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

Quynh-Nhu Numi Tran, Graham Low *

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

Received 14 January 2007; received in revised form 2 July 2007; accepted 17 July 2007
Available online 6 August 2007

Abstract

Ontologies offer significant benefits to multi-agent systems: interoperability, reusability, support for multi-agent system (MAS) development activities (such as system analysis and agent knowledge modeling) and support for MAS operation (such as agent communication and reasoning). This paper presents an ontology-based methodology, MOBMAS, for the analysis and design of multi-agent systems. MOBMAS is the first methodology that explicitly identifies and implements the various ways in which ontologies can be used in the MAS development process and integrated into the MAS model definitions. In this paper, we present comprehensive documentation and validation of MOBMAS.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Agent-oriented software engineering; Analysis and design methodologies; Ontology; Interoperability; Reusability

1. Introduction

Recognizing the importance of formal methodologies to the widespread commercial adoption and success of the agent technology, a number of agent-oriented software engineering (AOSE) methodologies have been proposed, e.g., MaSE [13], GAIA [86], PROMETHEUS [62] and TROPOS [5]. These methodologies are diverse in their scope, focus and approach. Nevertheless, an evaluation of well-known methodologies [75,77] revealed that most are deficient in at least one of the following areas of MAS development:

- agent internal design – design of agent mental constructs such as beliefs, goals, plans and actions;
- agent interaction design – design of interaction protocols and exchanged messages; and
- MAS organization modeling – design of acquaintances and authority relationships amongst agents or agents’ roles.

In addition the majority of existing AOSE methodologies do not consider interoperability with other systems in heterogeneous environments and design reuse. These are important concerns for any system, which will ultimately affect the take-up of agent technology by industry.

In a previous paper, we proposed a methodological framework for MAS development, which addresses these concerns via extensive support for ontology-centric AOSE [76]. The framework consists of a repository of development steps and modeling concepts that supports the key methodological requirements of AOSE. This framework was validated by a survey of AOSE practitioners and researchers and by a feature analysis of existing AOSE methodologies.

While limited ontological support is provided by a number of existing AOSE methodologies (e.g., MAS-CommonKADS [36], MaSE [13], and PASSI [10]), these methodologies do not incorporate the use of ontologies...
throughout the entire systems development lifecycle nor consider the diverse ways in which ontologies can be used to account for interoperability and verification during design.

As an instantiation of our framework, MOBMAS (Methodology for Ontology-Based Multi-Agent Systems) provides all the features that the framework aims to offer. In particular, MOBMAS is capable of supporting ontology-centric MAS development (which leads to the support for interoperability and reuse) and various other AOSE methodological requirements that are important to an AOSE methodology but which may not be well-supported by existing methodologies.

We developed MOBMAS by specializing, combining and ordering the steps of the framework into a development process, and by specifying a set of model definitions to represent the framework’s modeling concepts [76]. Where ever possible we reused and/or enhanced techniques and model definitions from existing AOSE methodologies. In so doing, we aimed to combine the strengths of existing AOSE methodologies into MOBMAS.

In this paper, we present comprehensive documentation and validation of MOBMAS. The remainder of the paper is organized as follows: Section 2 presents an overview of the MOBMAS development process. Sections 3–7 describe the five core activities in the development process of MOBMAS: Analysis, MAS Organization Design, Agent Internal Design, Agent Interaction Design, and Architecture Design. Each section elaborates on the activity’s associated steps, techniques and model definitions. Section 8 documents our work in validating the MOBMAS methodology. It was initially reviewed and refined based on the feedback of two experts in AOSE. The refined methodology was then used by two different developers to design a peer to peer community-based information sharing application. The feedback from the developers was also used to refine MOBMAS into its final version as documented in this paper. Lastly, a feature analysis was conducted to verify MOBMAS’ ability to support important AOSE methodological requirements [76]. In Section 9, we present our comparison of MOBMAS against other well-known AOSE methodologies.

2. Overview of MOBMAS

Conforming to the definition of a software engineering methodology [35], MOBMAS is comprised of a software engineering process that contains activities and associated steps to conduct the system development, techniques to assist the process steps and a definition of the models.

Fig. 1 shows the activities and steps in MOBMAS. Each MOBMAS step is associated with a model(s) or work product. The development process of MOBMAS is highly iterative and incremental, either within or between activities. In total, there are five activities, each focusing on a significant area of MAS development:

- **Analysis Activity:** Develops a conception for the future MAS, namely a first-cut identification of roles that comprise the MAS organization. Application ontologies required by the target MAS are also identified and modelled.
- **MAS Organization Design Activity:** Specifies the organizational structure for the target MAS and defines a set of agent classes that comprise the system. If the MAS is a heterogeneous system that incorporates non-agent resources, these resources are identified and their associated ontologies captured.
- **Agent Internal Design Activity:** Deals with the specification of each agent class, including its belief conceptualization, goals, events, plan templates and reflexive rules.
- **Agent Interaction Design Activity:** Designs the interactions between agent classes by selecting a suitable interaction mechanism for the target MAS and then defines the patterns of data exchanges amongst agent classes.
- **Architecture Design Activity:** Deals with various architecture-related issues, namely the identification of agent-environment interface requirements, the selection of the agent architecture, the identification of required infrastructure facilities, the instantiation of agent classes and the deployment configuration of agents.

In the following sections, we document each activity of MOBMAS, including its steps, potential techniques to perform each step and the output of each step. A Product Search application is used as an illustrative case study. This application searches for and retrieves product information from heterogeneous information resources provided by potential suppliers (such as suppliers’ databases or 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/her search query. Upon receiving a query, the system extracts keywords from it, searching through the resources to gather information for the query, and displays the final answer to the user. The system also accepts and records feedback from users, which may help improve its future performance.

3. Analysis Activity

The Analysis Activity of MOBMAS aims to develop an understanding of the target problem domain and thereby arrive at a first-cut identification of the roles that comprise the future MAS.

3.1. Step 1 – Develop System Task Model

3.1.1. Description

MOBMAS starts with the development of a System Task Model to capture the specifications of functionality that the target MAS should provide. These specifications are referred to as “system tasks”.

3.1.2. Potential techniques

The developer is referred to traditional requirements engineering techniques for identifying system tasks, e.g., [12,15,33,50,68].

3.1.3. Output

A System Task Model containing one or more System Task Diagrams is produced. Each System Task Diagram shows the identity of system tasks, conflicts amongst them (if any) and their functional decomposition (which can be either AND- or OR-decomposition, and “full” or “partial” decomposition). An example System Task Diagram for the Product Search application is presented in Fig. 2.

After a (preliminary) System Task Model is constructed, the model should be validated against the
Ontology Model, which is described in Step 4 of this activity. The ontologies specified in the Ontology Model may help to reveal new system tasks that have not yet been discovered, thus assisting in the refinement of the System Task Model.

3.2. Step 2 – Analyze Organizational Context (Optional)

3.2.1. Description

The organizational context refers to the human organization which the MAS aims to support, automate or monitor. The existing structure of this organizational context may assist the upcoming task of MAS role identification, because the MAS organizational structure (defined by roles) can mimic the existing structure of the organization which it supports. This step should only be performed if the existing structure of the human organization is clearly defined and has been accepted as an effective way to function.

3.2.2. Potential techniques

The developer should identify the existing “organizational units” (i.e., the positions or individuals or departments that exist in the organization) and the relationships between these units. For example, in the Product Search application, the developed MAS aims to support a human organization where organizational units are “enquirers” and “information providers” (e.g., product suppliers and search engines specialists).

3.2.3. Output

The structure of the MAS’ organizational context is captured in an Organizational Context Model via the Organizational Context Chart. This chart shows the identified organizational units and their relationships. An example Organizational Context Chart for the Product Search application is shown in Fig. 3.

3.3. Step 3 – Develop Role Model

3.3.1. Description

A role refers to a position in the MAS organization [14,20]. It serves as the building block for defining agent classes, because each agent class will be associated with one or more roles. Roles also define the expected behaviour of each agent class. This step identifies a set of roles comprising the target MAS and the potential acquaintances (or interaction pathways) between them.

3.3.2. Potential techniques

Roles are identified from system tasks and the organizational context’s structure (if Step 2 – “Analyze Organizational Context” has been performed).

Each system task in the System Task Model can be assigned to one role. For example, system task “Retrieve information from resources” of the Product Search MAS
(Fig. 2) can be mapped to the role “InfoSource Wrapper” (Fig. 4). However, each role can take on 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 cohesion and loose coupling [49,61]. For instance, in the Product Search application, all system tasks dealing with user interactions can be assigned to the role “User Interface” (Fig. 4). A single system task may be assigned to multiple roles if the system task requires the collective, shared effort of various roles; all the participating roles are equally accountable for the accomplishment of the task; or the control of the task needs to be shared among the participating roles.

If Step 2 – “Analyze Organizational Context” has been performed, the existing structure of the organizational context should be examined to verify or refine the set of roles identified from the system tasks. In general, each leaf-node organizational unit in the Organizational Context Chart can be represented by a role.

For each role, “role-tasks” should be identified. These are tasks that a particular role is responsible for fulfilling. Role-tasks can be directly mapped from the system tasks delegated to the role during the process of role identification. This mapping is generally one-to-one. For example, the role-tasks of “User Interface” are “Accept user query”, “Display result for query”, “Receive user feedback” and “Display acknowledgement” (Fig. 4).

3.3.3. Output

The Role Diagram shows the specification of roles, acquaintances between roles and authority relationships governing inter-role acquaintances. The definition of authority relationships between roles is discussed in the MAS Organization Design Activity (Section 4.1). Fig. 4 shows the preliminary Role Diagram for the Product Search MAS.

3.4. Step 4 – Develop Ontology Model

3.4.1. Description

This step identifies and models the application ontologies required for the operation of agents in the target MAS. Application ontologies are specific to a particular application because concepts in these ontologies are related to both application domains and tasks [29,32,34]. MOB-MAS recommends classifying application ontologies as:

- “MAS Application ontologies” which conceptualize the application provided by the MAS. They define all the concepts and relations that the agents need to know and share about the MAS application (e.g., about the MAS domains and tasks).
- “Resource Application ontologies” which conceptualize the applications provided by resources in the MAS. If the resource is an information source (e.g., a database),

![Fig. 4. Role diagram.](image-url)
its Resource Application ontology defines the conceptual schema of the resource’s data. If the resource is a processing application system (e.g., a legacy system), its Resource Application ontology defines the concepts and relations that conceptualize the domains and services of the resource. Resource Application ontologies are only necessary for heterogeneous MASs that contain non-agent software resources. In these systems, only agents that directly interface with the resources need knowledge of the Resource Application ontologies.

3.4.2. Potential techniques

The developer is referred to ontology-engineering methodologies or guidelines for ontology development, e.g., [21,31,46,55,79]. MOBMAS suggests guidelines for identifying input information for the development of MAS Application ontologies, while Resource Application ontologies are considered in the MAS Organization Design Activity (Section 4).

MAS Application ontologies are generally built by selecting concepts from either, or both, of the relevant Domain ontologies and Task ontologies, thereafter specializing or instantiating these concepts to suit the MAS application [32,34]. Domain ontologies define concepts specific to a domain. For example, a Car Domain Ontology may define concepts such as “Make”, “Steering”, and “Price”. On the other hand, Task ontologies define domain-independent concepts that are related to generic tasks (e.g., negotiation task) or problem-solving methods (e.g., propose-and-revise method). For instance, a Negotiation Task Ontology may contain concepts “Offer” and “Utility rating”. As a specialization of the Domain and Task ontologies, a Car Application Ontology may define a concept “Car-price-offer”, which specializes concept “Price” from the Car Domain Ontology and concept “Offer” from the Negotiation Task Ontology.

To identify the relevant Domain ontologies, the System Task Model and Role Model should be examined. System tasks and role-tasks provide an overview of the MAS’ purpose and scope, thereby revealing the specific domains targeted by the MAS.

To determine if Task ontologies are needed, examine the knowledge required by the MAS role-tasks against the domain knowledge provided by Domain ontologies [29]. If there is a syntactic mismatch (where the knowledge required by the tasks can be satisfied by the domain knowledge, but the domain knowledge needs to be rearranged or at least renamed) the MAS Application ontology can be derived solely from the Domain ontologies. However, if there is a semantic mismatch (where the knowledge needed by role-tasks is not included in the Domain ontologies), these concepts should be included from the Task ontologies. For example, in the Product Search application, the 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, the developer should consult the Ranking-by-Weight Task Ontology to obtain task-specific concepts such as “Hypothesis”, “Weight” and “Rank” for the MAS Application ontology. 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 a Task ontology. The knowledge requirements of the tasks may not be apparent until the Agent Internal Design Activity (Section 5), implying the need for iterative refinement of the MAS Application ontologies.

When developing MAS Application ontologies, the developer should specify “ontological mappings” between the ontologies where necessary. The related MAS Application ontologies can either be mapped against each other, or against a common ontology. Normally, when there are more than two ontologies to be mapped, the second approach is preferred. Generating ontological mappings is a labor-intensive and error-prone task. The developer is referred to [16,39,51,70] for more information.

3.4.3. Output

An Ontology Model is produced that captures the MAS Application ontologies. This model is subject to ongoing refinement and verification throughout the development process.

Various modeling notations can be used. For illustration purposes, UML and OCL are used. The Ontology Model in this case consists of UML Ontology Diagrams, each of which models a MAS Application ontology as an UML class diagram [11]. For ontological mappings, MOBMAS suggests extending the notation for dependency relationships in UML with keywords stating the semantic correspondence. If there are axioms, rules or other assertions that specify constraints on ontological classes, attributes and relationships, these can be specified as OCL comment notes in the Ontology Diagrams. Fig. 5 shows the Car application ontology for the Product Search MAS.

While the System Task Model and Role Model assist in the development of MAS Application ontologies, the latter can be used to validate and refine the two MAS analysis models. In particular, the concepts defined in MAS Application ontologies may correspond to, or indicate, system tasks or roles. For example, if the concept “Keyword” has been defined in the MAS Application ontology of the Product Search system, the developer may uncover a system task “Extract keywords from user query” and add it to the System Task Model if not already included.

3.5. Step 5 – Identify Ontology-Management Role

3.5.1. Description

Shared ontologies are typically stored at ontology servers in a MAS [27]. This step determines whether the ontology servers will be directly accessed by client agents as a common knowledge base, or whether access is controlled by a dedicated “ontology manager” agent.
3.5.2. Potential techniques

MOBMAS highlights the advantages and disadvantages of each approach. The “Ontology Manager” agent approach 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 authorized to obtain a requested ontology or to update an ontology. Conversely, allowing agents direct access to ontology servers provides design simplicity. However, any agent can access or change the ontologies (unless the ontologies are set to “read only” mode) and the agents have to perform the ontological reasoning and mapping activities by themselves to satisfy their ontology-related queries.

3.5.3. Output

If the ontology-manager-agent approach is adopted, the Role Diagram should be updated to show the Ontology Manager role (Fig. 4). The developer normally does not need to design the Ontology Manager agents because these are often application-independent components that come with the implementation framework (e.g., FIPA-based platforms such as JACK [67], JADE [19], FIPA-OS [18], and ZEUS [73]).

4. MAS Organization Design Activity

This activity is concerned with refining the organizational structure of the target MAS and identifying a set of agent classes comprising the MAS. If the MAS is a heterogeneous system that contains non-agent resources, these resources need to be identified and their applications conceptualized.

4.1. Step I – Specify MAS Organizational Structure

4.1.1. Description

In MOBMAS, MAS organizational structure is defined by the Role Model, where roles, inter-role acquaintances and authority relationships are specified. This step refines the preliminary MAS organizational structure developed in step “Develop Role Model” of the Analysis Activity (Section 3.3), by selecting an organizational style and defining the authority relationship(s) between roles.

4.1.2. Potential techniques

There are many potential organizational styles that a MAS can adopt, e.g., [40,47,71]. The four basic styles are peer-to-peer, hierarchical, federation and hybrid [63]. Various factors should be considered when choosing the MAS organizational style, including the existing structure of the real-world organization which the MAS supports, modularity, non-functional requirements (e.g., security) and the number of roles in the system. The chosen organizational style will determine the authority relationships amongst roles. Two basic types of authority relationships are “peer” and “control”. The former represents a peer-to-peer relationship where two roles have equal status. The latter indicates a superior-subordinate relationship where one role has authority over another.
4.1.3. Output

The Role Model is updated to show the identified authority relationships between roles, and to include new roles and/or inter-role acquaintances. Fig. 6 shows the Role Diagram for the Product Search MAS. The system adopts a primarily peer-to-peer structure, although it contains a superior-subordinate relationship between “Searcher” and “InfoSource Wrapper” roles.

4.2. Step 2 – Develop Agent Class Model

4.2.1. Description

In MOBMAS, agent classes are built upon roles. This step identifies a set of agent classes comprising the target MAS by determining which roles are to be performed by which agent classes.

4.2.2. Potential techniques

Roles are generally assigned to agent classes via one-to-one mappings. However, multiple roles may be mapped to a single agent class provided that the resulting agent class does not have disparate functionality. Other factors to consider are efficiency and non-concurrency requirements.

Each agent class should be characterized as either static or dynamic. A static agent class is one whose instances play all of the assigned roles throughout their lifetime. On the other hand, a dynamic agent class is one whose instances may change their active roles from one time to another [57].

4.2.3. Output

All agent classes in a MAS should be described in the Agent Class Model, which is depicted by two notational components:

- Agent Class Diagram which defines each agent class, including its dynamics, roles, belief conceptualization, goals and events.
- Agent Relationship Diagram which shows the acquaintances between agent classes (i.e., their interaction pathways), connections between agent classes and their wrapped resources (if any), and instantiation cardinality of each agent class.

These diagrams are developed in an ongoing manner throughout the development process. At this stage, the Agent Class Diagram is left empty because the agent internal constructs are not yet defined. Fig. 7 presents the preliminary Agent Relationship Diagram for the Product Search MAS.

4.3. Step 3 – Specify Resources (Optional)

4.3.1. Description

This step is required if the target MAS is a heterogeneous system which contains non-agent resources that provide application-specific information and/or services to the agents. Resources may include information sources (e.g., databases or web servers) and processing application systems (e.g., legacy systems) [24].
4.3.2. Potential techniques

Resources can be identified by reviewing the System Task Model or Role Model. These models describe the functionality of the target application, thereby revealing the major resources that accompany, or are required by, the target MAS system. For example, the potential resources for the Product Search MAS are various external databases, web servers and search engines on cars. However, more resources may be discovered during the Agent Internal Design Activity (Section 5).

4.3.3. Output

All resources in a MAS should be modeled in the Resource Model using Resource Diagram(s). A Resource Diagram shows the description of each resource and the wrapper agent classes. MOBMAS suggests two basic dimensions for resource description: resource type (i.e., category of the resource) and Resource Application ontology (i.e., the name of the ontology conceptualizing the data, domains and/or services of the resource). Fig. 8 shows a Resource Diagram for the Product Search MAS.

The Agent Relationship Diagram should be extended to show newly identified resources and their connections with wrapper agent classes (Fig. 7). The Role Model may also need to be revised to add necessary resource-related roles, such as “Wrapper” roles. If the Role Model is updated, both the Agent Class Diagram and the Agent Relationship Diagram need to be updated to show new agent classes and/or new role assignments to existing agent classes.

4.4. Step 4 – Extend Ontology Model to include resource application ontologies (optional)

4.4.1. Description

If the target MAS contains non-agent resources, the developer should extend the Ontology Model to include...
ontologies conceptualizing the resources’ data, domains and/or services.

4.4.2. Potential techniques

Generally, each resource should be conceptualized by a separate Resource Application ontology. Resource Application ontologies should then be mapped against relevant MAS Application ontologies, because these mappings allow the wrapper agents to translate ACL messages into resource-level commands, and from resource-level results back to ACL messages. These mappings also allow for the interoperability between heterogeneous resources. Specifically, information retrieved from heterogeneous resources can be integrated using a MAS Application ontology as an inter-lingua.

4.4.3. Output

The Ontology Model should be updated with the Resource Application ontologies. Fig. 9 shows an example Resource Application ontology for the Car database used by the Product Search MAS (named “CarInfo Resource Ontology”; cf. Fig. 8). The diagram also shows the ontological mappings between “CarInfo Resource Ontology” and “Car Application Ontology” (cf. Fig. 5).

5. Agent Internal Design Activity

This activity deals with the internal design of each agent class, namely the specification of each agent class’s belief conceptualization, agent-goals, events, plan templates and reflexive rules.

5.1. Step 1 – Specify agent class belief conceptualization

5.1.1. Description

Agent beliefs refer to the information that an agent holds about the world [66]. Agent beliefs exist at two levels of abstraction: belief state and belief conceptualization [42]. Belief states are run-time factual knowledge about particular states of the agent’s world. Meanwhile, belief conceptualization is the conceptual knowledge of the world described in the belief states. At design time, it is only feasible to define the “initial” belief conceptualization of each agent class because agents may dynamically update their conceptualization during their lifetime.

5.1.2. Potential techniques

The belief conceptualization of an agent class comprises the ontologies conceptualizing its application (i.e., its domains and tasks) and/or its wrapped resources’ applications. Only agent wrapper class(es) for 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 [24,38]. Accordingly, the specification of an agent class’s belief conceptualization involves the identification of which MAS Application ontologies and/or Resource Application ontologies to which the agent class should commit. In many cases, the agent class only needs to commit to a fragment of a particular MAS Application ontology or Resource Application ontology.

In general, an agent class should commit to a particular (part of) ontology if its functionality is related to the domain, task or resource that the (part of) ontology conceptualizes. In MOBMAS, an agent class’s functionality is reflected via its roles, role-tasks and acquaintances with other agent classes. As such, the Role Model and Agent Relationship Diagram should be examined. For example, the “Wrapper” agent class in the Product Search MAS plays the “InfoSource Wrapper” role (Fig. 7). It should therefore commit to the CarInfo Resource Ontology (Fig. 8) in order to execute its role-task “Retrieve information from resources”. The “Wrapper” agent class should also commit to the Car Application Ontology in order to communicate car-related messages with the “Searcher” agent class. Note that not all ontological commitments of an agent class are apparent at this stage. Step “Develop Agent Behaviour Model” (Section 5.4) and the Agent Interaction Design Activity (Section 6) provide additional insight into the knowledge requirements of each agent class.

5.1.3. Output

The Agent Class Diagram is updated to show the ontologies making up the belief conceptualization of each agent class. Fig. 10 presents the updated Agent Class Diagram of the “Searcher” agent class for the Product Search MAS.

5.2. Step 2 – Specify agent-goals

5.2.1. Description

An agent-goal is a state of the world that an agent class would like to achieve or satisfy [83]. This step aims at identifying the agent-goals of each agent class.

---

Fig. 9. CarInfo resource ontology and its mappings to car application ontology.
5.2.2. Potential techniques

Agent-goals can be identified directly from the agent class’s role-tasks. The state of the world that each role-task seeks to achieve, satisfy or maintain indicates an agent-goal.

5.2.3. Output

The Agent Class Diagram of each agent class is updated to show its agent-goals (Fig. 10). If a particular agent class is found to pursue multiple-related agent-goals, an Agent-Goal Diagram can be developed to capture the relationships between these agent-goals. Fig. 11 illustrates the Agent-Goal Diagram for the “Searcher” agent class in the Product Search MAS (cf. Fig. 10).

5.3. Step 3 – Specify events

5.3.1. Description

An event is a significant occurrence in the environment which activates an agent to pursue its goals, or changes the agent’s course of actions in achieving the goals [81]. This step identifies the events to be dealt with by instances of each agent class.

5.3.2. Potential techniques

Events arise from various sources: from agents via the execution of their actions, from human users via their inputs into the system, or from non-agent resources via the execution of their services. For each agent class, it is necessary to identify these events.

5.3.3. Output

The Agent Class Diagram is updated to show the events affecting each agent class (Fig. 10).

5.4. Step 4 – Develop Agent Behaviour Model

5.4.1. Description

This step specifies how each agent class behaves in order to achieve or satisfy each agent-goal. Two major styles of agent behaviour are “planning” and “reflexive acting” [65,84]. Planning and reflexive acting are often referred to as “deliberative” and “reactive” behaviour respectively (e.g., [7,69,84]).

Planning requires an agent to perform 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, and the failure or success of past actions. Reflexive acting, on the other hand, allows the agent to behave in a hard-wired situation-action manner, similar to reflexes. The agent simply follows predefined reflexive rules to determine which actions it should execute next.

The developer should firstly identify which style of behaviour is more appropriate to each agent class for the achievement of each of its agent-goals. Note that an agent class may adopt different styles of behaviour for different agent-goals. Factors to consider when choosing the behavioural style include complexity of reasoning required by an agent-goal, real-time requirement of an agent-goal, and predictability of environment situations.

For each agent class, depending on the chosen agent behavioural style, either (or both) of the following two styles may be adopted.

- **Planning**
  - Requires logical reasoning to choose among possible courses of actions
  - Takes into account current state of the environment, events, and past actions

- **Reflexive acting**
  - Based on predefined rules
  - Quick and efficient

Fig. 10. Agent class diagram for “Searcher” agent class.

Fig. 11. Agent goal diagram of “Searcher” agent class.
sub-steps should be performed: “Develop Agent Plan Templates” and “Develop Reflexive Rule Specification”. The outputs from these sub-steps comprise the Agent Behaviour Model.

5.5. Sub-step 4.1 – Develop Agent Plan Templates

5.5.1. Description
Generally, agent plans are formed on the fly at run-time by planners, reasoners or means-end analyzers provided by the agent implementation platforms (e.g., STRIPS [22], IPEM [1], AUTODRIVE [82], and IRMA [4] platforms). At design time, the developer can assist this run-time plan formation by identifying the important information that will be needed by the planners to build plans for each agent-goal. This sub-step develops Agent Plan Templates to capture this input information.

5.5.2. Potential techniques
For each agent-goal that requires planning behaviour, the developer should identify the triggering events, the set of actions an agent can perform, events that affect an agent’s course of action, commitment strategy and conflict resolution strategy.

5.5.3. Output
An Agent Plan Template should be developed for each agent-goal that requires planning behaviour. An example Agent Plan Template is presented in Fig. 12. The datatype for each variable in the template should be specified. For example, the datatype of the variable “keywords” is “User query.Keyword” (interpreted as “Keyword” of “User query”), where “User query” and “Keyword” are concepts from the application ontology committed by the agent class (in this case, the Query Application Ontology). The datatypes are highlighted in italic in Fig. 12.

<table>
<thead>
<tr>
<th>Initial state: (keywords: User query.Keyword) are known</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target agent-goal: Information is gathered from resources</td>
</tr>
<tr>
<td>Commitment Strategy: single-minded</td>
</tr>
<tr>
<td>Sub-agent-goal: “Appropriate resources are found”, cf. sub-plan X</td>
</tr>
<tr>
<td>Action 1: sendQueryToWrapper (keywords: User query.Keyword)</td>
</tr>
<tr>
<td>Pre-condition: Sub-agent-goal “Appropriate resources are found” is achieved successfully, and (resourceNo: Integer) &gt; 0</td>
</tr>
<tr>
<td>Effect: Event 2 incurs</td>
</tr>
<tr>
<td>Pre-condition: (message: Result list) is received from Wrapper agents, and (carResultArray: [Car]*) is empty</td>
</tr>
<tr>
<td>Effect: (carResultArray: [Car]*) is populated</td>
</tr>
<tr>
<td>Action 3: cancelSearch()</td>
</tr>
<tr>
<td>Pre-condition: Event 1 incurs</td>
</tr>
<tr>
<td>Effect: agent-goal is forfeited</td>
</tr>
<tr>
<td>Event 1: input of user’s cancel request</td>
</tr>
<tr>
<td>Event 2: incoming message from Wrapper agents</td>
</tr>
</tbody>
</table>

Fig. 12. Agent plan template for agent-goal “Information is gathered from resources” of “Searcher” agent class.

In cases where a tentative course of action exists for achieving a particular agent-goal, this sequence can be captured in an Agent Plan Diagram. The notation of Kinny [44] is adopted for the Agent Plan Diagram. Each state in the diagram represents a sub-agent-goal or an action, while each transition occurs when an event happens and/or a condition applies (Fig. 13).

5.6. Sub-step 4.2 – Develop Reflexive Rule specifications

5.6.1. Description
If an agent class achieves its agent-goal via reflexive behaviour, “reflexive rules” should be specified. These rules are basically (sequences of) “if-then” rules that couple stimuli and/or states of the environment with the actions to be executed by agents.

5.6.2. Potential techniques
For each agent-goal that requires reflexive behaviour, the developer should identify the set of actions an agent can perform, events and/or internal processing triggers that affect an agent’s course of action and guard conditions.

5.6.3. Output
Reflexive rules are defined in Reflexive Rule Specifications. MOBMAS adopts the UML Activity diagram notation for Reflexive Rule Specifications. Each action is depicted as an UML activity, while events, internal processing triggers and guard conditions are specified alongside the transition flows (Fig. 14).

In both sub-steps, the developed Agent Behaviour Model should be used to validate and refine the Ontology Model, and vice versa. Specifically, the datatypes of all variables in Agent Plan Templates and Reflexive Rule Specifications should be specified as concepts in the MAS Application ontologies and the Resource Application...
ontologies. Conversely, only concepts defined in the MAS Application ontologies and Resource Application ontologies should be used as datatypes for the variables in Agent Plan Templates and Reflexive Rule Specifications. For example, the terms used to datatype the variable “carModel” in Fig. 12, i.e., “Car” and “Model”, are obtained from the Car Application Ontology (cf. Fig. 5).

The Agent Behaviour Model should also be used to validate and refine the Agent Class Model because all agent-goals and events mentioned in the Agent Plan Templates and Reflexive Rule Specifications should be listed in the Agent Class Diagrams. The belief conceptualization of each agent class should also be checked to ensure that it contains the necessary application ontologies to describe the objects/entities mentioned in the Agent Behaviour Model.

6. Agent Interaction Design Activity

This activity 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.

6.1. Step 1 – Select Interaction Mechanism

6.1.1. Description

The developer decides which interaction mechanism is best suited to the target MAS. Two basic mechanisms of agent interaction are direct and indirect [80]. In direct interaction, agents exchange data by sending communication messages directly to each other following some interaction protocols. These messages are typically expressed in agent communication languages (ACL) such as KQML [78] and FIPA-ACL [23]. In indirect interaction, agents exchange data indirectly through a communication abstraction. A well-known abstraction is a tuplespace. Agents interacting through a tuplespace would insert “tuples” into, and remove them from, the shared tuplespace in an associative way. The tuplespace mechanism has advanced into the “tuple-centre” mechanism, where the shared space is no longer a mere communication channel, but a programmable reactive interaction medium [59,60].

6.1.2. Potential techniques

The direct interaction mechanism is more commonly used than the tuplespace/tuple-centre mechanism. This is because the direct mechanism has been widely adopted for object-oriented development, and many protocols have been proposed for direct agent interactions (e.g., FIPA 2002). In addition, the tuplespace/tuple-centre mechanism exhibits a strong centralized design due to the tuple-cen-
6.2. Step 2 – Develop Agent Interaction Model

6.2.1. For direct interaction mechanism

6.2.1.1. Description. For the direct interaction mechanism, the task of developing the Agent Interaction Model involves the task of defining the agent interaction protocols.

6.2.1.2. Potential techniques. The developer can use the Agent Behaviour Model to assist in defining the interaction protocols. Each communicative action specified in the Agent Behaviour Model (cf. Section 5.4) indicates an ACL message sent from one agent to another. Also, any event mentioned in the Agent Behaviour Model may itself be a message exchanged between agents. The interaction protocols should be specified to resolve any conflicts between interacting agents (if any), and ensure that all coordination rules governing the interaction are enforced. The developer can reuse and customize the catalogued patterns of interaction protocols provided by existing libraries, e.g., [25].

Although interaction protocols are normally directly hard-coded into the agents at implementation time, it is sometimes desirable to allow agents to dynamically acquire these protocols at run-time rather than having a priori knowledge at design time [72]. This is especially desirable if the agents exist in an open and dynamic environment. To enable the run-time acquisition of protocols by agents, the developer should:

- Define an ontology that conceptualizes the potential interaction protocols, and distribute this ontology to all relevant agent classes at design time. The ontology should contain concepts for describing interaction protocols, e.g., concepts “Protocol Name”, “Participating party”, “Message” and “Rule”. An example of such ontology is presented in [72].
- Define the interaction protocols in term of the shared ontology. This involves instantiating the concepts in the ontology with specific values so as to describe the target protocol.

6.2.1.3. Output. The output of this step is the Agent Interaction Model. For the direct interaction mechanism, the model is represented by a set of Interaction Protocol Diagrams, each graphically describing an interaction protocol. MOBMAS adopts the notation of AUML Sequence Diagram proposed by [56]. Each ACL message is defined in terms of a performative (e.g., inform, query-if, refuse), a list of arguments (whose datatypes must be specified), a guard-condition (if any) and a sequence-expression (if applicable) [25]. Fig. 15 presents an example Interaction Protocol Diagram for the Product Search MAS. It models the interaction protocol between a “Searcher” agent and a “Wrapper” agent. The datatype of each variable in the message is specified in italic.

6.2.2. For indirect interaction mechanism

6.2.2.1. Description. The task of agent interaction modeling for the indirect interaction mechanism involves the tasks of modeling the interactions between agents and the shared tuplespace/tuple-centre, and modeling the tuple-centre’s behaviour (if a programmable tuple-centre is used).

6.2.2.2. Potential techniques. For each potential conversation between agents, the developer should specify the tuples to be exchanged between the participating agents and the query protocol.
tuplespace/tuple-centre during the conversation. These tuples can be revealed by examining the communicative actions and events specified in the Agent Behaviour Model. Each tuple should be defined in terms of a communication primitive and a list of arguments whose datatypes must be specified. If a tuple-centre is used rather than a tuplespace, the reaction of the tuple-centre towards incoming tuples from agents should be specified.

6.2.2.3. Output. The Agent Interaction Model for the indirect interaction mechanism should contain Agent-TupleSpace Interaction Diagrams and (if a tuple-centre is used) Tuple-Centre Behaviour Diagrams. MOBMAS adopts the notation of AUML Sequence Diagram for Agent-TupleSpace Interaction Diagrams. The only difference is that the exchanged elements are tuples instead of ACL messages. Fig. 16 presents an example Agent-TupleCentre Interaction Diagram for a conversation between a “Searcher” agent and a “Wrapper” agent in the Product Search MAS. The conversation is the same as that previously described by Fig. 15, but this time it is via the shared tuplespace/tuple-centre. Again, the datatype of each variable in the tuples is specified (in italic).

MOBMAS adopts the notation of UML Statechart Diagram for the Tuple-Centre Behaviour Diagram. Each state 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 such as a tuple being inserted by an agent. An example Tuple-Centre Behaviour Diagram is presented in Fig. 17. It depicts the tuple-centre’s behaviour during the conversation between “Searcher” and “Wrapper” agents in the Product Search MAS where the tuple-centre can be programmed such that it allows the “Searcher” agent to get some responses (if any) within 10 s, and it only allows the “Searcher” agent to read the responses when either all “Wrapper” agents have responded, or when a certain period of timer has elapsed.

In both direct and indirect interaction mechanisms, the developer should validate the Agent Interaction Model against the Ontology Model, and vice versa. Specifically, the datatypes of all arguments in ACL messages and tuples must be defined as concepts in the MAS Application ontologies which are shared between the interacting agents. Vice versa, only concepts defined in the shared MAS Application ontologies can be used as datatypes for variable arguments in ACL messages and tuples. This requirement ensures that the semantics of information conveyed in ACL messages and tuples can be consistently interpreted by interacting agents. For example, the datatype of argument “carStock” in the ACL message presented in
Fig. 15 is “Car.number-in-stock”, where the terms “Car” and “number-in-stock” are obtained from the Car Application Ontology shared by “Searcher” and “Wrapper” agents (cf. Fig. 5).

The Agent Relationship Diagram should be updated to show new/changed interaction pathways between agent classes (if any) and various descriptive information about each interaction pathway, namely the interaction diagrams and the ontologies governing the semantics of these interactions. Fig. 18 presents the updated Agent Relationship Diagram for the Product Search MAS.

7. Architecture Design Activity

This activity deals with various design issues relating to agent architecture and MAS architecture. Its steps produce different notational components that constitute the Architecture Model.

7.1. Step 1 – Identify Agent-Environment Interface requirements

7.1.1. Description

In order to select appropriate sensors, effectors, agent architectures and an implementation platform for agents at implementation time, the developer should investigate the characteristics of the agents’ perception, effect and communication at design time.

7.1.2. Potential techniques

With regard to perception and effect, the developer should examine whether they are related to the physical or virtual world, the degree of complexity of perceptual inputs and effect outputs and the interaction with a human user. With regard to agent communication needs, most existing agent architectures and implementation platforms provide built-in support for basic communication operations such as a message transport service or a tuple exchange service. However, if the target MAS has special communication requirements such as encryption of exchanged messages or mobile and ubiquitous communication, these requirements should be identified.

7.1.3. Output

All requirements relating to sensors, effectors and communication services are documented in an Agent-Environment Interface Requirement Specification which constitutes the Architecture Model.

7.2. Step 2 – Select Agent Architecture

7.2.1. Description

This step decides the most appropriate architecture(s) for agents in the target MAS. Abstract internal constructs of each agent class (namely, belief conceptualization, agent-goals, plans and reflexive rules) are mapped onto the architectural modules during implementation.
7.2. Potential techniques

Given the availability of a large number of agent architectures, MOBMAS suggests using an existing architecture(s). Factors to be considered in this decision include whether the behaviour is planning or reflexive, required agent behavioural capabilities, desired style of control, knowledge representation mechanism, support for scalability and agent-environment interaction requirements.

7.2.3. Output

The selected agent architecture(s) is graphically modeled in an Agent Architecture Diagram. The diagram should show the architectural modules (or layers or subsystems) and the potential flows of data between these modules.

7.3. Step 3 – Specify MAS Infrastructure Facilities

7.3.1. Description

This step identifies the system components that are needed to provide system-specific services to agents in the MAS, referred to as infrastructure facilities.

7.3.2. Potential techniques

Various potential infrastructure facilities needed by a MAS are network facilities (e.g., agent naming service, agent migration service), coordination facilities (e.g., agent directory service, message transport service) and knowledge facilities (e.g., ontology servers, problem-solving methods servers) [37]. MOBMAS suggests either employing the “built-in” facilities provided by the MAS implementation platform; or providing and managing customized facilities via the use of dedicated agents.

7.3.3. Output

The specification of all necessary infrastructure facilities is included in an Infrastructure Facility Specification.

7.4. Step 4 – Instantiate Agent Classes

7.4.1. Description

This step determines the cardinality of instances for each agent class.

7.4.2. Potential techniques

Common types of cardinality for agent instantiation detailed in [85].

7.4.3. Output

The instantiation cardinality of each agent class is specified as an annotation next to the agent class name in the Agent Relationship Diagram (Fig. 18).

7.5. Step 5 – Develop MAS Deployment Diagram

7.5.1. Description

This step describes how the logical MAS architecture, which is specified in the Agent Relationship Diagram, can be actuated in the operational environment.
7.5.2. Potential techniques

The developer should specify the following deployment details: agent platforms, nodes (i.e., hosts on each agent platform), agent instances located at each node, connections between nodes and acquaintances between agent instances.

The deployment configurations can be determined by considering factors such as the message traffic between nodes (estimated from the Agent Interaction Model) and the required processing power of each node to accommodate the behavior of agents (estimated from the Agent Behaviour Model).

7.5.3. Output

A MAS Deployment Diagram should be constructed to capture the above deployment details. This diagram constitutes the Architecture Model. For modeling notation, MOBMAS adopts the notation of AUML Deployment Diagram [26]. Fig. 19 presents an example MAS Deployment Diagram for the Product Search MAS.

8. Validation of MOBMAS

The MOBMAS methodology was initially reviewed and refined based on the feedback of two experts in AOSE. The refined methodology was then used by two different developers to design a peer to peer community-based information sharing application based on [45,52]. The feedback from the developers was used to refine MOBMAS into its final version as documented in this paper. Both the expert reviews and test-uses by developers were conducted in a sequential order. Evaluation of the first expert/developer was used to refine MOBMAS before the second expert/developer was asked to evaluate/use the refined version.

This sequential and independent procedure prevented the possibility of two experts/developers identifying the same areas for improvement, and helped to identify new areas of improvement that might arise from the refinement of the methodology after the first review/test-use. In addition, the refinements made to MOBMAS as a result of the second expert’s/developer’s feedback were also discussed with the first expert/developer to ensure that no conflicts of opinions occurred.

Lastly a feature analysis was conducted on the final version of MOBMAS to verify MOBMAS’ ability to support important AOSE methodological features, steps and modeling concepts in our framework [75,77] (see Section 9.1). Tables 1–3 show that MOBMAS supports all the desired steps and modeling concepts from our framework. For each concept, the table shows the name of the model(s) that MOBMAS uses to represent or capture the concept.

9. Conclusions

9.1. Related work

We compared MOBMAS against sixteen well known methodologies: MaSE [13], MASSIVE [48], SODA [58], GAIA [86], MESSAGE [28], Methodology for BDI Agent [43], INGENIAS [64], Methodology with High-Level and Intermediate Levels [17], Methodology for Enterprise Integration [41], PROMETHEUS [62], PASSI [9], ADELFE [3], COMOMAS [30], MAS-CommonKADS [36], CASSIOPEIA [8], and TROPOS [5,6].

Of the 16 methodologies, only four were found to integrate ontologies into their MAS design, namely MAS-CommonKADS, MESSAGE, MaSE and PASSI.
Table 1
Support of MOBMAS for desirable AOSE features

<table>
<thead>
<tr>
<th>Desirable features</th>
<th>Support by MOBMAS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Desirable features of AOSE development process</strong></td>
<td></td>
</tr>
<tr>
<td>1. Specification of a system development lifecycle</td>
<td>MOBMAS adopts an iterative and incremental life cycle</td>
</tr>
<tr>
<td>2. Support for verification and validation</td>
<td>MOBMAS performs consistency checking between many of its models (e.g., between Agent</td>
</tr>
<tr>
<td></td>
<td>Class Model, Agent Behavior Model, Agent Interaction Model and Ontology Model)</td>
</tr>
<tr>
<td>3. Specification of steps for the development process</td>
<td>See Table 2</td>
</tr>
<tr>
<td>4. Specification of model kinds and/or notational</td>
<td>See Table 3</td>
</tr>
<tr>
<td>components</td>
<td></td>
</tr>
<tr>
<td>5. Specification of techniques and heuristics for</td>
<td>Each step of MOBMAS comes with a set of potential techniques</td>
</tr>
<tr>
<td>performing each process step and producing each model</td>
<td></td>
</tr>
<tr>
<td>kind</td>
<td></td>
</tr>
<tr>
<td>6. Support for refinability</td>
<td>All models in MOBMAS are iteratively refined and expanded throughout the system</td>
</tr>
<tr>
<td></td>
<td>development process (e.g., Role Model, Agent Class Model, Environment Model and Ontology</td>
</tr>
<tr>
<td></td>
<td>Model)</td>
</tr>
<tr>
<td><strong>Desirable features of AOSE models</strong></td>
<td></td>
</tr>
<tr>
<td>1. High degree of completeness/expressiveness</td>
<td>MOBMAS models can represent the target system from both static aspects such as agent/role</td>
</tr>
<tr>
<td></td>
<td>acquaintances, agent internal structure and system structure, and from dynamic aspects such</td>
</tr>
<tr>
<td></td>
<td>as interaction protocols and agent behavior MOBMAS models can also capture/represent a wide</td>
</tr>
<tr>
<td></td>
<td>variety of concepts (cf. Table 3)</td>
</tr>
<tr>
<td>2. High degree of formalization/preciseness</td>
<td>MOBMAS clearly defines the semantics and syntax of its notation</td>
</tr>
<tr>
<td>3. Provision of guidelines/logics for model derivation</td>
<td>MOBMAS provides techniques for transforming between its models and notational</td>
</tr>
<tr>
<td></td>
<td>components (e.g., from System Task Model to Role Model, or from Role Model and Ontology Model to</td>
</tr>
<tr>
<td></td>
<td>Agent Class Model)</td>
</tr>
<tr>
<td>4. Guarantee of consistency</td>
<td>MOBMAS provides rules for consistency checking among many of its model (e.g., between</td>
</tr>
<tr>
<td></td>
<td>Agent Behavior Model and Ontology Model, or between Agent Interaction Model and Ontology</td>
</tr>
<tr>
<td></td>
<td>Model)</td>
</tr>
<tr>
<td>5. Support for modularity</td>
<td>MOBMAS models agent classes as entities that encapsulate roles, goals, agent plan templates, reflexive rule specifications and ontologies</td>
</tr>
<tr>
<td>6. Manageable number of concepts in each model</td>
<td>Each MOBMAS model and notational component represents a manageable number of concepts</td>
</tr>
<tr>
<td>kind and each notational component</td>
<td></td>
</tr>
<tr>
<td>7. Model kinds expressed at various level of abstraction and detail</td>
<td>Many models of MOBMAS can be developed at various levels of detail (e.g., Agent Class Model, Role Model and Ontology Model)</td>
</tr>
<tr>
<td>8. Support for reuse</td>
<td>MOBMAS allows the developer to reuse various modeling elements such as role patterns, protocol templates, organizational structures and ontologies</td>
</tr>
<tr>
<td><strong>Desirable agent properties to be supported by AOSE models</strong></td>
<td></td>
</tr>
<tr>
<td>1. Autonomy</td>
<td>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)</td>
</tr>
<tr>
<td>2. Adaptability</td>
<td>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 behavior capabilities of an agent) when selecting an agent architecture</td>
</tr>
<tr>
<td>3. Cooperative behavior</td>
<td>MOBMAS supports the modeling of agent acquaintances and interaction protocols</td>
</tr>
<tr>
<td>4. Inferential capability</td>
<td>MOBMAS supports the modeling of agent plan templates and reflexive rules which help the agents to determine what to do to achieve its active goals at run-time</td>
</tr>
<tr>
<td>5. Knowledge-level communication ability</td>
<td>MOBMAS supports the modeling of ACL communication messages between agents, which are based upon speech-act performatives</td>
</tr>
<tr>
<td>6. Reactivity</td>
<td>MOBMAS models events and agents’ behavior to react to these events</td>
</tr>
<tr>
<td>7. Deliberative behavior</td>
<td>MOBMAS models agent classes as entities with purposes (represented as agent goals) and supports the modeling of agent plans for accomplishing these purposes in a deliberative manner</td>
</tr>
<tr>
<td><strong>Desirable features of AOSE methodology as a whole</strong></td>
<td></td>
</tr>
<tr>
<td>1. Support for open systems</td>
<td>MOBMAS facilitates the operation of newly added agents at run-time by explicitly modeling 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</td>
</tr>
<tr>
<td>2. Support for dynamic systems</td>
<td>MOBMAS supports the dynamic role-playing behavior of agents and offers model to capture/represent this dynamics. MOBMAS also considers the issue of system dynamics when selecting the interaction mechanism for the target MAS</td>
</tr>
</tbody>
</table>
| 3. 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 (e.g., “Model domain conceptualization”, “Define content of exchanged messages” and “Define agent information constructs”) and integrating ontologies into the model definitions of various models (i.e., Ontology Model, Agent Class Model, Agent Behavior Model and Agent Interaction Model)

(continued on next page)
However, these methodologies do not incorporate the use of ontologies throughout the entire systems development lifecycle nor consider the diverse ways in which ontology can be used to account for interoperability and verification at design time [74,76]. MOBMAS uses ontologies in the development process to facilitate the process of constructing and validating the MAS analysis and design models. Specifically, ontologies are used to help identify and validate the system tasks in System Task Model, actions of agent classes in Agent Behaviour Model, and exchanged messages between agents in Agent Interaction Model. MOBMAS also dedicates one of its models, namely the Ontology Model, to the representation of ontologies needed by agents in the target MAS. The Agent Class Model uses ontologies to conceptualize the agents’ beliefs. Agent Behaviour Model and Agent Interaction Model are also built upon ontologies: ontological concepts are used to formulate agent classes’ goals, plans, actions and content of communication messages.

By extensively exploiting ontologies, MOBMAS is able to provide various ontology-related strengths that are not provided, or provided to a lesser extent, by the other methodologies due to their lack of, or lesser, support for ontology. These strengths include:

- **Support for interoperability:** The MASs developed using MOBMAS can avoid interoperability issues between heterogeneous agents and between heterogeneous resources. This is because the knowledge or application of each agent and resource is explicitly conceptualized in ontologies, and the semantic mappings between these heterogeneous knowledge/applications are defined via ontological mappings. With a few exceptions (e.g., MaSE and SODA) most existing methodologies do not address the interoperability issues between MAS heterogeneous components. However, MASE does not consider non-agent resources.

- **Support for reusability:** The Ontology Model of MOBMAS offers a conceptual description of the target application. Thus MAS development projects can examine this model to determine whether, and which part(s) of, a past MAS design can be reused. Moreover, since the core design models produced by MOBMAS are built...
upon ontologies (namely, Agent Class Model, Agent Behaviour Model, and Agent Interaction Model), the developer can adapt past design models to a new application by changing the ontologies involved. In addition, MOBMAS’ support for interoperability (as discussed above) allows legacy agents and/or resources to be reused by a MAS system.

- **Support for verification of MAS analysis and design models:** MOBMAS uses ontologies to help verify the correctness and completeness of its MAS analysis and design models, namely System Task Model, Role Model, Agent Behaviour Model and Agent Interaction Model. Since ontologies are constructed by a separate development effort, they can serve as a reliable tool for verification. Some of the other methodologies offer support for verification: MaSE, MESSAGE, INGENIAS, PASSI, PROMETHEUS, ADELFE, and TROPOS [75,77]. However, their mechanism of verification is less reliable, because their MAS analysis and design models are verified internally and not against ontologies.

- **Well-structured, modular modeling of agents’ local knowledge:** Ontologies are used by MOBMAS as the building blocks for defining agents’ conceptual knowledge. This modeling mechanism results in agent knowledge models that are more structured and modular than those produced by the other methodologies. The latter do not organize agents’ knowledge into any modular conceptual structures.

- **Support for semantically consistent communication between agents:** By defining exchanged messages and tuples in terms of ontological concepts shared between the communicating agents, MOBMAS ensures that the agents are able to interpret the exchanged messages in a consistent manner. With the exception of MaSE and PASSI, the other methodologies do not provide this capability since they fail to integrate ontologies into agent interaction design.

- **Support for agent reasoning:** 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. This enables agents’ reasoning (which operationalizes the agents’ problem-solving knowledge) to utilize the ontology-based domain-related knowledge of agents at run-time. None of the other methodologies are found to explicitly associate agents’ problem-solving knowledge with agents’ ontological knowledge at design 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 resources has been formally documented in the ontologies of the Ontology Model. Other core models such as Agent Class Model, Agent Behaviour Model, and Agent Interaction Model are also consistently defined in terms of these ontologies. This support for maintainability is not demonstrated in the other AOSE methodologies.

We also compared MOBMAS against the sixteen methodologies using a comprehensive, multi-dimensional feature analysis framework that assesses the methods regarding their support for MAS-development steps, mod-
Table 4
Comparison of support for process steps

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MASE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MASSIVE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SODA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAIA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>INGENIAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDIM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HLIM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MEI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PROMETHEUS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PASSI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADELFE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COMOMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MAS-COMMONKADS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CASSIOPEIA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COMOMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MAS-COMMONKADS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TROPOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ORMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5
Comparison of support for modeling concepts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MASE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MASSIVE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SODA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAIA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>INGENIAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDIM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HLIM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MEI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PROMETHEUS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PASSI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADELFE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COMOMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MAS-COMMONKADS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CASSIOPEIA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COMOMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MAS-COMMONKADS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TROPOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ORMAS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Specification of a system development lifecycle</th>
<th>Support for verification &amp; validation</th>
<th>Different levels of modelling abstraction</th>
<th>Provision of guidelines/logics for model</th>
<th>Guarantee of consistency between models</th>
<th>Support for open systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASE</td>
<td>Iterative across all phases</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MASSIVE</td>
<td>Iterative View Engineering process</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Possibly</td>
</tr>
<tr>
<td>SODA</td>
<td>Not specified</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GAIA</td>
<td>Iterative within each phase but sequential between phases</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Rational Unified Process</td>
<td>Mentioned as future enhancement</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IINGENIAS</td>
<td>Unified software development process</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BDIM</td>
<td>Not specified</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HLIM</td>
<td>Iterative within and across the phases</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MEI</td>
<td>Not specified</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PROMETHEUS</td>
<td>Iterative across all phases</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PASSI</td>
<td>Iterative across and within all phases</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ADELFIE</td>
<td>Rational Unified Process</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COMOMAS</td>
<td>Not specified</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td>Cyclic risk-driven process</td>
<td>Mentioned but no clear guidelines</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CASSIOPEIA</td>
<td>Not specified</td>
<td>No</td>
<td>No</td>
<td>NA</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TROPOS</td>
<td>Iterative and incremental</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MOBMAS</td>
<td>Iterative and incremental</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
eling concepts and methodological features [75,77]. MOBMAS is found to be supportive of an extensive set of process steps and modeling concepts that span all key areas of MAS development, namely analysis, agent internal design, agent interaction design, and MAS organizational design. A comparison between MOBMAS and the sixteen other methodologies regarding their support for key AOSE steps and modeling concepts is presented in Tables 4 and 5. As revealed by Tables 4 and 5, none of the other methodologies addresses all key steps and modeling concepts of MAS development. Not only does MOBMAS endeavor to support all, it also combines the strengths of these methodologies, by reusing and enhancing their techniques and modeling definitions where appropriate.

MOBMAS also provides support for other important high-level methodological features such as explicit specification of the system development process, facilitation for verification, and provision of levels of abstractions in modeling. Not all of these features are supported by all other AOSE methodologies. A comparison of MOBMAS and the sixteen methodologies regarding these features is presented in Table 6.

9.2. Future work

While we have undertaken extensive validation of MOBMAS, more is desirable. It is intended to validate MOBMAS by using it to develop other MAS systems. We also recognize that MOBMAS may lack support in certain areas such as security, e.g., secure TROPOS [53,54]. We intend to extend MOBMAS to provide this support.

Currently, we do not have a tool to enforce the checking between the models. We are in the process of formalizing the current manual checking.

References


[70] G. Stumme, A. Maedche, Ontology merging for federated ontologies on the semantic web, Workshop on Ontologies and Information Sharing (ICAI’01), Seattle, USA, 2001.


[74] Q.N.N. Tran, MOBMAS: A Methodology for Ontology-based Multi-Agent Systems Development, School of Information Systems, Tech-
nology and Management, University of New South Wales, PhD, 2006.


