A Unified Graphical Notation for AOSE*

Lin Padgham[1], Michael Winikoff[1], Scott DeLoach[2], and Massimo Cossentino[3]

[1] RMIT University, Australia
{lin.padgham,michael.winikoff}@rmit.edu.au
[2] Kansas State University, USA
sdeloach@ksu.edu
[3] ICAR-CNR, Italy
cossentino@pa.icar.cnr.it

Abstract. Over the last five years a number of agent system development methodologies have been proposed and developed, with a number of them becoming well established and used beyond the group developing them. They all deal with similar concepts, but the notations used differ substantially. In this work we develop a standardized graphical notation for four prominent agent development methodologies, using principles of graphical notation suggested by Rumbaugh. We briefly illustrate the graphical design views produced in the different methodologies, on a conference management system example, using the standardized notation. We then discuss some of the similarities and differences on the basis of the design artifacts produced - which are now much more readily comparable than previously. This is a first step in being able to readily incorporate steps from different methodologies, depending on the needs of the application. It also helps to make the material more readily accessible to a wider audience.

1 Introduction

In recent years, it has become accepted that in order to effectively develop agent systems, it is necessary to have methodologies and notations that deal specifically with agent concepts and agent design issues. As a result, over the last several years many *Agent Oriented Software Engineering* (AOSE) methodologies have been developed or proposed, with some of the most well known including Gaia [1], O-MaSE (based on the earlier MaSE) [2], Tropos [3], Prometheus [4] and PASSI [5].

Important aspects of mature methodologies include the particular tools and diagrams that are used to develop and capture the analysis and design of the system being developed [6]. While there are a number of similarities between different methodologies cited above, each has its own particular strengths and nuances. It is certainly conceivable that a developer would wish to incorporate aspects of different methodologies into

* We acknowledge the input of Paolo Giorgini in discussing and determining the notation presented, and AgentLink which organized the Technical Forum. Padgham and Winikoff acknowledge the support of The Australian Research Council and the Australian Department of Education, Science and Training, as well as Agent Oriented Software under grants LP0453486 and CG040014. Scott DeLoach acknowledges the support of the US National Science Foundation under Grant No. 0347545 and by the US Air Force Office of Scientific Research.

a development process. In fact, this is the vision of method engineering [7] where the goal is to mix and match the activities, tasks and techniques of various methodologies according to the needs of a particular project. However, it is currently difficult to compare or use the diagrams and techniques from different AOSE methodologies because each methodology uses its own concepts, notations, and techniques.

The vision of method engineering is generally achievable using mature technologies such as object orientation where the basic concepts (objects, classes, associations, inheritance, etc.) and notations (UML) are well understood and generally agreed upon [8]. As a result of this maturity and agreement in the object oriented community, there are several well known activities, tasks and techniques that can be applied in a number of ways on various projects. There are also several commercially available tools that can be used together or separately to support a variety of approaches to developing object oriented systems [9]. Duplicating the success of object orientation requires two key elements: a common notation and a common metamodel

The goal of this paper is to take a first step toward the level of maturity evidenced in the object-oriented community. In this first step, the developers of a number of the most detailed and prominent AOSE methodologies have worked together to produce a common notation.[4] While this first step is modest, we believe that a shared graphical notation is a first step toward making AOSE methodological work applicable to industry consumers. This notation will be used in each of our future individual methodological work and will be integrated into existing and new tools supporting that work.

In progress toward the second key element, there has been work done attempting to define a common metamodel for multi-agent methods and techniques [11]. Some efforts have also been spent in the field of standardization within the FIPA organization. Two different technical committees (Modeling and Methodology) worked on that and results describing their points of view can be found in [12, 13]. However, while basic agent-oriented concepts have some commonality, we are far from having community-wide consensus on the majority of agent and multi-agent concepts. Thus, the common metamodels tend to be overly complex and of limited practical usefulness. Even though our goal may be considered more limited, from a practical standpoint, it is at least equally important and can provide a good first step in reaching such a community-wide consensus on the most important agent concepts.

While this paper does identify common concepts that we all use or wish to include in our notational set, the precise definitions and ways they are used do differ somewhat from methodology to methodology. Until there is a community wide agreement on these concepts, we believe these differences should be allowed to exist.

In the rest of this paper we present the new notation, motivating the choices we have made, followed by an example of a conference management system where we illustrate the design diagrams that can be produced by the various methodologies using the new notation. We finish with a brief discussion of the importance of working together across research groups to provide an engineering methodology that is accessible to practitioners wishing to build complex agent systems.

[4] We also worked with Paolo Giorgini, considering the Tropos methodology in the development of the common notation. We did not use Gaia because it does not make use of graphical models. We also did not include less prominent AOSE methodologies such as [10] for the time being.

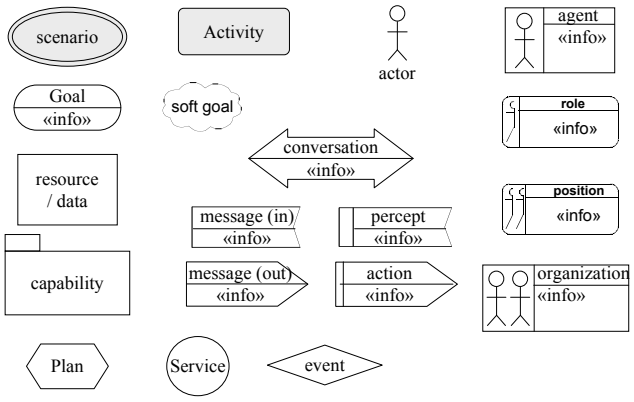


Fig. 1. Proposed Notation. The shaded symbols (use case and activity) as well as the actor symbol are existing UML symbols.

2 The Unified Graphical Notation

We begin this section by describing general criteria for developing (graphical) notations suitable for the analysis and design of complex software systems. The article by Rumbaugh [14], one of the developers of the widely-used UML notation, gives the following list of desiderata for developing notations. This list illustrates the trade-offs that must be made when different desired properties conflict.

1. Clear mapping of concepts to symbols
2. No overloading of symbols
3. Uniform mapping of concepts to symbols
4. Easy to draw by hand
5. Looks good when printed
6. Must fax and copy well using monochrome images
7. Consistent with past practice
8. Self consistent
9. Distinctions not too subtle
10. Users can remember it
11. Common cases appear simple
12. Suppressible details.

In the remainder of this section, we present the notation that we have developed and explain the rationale for our decisions. As is often the case, there are sometimes trade-offs, but we believe we have now developed a notation that satisfies desirable properties for usability, clarity, etc.

Our notation uses a common type of diagram where the entities of interest are depicted as nodes and distinctive shapes are used to differentiate different types of nodes. Figure 1 presents an overview of our proposed notation. Relationships between entities are depicted by links, which can be decorated with a label giving the link type (e.g.

"≪precedes≫", "≪initiates≫"). The decorations are optional and in many cases can be derived from the types of the entities. For example, in Prometheus an arrow from a percept to an agent is always a ≪receives≫ relationship.

This "graph-based" notation is standard in all types of engineering and is especially well suited to capturing system structure. However, capturing system *behavior* may be best done with non-graph-based models such as AUML sequence diagrams [15]. In this paper we do not tackle this type of diagram: since the AUML sequence diagram is well-defined and widely used, it makes little sense to propose a replacement for it. Other diagrams capturing system behavior, such as the Prometheus process diagrams, can be drawn with the proposed new notation.

Below we explain each type of node in our proposed notation. For each node we explain our reason for choosing the depiction given in Figure 1, and relate it to the concepts it can be used to represent. However, before describing the graphical notation, we briefly motivate our choice of concepts.

In selecting the concepts to be represented in our notation, we chose concepts that were required to model agent-based system as indicated by their use in the four methodologies participating in the discussion, as well as other agent based methodologies of which we were aware. In identifying "required" concepts, we related the concepts used to design and build agent systems to the *defining properties* of agents [16][5]:

- Agents are *autonomous* – the key concept here is the notion of an *agent* itself, as an autonomous entity (distinct from objects).
- Agents are *situated* – the minimal key design concepts are the interface to the agent system's environment, in terms of *actions* performed by agents that affect the environment, and *percepts*[6], that get information from the environment. Clearly, more sophisticated concepts can be used to characterize the environment.
- Agents are *proactive* – the corresponding concept is *goals*.
- Agents are *reactive* – the corresponding concept is the notion of an *event*, a "significant occurrence".
- Agents are *social* – here a wide range of concepts could be used, ranging from the minimal one of messages, through to a range of organizational models. We choose to use the concepts of *messages*, *conversations*[7], *roles*, *positions* and *organizations*, where positions are placeholders for one or more roles within an organization, and an organization can include particular forms of organization, such as a team, or an e-institution.

In addition to these clearly required concepts, we added the following commonly used concepts:

- *Soft-goals*: goals that do not have a clear satisfiability definition, used in a number of methodologies, both agent-oriented and non-agent-oriented, for modelling non-functional requirements such as security, usability, flexibility. We include soft-goals since they are clearly useful, and since they fit in very well with agent-based design, where agents have goals.

[5] Sturm *et al.* [10] proposed a similar set of concepts, based on our earlier work [16].

[6] From the Latin *perceptum*, same root as the word "perceive".

[7] Also known as "protocols" or as "interaction protocols".

- *Actor*: an external entity, which can be human or software. This concept is useful in early analysis, and is well established in existing practice.
- *Capability*: a concept often used in discussing agents and implemented first by the JACK agent-oriented programming language [17] and subsequently adopted and extended by Jadex. Capabilities are a modularization construct for agents which can contain things such as plans, events, data, and sub-capabilities.
- *Plan*: sometimes termed tasks, plans are a key concept in BDI agent platforms, and in other plan-based implementation platforms. Hence, it is clearly important for (detailed) design to support plans.
- *Resource/data*: like any other software, agents normally need to store data in some form and/or use existing resources. For notation purposes we a use a single symbol to depict data or resources, without distinguishing between resources (e.g. a printer) and data, or between different data formats (objects, belief sets, relational databases).
- *Service*: the use of services are becoming very popular in information systems design using what are called service-based multi-agent systems. Although services are currently only well-defined in PASSI, we believe that this is a growing area and thus it is important to be able to depict existing services that will be used.

Having identified the concepts used to define agent systems, we now turn to considering how to graphically depict these concepts (see Figure 1) in order to more easily model agent system designs. According to the desiderata identified by Rumbaugh [14], each of the key concepts should be mapped to a distinct symbol satisfying the first two criteria ("Clear mapping of concepts to symbols" and "No overloading of symbols"). In addition, we strived to select symbols that emphasize similarities between related concepts (e.g. between outgoing messages and actions) whilst using clearly distinct symbols for concepts that are clearly dissimilar ("Uniform mapping of concepts to symbols"). For example, the symbols for a plan and for an agent are completely different. The symbols selected are also easily drawn by hand; our notation does not rely on shading, line thickness, or any other distinctions that are subtle, confusing, or that do not copy/fax well. The only distinction between symbol shapes that is somewhat subtle, the use of rounded corners in roles and positions, is reinforced by the use of a modification to the stick figure within the symbol. Further, as is discussed below, we have strived for consistency with past practice, where appropriate. In particular, we have used the UML notation where it made sense to do so. However, as will be seen in the following sections, many of the concepts used to engineer agent systems do not exist in UML, and in this case we believe that it is important to have new and clearly distinct symbols for concepts that are new and clearly distinct. Finally we describe a notational mechanism for achieving scalability by suppressing details.

Goal and Softgoal: Perhaps because goals are a new concept in agents, and one of the differences that clearly distinguish agents from objects, there is no consensus on how to depict them graphically. For instance, O-MaSE depicts goals as a rectangle with a number and a name, Tropos uses a fully rounded box (a "pill" or "lozenge" shape), and Prometheus uses an oval. Since one of our aims is to be compatible with

existing standards, and since GRL[8] appears to be in the process of being standardized[9] we choose to use a pill/lozenge shape for goals.[10] For softgoals the standard is to use a cloud shape. Although this is not always easy to draw using tools, we cannot justify inventing a new symbol when a widely used symbol already exists for the concept.

Scenario: Scenarios are closely related to use cases, and hence we want a symbol that is close to the existing UML symbol for a use case (an oval). However, we also want to avoid overloading symbols, thus we have elected to use a double-lined oval.

Entities: Actors, Agents, Roles: Actors are a well-established concept with a well-established notation (the stick figure) which we adopt. For agents it is important to have a symbol that is distinct from the UML class symbol. However, despite the importance of the agent concept to AOSE, there is no consensus on its depiction: Prometheus uses a rectangle containing a stick figure, whereas Tropos uses a circle. For our notation, we propose that the Prometheus notation of the stick figure in a rectangle be adopted. Including the stick figure suggests a relationship with actors, and reinforces that, like humans, agents are active autonomous entities. Roles are an abstraction of agents and, in fact, are used in two ways within the AOSE community: as a social notion and as a component. In the *social* approach, agents are assigned play roles within some organization. In the *component* approach, which has been used in both O-MaSE and Prometheus, agents are designed by grouping roles. To help define the role symbol, we adopted a general notational principle that states that when there are two concepts and one is an abstraction of the other, we use the same symbol for the abstracted concept, but with rounded corners. Thus, our proposed symbol for a role is the same as an agent but with rounded corners. To further emphasize that roles are abstract and are not complete agents, we embed a "half stick figure" instead of the full actor symbol used in the agent symbol.

Intra-Agent: Plan/Task, Capability/Module: Plans (sometimes called tasks, e.g. in MESSAGE and Tropos) are depicted by a range of symbols. Using a similar reasoning to goals, we adopt the GRL/Tropos/*i** symbol: a hexagon. For capabilities we adopt UML's package symbol, since capabilities are conceptually package-like: they contain other entities.

Events and Messages: In Prometheus events and messages are both depicted as envelopes, which are memorable, easy to recognize, and easy to draw by hand. However this notation has three problems: firstly it is confusing to draw intra-agent events as messages, secondly it is not clear from the symbol whether the message is incoming or outgoing, and thirdly, the envelope symbol is not consistent with related UML notation. Thus, we propose to adopt the UML notation for *send signal actions* and *accept event actions* to depict sending and receiving messages respectively. This notation allows us to distinguish incoming and outgoing messages and provides consistency with standard practice. For events, we use a new symbol (a diamond). Events can be used to represent either inter- or intra-agent events.

---

[8] `http://www.cs.toronto.edu/km/GRL/`

[9] By the international telecommunications union (ITU). The proposed standardization brings together Use Case Maps (UCMs) and the Goal-Oriented Requirement Language (GRL) under the name User Requirements Notation (URN).

[10] GRL and Tropos use the same notation, due to their common ancestry, *i**.

Environment: Percepts, Actions, Resource: Because we assume agents are situated in their environment, it can be argued that sending and receiving messages are actually special instances of the general notions of performing actions on the environment and receiving percepts from the environment. Therefore, we choose to use symbols for actions and percepts that are variants of the message symbols described above. The addition of a vertical bar as the distinguishing characteristic is somewhat arbitrary, but was chosen to be an obvious difference that is easy to draw. Resources are represented as simple rectangles in Tropos. Due to its simplicity and clarity, it makes sense to use this notation in the general sense as well. In addition, the rectangle symbol is similar to UML classes, which are also rectangular in shape. Therefore, the resource symbol can be seen as a generalization of the UML class.

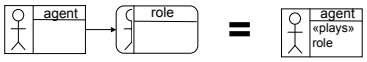
Social concepts: Conversation, Organization, Position: Due to its mnemonic value, the Prometheus symbol for a protocol (a large double headed arrow, denoting bi-directional communication), is proposed for our notation. However, to clarify the concept it represents, we term the concept a "conversation" rather than a "protocol". Because an organization is generally associated with a group of agents, it seemed natural to modify the agent symbol to represent this grouping. Thus, the agent symbol (a box with a single stick figure) is modified by replacing the single stick figure with multiple stick figures to represent an organization. When we decided on the symbol for a position, which is an organization's place holder for a role (one or more), we used the organisation symbol, modified in the same way as the role symbol was modified from the agent symbol: a round-cornered rectangle with half stick figures.

Service: Since there is no accepted symbol for a service we propose a new symbol. In addition, since the concept of a service is not closely related to any of the other agent concepts discussed so far, we wanted a distinct symbol that is simple to draw. Thus, we propose using a simple circle to represent services. While the choice is somewhat arbitrary, it can be argued that a service is similar to a UML *interface* as it describes how to interact with the agent providing the service.

Links: Although some notations, such as Tropos and *i**, use a wide range of different link/arrow types, we do not believe this to be a good idea because they can be hard to draw by hand, are quite subtly different, and can be difficult for users to remember. Instead, we propose a single arrow type which is (optionally) enriched with textual annotations where desired to indicate different link types. We have identified a variety of useful links; however, this (partial) list can be easily extended as long as the meaning of the link is defined. Annotations include: a role ≪achieves≫ a goal, an event ≪occurs≫ during the pursuit of a goal, an event ≪triggers≫ the creation of a new goal, and a goal ≪precedes≫ another goal.

Scalability: Collapsing Links: For a design notation to be practical and usable for the design of large systems it must *scale* to large designs. There are a number of abstraction and packaging concepts in the notation that support this (such as organisation, protocol and capability). In addition we propose the use of "collapsible" links. As is shown below, many of the symbols have an information section, which can be used to indicate links with other entities such as an agent that ≪plays≫ a role. These collapsable links can be used to replace links to symbols. For example, below the left part of the figure, showing an agent with a link to a role symbol, is equivalent to the right side of the figure

where the role symbol has been removed and the relationship indicated in the agent's information section.



The design notation can also be used to develop models which capture different aspects of the system, partitioning or abstracting to obtain scalability. For example a system overview diagram shows no agent internals, whereas agent overview diagrams can partition the system into a set of separate diagrams, one for each agent type.

3 Using the Notation

To illustrate the use of the unified notation across the different methodologies, we present examples of various diagrams taken from our methodologies based on a common exemplar system. Space limitations preclude us from presenting a wide range of diagrams from each methodology and we hope that the diagrams included are sufficient to give some of the flavour of how the proposed notation would be used.

The example we use in this paper is the popular multiagent conference management system, which was first proposed by [18] in 1998. It has since been widely used as it is suitable for illustrating a wide variety of aspects of multi-agent system analysis and design. The version of the system we are following is based on the version used in [19].

The conference management system is a multiagent system that supports the management of conferences that require the coordination of several individuals and groups to handle the paper selection process. This process includes paper submission, paper reviews, paper selection, author notification, final paper collection, and the printing of the proceedings. Authors may submit papers to the system up until the submission deadline. Once the submission deadline has passed, members of the program committee (PC) review the papers by either contacting referees and asking them to review a number of the papers, or by reviewing them themselves. Once all the reviews are complete, a final decision is made on whether to accept or reject each paper. Each author is notified of this decision and authors with accepted papers are asked to produce a final version that must be submitted to the system. All final copies are collected and sent to the printer for publication in the conference proceedings.

In the remainder of this section, we present several of the models used in our methodologies to capture various aspects of the conference management system analysis and design. However, each of the models uses the unified notation to illustrate how the different models might possibly be used together even though they are from different methodologies.

The Prometheus Goal Overview Diagram, as shown in Figure 2, shows how the overall goal of the system is refined into a hierarchical goal tree where subgoals define how their parent goal may be achieved. In this case, the overall goal *manage_conference* is refined into four subgoals: *get_papers*, *review*, *select_papers*, and *print_proceedings*. Each of these goals is further refined into subgoals providing more insight into how the goals will be achieved.

The O-MaSE Goal Model shown in Figure 3 is similar to the Prometheus Goal Overview Diagram in function; however, it provides a richer set of constructs with which to model the goal structure. As in the Prometheus model, the O-MaSE Goal Model has a top level goal of *Manage Conference Submissions*, which is broken down into five conjunctive sub-goals: *Get Papers*, *Assign Papers*, *Review Paper*, *Select Papers*, and *Print Proceedings*. The "precedes" relation between the *Collect Papers* and *Distribute Papers* goals indicates that the *Collect Papers* goal must be achieved before work may begin towards the achievement of *Distribute Papers*. The "occurs" and "triggers" relation between the *Partition Papers* and *Assign Reviewers* goals and the *created(set)* event indicates that the *created(set)* event may occur during achievement of the *Partition Papers* goal and when it does, it triggers the creation of a new *Assign Reviewers* goal that is parameterized based on some *set* of papers to be assigned to reviewers. As can be seen from Figures 2 and 3, the Prometheus Goal Overview Diagram provides a simpler and clearer model of system goals while the O-MaSE model provides additional constructs that provide a more detailed definition of system operation. Clearly, each model has situations where its use is warranted and the ability to choose between these models could be of great benefit to system designers.

The O-MaSE role model is derived from the goal model and depicts the relationships between the roles in the conference management system, as shown in Figure 4. In Figure 4, the goal(s) that each role may achieve are annotated via an embedded ≪achieves≫ relation in the body of each role symbol. Thus, the *Assigner* role is used to achieve the *Assigns Reviewers* goal. We also use a directed arrow to represent a conversation between roles with the arrows pointing from the initiator to the responder. The details of these conversations are defined using the commonly accepted AUML interaction diagrams [15]. Interactions with the external environment are represented as conversations with external actors.

The O-MaSE Agent Model (Figure 5) also shows assignment of roles to agents (via the ≪plays≫ embedded relation) but also shows the initiation and participation in specific conversations. (An alternate *implicit conversation* notation is shown in Figure 6). In both cases, the conversations between the agent types provide an overview of the entire system architecture.

The Prometheus System Overview Diagram (Figure 7) captures the architecture of the system, showing agent types, the conversations between them, and the interface to the environment in the form of percepts and actions. The System Overview diagram is generated automatically by the design tool (though layout must be done manually), based on the protocol specifications, and on the role specifications which form the agent.

The PASSI Multi-Agent Structure diagram (Figure 8) captures similar concepts as the O-MaSE Agent Model and the Prometheus System Overview. When following PASSI, the diagram contains no new information and is usually generated automatically by the PASSI ToolKit. Unique to the PASSI diagram is the use of ≪task≫ and ≪knowledge≫ keywords in the agent notation, which clearly highlights the extensibility of our current notation. While the O-MaSE Agent Model uses the ≪plays≫ keyword to denote the roles an agent may play, the PASSI approach is focused more on capturing the knowledge required by the agent (≪knowledge≫) and the tasks performed by the agents (≪task≫). Again, the commonality of the notation would allow

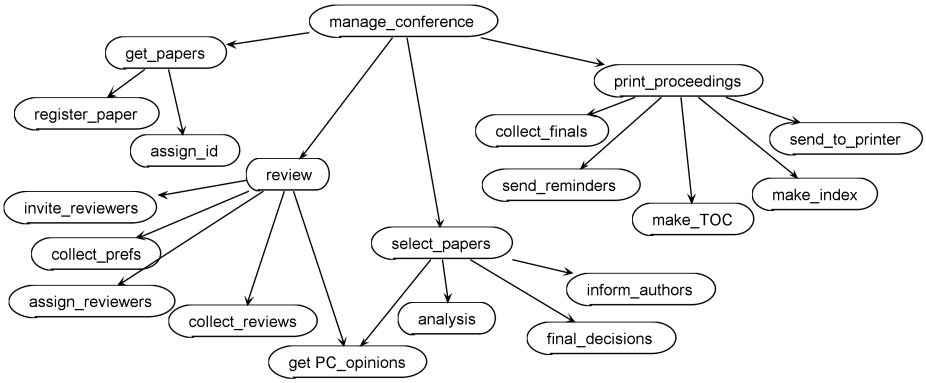


Fig. 2. Prometheus Goal Overview Diagram

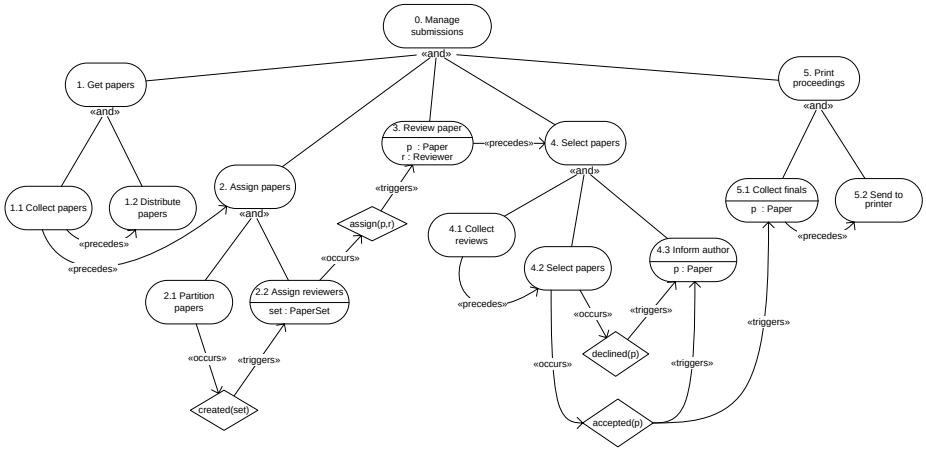


Fig. 3. O-MaSE Goal Model

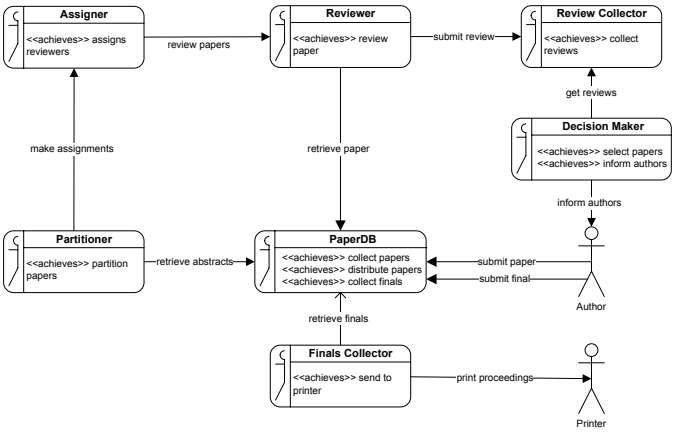


Fig. 4. O-MaSE Role Model

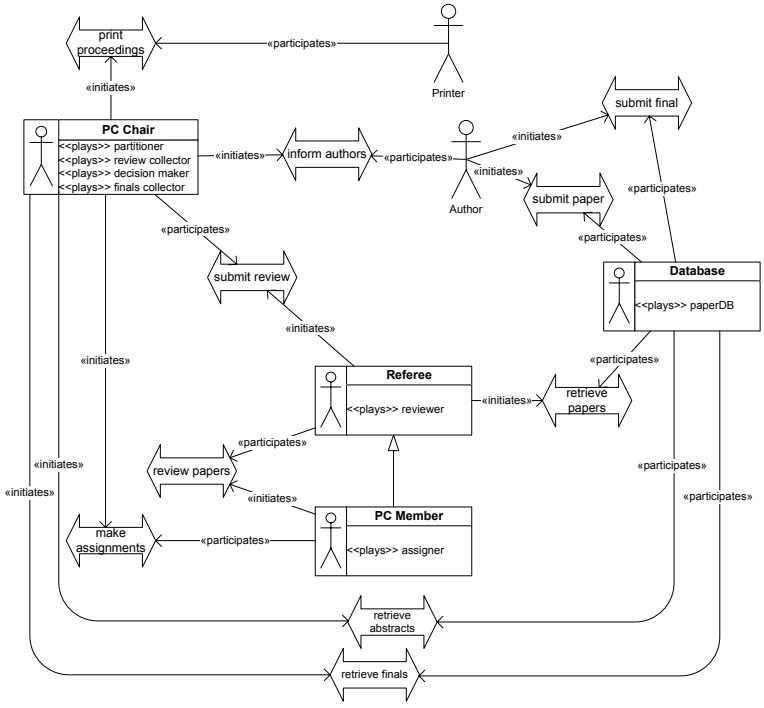


Fig. 5. O-MaSE Agent Model

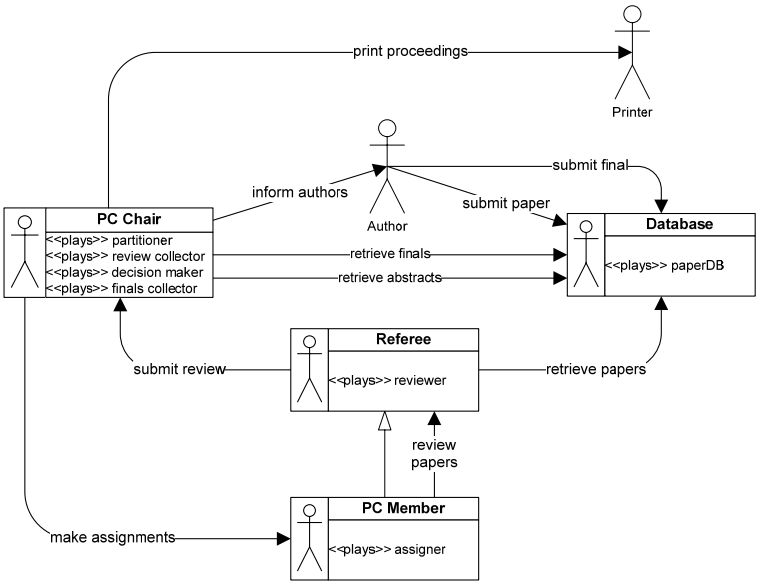


Fig. 6. O-MaSE Agent Model with Implicit Conversations

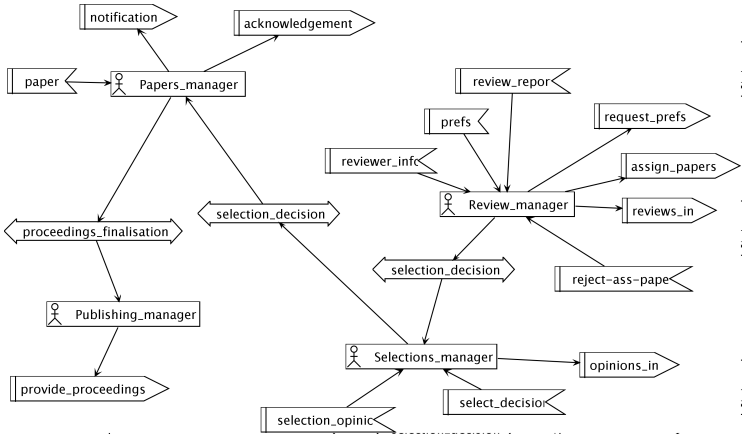


Fig. 7. Prometheus System Overview Diagram

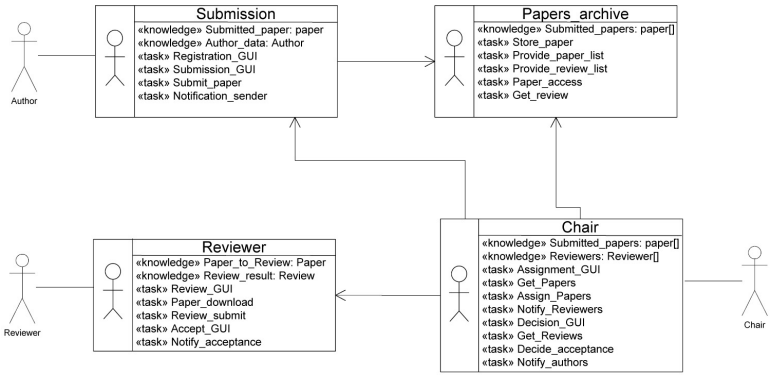


Fig. 8. PASSI Agent Structure Diagram

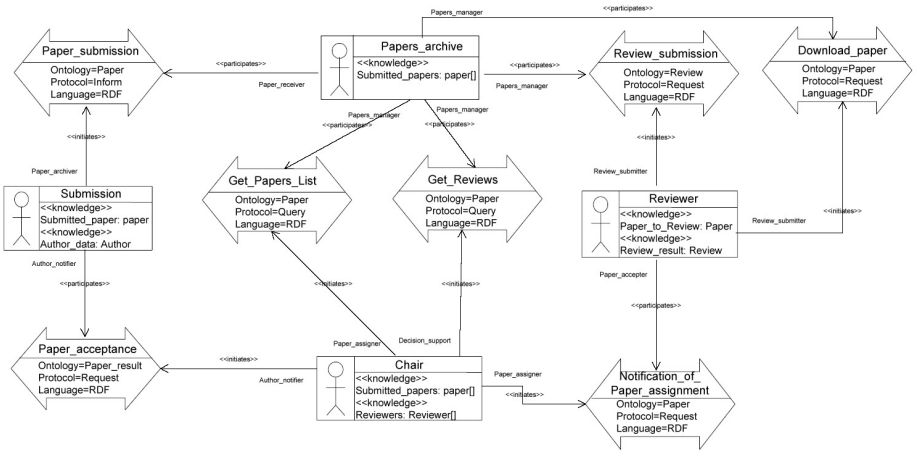


Fig. 9. PASSI Communication Ontology Description Diagram

the designer to use the most useful aspects of the various methods and diagram types to express the system design.

The PASSI Communication Ontology Description diagram (Figure 9) is essentially composed of communications and agents. For each communication, the designer can introduce three parameters: the ontological elements exchanged in message contents (represented by the Ontology parameter), the agent interaction protocol (represented by the Protocol parameter), and the content language (represented by the Language parameter).

The Prometheus System Overview, the O-MaSE Agent Model and the PASSI Agent Structure and Communication Ontology Description diagrams all show conversations between agent types. The difference lies in how they represent interaction with the environment. The Prometheus models show an explicit representation of individual ac-

tions/percepts while the O-MaSE and PASSI models represent interactions via conversations with external actors.

4 Discussion and Future Work

We can see that once the gratuitous incompatibility of notation is removed, it becomes much easier to see both the similarities and the differences, and to consider extending one methodology with aspects of another. It is clear from the example and associated diagrams that O-MaSE and Prometheus are quite close, at least at the level of system specification and architectural design, whereas PASSI is more dissimilar:

- Both O-MaSE and Prometheus capture goals in a goal overview diagram. The Prometheus notation is simpler whereas the O-MaSE notation captures additional relationships, such as one goal triggering another.
- O-MaSE, Prometheus and PASSI all have a diagram that captures the roles in the system and in the case of Prometheus and O-MaSE these both indicate the assignment of goals to roles.
- The System Overview Diagram of Prometheus and the Agent Model of O-MaSE are virtually identical apart from O-MaSE showing actors, whereas Prometheus shows actions and percepts. PASSI on the other hand has a simpler Agent Structure diagram with a separate diagram for the communication ontology.

Although there is still some way to go before portions of the methodologies would be fully interchangeable, the unified notation does allow us to more readily see possibilities for borrowing from each other. Most importantly, the unified notation has the potential to allow users and developers to more readily understand the various methodologies and associated diagrams, as they do not need to learn a new 'language' for each approach.

In order to move towards this new unified notation, the authors are committed to using this notation, and to moving our respective CASE tools towards using this notation. Indeed, there already is a version of the Prometheus Design Tool that uses the new notation, and this was used to generate Figure 7.

In future work we hope to specify XML representations for certain diagrams, that will facilitate sufficient mapping between underlying models to allow some sharing of tools. We also hope that further collaboration and exploration can lead to further integration of our approaches to the benefit of industry developers wishing to use these technologies.

References

1. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. ACM Transactions on Software Engineering and Methodology **12**(3) (July 2003) 317–370
2. DeLoach, S.A.: Engineering organization-based multiagent systems. In Garcia, A.F., Choren, R., de Lucena, C.J.P., Giorgini, P., Holvoet, T., Romanovsky, A.B., eds.: SELMAS. Volume 3914 of Lecture Notes in Computer Science., Springer (2005) 109–125

3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems **8** (May 2004) 203–236
4. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons, Chichester, UK (2004) ISBN 0-470-86120-7.
5. Cossentino, M.: From requirements to code with the PASSI methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Inc. (2005) 79–106
6. Sturm, A., Shehory, O.: A framework for evaluating agent-oriented methodologies. In Giorgini, P., Winikoff, M., eds.: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, Melbourne, Australia (July 2003) 60–67
7. Henderson-Sellers, B.: Method engineering for OO systems development. Commun. ACM **46**(10) (2003) 73–78
8. Bernon, C., Cossentino, M., Gleizes, M., Turci, P., Zambonelli, F.: A study of some multi-agent metamodels. In: Agent Oriented Software Engineering (AOSE'04). (2004)
9. Object Management Group: UML Resource Page. `http://www.uml.org/` (2006)
10. Sturm, A., Dori, D., Shehory, O.: Single-model method for specifying multi-agent systems. In: The Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS). (2003) 121–128
11. Bernon, C., Cossentino, M., Pavón, J.: Agent-oriented software engineering. Knowl. Eng. Rev. **20**(2) (2005) 99–116
12. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. In: Agent-Oriented Software Engineering Workshop, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (2005)
13. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardization to research. International Journal on Agent Oriented Software Engineering **1**(1) (2007)
14. Rumbaugh, J.: Notation notes: Principles for choosing notation. Journal of Object Oriented Programming **9**(2) (May 1996) 11–14
15. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Proceedings of the Fifth International Workshop on Agent Oriented Software Engineering (AOSE). (July 2004)
16. Winikoff, M., Padgham, L., Harland, J.: Simplifying the development of intelligent agents. In: AI2001: Advances in Artificial Intelligence. 14th Australian Joint Conference on Artificial Intelligence, Springer, LNAI 2256 (December 2001) 555–568
17. Busetta, P., Howden, N., Rönnquist, R., Hodgson, A.: Structuring BDI agents in functional clusters. In: Agent Theories, Architectures, and Languages (ATAL-99), Springer-Verlag (2000) 277–289 LNCS 1757.
18. Ciancarini, P., Niestrasz, O., Tolksdorf, R.: A case study in coordination: Conference Management on the Internet. `ftp://cs.unibo.it/pub/cianca/coordina.ps.gz` (1998)
19. DeLoach, S.: Modeling organizational rules in the multi-agent systems engineering methodology. In: AI '02: Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence, London, UK, Springer-Verlag (2002) 1–15