Towards a next-generation AOSE methodology

Hoa Khanh Dam[a], Michael Winikoff[b]

[a] *University of Wollongong, Australia.*
[b] *University of Otago, Dunedin, New Zealand.*

Abstract

Numerous methodologies for developing agent-based systems have been proposed in the literature. This proliferation creates a challenge to practitioners who need to select a methodology to adopt. This situation is analogous to that of object-oriented methodologies and notations pre-UML, and we argue that the time is ripe to begin the development of a next generation agent-oriented software engineering (AOSE) methodology, leading ultimately towards a unified AOSE methodology. This paper proposes process and models for a next generation AOSE methodology. Our proposal is based on a comparative analysis of seven prominent AOSE methodologies, which identified strengths, weaknesses, commonalities and differences.

Key words: Agent-Oriented Software Engineering, Methodology Standardisation.

Email addresses: hoa@uow.edu.au (Hoa Khanh Dam), michael.winikoff@otago.ac.nz (Michael Winikoff)

URL: http://www.uow.edu.au/∼hoa/ (Hoa Khanh Dam), http://www.winikoff.net (Michael Winikoff)

1. Introduction

Since the 1980s, agent technology has attracted an increasing amount of interest from the research and business communities, and the practical utility of agents has been demonstrated in a wide range of domains [1, 2].

However, a continuing issue in the development of agent-based systems concerns agent-oriented methodologies. It is generally accepted that analysis and design of agent-based systems requires an agent-oriented software engineering (AOSE) methodology (e.g. [3]), and although it was noted only 8 years ago that "*One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems*" [3, Page 4], the field has progressed considerably since then, and there are now many mature AOSE methodologies [4, 5] including MaSE [6] (and its successor O-MaSE [7]), Tropos [8], Gaia [9], Prometheus [10], INGENIAS [11, 12], ADEM[1] (and its modelling language AML[2] [13]), and PASSI[3] [15]. Indeed, the proliferation of methodologies has motivated work on comparisons of methodologies.

We argue that the time has come to work towards a unified AOSE methodology. There are two factors that lead us to this conclusion. Firstly, there is a problem. The proliferation of methodologies has created a challenge to practitioners in that they need to select a methodology from the large number of existing methodologies. Secondly, there is an opportunity. Whereas only a few years ago there were few AOSE methodologies that were mature and described in a high level of detail, there are now a number of methodologies that are described in sufficient detail to be useful to a practitioner; are mature, having been refined through extensive use; and have tool support. In other words, there is now a need to develop a unified AOSE methodology, and the field is sufficiently mature to enable a good unified methodology to be developed.

Before proceeding further, it is important to define what we mean by "methodology". Our position is pragmatic: a methodology comprises what a practitioner needs to support them in designing agent-based systems. Specifically, we view a

[1] http://www.whitestein.com/adem

[2] In the remainder of this paper we use "ADEM" to refer to the methodology, "AML" to refer to the modeling notation, and "ADEM/AML" to refer to their combination.

[3] There is also an update, PASSI2 [14] (http://www.pa.icar.cnr.it/passi/PassiDue/passiDueIndex.html), which clarifies certain aspects that were in the original version of PASSI, and adds an Agent Structure Exploration step.

methodology as including not just a **notation**, but also the underlying **concepts**, an indication of which **models** are to be developed (using the notation), an overall **process**, and detailed **guidelines** regarding steps of the process. Tool support is highly desirable but is not universally considered as an integral part of a methodology.

There has, in fact, been some initial work which has taken a number of prominent AOSE methodologies, and defined a proposed unified set of underlying *concepts*, and a *notation* [16]. We take this work a step further, focussing on the *models* and *process* of a next generation methodology, which is intended to be a step towards a unified methodology. In order to develop a well-founded proposal for the models and process of a next generation methodology we use a comparative survey of methodologies. This gives structure and rigour to the crucial step of understanding the relationship between various key methodologies, including their strengths, weaknesses, domain of applicability, and their commonalities and differences.

The aim of this paper is thus to move one step closer to a standard methodology by proposing a common process and set of models. We do not aim to propose a final standard methodology, but to propose a first draft, which we hope will lead to discussion in the relevant community, and that, ultimately, this discussion and collaboration might lead to a real unified AOSE methodology.

There is closely related work under the banner of method engineering [17]. Method engineering aims to provide a software engineer with means of constructing the right methodology for their context out of methodology fragments. Whilst there are contexts in which method engineering is valuable, the use of method engineering in practice is relatively low, and it has never been widely accepted or practised by software engineers due to its inherent complexity [18]. Furthermore, in practice the users of a project-specific method created from fragments that usually come from different methods may not be familiar with all the fragments and thus would require additional training. Therefore, method engineering is commonly viewed by practitioners as having a costly overhead. In contrast, we aim to develop a single unified methodology, which can be tailored by an organization to meet their specific project needs, as evidenced by successful work (e.g. [19]) in the method tailoring literature. The cost and complexity issues inherent in method engineering are eliminated since the base methodology is well-known and readily adopted by the development team.

The rest of this paper is organised as follows. Section 2 describes the approach that we take, including both a review of existing techniques for comparing software engineering methodologies, and a description of the criteria for selecting

methodologies, and their application. Section 3 then proposes a process and models for a next generation AOSE methodology, including discussion of the results of our comparion. We conclude and lay out some future directions for our work in section 4.

2. Approach

Since our aim is to provide a step towards a unified methodology, we have gathered together the work from various prominent AOSE methodologies. In doing so, it is important to understand the relationship between them, including each methodology's strengths, weaknesses, and domain of applicability. In addition, a crucial task is to identify the key commonalities (among all or most of the selected methodologies) and specific features that only a few (or one) methodology has but which are well motivated. These involve a systematic comparison of existing AOSE methodologies. Therefore, we have firstly reviewed the literature of evaluation methods to choose an appropriate comparison method for our study (section 2.1). We then design a set of criteria and carefully select a number of existing AOSE methodologies (for evaluation/comparison) that meet those criteria (section 2.2).

2.1. Comparing Methodologies

A well-known and widely-used approach to evaluate methodologies is the feature-based comparison which involves identifying a set of features and assessing a given methodology against such features [20, page 19]. Deriving a set of features is a difficult task since there is no universal agreement on a standard set of features. Assessing a methodology against a framework of attributes and features involves some judgement of how well it supports, or to what extent it has, a specific attribute or feature. There are a number of ways in which this can be done. These methods vary in their cost and reliability, and are classified in four major groups [21], ranging from the low-cost highly-subjective *screening mode* evaluation to more reliable and expensive *formal experiments*. In between, there are also the *case study* approach, which involves the actual use of the methodology, and the *survey* approach which is potentially cheaper. The feature-based approach offers several advantages: it does not require a measurement programme to be in place and it can be conducted with varying levels of detail. The limitations of the approach are primarily related to the inherent subjectivity of the assessment.

The feature-based approach has been the most commonly used approach in comparing AOSE methodologies (e.g. [22, 23, 24]), using a *screening mode*. She-

hory and Sturm [22] compared a number of AOSE methodologies using a set of criteria covering general software engineering issues and specific agent concepts. Similarly, Cernuzzi and Rossi [23] also compared two AOSE methodologies using a tree of features. More recently, Tran *et al.* [24] used a survey to attempt to ensure a complete set of features, and then used the feature set to evaluate ten methodologies. Bernon *et al.* [25] compared the meta-models of several existing AOSE methodologies (ADELFE, Gaia, INGENIAS, PASSI, RICA and Tropos) to find commonalities between them and used these to propose a unified meta-model for multi-agent systems.

Generally speaking, much of the work in comparing AOSE methodologies has low reliability and suffers from a number of problems. One problem is that there is subjectivity in selecting features and criteria. However, a more significant problem is that there is subjectivity in assessing a methodology against the criteria, since the assessment is typically done by the authors (who may also be the creators of one of the methodologies being assessed).

One exception to this is our earlier work [26] in which the assessment of methodologies was done using both a survey method (surveying the creators of each of the methodologies), as well as a case study method (using five students, each of whom designed a given system using a different methodology, and then reported on their experience).

Building on this work, in this paper we perform a *structural* comparison: rather than comparing AOSE methodologies based on their features (e.g. supporting agent concepts such as autonomy, mental attitudes, and pro-activeness or the clarity and understandability of the modelling language), we explore what models and processes the methodologies share, and what are the distinguishing aspects of each of them. This supports the proposal of a next generation methodology by identifying processes and models that are commonly used (i.e. the common intersection), and also by highlighting features that are only used in some methodologies, but which may be worthy of inclusion in the proposed next generation methodology.

2.2. Selecting Methodologies

Given the existence of many agent-oriented methodologies in the literature, we adopt a *multi-stage selection* approach [20], in which the methodologies are considered in multiple stages. The initial stage reduces the set of methodologies (similar to short listing job applicants) by using the following criteria:

Documentation: The selected methodology needs to be described in sufficient

detail. For example, it needs to have been presented in books, journal papers, or detailed technical reports rather than just a (single short) conference paper.

Maturity: The selected methodology needs to have seen significant use and refinement over time, including usage by people other than its creators and their colleagues/students. Specifically, we consider a methodology to have seen significant use if it has been used for non-toy examples, which demonstrates that it scales to real applications, and if it has been used by people other than its creators, which demonstrates that it is usable (e.g. well documented, not too complex). Note that there is clearly subjectivity in how to decide whether a given system is "non-toy". We use the heuristic that if the design fits into a conference paper, then the application is a "toy".

Tool support: Methodologies that have supporting tools are preferred over those that do not. Since part of our comparison process involved the practical use of each selected methodology to design a given system [26], the availability of tool support is a practical advantage. It is also a good indication of the maturity and of the development effort that has gone into a methodology. We also require that tool support is ongoing in order to ensure that tools remain usable and viable as environments change. This requirement excludes some older methodologies that are no longer under active development, such as MESSAGE.

Of the many AOSE methodologies in the literature, seven methodologies stand out as meeting these criteria (especially the maturity one): MaSE, Prometheus, Tropos, ADEM, Gaia, PASSI, and INGENIAS. Note that since MaSE has now been succeeded by O-MaSE, we use O-MaSE for our comparison, rather than the older MaSE. There are a number of methodologies that meet some of these criteria, but are weaker on the maturity criteria (e.g. ADELFE [27], ROADMAP [28]), and future work would include extending the analysis to include these methodologies.

3. Towards a Next Generation Methodology

Having selected seven agent-oriented methodologies, namely Gaia, Tropos, O-MaSE, ADEM, INGENIAS, PASSI and Prometheus, we have performed a structural analysis of their process and models. More specifically, we examined their similarities and distinguishing differences with respect to their process and

models. On that basis, in this section we propose a process and models for a next generation methodology, which aims to be a step towards a unified AOSE methodology.

To briefly summarise, we found that there were commonalities between the seven methodologies which were:

- the use of a goal model (an exception[4] was PASSI, which does not use goals);
- use case scenarios (except for Gaia and Tropos);
- forming agents by grouping smaller entities[5] ("roles" or "capabilities", or, in PASSI, use cases);
- using a static and dynamic model of the system, with the dynamic model typically being AUML interaction protocols[6]; and
- using dynamic and static models for individual agents.

We also found a number of unique features that were only present in some of the methodologies: early requirements (Tropos), an environmental model (INGENIAS, Prometheus, ADEM/AML, and to some extent MaSE), a deployment model (MaSE, ADEM/AML, and PASSI), using an ontology model to specify the concepts used in the design (O-MaSE, ADEM/AML, and PASSI), a mental diagram (AML) which captures the mental attitudes of entities (i.e. beliefs, goals, plans), modeling of services (AML), and using data coupling to guide the formation of agent types (Prometheus). Since these differences are generally well motivated, we include most of them in the proposed next generation methodology (See Figure 1). The differences that we do not include are those which are less general. For instance, the way in which AML models services and its mental diagram are both specific to certain types of systems, and, as we discuss in Section 3.5, these are valid options, but so are many other options. Therefore, we feel

[4]The original Gaia does not use goals explicitly, but its responsibility and safety conditions can be seen as goals. The revised Gaia discusses goals, but sees the identification of goals as being out of scope.

[5]It is not clear whether ADEM does this: the brief description of ADEM given in the AML book [13, Section 7.2] does not provide a clear process or guidance, since ADEM adopts a method engineering approach, and hence provides a framework rather than a methodology instance *per se*. The longer description of ADEM—a 2005 technical report—does not appear to be available.

[6]An exception is Gaia which is somewhat weak on capturing the dynamics of the system-to-be.

it is inappropriate to propose the use of any particular option, since none is clearly better than the others.

Figure 1 shows the models produced ("work products") depicted as rounded rectangles. It also shows two intermediate design artefacts: roles and agent acquaintance diagrams. The figure also shows the overall process followed: directed arrows indicate that a given model is developed based on a previously developed model. For example, the System Dynamics model is developed based on the Use Case Scenarios and on the Goal Model. Where arrows are bi-directional, this indicates that the models in question are developed hand-in-hand in an incremental and iterative manner. For example, the Goal Model and the Use Case Scenarios each assists in developing the other.

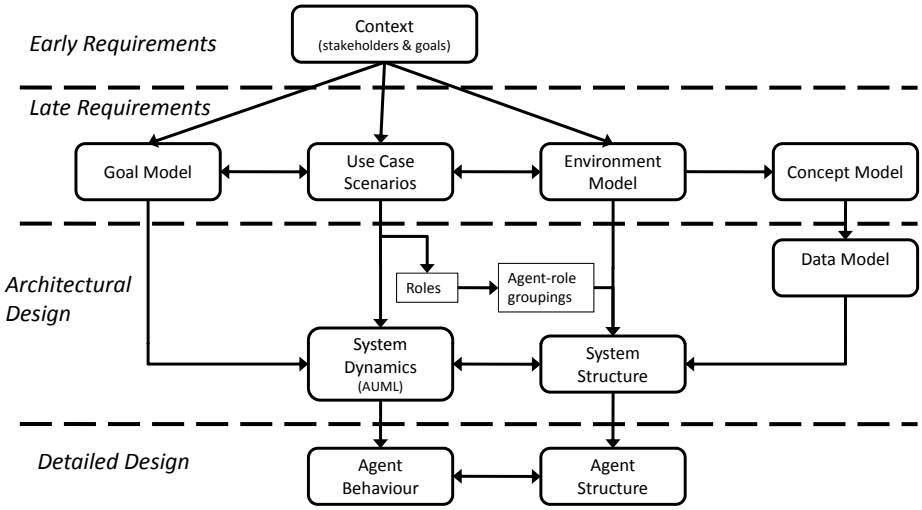

Figure 1: Overview of a Next Generation AOSE Methodology (omits implementation)

We note that in order for a methodology to scale to larger designs it is crucial to provide a range of mechanisms for dealing with complexity. These include the use of capabilities as a hierarchical structuring mechanism within agents (Prometheus), the use of "zooming" (Soda [29]), and the use of protocol nodes in the static system structure to abstract and group individual messages (Prometheus). Finally, note that although the following description follows a sequential presentation, the application of the methodology would normally be expected to be incremental and iterative.

3.1. A unified notation

As noted in Section 1, there has been earlier work on common concepts and an underlying notation [16], and many of the models that we propose would be expressed using the unified notation of Padgham *et al* [16]. The exceptions are the models relating to the dynamics, which would use AUML Interaction Protocols at the system level and some other notation (e.g. activity diagrams) at the agent level. Additionally, the concept model could use a UML class diagram, or a suitable ontology modeling notation (e.g. Protégé), and the data model could use an ER diagram or a UML class diagram.

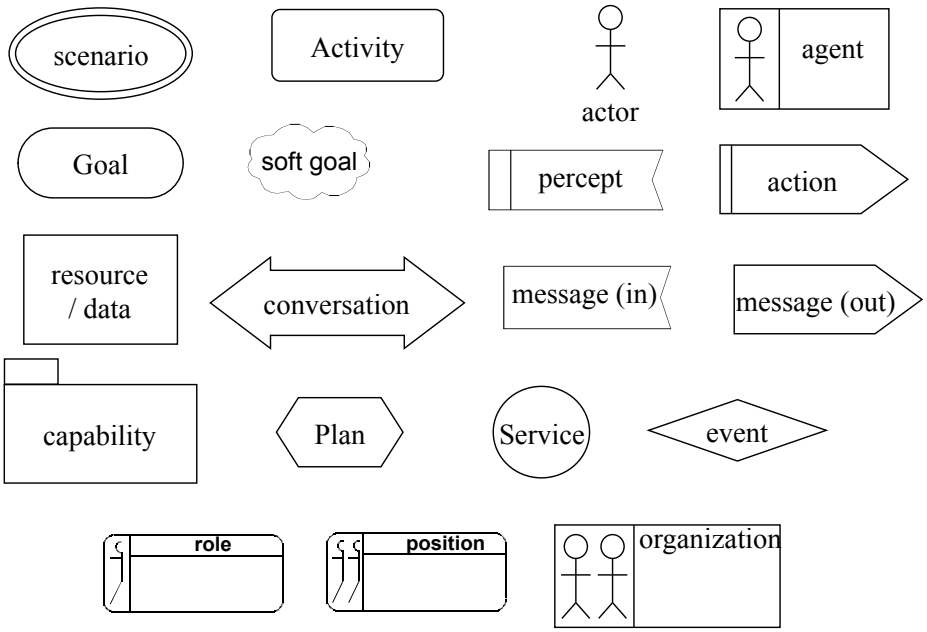

Figure 2: A unified notation (redrawn from [16])

The proposed unified notation uses a diagram type where different entities are depicted as nodes (with different concepts having different node shapes), and where relationships between entities are shown using directed edges. These edges can optionally be decorated with an annotation to clarify the relationship type (e.g. that one goal precedes or initiates another), but in many cases the relationship type is clear from the entity types involved. For example, an arrow from one goal to another indicates a sub-goal relationship. This "graph-based" notation

is standard in all types of engineering and is especially well suited to capturing system structure.

The notation was developed by considering firstly the different concepts that are used in various AOSE methodologies. For example, since agents are proactive the concept of "goal" is important to designing agents. Then, for each concept, careful consideration was given to selecting an appropriate symbol based on prior use, and to maintaining a clear distinction between different symbols [30]. For example, the symbols for goal, softgoal, actor and resources were adopted from Tropos (which in turn adopted them from *i**). On the other hand, the symbol for a plan (a hexagon) was selected because it is visually distinct from the other symbols used in the notation. Figure 2 shows the symbols proposed.

The proposed notation was used by Padgham *et al.* [16] to model a range of diagrams from the O-MaSE, Prometheus and PASSI methodologies. This demonstrated that the notation is able to cover the needs of a range of methodologies.

Another possible notation that could be considered is AML. It is worth considering because it was developed after the other methodologies, and was influenced by them. However, whilst AML is comprehensive, it is the work of a single group, rather than representative of a collaboration across groups, and it is quite complex. Its complexity is acknowledged by its authors who propose to define a simplified subset of the notation ("AML-lite" [13, Page 332]). However, it does not appear that any progress has been made in developing AML-lite.

3.2. Early Requirements

A unique feature of Tropos is its focus on early requirements, by adopting ideas from the requirements engineering community, specifically Eric Yu's *i** [31]. The Tropos early requirements phase involves examining the organisational setting including the domain stakeholders, their respective goals, intentions and their inter-dependencies. The next generation methodology should have an early requirements phase, which can be adopted from Tropos. The overall aim of the early requirements phase is to elicit and capture the organisational and social context in which the system-to-be will exist. This is done in terms of the stakeholders and their goals.

3.3. Late Requirements

The proposed late requirements phase develops four inter-related models. Initially, the Environment and Goal models are created based on the results of the

Early Requirements phase, but then they are refined iteratively, aided by considering desired system behaviour, expressed in terms of use case scenarios. The Concept model is developed based on the use case scenarios, and is then refined.

3.3.1. Goal model

Agents' goals are arguably one of the most important concepts of agency, and goals contribute to the pro-activeness of agents. Most of the methodologies (except for PASSI) agree on the importance of modelling goals: capturing goals is one of the first process steps in O-MaSE, is part of the system specification in Prometheus, and is a central modelling activity in Tropos. Identifying goals is also an important component in developing the Tasks/Goals viewpoint (one of the five viewpoints) in INGENIAS. Goals are in the form of roles' responsibilities in Gaia, which is more concrete than goals in the other methodologies. ADEM uses goals to model requirements in terms of a Business Goals Model and a System Goal-Based Requirements Model, both of which use the same Goal-Based Requirements Diagram provided by AML.

However, whilst the methodologies (mostly) agree on the importance of goals, they differ in the notation used, its richness, and the sort of information that is captured. The simplest is Prometheus, which uses a single goal type (not formally defined) and basic AND/OR decomposition. The goal model in O-MaSE captures not just AND/OR decomposition, but also sequential constraints among goals and events which trigger the creation of a goal. In Tropos, in addition to basic AND/OR decomposition, the goal decomposition process is also achieved by means-end analysis (plans, resources, and softgoals which provide the means for fulfilling a given goal) and contribution analysis (identifying goals that make either positive or negative contributions to other goals). Similarly, the Tasks/Goals viewpoint in INGENIAS describes not only the decomposition of goals but also details of tasks (to satisfy goals) such as resources used, inputs and outputs. AML provides a rich notation that is based on previous work (e.g. *i**, GRL, Tropos, KAOS, etc.). It provides both hard ("decidable") and soft ("undecidable") goals. Goals can have various properties, such as pre-conditions, invariants, and post-conditions. Different goal types (e.g. achieve, maintain) can be indicated using a naming convention.

The next generation methodology should support capturing goals in three steps: identifying goals, structuring them and representing them. It is also important to examine stakeholders' intentions and their dependencies via the tasks and resources they use to achieve goals (as done in the late requirements phase of Tropos). Goals can be structured and represented as a goal hierarchy diagram

similar to the one used in O-MaSE or Prometheus, using the proposed unified notation [16]. Tropos' notation is richer, and has features that are not supported by the notation of Padgham *et al.* [16]. Furthermore, there is a standard notation that has been proposed, the Goal-oriented Requirement Language (GRL) [32]. We therefore suggest that if the goal model being developed is a simple hierarchy of goals, then the notation of Padgham *et al.* would suffice, but for more complex goal models the GRL notation is eminently suitable, and there seems to be little reason to consider developing an alternative notation.

3.3.2. Use case scenarios

Use cases have been proven to be an effective technique in object-oriented methodologies in eliciting requirements. Use case scenarios (sometimes abbreviated to "scenarios") are a detailed description of one particular example sequence of events associated with a use case. Of the seven AOSE methodologies, Prometheus, INGENIAS and PASSI make use of this technique[7]. Specifically, INGENIAS and PASSI suggest the use of UML-like use case diagrams, whereas Prometheus provides use case scenarios. AML extends the RUP Use-Case Model with the ability to indicate responsibilities. However, although it is obviously possible to use a range of existing means (e.g. textual descriptors, interaction diagrams) to describe the details of use case scenarios, the description of ADEM does not clearly indicate whether providing a detailed description of a sequence of events is recommended by the methodology.

It is generally accepted that developing use case scenarios is also useful as a means for eliciting requirements, and ensuring that the goals captured are sufficient to support desired system behaviours. Furthermore, Henderson-Sellers [33] argues that use case scenarios are useful, and that it is important to use a "*textual description of each use case to facilitate understanding*" (page 306). We therefore argue that the unified methodology should support *both* techniques (e.g. use cases and use case scenarios) to help the analysts identify the key communications/interactions between entities.

Use cases can be captured using UML use case diagrams, whereas a range of notations could be used for describing use case scenarios, varying in the level of detail, all the way from free text, to the structured forms used by Prometheus which cover related information such as incoming triggers, goals, messages, and

[7] O-MaSE does not explicitly define a use case template, but uses UML-like sequence diagrams as a notation to express the communication paths between roles on the basis of the identified use cases.

events.

3.3.3. Environment model

Since an agent-based system is placed in an environment which it interacts with, situatedness is a key property of agents. Indeed, the 2003 Gaia paper notes that "*an explicit modeling of the environment is very important: not taking it into account (as, e.g., in the previous version of Gaia methodology [34]) may complicate the overall design . . .* " [9, Page 328].

However, we found that none of the selected methodologies addresses this aspect very well. The 2003 version of Gaia [9] models the environment simply, in terms of variables (or tuples) that the agents can read and write (and consume). Prometheus and INGENIAS seem to represent the environment better than most of the remaining methodologies. Prometheus' analysis overview diagram provides a relatively complete view of the environment in terms of actors, percepts and actions, whereas INGENIAS' environment viewpoint defines the entities (i.e. resources, other agents, and applications) with which an agent system interacts. O-MaSE has several improvements over MaSE in this regard by introducing actors (i.e. the environment) into the role model and having a domain model. Tropos only represents the resources (physical or information entity) as an entity whereas PASSI does not explicitly support a description of the environment. AML distinguishes between the system internal and external environment, the latter corresponding to the usual usage of the term "environment". AML models the external environment in terms of UML Actors, but provides additional features for modeling actors, such as mental models. The ADEM methodology description that is available does not provide much detail on how and where the (external) environment is modeled, but from Table 7-2 [13, Pages 112-116], one can see that aspects of the (external) environment may be captured in a number of places including the Actor Detail artifact (which uses the AML Entity Diagram to capture details of the actor), and the MAS Model artifact (which includes an Interaction Model artifact that uses interaction and/or service diagrams to capture interactions between entities and actors).

Developing an environment should be an important task in the process of the unified methodology. This task involves identifying actors and their interaction with the system in terms of percepts (inputs from the actor to the agent system) and actions (outputs from the system to actors). The unified methodology may adopt Prometheus' analysis overview diagram to capture this information. We further propose that the next generation methodology should deal explicitly with characteristics of the environment that may affect design decisions, such as

whether the environment is inaccessible, nondeterministic, dynamic, continuous, etc. Note that there has been a range of work on modelling the environment and the interface between agents and their environment (e.g. [35]).

3.3.4. Concept model

Whereas the environment model specifies how the agent system-to-be interacts with its environment, the *concept* model captures the concepts used by the agent system, that is, its ontology. Of the seven methodologies, O-MaSE, ADEM and PASSI both capture the ontology. O-MaSE has a domain model which captures the key entities. These are represented in terms of domain object types that agents interact with and reason about. The domain model also shows the relationships between those object types and between them and the agents. Object types are defined by a name and a set of attributes and are later used in other models such as the goal model, the policy model and the action model. Similarly, PASSI's Domain Ontology Description (DOD) is also represented as a UML class diagram which includes concepts (categories, entities of the domain), predicates (assertions on properties of concepts) and actions (that agents can perform in the domain). PASSI also has a Communication Ontology Diagram which shows communication paths between agent types, and for each communication, indicates the concepts being used to communicate. AML also uses (an extension of) UML to capture the ontology. Note that AML and PASSI's use of UML as a notation for capturing the ontology does not force the adoption of OO design principles.

In the requirement phase of the unified methodology, domain specific concepts, their attributes and relationships should be captured. We propose that the unified methodology uses a UML class diagram to describe the ontology of the domain, although other ontology modeling notations (e.g. Protégé) could also be used.

3.4. Architecture Design

The architectural design phase works towards defining the overall system's (static) structure, and its (dynamic) behaviour. These are captured using a system dynamics model (typically AUML interaction protocols), and a system structure model (which can be expressed using the proposed unified notation [16]).

In terms of process, one crucial step is deciding on the agent types. As discussed below (Section 3.4.2), this is often done by composing agents from smaller components, using considerations of coupling and cohesion. Once agent types have been determined, the system's structure can be specified using the agent

types and the system's interface with the environment as a starting point. In parallel, the system's dynamics are developed from the use case scenarios and goal model (Section 3.4.3). Additionally, the data model is developed from the concept model (Section 3.4.1).

3.4.1. Data model

The designers need to determine what information is needed by the system (including individual agents in the system). Based on domain concepts defined previously in the requirements phase, the designers then develop a data model.

The difference between the concept model and the data model is that the former captures the ontology, i.e. the concepts that are used when discussing the system. The latter specifies (in more detail) the data that the system itself will store. Not all concepts will correspond to data stored in the system. For example, in a transport scheduling system, the concepts might include different types of machines (e.g. trucks, cranes), but these may be mapped to agents, rather than to stored data. Conversely, some of the data stored in the system may not correspond to domain-level concepts.

In terms of process, one begins with the concepts, determines which of these will need to be stored, and specifies the details of what is to be stored. Determining which concepts correspond to stored data can be done with the assistance of the system interactions and the environment model: when an agent receives information that it needs to use later, the information will need to be stored.

In terms of notation, common notations such as ER diagrams or UML class diagrams can be used to represent the data model.

3.4.2. System structure

Since agents are the key entities in agent-based systems, one of the crucial requirements of AOSE methodologies is to assist the developers to identify the agents constituting the system. A common technique used in almost all of the methodologies to deal with agent identification is to start from entities that are smaller than agents and then group such entities to form agents. These entities are capabilities in Tropos and roles in Prometheus, Gaia, INGENIAS and O-MaSE. PASSI is different from the other methodologies in that agents are identified by grouping use cases. Roles in Tropos have a different meaning, which refers to an abstract characterization of the behavior of an actor within a specific context. As noted earlier, the description of ADEM does not provide sufficiently detailed guidance to allow us to state whether it uses this technique or not.

The unified methodology adopts such techniques from the existing methodologies. More specifically, the (static) structure of the system is developed by defining "chunks of functionality", which we term "roles", and then forming agent types in terms of groupings of roles. The choice of grouping roles or functionalities into one agent depends on several factors (e.g. coupling and cohesion) and more importantly needs to be supported by a method that guides designers in identifying issues. In the unified methodology, the designers are expected to develop two intermediate models (not shown in Figure 1): agent acquaintance and data coupling diagrams. The agent acquaintance diagram (adopted from O-MaSE, Gaia and Prometheus) depicts the communication paths between agents. These are directed graphs in which nodes represent agents, and directed edges represent communication links. The agent acquaintance diagram assists designers in identifying the coupling and communication bottlenecks among agents in the systems. In addition, the data coupling diagram (adopted from Prometheus) depicts agents, data, and read/write relationships. The data coupling diagram is used to assist in forming agents by considering coupling and cohesion; for example, putting agents that write or read the same data together.

Once agent types are identified, the structure of the system is developed and refined by considering the dependencies between agent types for using resources, performing tasks or achieving goals. The proposed unified notation is suitable for this model.

3.4.3. System dynamics

In addition to capturing the static structure of the system, it is also important to capture the high-level *dynamics* of the system, i.e. the interactions and communication taking place between agents. O-MaSE, AML, Prometheus, INGENIAS and Tropos describe interactions at two different levels of granularity. They include a set of high level interactions, and a more detailed representation in terms of interaction protocols. Gaia only provides interactions at the level of the former. O-MaSE, Prometheus, and Tropos are similar in that they model high level interactions using sequence/interaction diagrams borrowed from UML sequence diagrams. In addition, use cases are used to develop such sequence/interaction diagrams. Nonetheless, there are differences: interaction diagrams in Prometheus and Tropos show interactions between agents whereas sequence diagrams in O-MaSE (and also Gaia) depicts interactions between roles. When moving down to the detailed level of interaction protocols, AML, O-MaSE, Prometheus, INGENIAS, PASSI and Tropos suggest the use of AUML interaction protocol diagrams. AML extends the sequence diagram notation with the ability to depict

role changes and attribute changes. In PASSI, the agents's behavior is further described in terms of sequence of tasks and communications and is represented as a UML activity diagram.

For the unified methodology, the interactions of the system can be captured at a high level using AUML interaction protocols (which are used in O-MaSE, Prometheus, Tropos, PASSI, ADEM, and INGENIAS). Developing interaction protocols is done by generalising use case scenarios and considering the communication that is required to realise the desired system behaviours (as specified by the use case scenarios). Finally, it has been argued that goals should be included in the system dynamics model [36], but we see this as an area for investigation, rather than as a proposal that is sufficiently well developed to be ready for standardisation.

3.5. Detailed Design

The detailed design focuses on defining individual agents' structure and behaviour, by defining their components and the connections between them. Ultimately, the aim is to have models that are close to implementation concepts, in the same way that an object-oriented design ends up with classes and methods, which are the constructs provided by object-oriented programming languages.

As with the system level, we propose to define two models: one to capture the (static) internal structure of agents, and one to capture the (dynamic) behaviour of agents. It is interesting to note that O-MaSE does not have a model corresponding to the former, whilst Prometheus does not really have a model corresponding to the latter (as noted earlier, process diagrams are defined but are not well supported). In other words, not all existing methodologies provide support for capturing both structure and behaviour, and so we believe it is important to argue for including both types of model. On the other hand, some methodologies do provide support for both. For example, PASSI provides both a Single-Agent Structure Definition and a Single-Agent Behaviour Description. The former captures the static structure of an agent using a single class diagram which shows the agent and its tasks. The latter does not prescribe a specific notation[8], but suggests a number of possible notations that the designer could use to capture the dynamic behaviour of individual agents. Similarly, AML provides notations that can be used to capture both the dynamic behaviour and the static structure of entities. In fact, it provides

[8] "*Designers are free to describe them in the most appropriate way (for example, using flow charts, state diagrams, or semi-formal text descriptions).*" [15, Page 101].

a number of possible models including the Service Diagram, the Service Protocol Sequence Diagram, Behavior Decomposition Diagram, and Mental Diagram. AML uses UML activity diagrams to capture dynamic processes, including the behaviour of participants in protocols, and the internals of plans.

INGENIAS' Agent Viewpoint describes the functionality of an agent in terms of the goals the agent pursues, the tasks it has to execute, and the roles it plays. In addition, the Agent Viewpoint depicts a mental state of an agent which consists of mental entities such as goals, facts, and beliefs. There is also a mental state processor that determines which task the agent should execute, and a mental state manager provides the mechanisms for creating, deleting, and modifying mental state entities and their relationships. However, it is not clear from INGENIAS documentation (e.g. [11]) how the mental state manager and processor are expressed.

In Tropos, each agent's plan is described using a UML activity diagram. O-MaSE also describes this micro-level of dynamics using finite state diagrams (Agent State Model and Plan Model). Prometheus defines process diagrams, but these are not described in detail or supported by the Prometheus Design Tool, and do not appear to be commonly used.

While the original Gaia intentionally excludes detailed design from its scope, the 2003 version defines an agent model and a service model. The service model used in the 2003 version of Gaia provides a representation of the inputs, outputs, pre-conditions, and post-conditions of each service (i.e. capability) provided by an agent. However, techniques for describing agent's planning or scheduling capabilities are not described.

The details of the models and the process clearly depend on the type of implementation platform being targeted. For example, if a BDI-style platform is the ultimate target for implementation, as is the case for Tropos and Prometheus, then the detailed design will be expressed in terms of plans, events, and beliefs.

Regarding the process, defining agent structure involves identifying capabilities (modules within an agent) and their relationships. Roles defined in the earlier stage can be helpful in defining capabilities. In addition, for a BDI-style implementation, plans, internal events and internal data within an agent and/or its capabilities need to be identified. For non-BDI implementations, the detailed design will not use plans, but will typically use some means of describing the processing that an agent carries out in response to a stimulus (such as a message). For example, in MaSE this is done using a finite state machine, and the behaviour described by the finite state machine is then mapped to code in the agent [6]. Defining agent behaviour involves modelling states and actions within the agent.

Interaction protocols defined in the earlier stage should be used as the basis to define agent behaviour.

Regarding the notation used for the models, the static structure of agents can be captured using the notation proposed by Padgham *et al.* [16]. The behaviour of agents can be captured using a range of possible existing notations including activity diagrams, state charts, finite state machines, and Petri nets (indeed, most of these notations have been used by some AOSE methodology). Other options, that are less widely used, include AML's mental diagrams, and its models for describing behaviours. However, of the various options for capturing the behaviour of agents, none is clearly better than the others. We therefore do not select a given notation, but note the options, and leave it to future discussion and standardisation efforts to select an appropriate notation (or notations).

3.6. Implementation

The results of the detailed design process are models that define the structure and behaviour of the agents in the system, in terms of appropriate concepts. For example, if the methodology targets BDI-like implementation platforms, then the detailed design will be expressed in terms of events and plans. The detailed design can be translated to code skeletons in an appropriate language.

All the methodologies (except for Gaia[9]) have supporting tools that can generate code skeletons from design models. This is an important feature of modern software engineering methodologies. Going further, the next generation methodology could support the emerging model-driven development paradigm [38] in terms of generating the entire code from design models. In fact, INGENIAS and its tool support (IDK) provide extensive support for model-driven development. However, in order to enable complete executable code to be generated from design models, the design models need to contain significant additional detail, which may result in design models becoming cluttered and less comprehensible, and requiring more work to create.

One aspect of implementation is the *deployment* of the system, and three of the methodologies, ADEM/AML, O-MaSE and PASSI, provide specific support for deployment, using a *deployment model* which captures how agent instances map to the hardware platforms used. For example, the deployment model can specify that a certain agent instance resides on one particular machine, but that a different agent instance is on a different machine. PASSI uses the UML deployment

[9]Gaia does not cover the implementation phase, and consequently neither does the supporting tool, Gaia4e [37]

diagram notation, but extends it with features to allow the *mobility* of agents to be specified. O-MaSE also provides some limited support for modelling mobility in terms of specifying a move activity in a concurrent task state. Similarly, AML provides a Deployment Diagram which allows the mapping of agent instances to execution environments to be specified. AML provides the ability to show both the movement of agents, and the cloning of agents.

Additionally, during implementation, testing and debugging methods are essential, which should relate to the concepts used in the analysis and design. There has been work on testing and debugging of agent systems in recent years (e.g. [39, 40, 41, 42]), but integration with methodologies and their tools still needs improvement. Another area that is not well supported by existing methodologies and their tools is software evolution [43].

4. Conclusion

As agent-oriented approaches represent an emerging paradigm in software engineering, there has been a strong demand to apply the agent paradigm in large and complex industrial applications and across different domains. In doing so, the availability of agent-oriented methodologies that support software engineers in developing agent-based systems is very important. In recent years, there has been a large number of methodologies developed for agent-oriented software engineering, which makes the selection of a methodology a challenge for practitioners. In addition, a number of methodologies are now mature and described in sufficient details to be useful for a practitioner. For all those reasons, we argue that it is useful to begin gathering together the work of various existing agent-oriented methodologies with the aim of developing a future unified methodology that is complete and fully supports the industrial needs for agent-based system development.

Thus, we have carried out a comparison of seven prominent agent-oriented methodologies in order to understand the relationship between them. In particular, our main purposes were: (a) assessing each methodology's strengths, weaknesses, and domain of applicability, and (b) identifying the similarities and differences among them in terms of processes and models that are necessary or useful in guiding the developing of agent-bases systems. Following this comparative analysis, we proposed a next generation methodology.

One aspect where our work has been (intentionally) limited is that we have only considered the sorts of simple agent structures that are provided by widely-used methodologies such as Tropos, MaSE, Prometheus, etc. The approach used

by these methodologies to capture the structure of agent systems is sufficient for a range of application types [44], but is limited in a number of respects. For example, the structure of a multi-agent system is considered to be static, rather than varying over time; and the nature of the relationships between agents (e.g. supervisor) is not captured. There has been a range of work that considers richer societal models (e.g. [45, 46, 47, 48, 49, 50]) and one area for future work is to consider these richer models. AML's notation is a good start in this direction, since it allows one to capture roles, how these are played by agents, and the relationships of roles in an organization unit.

Another area for future work involves expanding the number of selected methodologies and using the current framework to compare them. By doing so, we may improve the proposed next generation methodology by including supplementary models and techniques. In addition, this paper has considered the process and models but has not considered the detailed techniques which are a crucial topic for future work. Space has also precluded us from presenting a detailed example. Finally, our ultimate aim is the development of a unified methodology based on the proposal in this paper, and perform further empirical studies to investigate how such a unified methodology would be useful in practice.

References

[1] S. Munroe, T. Miller, R. A. Belecheanu, M. Pěchouček, P. McBurney, M. Luck, Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents, Knowledge Engineering Review 21 (4) (2006) 345–392. doi:10.1017/S0269888906001020.

[2] M. Pěchouček, V. Mařík, Industrial deployment of multi-agent technologies: review and selected case studies, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 17 (2008) 397–431. doi:10.1007/s10458-008-9050-0.

[3] M. Luck, P. McBurney, C. Preist, Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing), AgentLink, 2003, ISBN 0854 327886.

[4] B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, 2005.

[5] F. Bergenti, M.-P. Gleizes, F. Zambonelli (Eds.), Methodologies and Software Engineering for Agent Systems, Kluwer Academic Publishing (New York), 2004.

[6] S. A. DeLoach, M. F. Wood, C. H. Sparkman, Multiagent systems engineering, International Journal of Software Engineering and Knowledge Engineering 11 (3) (2001) 231–258.

[7] S. A. DeLoach, J. C. Garcia-Ojeda, O-MaSE: A customizable approach to developing multiagent development processes, International Journal of Agent-Oriented Software Engineering 4 (2010) 244–280. doi:10.1504/IJAOSE.2010.036984.

[8] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, Tropos: An agent-oriented software development methodology, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 8 (2004) 203–236.

[9] F. Zambonelli, N. R. Jennings, M. Wooldridge, Developing multiagent systems: The Gaia methodology, ACM Transactions on Software Engineering and Methodology 12 (3) (2003) 317–370. doi:10.1145/958961.958963.

[10] L. Padgham, M. Winikoff, Developing intelligent agent systems: A practical guide, John Wiley & Sons, Chichester, 2004, ISBN 0-470-86120-7.

[11] J. Pavon, J. J. Gomez-Sanz, R. Fuentes, The INGENIAS methodology and tools, in: B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, 2005, Ch. IX, pp. 236–276.

[12] J. Pavón, J. Gómez-Sanz, Agent oriented software engineering with INGENIAS, in: V. Marík, M. Pechoucek, J. Müller (Eds.), Multi-Agent Systems and Applications III, Vol. 2691 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 394–403.

[13] R. Cervenka, I. Trencansky, AML The Agent Modeling Language: A Comprehensive Approach to Modeling Multi-Agent Systems, Birkhäuser, 2007, ISBN 978-3-7643-8395-4.

[14] M. Cossentino, V. Seidita, PASSI2 - going towards maturity of the PASSI process, Technical Report RT-ICAR-PA-09-02 (December 2009).

[15] M. Cossentino, From requirements to code with the PASSI methodology, in: B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Inc., 2005, Ch. IV, pp. 79–106.

[16] L. Padgham, M. Winikoff, S. DeLoach, M. Cossentino, A unified graphical notation for AOSE, in: M. Luck, J. J. Gomez-Sanz (Eds.), Proceedings of the Ninth International Workshop on Agent Oriented Software Engineering, Estoril, Portugal, 2008, pp. 61–72.

[17] B. Henderson-Sellers, J. Ralyté, Situational method engineering: State-of-the-art review, Journal of Universal Computer Science 16 (3) (2010) 424–478.

[18] B. Fitzgerald, N. L. Russo, T. O'Kane, Software development method tailoring at Motorola, Commun. ACM 46 (2003) 64–70.

[19] M. Bajec, D. Vavpotič, M. Krisper, Practice-driven approach for creating project-specific software development methods, Inf. Softw. Technol. 49 (2007) 345–365. doi:10.1016/j.infsof.2006.05.007.

[20] D. Law, Methods for Comparing Methods: Techniques in Software Development, NCC Publications, 1988.

[21] B. Kitchenham, DESMET: a method for evaluating software engineering methods and tools, Technical Report TR96-09, University of Keele, U.K. (August 1996).

[22] O. Shehory, A. Sturm, Evaluation of modeling techniques for agent-based systems, in: J. P. Müller, E. Andre, S. Sen, C. Frasson (Eds.), Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press, 2001, pp. 624–631.

[23] L. Cernuzzi, G. Rossi, On the evaluation of agent oriented modeling methods, in: Proceedings of Agent Oriented Methodology Workshop, Seattle, 2002.

[24] Q.-N. N. Tran, G. C. Low, Comparison of ten agent-oriented methodologies, in: B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, 2005, Ch. XII, pp. 341–367.

[25] C. Bernon, M. Cossentino, J. Pavón, Agent-oriented software engineering, Knowledge Engineering Review 20 (2) (2006) 99–116.

[26] K. H. Dam, M. Winikoff, Comparing agent-oriented methodologies., in: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems (AOIS), Vol. 3030 of Lecture Notes in Computer Science, Springer, 2004, pp. 78–93.

[27] G. Picard, M.-P. Gleizes, The ADELFE methodology: Designing adaptive cooperative multi-agent systems, in: Bergenti et al. [5], Ch. 8, pp. 157–175.

[28] L. S. Sterling, K. Taveter, The Art of Agent Oriented Modeling, MIT Press, 2009.

[29] A. Molesini, E. Denti, A. Omicini, Agent-based conference management: a case study in SODA, International Journal of Agent-Oriented Software Engineering 4 (1) (2010) 1–31.

[30] J. Rumbaugh, Notation notes: Principles for choosing notation, Journal of Object-Oriented Programming (JOOP) 8 (10) (1996) 11–14.

[31] E. Yu, Modelling strategic relationships for process reengineering, PhD thesis, University of Toronto, Department of Computer Science (1995).

[32] L. Liu, E. Yu, Designing information systems in social context: a goal and scenario modelling approach, Information Systems 29 (2) (2004) 187 – 203, the 14th International Conference on Advanced Information Systems Engineering (CAiSE*02). doi:10.1016/S0306-4379(03)00052-8.

[33] B. Henderson-Sellers, Consolidating diagram types from several agent-oriented methodologies, in: Proceeding of the 2010 conference on New Trends in Software Methodologies, Tools and Techniques (SoMeT), IOS Press, Amsterdam, The Netherlands, The Netherlands, 2010, pp. 293–345. URL http://portal.acm.org/citation.cfm?id=1860875.1860901

[34] M. Wooldridge, N. Jennings, D. Kinny, The Gaia methodology for agent-oriented analysis and design, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 3 (3).

[35] D. Weyns, A. Omicini, J. Odell, Environment as a first-class abstraction in multiagent systems, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 14 (1) (2007) 5–30. doi:10.1007/s10458-006-0012-0.

[36] J. Khallouf, M. Winikoff, Goal-oriented design of agent systems: A refinement of prometheus and its evaluation, International Journal of Agent-Oriented Software Engineering 3 (1) (2009) 88–112.

[37] L. Cernuzzi, F. Zambonelli, Gaia4E: A Tool Supporting the Design of MAS using Gaia, in: J. Cordeiro, J. Filipe (Eds.), ICEIS 2009 - Proceedings of the 11th International Conference on Enterprise Information Systems, 2009, pp. 82–88.

[38] D. C. Schmidt, Guest editor's introduction: Model-driven engineering, Computer 39 (2) (2006) 25–31. doi:http://dx.doi.org/10.1109/MC.2006.58.

[39] L. Padgham, M. Winikoff, D. Poutakidis, Adding debugging support to the Prometheus methodology, Engineering Applications of Artificial Intelligence, special issue on Agent-oriented Software Development, 18 (2) (2005) 173–190.

[40] J. J. Gomez-Sanz, J. Botía, E. Serrano, J. Pavón, Testing and debugging of MAS interactions with INGENIAS, in: J. J. Gomez-Sanz, M. Luck (Eds.), Ninth International Workshop on Agent-Oriented Software Engineering (AOSE), 2008, pp. 133–144.

[41] C. D. Nguyen, A. Perini, P. Tonella, Experimental evaluation of ontology-based test generation for multi-agent systems, in: J. J. Gomez-Sanz, M. Luck (Eds.), Ninth International Workshop on Agent-Oriented Software Engineering (AOSE), 2008, pp. 165–176.

[42] Z. Zhang, J. Thangarajah, L. Padgham, Automated unit testing for agent systems, in: Second International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2007, pp. 10–18.

[43] H. K. Dam, M. Winikoff, An agent-oriented approach to change propagation in software maintenance, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) (2011). doi:10.1007/s10458-010-9163-0.

[44] V. Dignum, F. Dignum, Designing agent systems: state of the practice, International Journal of Agent-Oriented Software Engineering 4 (3) (2010) 224–243.

[45] B. Horling, V. Lesser, A Survey of Multi-Agent Organizational Paradigms, The Knowledge Engineering Review 19 (4) (2005) 281–316.

[46] V. Dignum (Ed.), Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, Hershey, New York, 2009, ISBN 978-1-60566-256-5.

[47] V. Dignum, A model for organizational interaction: Based on agents, founded in logic, PhD thesis, Utrecht University (2004).

[48] V. Dignum, F. Dignum, The knowledge market: Agent mediated knowledge sharing, in: Proceedings of the Third International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 03), 2003, pp. 168–179.

[49] J. L. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez-Aguilar, C. Sierra, Engineering open environments with electronic institutions, Eng. Appl. of AI 18 (2) (2005) 191–204.

[50] J. Hübner, J. Sichman, O. Boissier, Developing organised multiagent systems using the $\mathcal{M}$OISE$^+$ model: programming issues at the system and agent levels, International Journal of Agent-Oriented Software Engineering 1 (3/4) (2007) 370–395.