in semantics between "composition" and "aggregation" was also highlighted.

5. Fix the following notational errors:
   - An idle state or a decision point in the Tuple-Centre Behaviour Diagram (which is basically a UML State Chart) should be represented as a circle ◯ and not a diamond ◇, to adhere to UML notation.
   - Each lifeline in the Interaction Protocol Diagram and Agent-TC Interaction Diagram of the Agent Interaction Model Kind should represent an agent instance and not an agent class. Accordingly, there should be a colon (**:**) in front of the agent class name in the rectangle above the lifeline.

#### *7.2.3.2. Refinements of MOBMAS as a result of the second expert review*

1. Revise step "Develop System Task Model" and step "Analyse Organisational Context" of the "Analysis" activity (Section 6.2) to avoid any overlap in the applicable conditions of these steps. Specifically, in the initial version of MOBMAS, step "Develop System Task Model" was recommended if

   > "…the target MAS is a processing application system that does not exhibit any specific and apparent human organisational structure",

while step "Analyse Organisational Context" was suggested if

> "… the target MAS aims to mimic or support a human-like organisation and the human organisational structure is clear".

In the refined version of MOBMAS, the developer was recommended to "Analyse Organisational Context" if

> "… the target application satisfies **all** of the following conditions.
>
> - The structure of the real-world organisation is known and clear.
> - The real-world organisational structure is well-established, not likely to change, and has proven or been accepted to be an effective way to function. Accordingly, it is desirable for the future MAS to mimic this existing structure."

and to "Develop System Task Model" if

> "…the target application does not satisfy all of the above conditions, or if the developer is unsure of whether the target application satisfies all of these conditions."

2. Change the naming convention of "system-task" from "*to do something*" (which sounds like an objective) to "*do something*" (which sounds more like an activity, e.g. "Receive user query" or "Get information from resources").

3. Extend step "Develop Agent Behaviour Model" of the "Agent Internal Design" and step "Develop Agent Interaction Model" of the "Agent Interaction Design" to consider the adoption, and allow the developer to adopt, various techniques for conflict resolution within agent classes and between agent classes. Recommended techniques included priority conventions (Ioannidis and Sellis 1989), constraint relaxation (Sathi and Fox 1989), arbitration (Steep et al. 1981) and evidential reasoning (Carver and Lesser 1995), negotiation (Sycara 1988), voting (Ephrati and Rosenschein 1991), and standardization and social rules (Shoham and Tennenholtax 1992).

4. Extend step "Select Interaction Mechanism" of the "Agent Interaction Design" activity to add a new criterion for the comparison between the direct interaction mechanism via ACL and the indirect interaction mechanism via tuplespace/tuple-centre: security control. This criterion examines how the two mechanisms differ in their control and implementation of security.

5. Extend step “Develop Agent Interaction Model” of the “Agent Interaction Design” activity to consider the modes of synchronisation in agent interaction. Two common modes of synchronisation were described: synchronous and asynchronous.

6. Extend step “Identify agent-environment interface requirements” of the “Architecture Design” activity to include:
   - description of the different types of interactions between an agent and its environment (namely, perception, effect and communication with other agents);
   - consideration of the differences between physical and virtual environments in term of the requirements of the agents’ sensor and effector; and
   - consideration of the requirements of inter-agent communications.

7. Allow the developer to adopt any ontology modelling languages for the Ontology Model Kind. UML was only used by MOBMAS for illustration purpose.

8. Mention the possibility of a complex tree structure in the System Task Diagram of System Task Model Kind.

9. Extend step “Specify Resources” of the “MAS Organisation Design” activity to discuss the distinction between those resources which belong *internally* to the MAS system, and those resources which exist externally and are available to agents in other systems.

10. Change the notation of AND/OR decomposition in the System Task Diagram of System Task Model Kind from the uncommon notation introduced by TROPOS (Figure 7.2) to the well-known notation of AND/OR graphs (Figure 7.1).

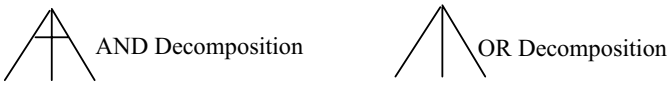



Figure 7.1 – Notation of AND/OR Graphs

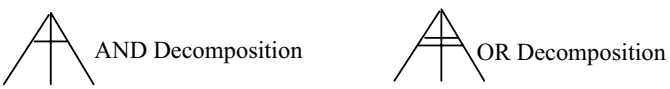



Figure 7.2 – TROPOS notation for AND/OR decomposition

# 7.3. APPLICATION OF MOBMAS

After MOBMAS was refined by Step 1 – "*Obtain expert reviews*" (Section 7.2), it was further evaluated and refined by Step 2 – "*Using MOBMAS on a test application*" (cf. Section 4.6). In this step, MOBMAS was given to two external developers who sequentially applied the methodology on a "Peer-to-Peer Information Sharing" application (Appendix H) and evaluated the methodology according to their experience of usage. The obtained feedback was investigated to refine MOBMAS into its final version as presented in Chapter 6. Compared to the expert reviews, the application of MOBMAS aimed to evaluate the methodology at a more detailed and elaborate level, specifically at *every step* and *every model kind* of the methodology.

Section 7.3.1 firstly describes the procedures of MOBMAS application. Section 7.3.2 then provides biographic information about each external developer. Section 7.3.3 finally reports on the refinements of MOBMAS as a consequence of the developers' evaluation.

## 7.3.1. Application procedures

Each of the two developers was provided with a full documentation of MOBMAS and requested to:

- apply it on a test application;
- produce a design of MAS by using the methodology and generate a number of major models to illustrate this design; and
- evaluate the methodology based on the usage of the methodology. Each developer was asked to:
  - identify the strengths of the methodology (optional);
  - identify the areas for improvement of the methodology;
  - recommend how to improve these areas (optional);
  - rate the "ease of understanding" and "ease of following" of each step of the methodology on a High-Medium-Low scale; and
  - rate the "ease of understanding" of each model kind of the methodology on a High-Medium-Low scale.

To help the developers to systematically and thoroughly record their evaluation, an evaluation form was designed. The form consists of five parts, each collecting the developers' evaluation of each activity of the methodology. The full evaluation form can be found in Appendix G.

MOBMAS was applied by the two developers in a *sequential* order. Evaluation from the first developer was obtained and used to refine MOBMAS before the second developer was asked to apply and evaluate this refined version. Reasons for this sequential application process are the same as those stated for the expert reviews in Section 7.2.1:

- to prevent the possibility of the two developers identifying the same areas for improvement of MOBMAS; and
- to help identifying new areas of improvement that potentially arose from the refinement of MOBMAS after the first application.

A face-to-face meeting was organised with each developer at the end of each application to get a walkthrough of his feedback. After MOBMAS was refined in accordance with each developer's feedback, follow-up meetings and email communication were carried out with the developer to ensure that he was satisfied with the refinement made to the methodology. In addition, the refinements made to MOBMAS as a result of the second developer's feedback were also discussed with the first developer to ensure that no conflicts of opinions incurred.

## 7.3.2. Developers' biography

The two developers who participated in the study were Dr. Ghassan Beydoun (first developer) and Dr. Cesar Gonzalez-Perez (second developer). Both developers have extensive knowledge and experience in software engineering in general and AO software engineering in particular.

- Dr. Ghassan Beydoun is currently a post-doc research fellow at the University of New South Wales, Sydney. He is working on a project funded by the Australian Research Council, entitled "FAME – Futuristic Agent-Oriented Method Engineering". The project aims to develop a repository of AOSE method components that can be used to build a MAS development methodology that suits

the application or organisation at hand. Dr. Beydoun's expertise also covers the realm of ontology, given his active research involvement in the areas of knowledge engineering, knowledge management, knowledge representation and acquisition. Although Dr Beydoun's research's location is the same as that of the conductor of this PhD study, he was not involved in the design and execution of this research in any way, except for being an independent tester.

- Dr. Cesar Gonzalez-Perez is currently a post-doc research fellow at the University of Technology, Sydney. He is another participant in the FAME project funded by the Australian Research Council, "Futuristic Agent-Oriented Method Engineering". His extensive knowledge and experience on software engineering include the involvement in developing the "International Standard Metamodel for Software Development Methodologies" (in progress) and "OPEN/Metis" - an integral, object-oriented software development framework. Dr. Gonzalez-Perez is also the founder and former technical director of NECO, a company based in Spain specialising in software development. While working at the same university as the two expert reviewers, Prof. Brian Henderson-Sellers and Prof. Mary-Anne Williams (c.f. Section 7.2.2), Dr Beydoun did not have any communication with these reviewers with regard to MOBMAS. His testing of the methodology was completely independent and unbiased by the preceeding expert reviews.

### 7.3.3. Refinements of MOBMAS

This section documents the refinements made to MOBMAS as a result of each developer's comments. Only the refinements made to the steps and model definitions of MOBMAS are presented. Refinements made to the writing style or wording to improve MOBMAS' comprehensibility, coherence and expressiveness are not listed. The two developers' comments are presented in Appendix G. These comments were recorded via an evaluation form. Appendix H documents the "Peer-to-Peer Information Sharing" application on which MOBMAS was applied, and the major models created by each developer for the application.

### *7.3.3.1. Refinements of MOBMAS as a result of Developer 1's comments*

1. Make step "Develop System Task Model" a compulsory step in the "Analysis" activity (Section 6.2). Step "Analyse Organisational Context", however, remained optional. The applicable condition of the latter step was left unchanged.

2. Extend step "Develop Role Model" of the "Analysis" activity to highlight the need for spiral, iterative development of System Task Model and Role Model.

3. Extend step "Develop Agent Class Model" of the "MAS Organisation Design" and step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity to provide support for the modelling of agent class' dynamics.

4. Extend step "Develop Ontology Model" of the "Analysis" activity to include the discussion of the use of ontologies to validate System Task Model and Role Model.

5. Extend step "Identify Ontology Manage Roles" of the "Analysis" activity to describe an alternative design approach where no ontology manager is needed. Advantages and disadvantages of this alternative design were documented.

6. Move step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity in front of step "Develop Agent Class Model" (of the same activity) to make the steps easier to follow.

7. Extend step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity to consider the "hybrid" style of organisational structure.

8. Merge steps "Identify Resource Application Ontologies" and "Develop Resource Application Ontologies" of the "MAS Organisation Design" activity into one step "Extend Ontology Model To Include Resource Application Ontologies".

9. Make steps “Specify resources” and “Extend Ontology Model To Include Resource Application Ontologies” of the “MAS Organisation Design” activity optional, because these steps are only applicable to heterogeneous systems.

10. Remove step “Develop System Overview Diagram” from the “MAS Organisation Design” activity. Instead, a sub-step “Update Agent Class Model” was defined within step “Specify resources” of the same activity to show the overall architecture of MAS.

11. Simplify Agent Class Diagram by omitting various compartments which were initially created to store the names of Agent Plan Templates for an agent class. The Agent Behaviour Model Kind was instead used to capture the specification of these Agent Plans.

12. Rename the relationships between agent classes in the Agent Relationship Diagram from “association” to “acquaintance” to clarify that these relationships represent interaction pathways.

13. Extend step “Specify Agent Class’ Belief Conceptualisation” of the “Agent Internal Design” activity to discuss the dynamics of the belief conceptualisations and belief state of agents at run-time.

14. Revise step “Develop Agent Behaviour Model” of the “Agent Internal Design” activity to re-define the information to be specified in the Agent Plan Template. Instead of modelling the exact sequences of actions to be performed by an agent to achieve an agent-goal, an Agent Plan Template should model a repository of *potential* actions and/or sub-agent-goals that *may* be selected by the agent to perform to achieve the target agent-goal at run-time. The selection of which actions to perform, and the sequencing of these actions, are delegated to the planners or reasoners or means-end analysers that are built in to the agent architecture/platform. The sequencing of sub-agent-goals may be modelled for an Agent Plan if this sequence is fixed and determinable at design time.

15. Extend sub-step "Develop Agent Plan Templates" of step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity to specify:
    - "commitment strategy" for each Agent Plan Template; and
    - "conflict resolution strategy" to deal with conflicts between agent-goals (if any).

16. Extend step "Develop Agent Interaction Model" of the "Agent Interaction Design" step to:
    - address the identification of "communicative actions" in the Agent Plan Templates and Reflexive Rule Specifications; and
    - highlight the need for consistency checking between Role Model, Agent Class Model and Agent Interaction Model (i.e. checking whether all the acquaintances between roles and agent classes have been captured, and whether all the authority relationships previously defined for roles and agent are still valid).

17. Extend step "Identify agent-environment interface requirements" of the "Architecture Design" activity to consider two other characteristics of the perception and effect activities of agent classes: the degree of complexity of the perceptual inputs and effect outputs, and the interaction with human user.

18. Revise step "Select agent architecture" of the "Architecture Design" activity to remove the statement that all agent architectures listed in MOBMAS are commercial products. Some agent architectures are actually available to only the academic realm (e.g. SOAR).

19. Extend step "Specify MAS Infrastructure Facilities" of the "Architecture Design" activity to address the case when MASs are closed and run on a single platform.

20. Remove the notation for agent mobility from the MAS Deployment Diagram, since MOBMAS does not address mobile agents. However, the capability of the MAS Deployment Diagram to model agent mobility is noted in a footnote.

### *7.3.3.2. Refinements of MOBMAS as a result of Developer 2's comments*

1. Provide an explicit definition for the term "(MAS) *environment*".

2. Use the term "*model kind*" to refer to the definition of a class of models. The term "*model*" is used to mean a particular work product that is created by the developer at the design time.

3. Use the term "*activity*" to refer to a group of steps in MOBMAS instead of the term "*phase*" as used in the earlier version of MOBMAS. This change helps to avoid the implication of temporal ordering between the activities (MOBMAS adopts an iterative and incremental development life cycle where the developer is allowed to move back and forth across activities).

4. Refine the diagram that shows MOBMAS' conceptual framework (i.e. Figure 6.1) to model the cardinality indicators of the relationships between MOBMAS' abstractions.

5. Rename the notational components of the Agent Behaviour Model Kind as "Agent Plan Template" and "Reflexive Rule Specification" instead of "Agent Plan" and "Reflexive Courses of Actions".

6. Use the term "authority relationship" to refer to the organisational relationships between roles (e.g. peer-to-peer relationship or superior-subordinate relationship) instead of the term "control regime" as used in the earlier version of MOBMAS.

7. Use the term "information sources" to refer to non-agent resources instead of "knowledge sources". The former term is more general than the latter because it encompasses resources such as databases and web servers that are not necessarily knowledge sources.

8. Use the term “belief conceptualisation” to refer to the conceptualisation of an agent class’ beliefs instead of the term “belief set”. The latter may be misunderstood as a set of beliefs rather than a conceptualisation of beliefs.

9. Revise step “Develop System Task Model” of the “Analysis” activity to delegate the identification of system-tasks to a separate Requirements Engineering effort (which is not part of MOBMAS) instead of identifying system-tasks from system goals as in the earlier version of MOBMAS. With this change, MOBMAS allows the developer to identify system-tasks from any constructs, thereby supporting the development of MASs that do not have clear goals.

10. Revise step “Analyse Organisational Context” of the “Analysis” activity to remove the use of the term “*real-world*” when referring to the structure of the MAS’ organisational context. This term may mislead the readers into thinking of only the “physical” world.

11. Refine the techniques for step “Develop Role Model” of the “Analysis” activity by:
    - making the heuristic “*strong internal coherence and loose coupling*” the primary criterion for grouping system-tasks to roles;
    - considering the mapping of system-tasks to roles when the system-tasks are fully and partially decomposed;
    - using the term “joint task” instead of “social task” to refer to a task that is collectively carried out by multiple parties. The term “joint task” helps to avoid any improper implications that may come with the term “social task”, e.g. the existence of subjective or intersubjective spaces; and
    - documenting the cardinality of the mapping relationship between system-tasks and role-tasks.

12. Revise Role Diagram and Agent Class Diagram to remove the use of guillemets in the label of each compartment in these diagrams. This removal helps to avoid any potential confusion with the UML stereotypes.

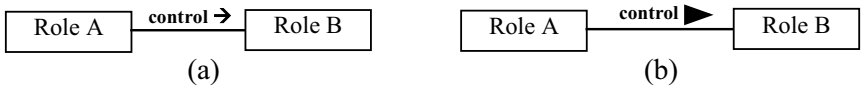
13. Revise step "Develop Ontology Model" of the "Analysis" activity to:
    - re-define an Application ontology as a *specialisation* of generic Domain ontologies and Task ontologies, not a composition of these two ontologies as inaccurately suggested in the earlier version of MOBMAS;
    - document the need for specifying cardinality of the relationships between ontological concepts in the Ontology Diagram;
    - provide examples for each type of ontological relationships; and
    - recommend the developer to study the extensive literature on "ontological mappings" for more options on the semantic correspondences between concepts besides the three basic correspondences – "equivalent", "subsumes" and "intersects" which are listed in MOBMAS.

14. Use UML "dependency relationship" to depict ontological mappings in the Ontology Diagram instead of UML "association relationship".

15. Refine step "Identify Ontology Management Role" of the "Analysis" activity to note that "ontology manager" agent classes may have been provided by the development platform as built-in components. These agent classes therefore do not have to be designed from scratch.

16. Extend step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity to add "modularity" and "non-functional requirements support" as additional factors to be considered when determining the organisational structure for the target MAS.

17. Change the notation of the superior-subordinate relationship between roles in the Role Diagram from Figure 7.3a to Figure 7.3b to improve its comprehensibility.




(a) (b)

Figure 7.3 – Old (a) and new (b) notation for superior-subordinate relationship between roles in Role Diagram

18. Revise step "Develop Agent Class Model" of the "MAS Organisation Design" activity to:
    - add "modularity" and "efficiency consideration" to the list of factors to be considered when assigning multiple roles to a single agent class; and
    - remove the consideration of the computation complexity of each agent class and the available processing power of each node (where the agent classes will be run) when assigning multiple roles to a single agent class. This removal is necessary because it is often unknown at the design time as to which hardware platform will implement the MAS system.

19. Modify Agent Class Diagram to model each agent class' dynamics as a property of the agent class instead of the property of the roles played by the agent class.

20. Modify Resource Diagram to remove the modelling of "communication properties" from the description of each resource, since these details (e.g. networking protocol and network address) are normally not available at the design time.

21. Revise step "Extend Ontology Model To Include Resource Application Ontologies" of the "MAS Organisation Design" activity to:
    - note that Resource Application ontologies can only be derived from the conceptual schema of the information stored in the resource, *not* directly corresponding to this conceptual schema; and
    - remove the guideline that, if a Resource Application ontology coincides with a MAS Application ontology, the MAS Application ontology can be used in place of the Resource Application ontology. Such a guideline will hinder the system's ability to accommodate future changes in the conceptual structure of the resource.

22. Extend step "Specify Agent Class' Belief Conceptualisation" of the "Agent Internal Design" activity to discuss the different ways by which the changes in an agent's belief conceptualisation can be propagated to other agents at run-time.

23. Revise step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity to rename the two major styles of agent behaviour as "planning" and "reflexive" instead of "proactive" and "reactive". This change helps to avoid any potential misunderstanding of the nature of these two styles. ("Proactive" and "reactive" may be understood as the two different modes of triggering of a piece of behaviour, while "planning" and "reflexive" refer to the two different levels of complexity in agent reasoning exhibited in a piece of behaviour).

24. Extend step "Develop Agent Behaviour Model" of the "Agent Internal Design" activity and step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity to address the case when the datatypes of the variables/arguments are "basic" datatypes such as String and Integer. These datatypes do not need to be defined in a MAS Application ontology.

25. Revise the "Agent Interaction Design" activity to re-classify the two major mechanisms of agent interactions into "direct" and "indirect" mechanisms, instead of "direct" and "tuplespace/tuple-centre" mechanisms as in the earlier version of MOBMAS. The refined version of MOBMAS acknowledged that the tuplespace/tuple-centre interaction method is only one of many methods of indirect interaction, even though it is the most commonly used method.

26. Rename the last activity of MOBMAS as "Architecture Design" instead of "Deployment Design" as in the earlier version of MOBMAS. This new name describes more accurately the steps specified in this activity.

27. Refine step "Select Agent Architecture" of the "Architecture Design" activity by merging the criterion "*Size of knowledge base*" (which was suggested for consideration when selecting agent architecture for the target MAS) into criterion "*Support for scalability of agent*".

28. Introduce a new model kind, Resource Model Kind, which is depicted by the Resource Diagram, and extend the Architecture Model Kind to include Agent-Environment Interface Requirements Specification and Infrastructure Facilities Specification notational components. This modification removed the need for an

Environment Model Kind, which was originally defined to encompass the three notational components Resource Diagram, Agent-Environment Interface Requirements Specification and Infrastructure Facilities Specification.

29. Extend Agent Relationship Diagram to model agent classes' instantiation cardinality (via the use of annotations next to each agent class' name). This extension removed the need for a separate Agent Instantiation Diagram as recommended in the earlier version of MOBMAS.

30. Refine step "Identify agent-environment interface requirements" of the "Architecture Design" activity to note that, when interacting with a physical environment, the sensor and effector of an agent needs to able to connect to, and control, the drivers of hardware components such as thermostats, infra-red sensor, wheels and grippers.

# 7.4. FEATURE ANALYSIS OF MOBMAS

After MOBMAS was refined into its final version by Step 2 – "*Using MOBMAS on a test application*" (Section 7.3), Step 3 – "*Performing a feature analysis of MOBMAS*" was conducted on this final version. The objectives of this step were to (cf. Section 4.6):

- verify whether MOBMAS is able to achieve its goal, that is, to provide support for ontology-based MAS development and various other important AOSE methodological requirements that are documented in Section 5.4.3 (which include the required *features*, *AOSE steps* and *modelling concepts*);
- reveal the origin of MOBMAS' techniques and model definitions; and
- compare MOBMAS with the existing AOSE methodologies in term of specific evaluation criteria. These criteria had previously been used to evaluate the existing AOSE methodologies in Section 5.4. The comparison also highlights the various ontology-related strengths of MOBMAS, which are not provided, or provided to a lesser extent, by the existing AOSE methodologies.

Section 7.4.1 firstly examines the support of MOBMAS for its methodological requirements (including the support for ontology-based MAS development) and reports

on the origin of MOBMAS techniques and model definitions. Section 7.4.2 then compares MOBMAS with the existing AOSE methodologies.

## 7.4.1. MOBMAS' Support for Methodological Requirements

The methodological requirements of MOBMAS consist of the **features**, **steps** and **modelling concepts** that are desirable to the development process, techniques and model kinds of MOBMAS. These requirements have been shown in Tables 5.33, 5.34 and 5.35 of Chapter 5 respectively.

The clarification of MOBMAS' support for the required ***features*** is presented in Table 7.4. Feature "*Support for ontology-based MAS development*", in particular, is examined in detail in Section 7.4.1.1.

The support of MOBMAS for the required ***steps*** is documented in Table 7.5. The table shows the correspondences between MOBMAS' *actual* steps and the required AOSE steps (listed in Table 5.34). The various AOSE steps that were previously identified as desirable *ontology-related* steps (cf. Section 5.5) have been highlighted with an adornment *(O)*.

Table 7.6 lastly clarifies MOBMAS' support for the required *modelling concepts*. For each concept, the table shows the name of the model kind(s) of MOBMAS which represents or captures the concept.

In all three tables (Tables 7.4, 7.5 and 7.6), column "*Origins of MOBMAS techniques and modelling definitions*" provides an account of:

- the techniques and/or model definitions that have been **reused** by MOBMAS from the existing AOSE methodologies (i.e. "*REUSE*" subheading). The names of these source methodologies are shown under the "*Sources*" subheading. They are a subset of the "*potential sources of techniques and model definitions*" previously identified in Tables 5.33, 5.34 and 5.35 of Chapter 5;

- the **enhancements** that were made by MOBMAS on the reused techniques and/or model definitions (i.e. "*ENHANCEMENT*" subheading); and/or

- the **new** techniques and/or model definitions that were developed by MOBMAS and not found in the existing AOSE methodologies (i.e. "*NEW*" subheading). The need for some of these techniques and model definitions has previously been identified in Tables 5.33, 5.34 and 5.35 of Chapter 5.

Table 7.4 – MOBMAS' support for the required features (cf. Table 5.33)

| ***Required features of MOBMAS development process*** | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 1. Specification of a system development lifecycle | MOBMAS adopts an iterative and incremental life cycle | **REUSE:**<br>Allowing the development process to be iterative and incremental life cycle within and across all phases. *Sources: MASE, HLIM, PROMETHEUS, TROPOS and suggestions of survey's participants (Section 5.3).* |
| 2. Support for verification and validation | MOBMAS performs consistency checking between many of its model kinds (e.g. between Agent Class Model Kind, Agent Behaviour Model Kind, Agent Interaction Model Kind and Ontology Model Kind) | **REUSE:**<br>Supporting verification and validation via consistency checking of model kinds. *Sources: MASE, MASSIVE, INGENIAS, PROMETHEUS, PASSI, ADELFE and TROPOS.*<br>**ENHANCEMENT:**<br>Providing specific guidelines for the consistency checking between particular model kinds |
| 3. Specification of steps for the development process | See Table 7.5 | See Table 7.5 |
| 4. Specification of model kinds and/or notational components | See Table 7.6 | See Table 7.6 |
| 5. Specification of techniques and heuristics for performing each process step and producing each model kind | See Table 7.5 | See Table 7.5 |
| 6. Support for refinability | All models in MOBMAS are iteratively refined and expanded throughout of the system development process (e.g. Role Model, Agent Class Model, Environment Model and Ontology Model) | **REUSE:**<br>Iteratively refining and expanding the different models of the methodology throughout different phases of the development process. *Sources: All existing methodologies except for SODA.* |

| *Required features of MOBMAS model definitions* | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 1. High degree of completeness/expressiveness | MOBMAS model kinds can represent the target system from both static aspects such as agent/role acquaintances, agent internal structure and system structure, and from dynamic aspects such as interaction protocols and agent behaviour. MOBMAS model kinds can also capture/represent a wide variety of concepts (cf. Table 7.3) | **REUSE:** Providing a comprehensive set of model kinds that represent the target system from both static aspects and dynamic aspects. The model kinds should also be able to capture/represent a wide variety of concepts. *Sources: MASE, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE, COMOMAS, MAS-CommonKADS and TROPOS.* |
| 2. High degree of formalisation/preciseness | MOBMAS clearly defines the semantics and syntax of its notation. Examples are provided for all model kinds and notational components | **REUSE:**<br>- Providing explanation of the semantics and syntax of the modelling notation. *Sources: MASE, GAIA, PROMETHEUS and TROPOS.*<br>- Adopting UML and AUML notation which are familiar to most system developers. *Sources: PASSI and ADELFE.* |
| 3. Provision of guidelines/logics for model derivation | MOBMAS provides techniques for transforming between its model kinds and notational components (e.g. from System Task Model Kind to Role Model Kind, from Role Model Kind and Ontology Model Kind to Agent Class Model Kind, and from Role Diagram and Resource Diagram to Agent Relationship Diagram) | **REUSE:** Allowing the developer to easily transform between the model kinds and notational components, or at least partially create a model kind or notational component from the information present in another. *Sources: All existing methodologies except CASSIOPEIA and MAS-CommonKADS.*<br>**ENHANCEMENT:** Suggesting specific guidelines for the transformation between particular model kinds and notational components |
| 4. Guarantee of consistency | MOBMAS provides rules for consistency checking among many of its model kinds (e.g. between Agent Behaviour Model Kind and Ontology Model Kind, between Agent Interaction Model Kind and Ontology Model Kind, between Agent Class Model Kind and Agent Behaviour Model Kind, and between System Task Model Kind, Role Model Kind and Ontology Model Kind) | **REUSE:** Documenting the need for the different model kinds and notational components which represent the same modelling concept(s) to be consistent with each other. *Sources: MASE, SODA, GAIA, INGENIAS, HLIM, PROMETHEUS, PASSI, ADELFE and TROPOS.*<br>**ENHANCEMENT:** Providing specific guidelines for the consistency checking between particular model kinds and notational components |

| *Required features of MOBMAS model definitions* | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 5. Support for modularity | MOBMAS models agent classes as entities that encapsulate roles, goals, agent plan templates, reflexive rule specifications and ontologies) | **REUSE:**<br>- Modelling agent classes as entities that encapsulate roles. *Sources: MASE, MASSIVE, SODA, BDIM, GAIA, MESSAGE, INGENIAS and CASSIOPEIA.*<br>- Modelling agent classes as entities that encapsulate goals. *Sources: TROPOS, MESSAGE, PASSI, HLIM, MEI and INGENIAS.*<br>**NEW:**<br>- Modelling agent classes' behaviour via modular agent plan templates and reflexive rule specifications<br>- Modelling agent classes' belief conceptualisation as being composed of ontologies |
| 6. Manageable number of concepts in each model kind and each notational component | Each MOBMAS model kind and notational component represents a manageable number of concepts | **REUSE:**<br>Not capturing too many concepts in one single model kind. Adopting well-known notation. If new notation is used, it should have simple semantics and syntax. *Sources: All existing methodologies except SODA and INGENIAS.* |
| 7. Model kinds expressed at various level of abstraction and detail | Many model kinds of MOBMAS can be developed at various levels of detail (e.g. Agent Class Model Kind, Role Model Kind and Ontology Model Kind) | **REUSE:**<br>- Developing Role Model Kind at various levels of detail. *Sources: GAIA and MASE.*<br>- Developing Agent Class Model Kind at various levels of detail. *Source: BDIM.*<br>**NEW:**<br>Developing Ontology Model Kind at various levels of detail |
| 8. Support for reuse | MOBMAS allows the developer to reuse various modelling elements such as role patterns, protocol templates, organisational structures and ontologies | **REUSE:**<br>Making it possible to reuse the following modelling elements: role patterns, protocol templates and organisational structure. *Sources: MASE, MASSIVE, GAIA, BDIM, PASSI and MAS-CommonKADS.*<br>**NEW:**<br>Promoting reusability with the use of ontology (cf. Section 7.4.2.4) |

| *Agent properties required to be captured/represented by MOBMAS model kinds* | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 1. Autonomy | MOBMAS models agent classes as entities with purposes (represented as agent roles and goals) and entities with internal control (represented as agent beliefs, plans and reflexive rules) | **REUSE:**<br>- Modelling agent classes as entities with purposes, which are represented as agent roles and goals. *Sources: MASE, MASSIVE, SODA, GAIA, PROMETHEUS, PASSI, INGENIAS and ADELFE.*<br>- Modelling agent classes as entities with internal control, which is captured via agent beliefs/knowledge and plans. *Sources: PROMETHEUS, MESSAGE, MAS-CommonKADS, TROPOS, BDIM, HLIM, MEI, COMOMAS and INGENIAS.* |
| 2. Adaptability | MOBMAS discusses the possibility of agent's ontologies being modified and propagated to other agents, e.g. as a result of learning, and mentions the need to consider learning (amongst other required behaviour capabilities of an agent) when selecting an agent architecture | **REUSE:**<br>Supporting adaptability by considering learning in selecting agent architecture. *Sources: MASSIVE.*<br>**NEW:** Discussing the possibility of agent's ontologies being modified and propagated to other agents, e.g. as a result of learning. |
| 3. Cooperative behaviour | MOBMAS supports the modelling of agent acquaintances and interaction protocols | **REUSE:**<br>Capturing cooperative behaviour via agent acquaintances and interaction protocols. *Sources: All existing methodologies.*<br>**NEW**:<br>Supporting both direct interactions via ACL and indirect interactions via tuplespace/tuple-centre |
| 4. Inferential capability | MOBMAS support s the modelling of agent plan templates and reflexive rules which help the agents to determine what to do to achieve its active goals at run-time | **REUSE:**<br>Supporting inferential capability by developing agent plans. *Sources: PROMETHEUS, BDIM, HLIM, MEI and TROPOS.* |
| 5. Knowledge-level communication ability | MOBMAS supports the modelling of ACL communication messages between agents, which are based upon speech-act performatives | **REUSE:**<br>Modelling agents' exchanged messages using ACL speech-acts performatives. *Sources: BDIM, ADELFE, HLIM and MESSAGE.*<br>**NEW:** Formulating agents' exchanged messages in terms of shared ontologies |

| *Agent properties required to be captured/represented by MOBMAS model kinds* | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 6. Reactivity | MOBMAS models events and agents' behaviour to react to these events | **REUSE:**<br>- Modelling "events" that incur during agent interactions and in agent internal processing. *Sources: MASE, MESSAGE, INGENIAS, PASSI, ADELFE and MAS-CommonKADS.*<br>- Developing agent behaviour to react to events. *Sources: PROMETHEUS, BDIM, HLIM, COMOMAS and TROPOS.* |
| 7. Deliberative behaviour | MOBMAS models agent classes as entities with purposes (represented as agent goals) and supports the modelling of agent plans for accomplishing these purposes in a deliberative manner | **REUSE:**<br>- Capturing deliberative behaviour via agents' goals. *Sources: BDIM and HLIM.*<br>- Modelling agent plans to achieve the agent-goals. *Sources: BDIM, HLIM, TROPOS and PROMETHEUS.* |
| ***Required features of MOBMAS as a whole*** | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
| 1. Support for open systems | MOBMAS facilitates the operation of newly-added agents at run-time by explicitly modelling the resources offered by the target MAS environment and allowing the specification of a "resource broker" agent which brokers the available resources to the newly added agents. MOBMAS also allows for the use of indirect interaction mechanisms which are particularly suitable to open systems. MOBMAS also offers an option to conceptualize the agent interaction protocols during the design time. This explicit conceptualization will allow any new agents to join the pre-existing conversations, and allow the interaction protocols to change over time. | **REUSE:**<br>Modelling resources of the MAS environment. *Sources: SODA and GAIA*.<br>**NEW:**<br>- Considering the design of a "*Resource Broker*" role/agent class<br>- Considering the adoption of an indirect interaction mechanism, which is particularly suitable to open MASs<br>- Considering system openness as a criterion when selecting the organisational structure style for target MAS |
| 2. Support for dynamic systems | MOBMAS supports the dynamic role-playing behaviour of agents and offers model kinds to capture/represent this dynamics. MOBMAS also considers the issue of system dynamics when selecting the interaction mechanism for the target MAS. | **REUSE:**<br>Supporting system dynamics by allowing for dynamic assignment of roles to agents. *Sources: MASSIVE, HLIM, MASE and PASSI.*<br>**NEW:**<br>Discussing the modelling of agent classes' dynamics in the Agent Class Model Kind and Agent Interaction Model Kind. |

| *Required features of MOBMAS as a whole* | **Support by MOBMAS** | **Origins of MOBMAS' techniques and model definitions** |
|---|---|---|
| 3. Support for ontology-based MAS development | See Section 7.4.1.1 | **REUSE:**<br>- Addressing the modelling of MAS Application ontologies. *Sources: MAS-CommonKADS, PASSI, MESSAGE and MASE.*<br>- Formulating agents' exchanged messages in accordance with a shared ontology. *Source: MASE and PASSI.*<br>**NEW:**<br>- Discussing the modelling of Resource Application ontologies<br>- Considering the use of ontologies to verify other model kinds of MOBMAS, e.g. System Task Model Kind, Agent Behaviour Model Kind and Agent Interaction Model Kind<br>- Modelling agents' belief conceptualisation as being composed of ontologies<br>- Modelling Agent Plan Templates, Reflexive Rule Specification and exchanged messages in terms of concepts defined in ontologies<br>- Discussing the mapping between ontologies<br>- Specifying the steps when the developer should start constructing each type of ontology in the MAS development process, and providing some guidelines on the construction of the ontologies. *Source: suggestions of survey's participants in Chapter 7.* |
| 4. Support for heterogeneous systems | MOBMAS deals with the modelling of non-agent resources apart from agent classes. The agent classes themselves are allowed to range from purely planning agents to purely reflexive agents, each with a different agent behavioural style. | **REUSE:**<br>Supporting system heterogeneity by addressing the modelling of non-agent software components. *Sources: INGENIAS, PROMETHEUS, GAIA and MASSIVE.*<br>**NEW:**<br>- Discussing the modelling of Resource Application Ontologies to conceptualise the information and/or services provided by each resource<br>- Addressing the design of both planning agents and reflexive agents |

Table 7.5 – MOBMAS' support for the required steps (cf. Table 5.34)

| ***Problem Domain Analysis steps*** | **Support by MOBMAS?** | **Origins of MOBMAS' techniques (for performing steps)** |
|---|---|---|
| 1. Identify system functionality (O) | See step "Develop System Task Model" of the "Analysis" activity | **REUSE:**<br>Incrementally decomposing system-tasks into sub-system-tasks. *Source: COMOMAS.*<br>**NEW:**<br>- Considering full versus partial decomposition of system-tasks<br>- Considering conflicts between system-tasks<br>- Using Ontology Model to validate System Task Model |
| 2. Identify roles | See step "Develop Role Model" of the "Analysis" activity | **REUSE:**<br>- Using system-tasks as inputs to identify roles. *Sources: MASSIVE and BDIM.*<br>- Mapping a "joint" system-task to a group of roles. *Source: SODA.*<br>- Applying the principle of "strong internal coherence and loose coupling" to group system-tasks to roles. *Sources: MASSIVE and PROMETHEUS.*<br>- Grouping different system-tasks to one role if the tasks share a lot of common data or interact intensely with each other. *Source: PROMETHEUS.*<br>- Assigning different system-tasks to different roles if they need to be executed on different processors and if there exist security and privacy requirements.<br>- Analysing the structure of the MAS' organisational context (if applicable) to identify roles. *Source: GAIA.*<br>**NEW:**<br>- Considering the mapping of system-tasks to roles when the system-tasks are fully and partially decomposed.<br>- Considering the identification of "Wrapper" roles, "Resource Broker" roles and "Ontology Manager" role. |
| 3. Identify agent classes | See step "Develop Agent Class Model" of the "MAS Organisation Design" activity | **REUSE:**<br>- Identifying agent classes from roles. *Sources: MASE, GAIA, MASSIVE, PASSI and suggestions of survey's participants in Section 5.3.*<br>- Applying one-to-one mapping from roles to agent classes. *Sources: MASE and GAIA.*<br>- Grouping roles to one agent class if roles interact intensely. *Sources: MASE and BDIM.*<br>- Evaluating the coherence of each agent class by checking whether the agent class can be easily described by a single name without any conjunctions. *Source: PROMETHEUS.*<br>**ENHANCEMENT:** Considering various modularity and efficiency issues in the assignment of roles to agent classes |

| ***Problem Domain Analysis steps*** | **Support by MOBMAS** | **Origins of MOBMAS' techniques (for performing steps)** |
|---|---|---|
| 4. Model domain conceptualisation (O) | See step "Develop Ontology Model" of the "Analysis" activity | **REUSE:**<br>Incrementally adding domain concepts as they arise throughout the MAS development process. *Source: MESSAGE.*<br>**NEW:**<br>Identify generic Domain ontologies and Task ontologies that can serve as inputs to the construction of MAS Application ontologies. |
| ***Agent Interaction Design steps*** | **Support by MOBMAS** | **Origins of MOBMAS' techniques (for performing steps)** |
| 1. Specify acquaintances between agent classes | See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity | **REUSE:**<br>Identifying agent acquaintances from acquaintances between roles. *Source: CASSIOPEIA.*<br>**NEW:** Deriving agent acquaintances from communicative actions in Agent Plan Templates and Reflexive Rule Specifications. |
| 2. Define interaction protocols | See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity | **REUSE:**<br>- Specifying coordination rules to govern agents' interactions for achieving identical agent-goals. *Source: SODA.*<br>- Allowing for reuse of interaction protocol templates from FIPA. *Sources: PROMETHEUS, MAS-CommonKADS and PASSI.*<br>- Indicating the ontology used to govern each interaction protocol. *Source: PASSI.*<br>**NEW:**<br>Considering both direct interactions via ACL and indirect interaction mechanism via tuplespace/tuple-centre |
| 3. Define content of exchanged messages (O) | See step "Develop Agent Interaction Model" of the "Agent Interaction Design" activity | **REUSE:**<br>- Specifying speech-act performatives for each exchanged message. *Sources: PASSI and MAS-CommonKADS.*<br>- Specifying arguments to be passed within each exchanged message. *Sources: MASE, PASSI and MAS-CommonKADS.*<br>- Formulating exchanged messages using concepts defined in MAS Application ontologies. *Source: PASSI.*<br>**ENHANCEMENT:**<br>- Providing extensive techniques for the specification of exchanged messages in each interaction mechanism (i.e. direct interaction via ACL and indirect interaction via tuplespace/tuple-centre)<br>- Specifying explicit rules to verify the content of the exchanged messages against the concepts defined in ontologies. |

| *Agent Internal Design steps* | **Support by MOBMAS** | **Origins of MOBMAS' techniques (for performing steps)** |
|---|---|---|
| 1. Specify agent architecture | See step "Select Agent Architecture" of the "Architecture Design" activity | **REUSE:**<br>Proposing a set of criteria for selecting agent architecture. *Source: MASSIVE.*<br>**ENHANCEMENT:**<br>Proposing a few additional criteria for selecting agent architecture that are not listed in MASSIVE, namely "required agent behavioural capabilities", "complexity of sensor input", "support for scalability" and "agent-environment interaction requirements".<br>**NEW:**<br>Listing various existing architectures from which the developer can consider for reuse |
| 2. Define agent informational constructs (O) | See step "Specify Agent Class' Belief Conceptualisation" of the "Agent Internal Design" activity | **REUSE:**<br>Discussing the differentiation between Belief conceptualisation and Belief States. *Source: BDIM.*<br>**NEW:**<br>- Defining agents' belief conceptualisation as being composed of ontologies<br>- Providing detailed techniques for determining which ontologies each agent should commit |
| 3. Define agent behavioural constructs (O) | See steps "Specify Agent Goals", "Specify Events" and "Develop Agent Behaviour Model" of the "Agent Internal Design" activity | **REUSE:**<br>Developing agent plans to achieve agent-goals. *Source: BDIM.*<br>**ENHANCEMENT:**<br>- Providing guidelines to ensure consistency between Agent Plan Templates, Reflexive Rule Specification and Agent Belief Conceptualisation<br>- Considering both planning behaviour and reflexive behaviour.<br>**NEW:**<br>Specifying the states and actions of Agent Plan Templates and Reflexive Rule Specification in accordance with the concepts defined in ontologies. |
| ***Overall System Design steps*** | **Support by MOBMAS** | **Origins of MOBMAS' techniques (for performing steps)** |
| 1. Specify system architecture (i.e. overview of all system components & their connections) | See sub-step "Update Agent Class Model" of step "Specify resources" of the "MAS Organisation Design" activity | **REUSE:**<br>Showing all system components and their connections. *Sources: INGENIAS, PROMETHEUS, PASSI and MAS-CommonKADS.* |

| *Overall System Design steps* | Support by MOBMAS | Origins of MOBMAS' techniques (for performing steps) |
|---|---|---|
| 2. Specify organisational structure/inter-agent authority relationships | See step "Specify MAS Organisational Structure" of the "MAS Organisation Design" activity | **REUSE:**<br>- Suggesting common styles of organisational structures which MAS may adopt, namely flat and hierarchical styles. *Source: MASSIVE.*<br>- Suggesting the developer to investigate the structure of the MAS' organisational context when choosing the organisational structure for the target MAS. *Source: GAIA.*<br>**ENHANCEMENT**:<br>- Extend the common styles of MAS organisational structure to include federation and hybrid styles<br>- Suggesting the developer to consider modularity issues, non-functional requirements and the number of roles in the system when choosing the organisational structure for the target MAS. |
| 3. Model MAS environment (O) | See steps "Specify Resources" and "Extend Ontology Model to include Resource application ontologies" of the "MAS Organisation Design" activity, and step " "Specify MAS Infrastructure Facilities" of the "Architecture Design" activity | **REUSE**<br>Modelling MAS environment by specifying resources offered by the environment. *Sources: SODA and GAIA.*<br>**ENHANCEMENT:** Providing specific guidelines for the identification and modelling of resources<br>**NEW:**<br>- Modelling Resource Application ontology for each resource<br>- Specifying infrastructure facilities for MAS |
| 4. Specify agent-environment interaction mechanism | See step "Identify Agent-Environment Interface Requirements" of the "Architecture Design" activity | **REUSE:**<br>Considering the use of conventional objects as sensors or effectors for agents. *Source: MEI.*<br>**ENHANCEMENT:**<br>Investigating all three different methods of agent-environment interaction: perception, effect and communication<br>**NEW:**<br>Suggesting various characteristics of the agents' perception, effect and communication that the developer should investigate when identifying the requirements of agent-environment interactions |
| 5. Instantiate agent classes | See step "Instantiate Agent Classes" of the "Architecture Design" activity | **REUSE**<br>Specifying cardinalities for agent classes' instantiation. *Source: GAIA.* |
| 6. Specify agent instances deployment | See step "Develop MAS Deployment Diagram" of the "Architecture Design" activity | **REUSE:**<br>Determining MAS deployment configuration by considering the communication traffic between agents, and the processing power available on particular machines or required by agents. *Source: MASE.*<br>**ENHANCEMENT:** Proposing a list of configuration details that need to be specified for MAS deployment. |

Table 7.6 – MOBMAS' support for the required modelling concepts (cf. Table 5.35)

| ***Problem Domain concepts*** | **Support by MOBMAS** | **Origins of MOBMAS' modelling techniques and notation** |
|---|---|---|
| 1. System functionality | System Task Model Kind | **REUSE:**<br>- Showing a hierarchy of system-tasks. *Sources: MASSIVE, COMOMAS and MAS-CommonKADS.*<br>- Notation for System Task Diagram. *Source: TROPOS.* |
| 2. Role | Role Model Kind | **REUSE:**<br>- Showing tasks of each role. *Sources: MASE, PASSI and GAIA.*<br>- Model communication paths between roles. *Sources: MASE, GAIA, PASSI and INGENIAS.*<br>**NEW:** Notation for Role Diagram (including the notation for "joint role-tasks" in Role Diagram) |
| 3. Domain conceptualisation | Ontology Model Kind | **REUSE:**<br>Representing ontological concepts as classes in UML Class Diagram, and ontological relations as relationships between UML classes. *Sources: MESSAGE, PASSI and MAS-CommonKADS.*<br>**NEW:** Modelling mappings between ontologies |
| ***Agent concepts*** | **Support by MOBMAS** | **Origins of MOBMAS' modelling techniques and notation** |
| 1. Agent-role assignment | Agent Class Model Kind | **REUSE:**<br>- Showing roles of each agent class. *Sources: MASE, GAIA and MESSAGE.*<br>- Modelling agent classes' dynamics in role playing behaviour. *Source: PASSI.*<br>**NEW:**<br>- Notation for Agent Class Diagram and Agent Relationship Diagram<br>- Notation for the modelling of agent classes' dynamics in Agent Class Diagram |
| 2. Agent goal/task | Agent Class Model Kind (Agent Class Diagram) and Agent Behaviour Model Kind (Agent Goal Diagram) | **REUSE:**<br>- Modelling agent-goals as states that an agent class aims to achieve. Sources: INGENIAS and BDIM.<br>- Notation for agent-goals. Source: TROPOS<br>**NEW:**<br>Showing a hierarchy of agent-goals for a particular agent class (if necessary) and the conflicts amongst these goals (if any) |

| *Agent concepts* | **Support by MOBMAS** | **Origins of MOBMAS' modelling techniques and notation** |
|---|---|---|
| 3. Agent belief/knowledge | Agent Class Model Kind (Agent Class Diagram) | **REUSE:**<br>Modelling belief conceptualisation for each agent class. *Source*: *BDIM*<br>**NEW:**<br>Modelling each agent class' belief conceptualisation in term of ontologies (which are in turn modelled in Ontology Model Kind) |
| 4. Agent plan/reasoning rule/problem solving method | Agent Behaviour Model Kind | **REUSE:**<br>- Modelling plans for agent classes. *Sources: BDIM, HLIM, MEI and TROPOS.*<br>- Notation for Agent Plan Diagram. *Sources: BDIM and TROPOS.*<br>- Specifying events and actions for each agent plan. *Sources: BDIM and PROMETHEUS.*<br>**ENHANCEMENT**: Considering the modelling of both planning behaviour and reflexive behaviour for agent classes<br>**NEW**: Notation for Agent Plan Template and Reflexive Rule Specification |
| 5. Agent architecture | Architecture Model Kind (Agent Architecture Diagram) | **REUSE:**<br>Showing the logical modules of agent architecture and potential flows of information between the modules. *Source: MASE*.<br>**NEW**: Notation for Agent Architecture Diagram |
| *Agent Interaction concepts* | **Support by MOBMAS** | **Origins of MOBMAS' modelling techniques and notation** |
| 1. Agent acquaintance | Agent Class Model Kind (Agent Relationship Diagram) | **REUSE:**<br>Representing agent acquaintances as connections between agent classes in Agent Relationship Diagram. *Sources: MASE, GAIA, MESSAGE and PASSI.*<br>**NEW:** Showing descriptive information about each agent acquaintance (e.g. interaction protocols and ontology that govern the acquaintance) |
| 2. Interaction protocol | Agent Interaction Model Kind | **REUSE:**<br>Using AUML Interaction Diagrams for the modelling of interaction protocols in "direct interaction mechanism". *Sources: MESSAGE, PROMETHEUS and PASSI*.<br>**NEW:**<br>- Adapting AUML Interaction Diagrams for the modelling of interactions between agents and a shared tuplespace/tuple-centre in the "indirect interaction mechanism"<br>- Modelling the behaviour of tuple-centre in the "indirect interaction mechanism" using UML State Chart |

| *Agent Interaction concepts* | **Supported by MOBMAS?** | **Origins of MOBMAS' modelling techniques and notation** |
|---|---|---|
| 3. Content of exchanged messages | Agent Interaction Model Kind | **REUSE:**<br>Specifying speech-act performative for each exchanged message. *Sources: MESSAGE, TROPOS and MAS-CommonKADS.*<br>**ENHANCEMENT:** Providing detailed techniques for modelling the content of each exchanged message (i.e. performatives, variables, data types) for each coordination mechanism (i.e. Tuple Centre Coordination Mechanism or Interaction-Protocol Coordination Mechanism) |
| ***Overall System Design concepts*** | **Supported by MOBMAS?** | **Origins of MOBMAS' modelling techniques and notation** |
| 1. System architecture | Agent Class Model Kind (Agent Relationship Diagram) | **REUSE:**<br>Showing an overview of all agent classes and resources in Agent Relationship Diagram. *Sources: PROMETHEUS, PASSI, INGENIAS and MAS-CommonKADS.*<br>**ENHANCEMENT:** Showing "wrapping" relationships between agent classes and resources in Agent Relationship Diagram |
| 2. Organisational structure/inter-agent authority relationship | Role Model Kind | **REUSE:**<br>Showing authority relationships between roles in Role Diagram. *Sources: GAIA, MESSAGE, INGENIAS and HLIM.*<br>**NEW:** Adapting UML dependency relationship for the modelling of inter-role authority relationships |
| 3. Environment resource/facility | Resource Model Kind | **REUSE:**<br>Modelling resources and their relationships with "wrapper" agent classes. *Source: INGENIAS.*<br>**NEW:**<br>- Notation for Resource Diagram<br>- Allowing the Resource Diagram to be adaptable to the target MAS development project (i.e. the configuration dimensions can be changed)<br>- Modelling Resource Application ontology for each resource in Ontology Model Kind<br>- Modelling MAS infrastructure facilities in "Infrastructure Facility Specification" |
| 4. Agent instantiation | Agent Class Model Kind (Agent Relationship Diagram) | **REUSE:**<br>Showing instantiation cardinality for each agent class *Sources: MASE, GAIA and PROMETHEUS.*<br>**NEW:**<br>Notation for modelling agent instantiation in Agent Relationship Diagram |
| 5. Agent instance deployment | Architecture Model Kind (MAS Deployment Diagram) | **REUSE:**<br>- Showing locations of agents. *Source: MASE*.<br>- Show processing nodes, agents at each node and connections between nodes/agents. *Source: PASSI*.<br>**ENHANCEMENT:** Adopting UML Deployment Diagram for the modelling of agent instance deployment. |

#### *7.4.1.1. MOBMAS' support for ontology-based MAS development*

MOBMAS offers extensive support for ontology-based MAS development, by using ontologies in various steps of the MAS development **process** and integrating ontologies into the model definitions of various **model kinds**.

Regarding the MAS *development process*, a large proportion of MOBMAS' steps are ontology-related. In fact, these MOBMAS' ontology-related steps correspond to those *desirable ontology-related AOSE steps* that were previously recommended for MOBMAS in Section 5.5. The following discussion recapitulates these desirable ontology-related AOSE steps (c.f. Section 5.5) and reveals their correspondences with MOBMAS' actual steps.

1. "*Identify system functionality*": This generic AOSE step is supported by MOBMAS via step "*Develop System Task Model*" (c.f. Section 6.2.1). Even though MOBMAS does not cover the process of system tasks elicitation, it suggests the developer to validate and refine the identified system tasks by examining the application ontologies captured in the Ontology Model (c.f. Section 6.2.4.1.c).

2. "*Model domain conceptualisation*": This desirable AOSE step corresponds to MOBMAS' step "*Develop Ontology Model*" (c.f. Section 6.2.4). This step produces an Ontology Model to capture all of the application ontologies of the target MAS and the semantic mappings between these ontologies.

3. "*Define content of exchanged messages*": This AOSE step is performed as part of step "*Develop Agent Interaction Model*" in MOBMAS (c.f. Section 6.5.2). MOBMAS uses ontological concepts to formulate the content of the exchanged messages (particularly the datatypes of the exchanged variables), and requires the developer to validate the formulated messages against the ontologies in the Ontology Model and vice versa.

4. "*Define agent information constructs*": This desirable AOSE step corresponds to MOBMAS' step "*Specify agent class' belief conceptualisation*" (c.f. Section 6.4.1). Ontologies are used by this step as the building blocks for modelling agents' conceptual beliefs.

5. "*Define agent behavioural constructs*": This generic AOSE step is supported via three MOBMAS' steps: "*Specify agent goals*", "*Specify events*" and "*Develop Agent Behaviour Model*" (c.f. Sections 6.4.2, 6.4.3 and 6.4.4). All of these steps refer to the concepts defined in the application ontologies whenever appropriate to define agents' goals, plans, reflexive rules and actions. Ontologies are also used to help identify and validate the agents' actions.

6. "*Model MAS environment*": This AOSE step is covered via three steps in MOBMAS: "*Specify resources*", "*Extend Ontology Model to include Resource application ontologies*" and "*Specify MAS infrastructure facilities*" (c.f. Sections 6.3.3, 6.3.4 and 6.3.3). The former two[82] involve identifying the ontologies which conceptualise each resource of MAS, and updating the Ontology Model to include these ontologies (together with their mappings).

As can be seen above, all desirable ontology-related steps that were identified in Section 5.5 are supported by MOBMAS. Consequently, it can be said that MOBMAS is capable of realising all the benefits of ontology in MAS design and operation as listed in Section 2.3.2. These capabilities will be further confirmed in Section 7.4.2.4. In addition, MOBMAS also addresses how the MAS development process can assist in the engineering of ontology. Specifically, the investigation of system's functionality, agent goals, plans, reflexive rules, actions and exchanged messages helps to identify and validate the concepts to be included in ontologies.

Regarding its *model definitions*, MOBMAS integrates ontologies into five of its nine model kinds, namely:

- *Ontology Model Kind*;
- *Agent Class Model Kind*;
- *Resource Model Kind*;
- *Agent Behaviour Model Kind*; and
- *Agent Interaction Model Kind.*

These model kinds are the direct products of the previously listed MOBMAS' ontology-related steps.

---

[82] Note that step "*Specify MAS infrastructure facilities*" does not need to involve ontologies.

## 7.4.2. Comparison of MOBMAS and Existing AOSE Methodologies

This section documents the comparison between MOBMAS and the existing AOSE methodologies, which uses the same evaluation framework as that used in the feature analysis of the existing AOSE methodologies in Section 5.4. This evaluation framework consists of 38 criteria (cf. Section 5.4.1), namely:

- 36 evaluation criteria on *features* (Table 5.21);
- one criterion on *steps* (Table 5.22); and
- one criterion on *modelling concepts* (Table 5.23).

The comparison between MOBMAS and the existing methodologies in term of these evaluation criteria is presented in Sections 7.4.2.1, 7.4.2.2 and 7.4.2.3. It should be noted that, some of the criteria actually deal with features, steps and modelling concepts that are *not* required from MOBMAS (i.e. those identified as "potential requirements" of MOBMAS but which were not eligible to become MOBMAS' "actual requirements" in Section 5.4.3). Consequently, MOBMAS' support for these criteria may be weak. These criteria are annotated with symbol "#" in Tables 7.4, 7.5, 7.6 and 7.7.

Section 7.4.2.4 finally provides an account of the important ontology-related strengths of MOBMAS. These strengths are either not provided, or provided to a lesser extent, by existing AOSE methodologies due to their lack, or low level, of support for ontology.

### *7.4.2.1. Comparison of support for Features*

- **Comparison of support for features relating to AOSE process:**
  Of the fourteen criteria in this category (Table 5.21), only eight criteria are examined in this section (Table 7.7). The comparison of the remaining six criteria[83] will be discussed in Section 7.4.2.2 alongside criterion "*Support for steps*", because these six criteria need to use the list of AOSE steps presented in Table 5.22 as a common yardstick.

[83] That is, criteria "*Specification of model kinds and/or notational components*", "*Definition of inputs and outputs for steps*", "*Specification of techniques and heuristics*", "*Ease of understanding of techniques*", "*Usability of techniques*" and "*Provision of examples for techniques*".

Justification of MOBMAS' support for the criteria "*Specification of a system development lifecycle*", "*Support for verification and validation*", "*Specification of steps for the development process*" and "*Support for refinability*" of Table 7.7 can be found in Table 7.4. For criteria "*Ease of understanding of the development process*" and "*Usability of the development process*", the evaluation of MOBMAS was obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G). The assessment of the first developer is denoted as "*D1*", while that of the second developer is labelled with "*D2*". For criterion "*Approach for MAS development*", MOBMAS is Role-Oriented (denoted as "*RO*") because it uses "role" as the building block for defining agent classes.

- **Comparison of support for features relating to AOSE model definitions:**
  Nine evaluation criteria were used to conduct this comparison (Table 5.21). The comparison results are presented in Table 7.. The support provided by MOBMAS for each criterion has been justified in Table 7.4, except for criterion "*Ease of understanding of model definitions*", whose evaluation was obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G).

- **Comparison of support for agent properties:**
  Nine agent properties were investigated in total (Table 5.21). The comparison between MOBMAS and the existing methodologies in term of the support for these properties is shown in Table 7.9. The justification for MOBMAS' support can be found in Table 7.4.

- **Comparison of support for features relating to the methodology as a whole**:
  There are six high-level, supplementary features that pertain to a MAS development methodology as a whole. The comparison between MOBMAS and the existing methodologies with regard to these features is presented in Table 7.10. The justification for MOBMAS' evaluation can be found in Table 7.4. For criterion "*Support for agility and robustness*", MOBMAS was given a "*possibly*" evaluation because it implicitly models the exceptional situations and the exception-handling

behaviour of agents, through the specification of interaction protocols in the "*Agent Interaction Design*" activity (Section 6.5).

Table 7.7 – Comparison of support for features relating to AOSE process

| | Specification of a system development (dev.) lifecycle | Support for verification & validation | Specification of steps for the dev. process | Support for refinability | Ease of understanding of the dev. process | Usability of the dev. process | Approach for MAS dev. |
|---|---|---|---|---|---|---|---|
| **MASE** | Iterative across all phases | Yes | Yes | Yes | High | High | RO |
| **MASSIVE** | Iterative View Engineering process | Yes | Yes | Yes | High | Medium | RO |
| **SODA** | Not specified | No | Yes | No | High | Low | RO |
| **GAIA** | Iterative within each phase but sequential between phases | No | Yes | Yes | High | Medium | RO |
| **MESSAGE** | Rational Unified Process | Mentioned as future enhancement | Yes | Yes | High | Medium | RO |
| **INGENIAS** | Unified software development process | Yes | Yes | Yes | High | High | RO |
| **BDIM** | Not specified | No | Yes | Yes | High | Low | RO |
| **HLIM** | Iterative within and across the phases | No | Yes | Yes | High | High | RO |
| **MEI** | Not specified | No | Yes | Yes | High | High | NRO |
| **PROMETHEUS** | Iterative across all phases | Yes | Yes | Yes | High | High | NRO |
| **PASSI** | Iterative across and within all phases | Yes | Yes | Yes | High | High | RO |
| **ADELFE** | Rational Unified Process | Yes | Yes | Yes | High | High | NRO |
| **COMOMAS** | Not specified | No | Yes | Yes | High | Medium | NRO |
| **MAS-CommonKADS** | Cyclic risk-driven process | Mentioned but no clear guidelines | Yes | Yes | High | Medium | NRO |
| **CASSIOPEIA** | Not specified | No | Yes | Yes | High | Medium | RO |
| **TROPOS** | Iterative and incremental | Yes | Yes | Yes | High | Medium | NRO |
| **MOBMAS** | Iterative and incremental | Yes | Yes | Yes | D1: High<br>D2: High | D1: High<br>D2: High | RO |

Table 7.8 – Comparison of support for features relating to AOSE model definitions

| | Completeness/ expressiveness | Formalization/ preciseness | Provision of guidelines/ logics for model derivation | Guarantee of consistency | Support for modularity | Management of complexity | Levels of abstraction | Support for reuse | Ease of understanding of model definitions |
|---|---|---|---|---|---|---|---|---|---|
| **MASE** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Yes | High |
| **MASSIVE** | Medium | a. High<br>b. Yes | Yes | a. No<br>b. Yes | Yes | Yes | Yes | Yes | High |
| **SODA** | Medium | a. Medium<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | No | Possibly | High |
| **GAIA** | Medium | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Yes | High |
| **MESSAGE** | Medium | a. High<br>b. Yes | Yes | a. No<br>b. Yes | Yes | Yes | Yes | Possibly | High |
| **INGENIAS** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | No | Yes | Possibly | Medium |
| **BDIM** | Medium | a. High<br>b. Yes | Yes | a. No<br>b. Yes | Yes | Yes | Yes | Yes | High |
| **HLIM** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Possibly | High |
| **MEI** | Medium | a. Low<br>b. Yes | Yes | a. No<br>b. Yes | Yes | Yes | Yes | Possibly | Medium |
| **PROMETHEUS** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Possibly | High |
| **PASSI** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | No | Yes | High |
| **ADELFE** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Possibly | High |
| **COMOMAS** | High | a. High<br>b. Yes | Yes | a. No<br>b. Yes | Yes | Yes | No | Possibly | High |
| **MAS-CommonKADS** | High | a. Medium<br>b. Yes | No | a. No<br>b. Yes | Yes | Yes | Yes | Yes | Medium |
| **CASSIOPEIA** | Medium | a. High<br>b. Yes | NA | a. No<br>b. NA | Yes | Yes | No | Possibly | High |
| **TROPOS** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Possibly | High |
| **MOBMAS** | High | a. High<br>b. Yes | Yes | a. Yes<br>b. Yes | Yes | Yes | Yes | Yes | D1: Medium<br>D2: High |

Table 7.9 – Comparison of support for agent properties

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

Table 7.10 – Comparison of support for features relating to the methodology as a whole

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

### *7.4.2.2. Comparison of support for Steps*

The assessment of MOBMAS with regard to the criterion "*Support for steps*" (Table 5.22) is presented in Table 7.11. This table also documents the evaluation of MOBMAS with regard to six other criteria which relate to the AOSE process but had not been examined in Section 7.4.2.1, namely:

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

All of these criteria use the list of AOSE steps presented in Table 5.22 as yardstick. This list had previously been used in the feature analysis of the existing methodologies in Section 5.4.2 and Appendix D. The meaning of all abbreviations in Table 7.11 is the same as that in Tables D.1 to D.14 in Appendix D. It should be noted that, the assessment of MOBMAS' support for the criteria "*Ease of understanding of techniques*" and "*Usability of techniques*" were obtained from the two developers who used MOBMAS on the "Peer-to-Peer Information Sharing" application (Section 7.3; Appendix G). The assessment of each developer is denoted as "*D1*" and "*D2*" respectively.

To compare MOBMAS with the existing 16 AOSE methodologies, readers are referred to Tables AppendixD.1 to AppendixD.16 in Appendix D, where the individual assessment of each existing methodology is documented. Since the evaluation findings of all methodologies are presented in the same format, a direct comparison between them can easily be made. A bird-eye view of the comparison between all methodologies is shown in Table 7.12. This table shows only the assessment of the criterion "*Usability of techniques"*.

Table 7.11 – MOBMAS' support for steps

| MOBMAS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Steps** | **Supported?** | **Model kinds/ Notational components?** | **Inputs/ Outputs?** | **Techniques for step** | **Techniques for modelling** | **Ease of understanding** | **Usability** | **Examples** |
| 1. Identify system functionality | E | System Task Model Kind | B | See Section 6.2.1 – "Develop System Task Model" | See Section 6.2.1.1 – "Notation of System Task Diagram" | D1: H D2: M | D1: H D2: H | Y |
| 2. Specify use case scenarios # | | | | | | | | |
| 3. Identify roles | E | Role Model Kind | B | See Section 6.2.3.1 – "Identify roles" and Section 6.2.3.2 – "Specify role-tasks" | See Section 6.2.3.3 – "Notation of Role Diagram" | D1: H D2: M | D1: H D2: H | Y |
| 4. Identify agent classes | E | Agent Class Model Kind | B | See Section 6.3.2.1 – "Identify agent classes" | See Section 6.3.2.2 – "Notation of Agent Class Model Kind" | D1: H D2: H | D1: H D2: M | Y |
| 5. Model domain conceptualisation | E | Ontology Model Kind | B | See Section 6.2.4.1 – "Develop MAS Application ontologies" | See Section 6.2.4.2 – "Language of Ontology Model Kind" | D1: M D2: H | D1: M D2: H | Y |
| 6. Specify acquaintances between agent classes | E | Agent Class Model Kind (Agent Relationship Diagram) | B | See Sections 6.5.2.1.c and 6.5.2.2.c – "Update Agent Class Model and Role Model" | See Section 6.3.2.2 – "Notation of Agent Class Model" | D1: H D2: H | D1: H D2: H | Y |
| 7. Define interaction protocols | E | Agent Interaction Model Kind | B | See Section 6.5.2 – "Develop Agent Interaction Model" | See Section 6.5.2 – "Develop Agent Interaction Model" | D1: H D2: H | D1: H D2: H | Y |
| 8. Define content of exchanged messages | E[7] | Agent Interaction Model Kind | B | See Section 6.5.2 – "Develop Agent Interaction Model" | See Section 6.5.2 – "Develop Agent Interaction Model" | D1: H D2: H | D1: H D2: H | Y |
| 9. Specify agent communication language# | I[8] | | | Assume the use of any ACL that use speech-act performatives (such as FIPA-ACL or KQML) | | | | |
| 10.Specify agent architecture | E | Architecture Model Kind (Agent Architecture Diagram) | B | See Section 6.6.2.1 – "Select agent architecture" | See Section 6.6.2.2 – "Develop Agent Architecture Diagram" | D1: H D2: H | D1: H D2: H | Y |
| 11.Define agent informational constructs (i.e. beliefs) | E | Agent Class Model Kind (Agent Class Diagram); Ontology Model Kind | B | See Section 6.4.1.1 – "Specify Belief Conceptualisation of Agent Classes" | See Section 6.4.1.2 – "Update Agent Class Model To Show Belief Conceptualisation" | D1: M D2: H | D1: H D2: H | Y |
| 12.Define agent behavioural constructs (e.g. goals, plans, actions, services) | E | Agent Behaviour Model Kind; Agent Class Model Kind (Agent Class Diagram) | B | See Section 6.4.2 – "Specify agent-goals", Section 6.4.3 – "Specify events" and Section 6.4.4 – "Develop Agent Behaviour Model" | See Section 6.4.2 – "Specify agent-goals", Section 6.4.3 – "Specify events" and Section 6.4.4 – "Develop Agent Behaviour Model" | D1: H D2: M | D1: M D2: H | Y |
| 13.Specify system architecture (i.e. overview of all components and their connections) | E | Agent Class Model Kin d (Agent Relationship Diagram) | B | See Section 6.3.3.4 – "Update Agent Class Model" | See Section 6.3.3.4 – "Update Agent Class Model" | D1: H D2: H | D1: H D2: H | Y |
| 14.Specify organisational structure/inter-agent authority relationships | E | Role Model Kind | B | See Section 6.3.1.1. – "Determine MAS Organisational Structure" | See Section 6.3.1.2 – "Update Role Model" | D1: H D2: H | D1: M D2: H | Y |
| 15.Model MAS environment | E | Resource Model Kind | B | See Section 6.3.3 – "Specify resources" | See Section 6.3.3.2 – "Notation of Resource Diagram" | D1: H D2: H | D1: H D2: H | Y |
| 16.Specify agent-environment interaction mechanism | E | Architecture Model Kind (Agent-Environment Interface Requirements Specification) | B | See Section 6.6.1 – "Identify agent-environment interface requirements" and Section 6.6.3 – "Specify MAS infrastructure facilities" | See Section 6.6.1 – "Identify agent-environment interface requirements" and Section 6.6.3 – "Specify MAS infrastructure facilities" | D1: M D2: H | D1: M D2: H | Y |
| 17.Specify agent inheritance and aggregation# | | | | | | | | |
| 18.Instantiate agent classes | E | Agent Class Model Kind (Agent Relationship Diagram) | B | See Section 6.6.4 – "Instantiate agent classes" | See Section 6.6.4 – "Instantiate agent classes" | D1: H D2: H | D1: H D2: H | Y |
| 19.Specify agent instances deployment | E | Architecture Model Kind (MAS Deployment Diagram) | B | See Section 6.6.5 – "Develop MAS Deployment Diagram" | See Section 6.6.5 – "Develop MAS Deployment Diagram" | D1: H D2: H | D1: M D2: H | Y |

Table 7.12 – Comparison re criterion "Usability of techniques"

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

### *7.4.2.3. Comparison of support for Modelling Concepts*

In this section, MOBMAS is compared with the existing AOSE methodologies in term of the criterion "*Support for concepts*" (Table 7.13). This criterion uses the list of AOSE modelling concepts presented in Table 5.23 as yardsticks. This list had previously been used in the feature analysis of the existing methodologies in Section 5.4 and Appendix D.

In Table 7.13, if a methodology provides support for a particular modelling concept, this support is represented by a tick✓. Readers are referred to Tables 5.29a and 5.29b for the names of the model kinds and/or notational components that model the concepts in each existing methodology. For MOBMAS, this information can be found in Table 7.6.

Table 7.13 – Comparison of support for modelling concepts

| | 1. System functionality | 2. Use case scenario# | 3. Role | 4. Domain conceptualisation | 5. Agent-role assignment | 6. Agent goal/task | 7. Agent belief/knowledge | 8. Agent plan/reasoning rule/problem solving method | 9. Agent capability/service# | 10. Agent percept/event# | 11. Agent architecture | 12. Agent acquaintance | 13. Interaction protocol | 14. Content of exchanged messages | 15. System architecture | 16. Organisational structure/inter-agent authority relationship | 17. Environment resource/facility | 18. Agent aggregation relationship# | 19. Agent inheritance relationship# | 20. Agent instantiation | 21. Agent instance deployment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **MASE** | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ | ✓ |
| **MASSIVE** | ✓ | | ✓ | | ✓ | | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | |
| **SODA** | ✓ | | ✓ | | ✓ | | | | | | | ✓ | ✓ | | | | ✓ | | | ✓ | ✓ |
| **GAIA** | ✓ | | ✓ | | ✓ | | | | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | |
| **MESSAGE** | ✓ | | ✓ | ✓ | ✓ | | | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| **INGENIAS** | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| **BDIM** | | | | | | ✓ | ✓ | ✓ | | | | ✓ | | ✓ | | | | ✓ | ✓ | ✓ | |
| **HLIM** | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | | | | | |
| **MEI** | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | | | | | | | | |
| **PROMETHEUS** | ✓ | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | | ✓ | |
| **PASSI** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ |
| **ADELFE** | ✓ | ✓ | | | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | |
| **CASSIOPEIA** | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | | |
| **COMOMAS** | ✓ | | | | | ✓ | ✓ | | | | | | ✓ | | | ✓ | | | | | |
| **MAS-COMMONKADS** | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| **TROPOS** | ✓ | | | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | ✓ | | | | | |
| **MOBMAS** | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |

### *7.4.2.4. Ontology-related strengths of MOBMAS*

By using ontologies in the MAS development process and integrating ontologies into the MAS model definitions, MOBMAS is able to exploit ontologies to enhance its *MAS development process* and *MAS development product*[84] with many important ontology-related strengths. These strengths are either not provided, or provided to a lesser extent, by the existing AOSE methodologies due to their lack, or low level, of support for ontology.

The following sections identify and justify the various ontology-related strengths of MOBMAS, organised into *process*-related strengths and *product*-related strengths. These strengths include those ontology's benefits that are identified in Section 2.3.2, and those additional benefits provided by MOBMAS.

#### Ontology-related strengths of MOBMAS *development process*

- *Highly reliable system analysis* (cf. Section 2.3.2.3): By acknowledging that an effective ontology analysis would facilitate the understanding of a particular domain, MOBMAS recommends using the ontological analysis effort of knowledge engineers to facilitate and validate the system analysis effort of system developers. In particular, ontologies can be used to help identify and validate the identification of system tasks in the System Task Model (cf. Section 6.2.4.1.c). In all other existing AOSE methodologies, the system analysis process is conducted solely by the system developer himself and not supplemented by any other analysis effort.

- *Effective modelling of application domain*: As discussed in Section 2.3.2.3, ontology provides a structured, human-readable and reusable representation mechanism for modelling application domain of a given MAS system. MOBMAS accordingly recommends the developer to use ontologies as the modelling mechanism for application domains (i.e. "MAS application ontologies"; cf. Section 6.2.4.1). Amongst the existing 16 AOSE methodologies, only MAS-CommonKADS, MESSAGE, MASE and PASSI exploit ontologies for application domain modelling (cf. Section 3.3.2).

[84] "Product" refers to the final MAS system whose design is developed by MOBMAS.

- *Well-structured, modular modelling of agents' local knowledge* (cf. Section 2.3.2.3): In MOBMAS, ontologies serve as the building blocks for defining agents' conceptual knowledge (cf. Section 6.4.1). This ontology-based modelling mechanism results in agent knowledge models that are much more structured and modular than those produced by the existing AOSE methodologies. The latter do not organise agents' knowledge into any modular conceptual structures.

- *Systematic modelling of agent local knowledge*: If an agent wishes to hold beliefs about a particular domain or resource, the local knowledge of that class should contain the corresponding ontology. By implementing this modelling mechanism, MOBMAS helps the developer to systematically and effectively design the knowledge model for each agent class (cf. Section 6.4.1). The process of agent knowledge modelling in the other existing AOSE methodologies is not as simple and effective, because these methodologies identify agent knowledge in a "bit-by-bit" manner, e.g. by investigating each agent goal, plan, interaction and/or use cases (e.g. BDIM, MESSAGE, INGENIAS, HLIM, COMOMAS and MAS-CommonKADS; cf. Appendix D).

- *Reliable specification of agents' behaviour*: While the existing AOSE methodologies only examine constructs such as agent roles, goals, interactions and use cases to identify agents' potential actions (e.g. GAIA, BDIM, HLIM, PROMETHEUS and MAS-CommonKADS; cf. Appendix D), MOBMAS also examines the ontologies committed by each agent (cf. Section 6.4.4). By using ontologies as an additional input, the developer may uncover actions that would otherwise be missed if he only investigates agent roles, goals, interactions and use cases.

- *Extensive verification and validation*: An ontology development effort is closely related and supplementary to a MAS development effort, because both involve a detailed investigation of the target application. As noted in Section 2.3.2.3, a strong ontological analysis leads to a more complete and accurate understanding of the target application. Accordingly, MOBMAS recommends that the developer should

exploit application ontologies to verify and validate the correctness and completeness of its MAS analysis and design models, namely System Task Model, Role Model, Agent Behaviour Model and Agent Interaction Model (cf. Sections 6.2.4.1.c, 6.4.4.1, 6.4.4.3 and 6.4.2.3). Since application ontologies are often constructed by a separate development team (e.g. domain experts or knowledge engineers), they can serve as a reliable tool for verification and validation. When examining the few existing AOSE methodologies that offer support for verification and validation (e.g. MASE, INGENIAS, PASSI, PROMETHEUS and TROPOS; cf. Table 5.24), it was found that each methodology simply uses its MAS analysis and design models to verify and validate against themselves. This mechanism of verification and validation is undoubtedly less reliable than the use of a separately-developed ontology model as seen in MOBMAS.

- *Support for (distributed) team-based development*: Sharing an ontology is basically sharing the same conceptual knowledge of a particular application domain, task or resource. This consensual knowledge is important to a MAS development project where multiple, distributed developers are involved. In MOBMAS, where ontologies are used as a major information source for many MAS analysis and design steps (e.g. "Develop System Task Model" step, "Specify Agent Class' Belief Conceptualisations" step, "Develop Agent Behaviour Model" step and "Develop Agent Interaction Model" step; cf. Sections 6.2.1, 6.4.1, 6.4.4 and 6.5.2 respectively), the different developers who engage in different development steps can share the same knowledge base when performing their individual work, thereby generating consistent work products despite of the distributed development contexts. In addition, as mentioned in Section 2.3.2.3, the mappings between different ontologies identify the associations amongst the different application domains, tasks and/or resources. This identification allows the developers to combine/integrate their work if each had focused on a different domain, task or resource.

## Ontology-related strengths of MOBMAS' development product

- *Support for interoperability* (cf. Section 2.3.2.1): The MASs resulted from MOBMAS can strongly support interoperability between heterogeneous agents and between heterogeneous resources, because in these MASs, the knowledge of heterogeneous agents has been explicitly conceptualised by ontologies (cf. Section 6.4.1), the information/applications of heterogeneous resources have also been explicitly conceptualised by ontologies (cf. Section 6.3.4), and the semantic mappings between these ontologies have been explicitly specified (cf. Sections 6.2.4.2 and 6.3.4.1). A detailed discussion of how these factors can support interoperability is already presented in Section 2.3.2.1. Amongst the existing AOSE methodologies, INGENIAS, PROMETHEUS, GAIA and MASSIVE are the only ones that mention the existence of non-agent resources in MAS. However, they do not discuss how the heterogeneous components of MAS can be interoperated. Meanwhile, even though MASE considers the use of ontology to support interoperability between heterogeneous agents, it does not mention the case of non-agent resources.

- *Support for reusability* (cf. Section 2.3.2.2): The Ontology Model of a MAS designed by MOBMAS offers a detailed description of the target application. Therefore, any future MAS development projects can simply examine this model to determine whether, and which part(s) of, a past MAS design can be reused. Moreover, since the core design models of MOBMAS are composed in terms of ontologies and ontological concepts (namely, Agent Belief Conceptualisation, Agent Behaviour Model and Agent Interaction Model), the developer can adapt the past MAS design models to a new application by simply changing the ontologies involved. In addition, MOBMAS has implemented the idea of using ontologies to decouple the modelling of agents' domain knowledge from agents' behavioural/problem-solving knowledge[85], thereby supporting the reuse of these two knowledge components across agents. Lastly, MOBMAS provides extensive support for interoperability (as discussed above). It therefore shows how legacy agents and/or resources can be reused by the current MAS system. In summary, compared

[85] In MOBMAS, agents' domain knowledge is captured via ontologies in Agent Belief Conceptualisations (cf. Section 6.4.1), while agents' behavioural constructs (i.e. plans, reflexive rules and actions) are defined in Agent Behaviour Model (cf. Section 6.4.4).

to the existing AOSE methodologies, MOBMAS discovers new ways of supporting reusability through its use of ontologies in MAS development.

- *Support for semantically-consistent communication between agents* (cf. Section 2.3.2.4): Ontology is essential to the successful communication between agents. If agents use the same ontology to compose and interpret the exchanged messages, they can convey the information in a uniform and consistent manner. With this understanding, MOBMAS requires the developer to "datatype" the variables in all exchanged ACL messages (or tuples) with concepts defined in the ontologies shared between the communicating parties (cf. Section 6.5.2). With this rule, agents in the resulting MASs will always be able to interpret the exchanged messages, and interpret them in a consistent manner. All of the existing AOSE methodologies, except for MASE and PASSI, do not provide this insurance, since they fail to recognise the importance of ontology in agent communication (cf. Section 3.3.2).

- *Support for communication between agents and resources* (cf. Section 2.3.2.4): An explicit conceptualisation of a resource will allow the wrapper agents to determine which vocabulary they should use to formulate the queries/commands to the resource and to interpret the queries' results, without having to access the resource's internal structure. This ontology-related benefit is naturally offered by a MAS produced by MOBMAS, because MOBMAS addresses the modelling of resources' conceptualisations through ontologies and the specification of mappings between resources' ontologies and MAS application ontologies (cf. Section 6.3.4). Even though four of the existing AOSE methodologies show some consideration for ontology (i.e. MAS-CommonKADS, MESSAGE, MASE and PASSI), they do not discuss the modelling of resources' conceptualisations, thus failing to use ontologies to facilitate agent-resource communication.

- *Support for agent reasoning* (cf. Section 2.3.2.4): In MOBMAS, the specification of agents' behavioural constructs (i.e. plans, reflexive rules and actions) makes reference to the agents' ontology-based knowledge (wherever appropriate) to allow for the agents' problem-solving knowledge to be linked with the agents' ontology-based domain-related knowledge. For example, ontological concepts are used to

define the knowledge requirements of each agent' plans and actions; cf. Section 6.4.4.3). This enables agents' reasoning (which operationalises the agents' problem-solving knowledge) to utilize the ontology-based domain-related knowledge of agents at run-time. No existing AOSE methodologies are found to explicitly associate agents' problem-solving knowledge with agents' ontological knowledge at design time. Accordingly, they cannot illustrate whether, and how, agent reasoning can utilize ontology-based knowledge at run-time.

- *Support for maintainability*: A MAS system produced by MOBMAS can easily be maintained, even by someone other than the original developer, because the specification of the underlying application domains, tasks and wrapped resources has been formally documented in the ontologies of the Ontology Model, and because the other core MAS design models such as Agent Belief Conceptualisation, Agent Behaviour Model and Agent Interaction Model are consistently defined in term of these ontologies. This support for maintainability is not demonstrated in the existing AOSE methodologies.

- *Support for extendibility*: When a MAS designed by MOBMAS needs to cover new domains, tasks or resources, its agents can easily extend their knowledge by adding new ontologies to their knowledge models. New Agent Plan Templates, Reflexive Rule Specifications and Interaction Diagrams can also be created by referring to the concepts defined in the new ontologies. Such ease of extendibility is not demonstrated in the existing AOSE methodologies.

- *High likelihood of a correct system*: This is the direct result of MOBMAS' extensive support for verification and validation during the design of core models such as Agent Behaviour Model (cf. Section "*Ontology-related strengths of the MAS development process of MOBMAS*").

# 7.5. SUMMARY

This chapter has documented the process of evaluating and refining MOMBAS, which progressively led to the final version of MOBMAS presented in Chapter 6. The

progressive evaluation and refinements of MOBMAS were conducted through the collection of two expert reviews, the use of MOBMAS on a test application by two external developers, and a feature analysis of MOBMAS. This feature analysis includes the justification of MOBMAS' comprehensive support for ontology-based MAS development and various other important AOSE methodological requirements, the comparison between MOBMAS and the existing AOSE methodologies, and the clarification of MOBMAS' ontology-related strengths.

# CHAPTER 8
# CONCLUSIONS

## 8.1. INTRODUCTION

This chapter concludes the thesis by recapitulating the contributions of this research to the literature on AOSE (Section 8.2). It also identifies the limitations of the process of conducting the research (Section 8.3) and suggests directions for future research (Section 8.4). The chapter is closed with some concluding remarks (Section 8.5).

## 8.2. CONTRIBUTIONS OF THE RESEARCH

The main contribution of this research is *the proposal of an AOSE methodology for the analysis and design of ontology-based MASs, which is named "MOBMAS" – "Methodology for Ontology-Based Multi-Agent Systems"*. MOBMAS offers a **software engineering process** comprising of *activities* and *steps* to conduct the analysis and design of an ontology-based MAS, **techniques** to perform these steps and **model kinds** to represent the software artifacts. The methodology is capable of supporting *ontology-based MAS development* and various other *AOSE methodological requirements* which are important to an AOSE methodology but which may not be well-supported by the existing methodologies.

- With regard to the **support for ontology-based MAS development**, MOBMAS surpasses all of the existing AOSE methodologies. Even though four of the existing methodologies were found to integrate ontologies into MAS design, they fail to identify and implement the diverse potential ways in which ontologies can be used in the MAS development process and/or included in the MAS model definitions (cf. Section 3.3.2). Meanwhile, MOBMAS, with its comprehensive acknowledgement of ontology's significant benefits to interoperability, reusability, MAS development activities (particularly system analysis and agent knowledge modelling) and MAS

operation (specifically communication and agent reasoning), has extensively incorporated ontologies into its MAS development process and model definitions.

- Regarding the *MAS development process*, MOBMAS makes use of application ontologies to facilitate the process of constructing and validating its MAS analysis and design models. In particular, application ontologies are used to help identify and validate the system tasks of the target MAS, actions of agent classes and exchanged messages between agents. Moreover, MOBMAS also enables the MAS development process to, in return, support the development of application ontologies. Specifically, the analysis of MAS system tasks and the detailed design of agent classes' goals, plans, actions and exchanged messages help to identify and validate the concepts defined in the application ontologies.

- Regarding the *MAS model definitions*, MOBMAS dedicates one of its model kinds, namely "Ontology Model Kind", to the representation of application ontologies. This model kind captures all of the application ontologies that are necessary for agents in the target MAS to operate. Agents' knowledge is then modelled in term of these ontologies. Agent behaviour modelling and interaction modelling are also based upon ontologies: concepts in the application ontologies are used to formulate agent classes' goals, plans, actions and content of communication messages. MOBMAS also models the conceptualisation of non-agent resources (i.e. Resource Application ontologies), and the mappings between these Resource Application ontologies and the MAS Application ontologies.

By extensively exploiting ontology as described above, MOBMAS is able to enhance its *MAS development process* and *MAS development product* with many important ontology-related strengths (cf. Section 7.4.2.4). These strengths include those widely-acknowledged benefits of ontology to MASs (i.e. support for interoperability, reusability, system analysis, agent knowledge modelling, communication and agent reasoning; cf. Section 2.3.2), and those additional benefits of ontology as uncovered by MOBMAS (e.g. support for verification and validation, maintainability, extendibility and reliability). These ontology-related strengths are either not provided, or provided to a lesser extent, by the existing AOSE methodologies due to their lack, or low level, of support for ontology (cf. Sections 3.3.2 and 7.4.2.4).

- With regard to the **general support for MAS analysis and design**, no individual AOSE methodology was found to address *all* of the important methodological requirements of an AOSE methodology (which were identified by this research from an investigation of the AOSE literature and confirmed by the practitioners and researchers in the field). MOBMAS, on the other hand, endeavours to support all of these requirements by combining the strengths of the existing AOSE methodologies (i.e. by reusing and enhancing the various strong techniques and model definitions of the existing methodologies where appropriate) and proposing new techniques and model definitions where necessary.

Given the above major improvements of MOBMAS over the existing AOSE methodologies, this research helps to cultivate the maturity of the AOSE paradigm, which is still far away from reaching the maturity level of other conventional software engineering paradigms such as OO software engineering.

In addition, apart from MOBMAS, this research also makes other notable contributions to the literature on AOSE.

- It recommends a list of *methodological requirements for an AOSE methodology*. These requirements consist of a set of *features* that an AOSE methodology should support, a set of *steps* that the MAS development process should include, and a set of *modelling concepts* that MAS development model kinds should represent. They were identified by investigating the literature on AOSE, namely, the various evaluation frameworks on AOSE methodologies and conventional system development methodologies, as well as the documentation of the various existing AOSE methodologies. These requirements were also validated by conducting a survey on practitioners and researchers in the field.

  The identification of AOSE methodological requirements has a significant contribution to the future research in AOSE, because it establishes a sensible starting point for the development of new AOSE methodologies, namely new AOSE development process, techniques and model definitions. To date, no study has been found that attempts to identify these AOSE methodological requirements. This research therefore represents a pioneering effort in this area.

- This research also proposed a comprehensive and multi-dimensional *feature analysis framework* for the evaluation and comparison of AOSE methodologies. Developed from the synthesis of various existing evaluation frameworks (both for AOSE methodologies and for conventional system development methodologies), the novelty of the proposed framework lies in the high degree of its completeness and relevance. The framework consists of evaluation criteria that assess an AOSE methodology from both the dimensions of system engineering and those specific to AOSE. It also pays attention to all three major elements of a system development methodology: development process, techniques and model definitions.

# 8.3. LIMITATIONS OF THE RESEARCH

## 8.3.1. Limitations of the survey on practitioners and researchers

The survey was conducted by this research to validate the methodological requirements proposed for an AOSE methodology (Section 5.3). Since the AOSE paradigm is still very young, the number of practitioners and researchers who participated in the survey was expected to be small. Eventually the survey sample was 41. Although this is not a small number, a larger sample size would provide a more reliable assessment of the professional opinions on the importance of the proposed AOSE requirements.

## 8.3.2. Limitations of the feature analysis on the existing AOSE methodologies

This research evaluated the 16 existing AOSE methodologies in term of their support for each AOSE methodological requirement (Section 5.4). This investigation was based on the published documentation of each methodology. Even though this material provided a relatively comprehensive description of each methodology, it might omit discussion of the methodology's support for some particular features (e.g. which software development lifecycle the methodology adopts, or whether a methodology is capable of supporting dynamic systems). Accordingly, this research sometimes had to deduce a methodology's support for a particular methodological requirement from the

published documentation (if possible), or concluded that the methodology does not support the feature. This evaluation may be subject to error.

In addition, with regard to the evaluation of criteria "*Usability of the development process*" and "*Usability of techniques*", this research arrived at its assessment by *non-empirically* reviewing the methodology's steps and model definitions. A more reliable evaluation would be to empirically apply the 16 methodologies on an (identical) application. Unfortunately, the constraints in time and resources prevented this research from conducting such an empirical evaluation.

### 8.3.3. Limitations of the comparison between MOBMAS and the existing AOSE methodologies

The comparison of MOBMAS and the 16 existing methodologies in term of criteria "*Usability of the development process*" and "*Usability of techniques*" is potentially biased, because MOBMAS' usability was *empirically* assessed by the two external developers who used the methodology on an application, while the usability of the existing methodologies was *non-empirically* assessed by the researcher. As such, the evaluators were different and the methods of usability evaluation were also different. To obtain an ideal comparison, the two developers who evaluated MOBMAS should also use all of the 16 existing AOSE methodologies on the same application, thereby comparing the usability of all methodologies. However, the constraints in time and resources prevented this research from conducting such an empirical comparison.

## 8.4. SUGGESTIONS FOR FUTURE RESEARCH

Since the main output of this research is the proposal of an AOSE methodology, there are basically two general directions of future research: making extensions to the proposed methodology, and applying the methodology to a variety of applications.

### 8.4.1. Extending MOBMAS

MOBMAS may be extended in two major ways.

- *Adding more techniques and/or modelling notation to support the currently provided features, steps and modelling concepts*: As research on AOSE continually grows, new techniques and/or modelling notation may arise for supporting the features, steps and modelling concepts that are currently addressed in MOBMAS (e.g. new techniques for the identification of agent classes, new mechanisms for agent interactions, or new notation for ontology modelling). These new ideas should be recognised and included in MOBMAS (if applicable) to improve MOBMAS' powerfulness.

- *Adding support for new features, steps and modelling concepts*: Currently, MOBMAS provides support for a variety of features, steps and modelling concepts that have been determined to be important to an AOSE methodology. However, to extend MOBMAS' capability and applicability, new features, steps and modelling concepts can be added to MOBMAS. For example, new techniques and model definitions can be introduced to provide support for the development of MASs with mobile agents, or agents with personality.

### 8.4.2. Applying MOBMAS to a variety of applications

In this dissertation, MOBMAS has been applied to a "Peer-to-Peer Information Sharing" application by two external developers (Appendix H). To further validate MOBMAS, the methodology should be tested on other demonstrative applications, and/or employed in many real-world development projects. Preferably, MOBMAS should be tested and/or used on applications of diverse domains, sizes and/or degrees of complexity. In addition, a potential revenue for future research is to apply MOBMAS on the same application as that previously used by an existing MAS development methodology(ies). This would facilitate a reliable comparison between MOBMAS and the existing methodology(ies) in term of usability (as had been discussed in Section 8.3.3).

# 8.5. CONCLUDING REMARKS

In summary, this research has proposed a software engineering methodology for the analysis and design of ontology-based MASs. This methodology improves on the existing AOSE methodologies in terms of its comprehensive support for ontology-based MAS development, and its support for various other important features, steps and modelling concepts of MAS analysis and design that are not well-supported by the existing methodologies. The proposed methodology has been applied to a "Peer-to-Peer Information Sharing" application by two external developers. It is hoped that the methodology will be applied to many other MAS development projects and widely recognised and adopted by the AOSE community.