Developing a Multiagent Conference Management System Using the O-MaSE Process Framework

Scott A. DeLoach

Department of Computing and Information Sciences, Kansas State University
234 Nichols Hall, Manhattan, KS 66506
sdeloach@cis.ksu.edu

Abstract. This paper describes how the Organization-based Multiagent Systems Engineering (O-MaSE) methodology can be applied to an exemplar multiagent system, the Conference Management System. First, a custom process for the CMS application is created using the O-MaSE Process Framework. Then, each task identified in the O-MaSE compliant process is performed and the appropriate models are generated. For the CMS system, we begin by creating a Goal Model via the Model Goals and Goal Refinement tasks. Once the Goal Model is complete, we create an Organization Model to capture all the interfaces to external actors and systems. After that, a Role Model is created to capture the functionality and the logical architecture of the system. Next, based on the Role Model, an Agent Class Model is created. The details of the agents and protocols identified in the Agent Class Model are further refined into several Protocol Models and Agent Plan Models.

1 Introduction

The purpose of this paper is to describe how the Organization-based Multiagent Systems Engineering (O-MaSE) methodology [4] can be applied to a particular example multiagent system, the Conference Management System, which is described in Section 2.

Because O-MaSE is really a framework for creating custom multiagent systems development processes and not a single process, the first step is to define an appropriate O-MaSE compliant process. The O-MaSE compliant process described in this paper is presented in Section 3. Once our custom process is defined, we present each step of the process in Section 4. Finally, we finish in Section 5 with our conclusions and future work.

2 Conference Management System

The Conference Management System (CMS) example used in this paper was originally defined in [8]. The CMS is an open multiagent system supporting the management of various sized conferences requiring the coordination of several

individuals and groups. The process begins when paper authors submit their papers. After the deadline for submissions has passed, the program committee (PC) reviews the papers by either contacting referees and asking them to review a number of the papers, or reviewing them themselves. After the reviews are complete, a decision on whether to accept or reject each paper is made. After the decisions are made, authors are notified of the decisions and, if accepted, are asked to produce a final version of their paper. Once the final copies are collected, they are sent to the printer for inclusion in the conference proceedings. The conference management system consists of an organization of agents whose membership represents humans (authors, reviewers, decision makers, review collectors, etc.).

3 O-MaSE Process

The first step in using O-MaSE to define a CMS system is the creation of an appropriate custom process for the CMS application using the O-MaSE Process Framework [4]. As a detailed discussion of the task selection criteria is beyond the scope of this paper, it should be pointed out that the CMS system is made up of agents representing specific humans playing specific roles in the organization and thus, there is no requirement for an autonomously adaptive system. Thus, the definition of individual capabilities of the roles and agents are not required.

The process defined for designing the CMS system is shown in Figure 1. Assuming there exists some kind of system requirements or system definition, we begin by creating a goal tree and refining it into a Goal Model for Dynamic Systems (GMoDS) via the Model Goals and Goal Refinement tasks. Once the Goal Model is complete, we create an Organization Model to capture all the interfaces to external actors and systems. Once the Organization Model is complete, it and the GMoDS Goal Model are used to create the initial Role Model. Based on the Role Model, an Agent Class Model is created. The details of the agents and protocols identified in the Agent Class Model are further refined into several Protocol Models and Agent Plan Models. While we chose to define the Protocol Models based on the Agent Class Model, we could have also defined the protocols after creating the Organization Model or the Role Model, as each of those identifies protocols as well.

4 Modeling CMS in O-MaSE

4.1 Goal Model

The first step in our O-MaSE compliant process is to create an initial Goal Model that captures the essential requirements of the CMS system as defined in the system definition or requirements documents. The initial Goal Model for the CMS system is shown in Figure 2. The Model Goals task uses traditional AND/OR refinement to decompose the top-level CMS goal, `Manage Submissions`, into six AND-refined subgoals: `Get papers`, `Assign papers`, `Review papers`, `Select`

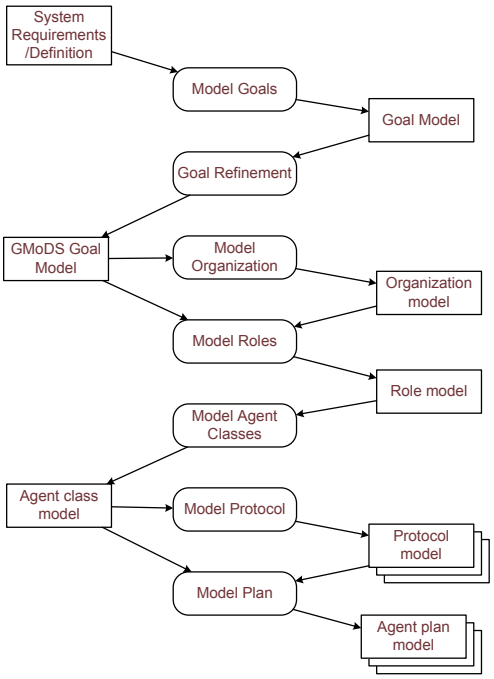

Fig. 1. CMS O-MaSE Compliant Process

`papers`, `Inform authors`, and `Print proceedings`. The UML aggregation notation is used to represent AND-refinement while the UML generalization notation is used to represent OR-refinement. Each goal in the model is annotated by the keyword ≪goal≫. All the subgoals except `Review papers` are further decomposed into subgoals that define what must be accomplished in order to achieve the given goal. For instance, the `Select papers` goal is AND-refined into a `Collect reviews` goal and a `Make decision` goal. Notice that the `Inform authors` goal is OR-refined into an `Inform declined` and `Inform accepted` subgoals. Obviously, the subgoal used to satisfy the `Inform authors` goal is based on the decision made whether to accept or reject the paper.

The Goal Refinement task takes the initial Goal Model and adds additional information to capture the dynamism associated with the CMS system. Specifically, we refine our initial model into a model based on the Goal Model for Dynamic Systems (GMoDS) [6]. GMoDS introduces three concepts into AND/OR goal modeling approaches to handle goal sequencing, the creation of goal instances, and parameterized goals. Sequencing of goals is provided by goal precedence, which specifies that one goal must be achieved before a second goal can be achieved. Goal instances are created based on events that occur during system operation; these events are termed goal *triggers*. Goals without a specific trigger are created at system initialization, while other goals are created when specific events occur. Finally, goals can be parameterized to fully define what

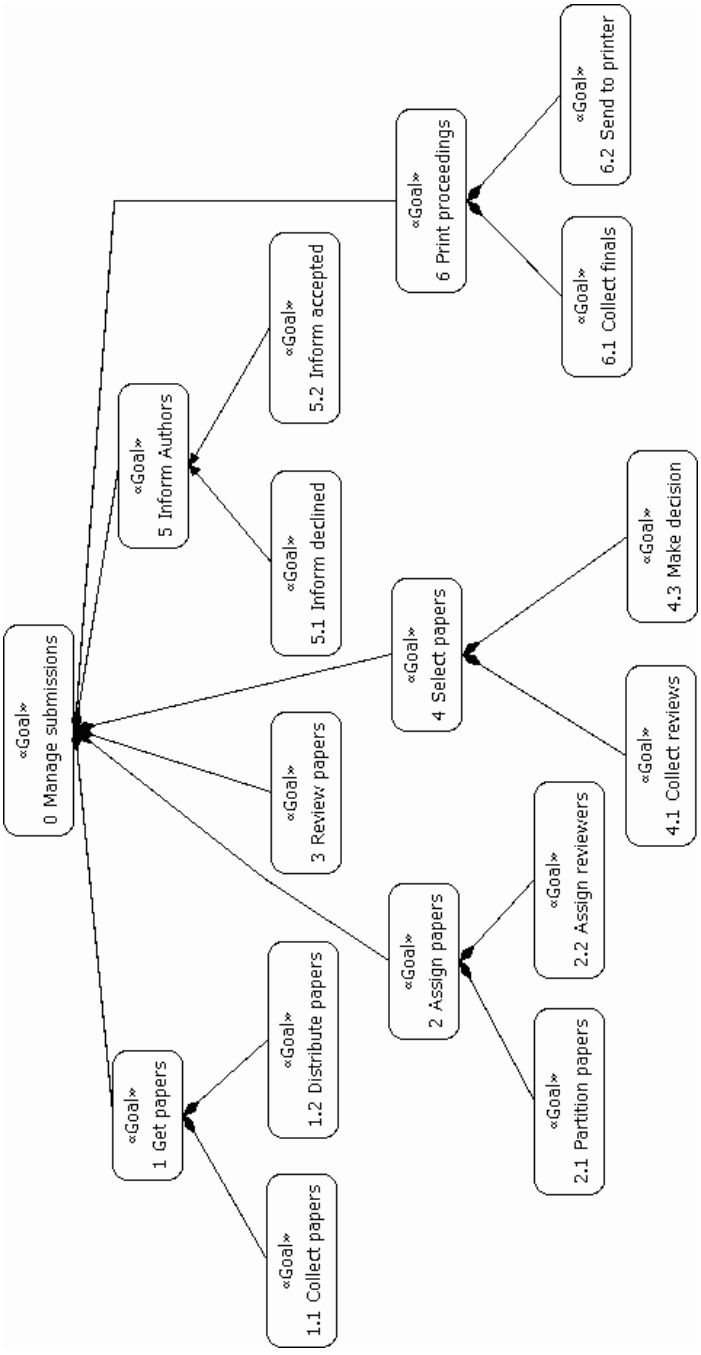

Fig. 2. CMS AND/OR Goal Model

the purpose of the goal is. For instance, in the CMS system, we have a goal to `Review papers`. However, this goal is ambiguous until we specify which set of papers to be reviewed. Thus, we add a parameter to the goal to specify the papers to be reviewed.

The GMoDS Goal Model for the CMS system is shown in Figure 3. The GMoDS model has the same basic shape as shown in Figure 2, but with additional arrows between goals showing precedence and goal triggering as well parameters for several goals. In the Figure 2, precedence between goals is shown by an arrow labeled with the ≪precedes≫ keyword while triggers are represented by arrows between goals with an event name and a set of parameters in the form *event*($p_1, ...p_n$). Reading Figure 3 we can see that the `Collect papers` goal precedes both the `Distribute papers` and `Assign papers` goals. Thus, once the `Collect papers` goal is achieved, the papers may be distributed and the `Partition papers` goal (a sub-goal of `Assign papers`) can begin. The trigger between `Partition papers` and `Assign reviewers` denotes that each time a set of papers is *created* during the pursuit of the `Partition papers` goal, a new goal is instantiated for that set. As each `Assign reviewers` goal is instantiated, pursuit of it can begin. As each assignment is made, the *assign(p,r)* trigger creates a new goal to `Review papers` for each paper set and reviewer assigned.

When all the `Review papers` goals have been achieved, the `Select papers` goal can be pursued via its subgoals: `Collect reviews` and `Make decision`. When the `Collect reviews` goal is achieved, then the `Make decision` goal can be pursued. As a decision is made on each paper, a *declined(p,a)* or *accepted(p,a)* event occurs. If a paper is declined, an `Inform decline` goal for that paper is instantiated while if a paper is accepted, both an `Inform accepted` and `Collect finals` goals are instantiated for that paper. Once all the `Collect finals` goals are achieved, then the `Send to printer` goal can be pursued. Assuming the `Inform authors` goals have been achieved, achievement of the `Send to printer` goal achieves all the sub-goals and the overall system goal is achieved.

4.2 Organization Model

The Organization Model is created using the Model Organization task, which takes as input the GMoDS Goal Model derived in the previous task. The aim of this task is to identify system's (which is referred to as the organization) interfaces with external actors. In the case of the CMS system (see Figure 4), the system interfaces with the committee (including the `PC chair` and the `reviewers`), the `Authors`, and the `Printer`. The various ways that the actors interact with the system are modeled as protocols, which are represented by arrows from the initiator of the protocol to the responder. The initiator and responder of an protocol must be either an external actor or the organization. The system is represented as an organization, which is denoted using the ≪Organization≫ keyword.

As stated above, the `CMS` organization interacts with `Authors`, the `PC chair`, `Reviewers`, and the `Printer`. Each of these are shown as actors in Figure 4. Using the system description, the protocols required for interaction between the

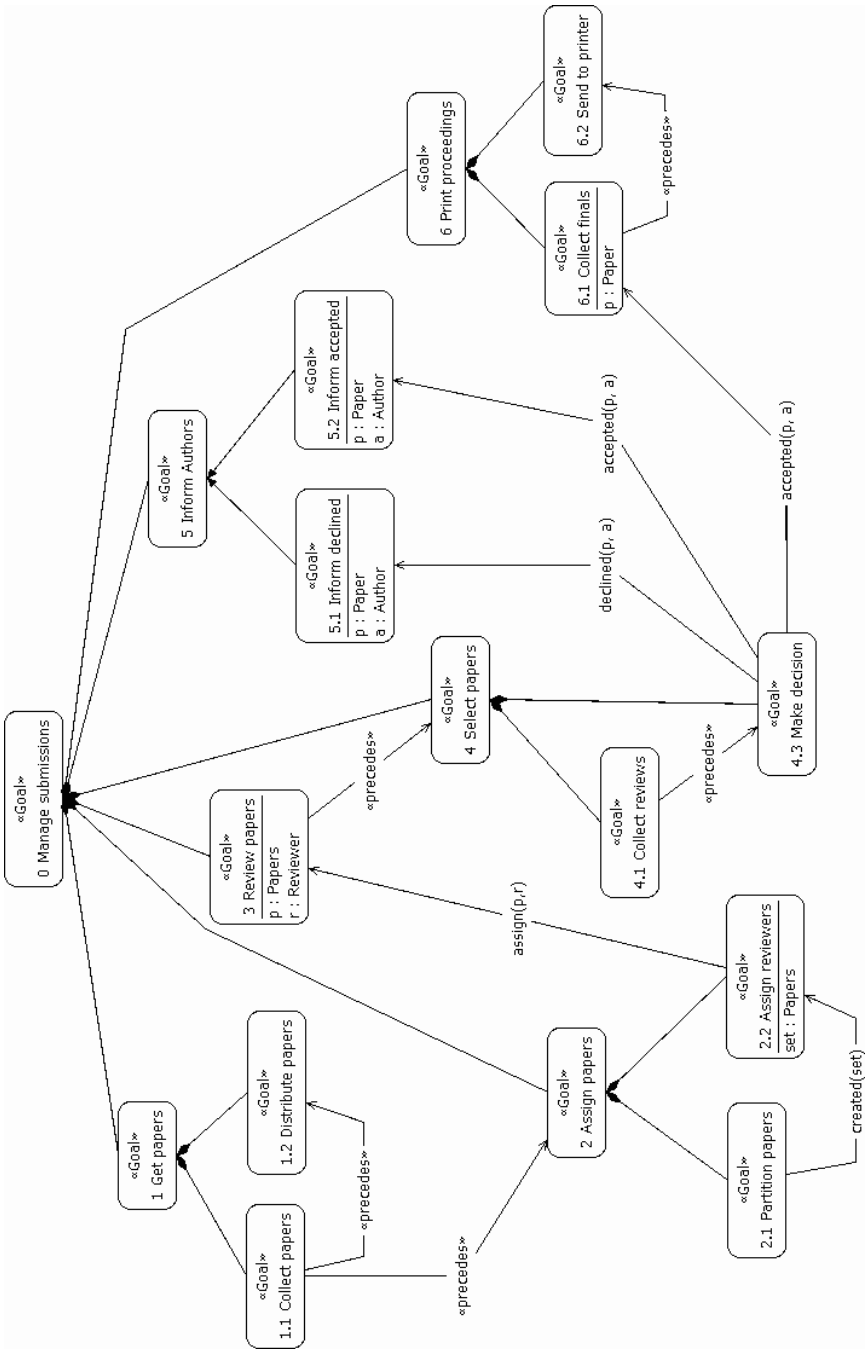

Fig. 3. CMS GMoDS Goal Model

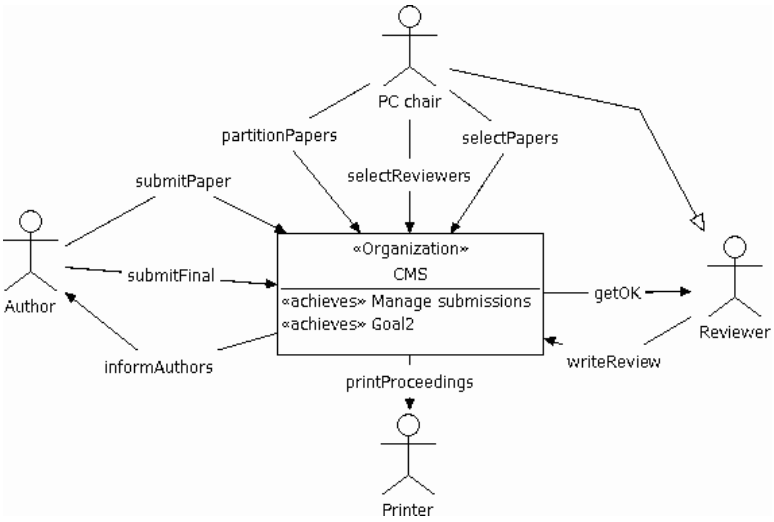

Fig. 4. CMS Organization Model

organization and the actors are identified. In the `CMS` system, an `Author` submits papers to the system using the `submitPaper` protocol. After being reviewed, the `CMS` notifies the `Author` whether their paper is accepted or rejected via the `informAuthor` protocol. If the paper was accepted, the `Author` then submits the final version of the paper using the `submitFinal` protocol. The `PC chair` actor works with the `CMS` by partitioning papers into sets via the `partitionPaper` protocol and then assigns various reviewers to review those sets of papers via the `selectReviewers` protocol. Once the reviews are complete, the `PC chair` makes the final selections via the `selectPapers` protocols. The `Reviewers` accept or reject their assignments via the `getOK` protocol and submit their reviews via the `writeReview` protocol. Finally, the final papers are sent to the `Printer` for printing via the `printProceedings` protocol.

4.3 Role Model

The Role Model is developed using the Organization Model and the GMoDS Goal Model defined earlier in the process. This focus of the Role Modeling task is to identify the roles required in the organization and their interactions (defined via protocols) with each other. The actors from the Organization Model should show up as actors in the Role Model and the protocols between the actors and the organization must be mapped to protocols between those actors and specific roles in the system. Thus, the Role Model is a refinement of the Organization Model. In addition, each leaf goal in the GMoDS Goal Model must be assigned to a role in the Role Model that can achieve it, as denoted by the ≪achieves≫ keyword in the body of the role. Thus, each role should achieve at least one leaf goal, although in general, a role may achieve multiple leaf goals.

The Role Model for the CMS system is shown in Figure 5. In the CMS system, there are seven roles: the `PaperDB`, the `Partioner`, the `Assigner`, the `PCreviewer`, the `ReviewCollector`, `FinalsCollector`, and `DecisionMaker`. The `PaperDB` role acts as the collection and distribution mechanism in the CMS. `Authors` submit papers to the `PaperDB`, while the `Partitioner`, `PCreviewer`, and `FinalsCollector` roles access the papers, abstracts, and final versions via protocols with the `PaperDB`. When all the papers have been submitted, the `PC chair` interacts with the `Partioner` role to look at the various abstracts and assign them to groups to be assigned reviewers. Once this task is complete, the `PC chair` interacts with the `Assigner` role to select reviewers to assign to each set of papers. The `Assigner` role then interacts with the `PCreviewer` role via the `reviewPapers` protocol, which interacts with the `Reviewer` via the `getOK` protocol. The `Reviewer` then reviews the papers and submits them to the `PCreviewer` role using the `writeReview` protocol. The `PCreviewer` role then sends the reviews to the `ReviewCollector` role. Once all the reviews have been submitted, the `PC chair` interacts with the `DecisionMaker` role to select papers for the conference. The status of the papers are relayed to their authors by the `DecisionMaker` role via the `informAuthors` protocol. Once the `Author` completes the final version, the paper is submitted to the `PaperDB` via the `submitFinal` protocol. When all the final papers have been submitted, the papers are then forwarded to the `Printer` from the `FinalsCollector` via the `printProceedings` protocol.

4.4 Agent Class Model

Once the roles have been defined, the analysis phase is complete and the analysis models are transformed into design models that more closely match the final implementation form. For the CMS system, this includes creating an Agent Class Model via the Model Agent Classes task. The goal of this task is to translate the role model, which captures basic system functionality, into a form more amenable to implementation. In short, this means mapping roles to agent classes. The result of this mapping for the CMS system is shown in Figure 6. The roles that each agent has been assigned to play are embedded in the body of the agent classes and are prefixed with the keyword ≪plays≫. The agent classes are denoted by the ≪Agent≫ keyword.

While the assignment of roles to agents is made by the designer, typical software engineering concepts such as coupling and cohesion should be used to evaluate the assignment. In the CMS system, two agent classes play two roles, while the other two classes play a single role each. The `PCmember` agent has been assigned to play both the `Assigner` and `Partioner` roles and thus interacts with the `PC chair`. Likewise, the `PCchair` agent also plays two roles – `ReviewCollector` and `DecisionMaker` – while also interacting with the `PC chair`. The `Referee` agent plays the `PCreviewer` role and interacts with the `Reviewer`, while the `Database` agent plays the `PaperDB` role and interacts with the `Authors` and the `Printer`. Notice that the protocols between roles in the

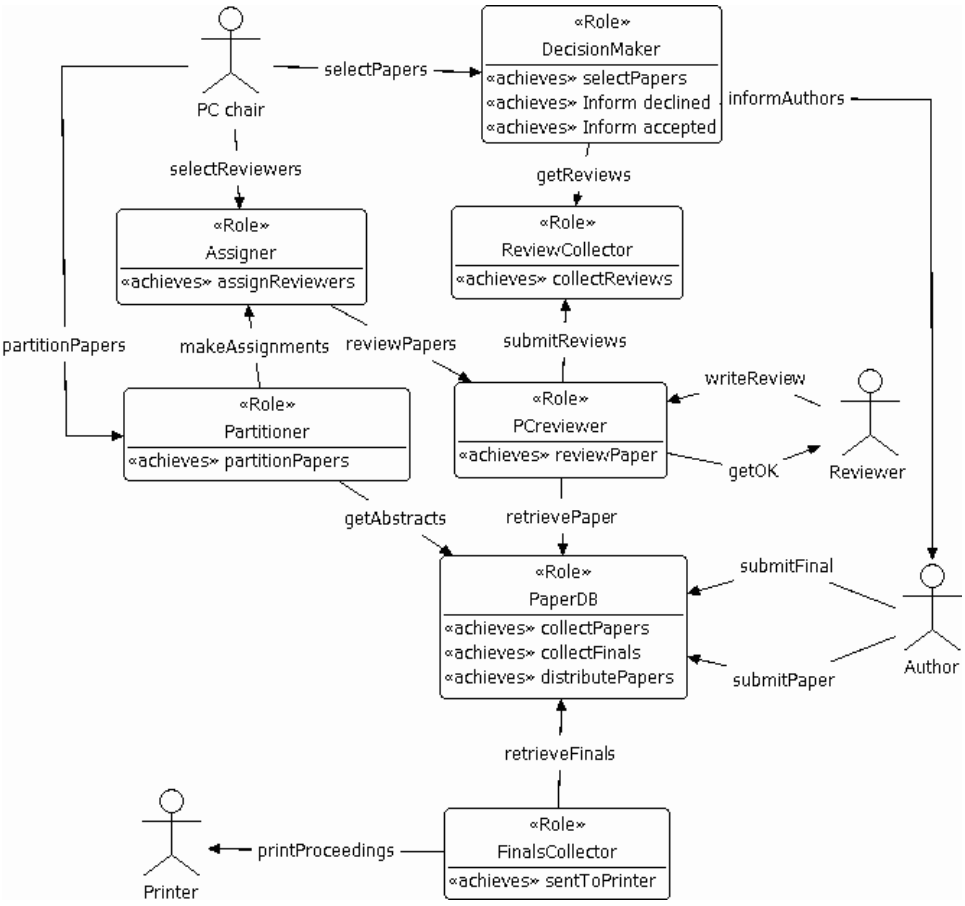

Fig. 5. CMS Role Model

Role Model have been mapped to protocols between the appropriate agents in the Agent Class Model.

After the Agent Model is complete, the agent classes and protocols have been identified, but not defined. The remaining two tasks – Model Protocols and Model Plans – are used to define the low-level design of the individual agents. The Model Protocols task is performed first, followed by the Model Plans task.

4.5 Protocol Models

The goal of the Model Protocols task is to define the details of the protocols identified in the Role Model and Agent Class Model. The Protocol Model defines the protocol in terms of messages passed between agents or between agents and external actors. As there were 13 protocols identified in the Agent Class Model (Figure 6), we must define each of the 13 protocols with individual Protocol

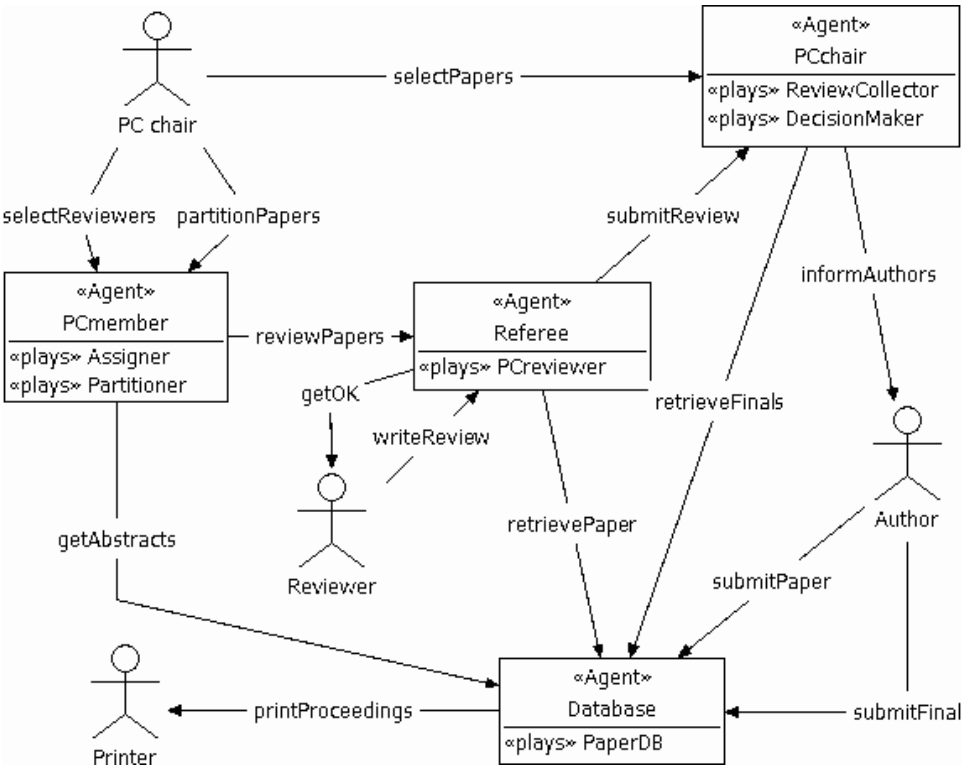

Fig. 6. CMS Agent Model

Models. The protocols are modeled using the AUML Interaction Diagrams [5], which allow us to specify message sequences, alternatives, loops, and references to other protocols.

Due to space constraints, we will only present three of the 13 protocol models: `reviewPapers`, `submitReviews`, and `retrievePapers`. Figure 7 shows the `reviewPapers` protocol, which defines the interaction between the `PCmember` and `Referee` agents, which are specified by the ≪Agent≫ keyword (protocols can also be specified between agents and actors using the same method). This protocol is very simple. The `PCmember` sends a `reviewpapers` message with a list of *paperIDs* for the `Referee` to review. The `Referee` may respond by either accepting or declining to review the set using the `accept` and `decline` messages respectively.

Figure 8 shows the `submitReviews` protocol, which defines the interaction between the `Referee` agent and the `PCchair` agent. In this protocol, the `Referee` sends several reviews via a `submit` message to the `PCchair` followed by a `done` message. The `PCchair` does not send a message in response.

Figure 9 shows the `retrievePapers` protocol, which defines a simple request protocol between the `Referee` and `Database` agents. According to the protocol,

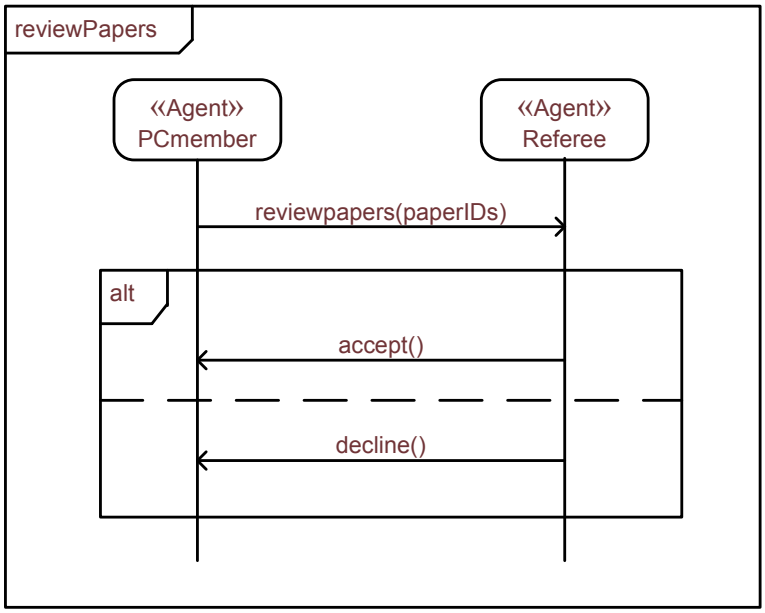

Fig. 7. CMS reviewPapers Protocol Model

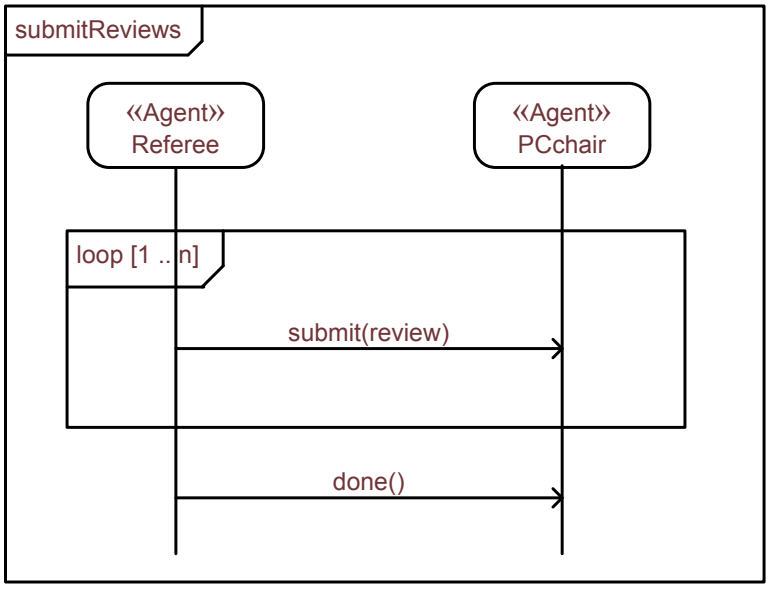

Fig. 8. CMS submitReviews Protocol Model

the `Referee` issues a request to the `Database` for a set of papers via a `request` message. The `Database` simply responds with the appropriate set of papers in a `receive` message.

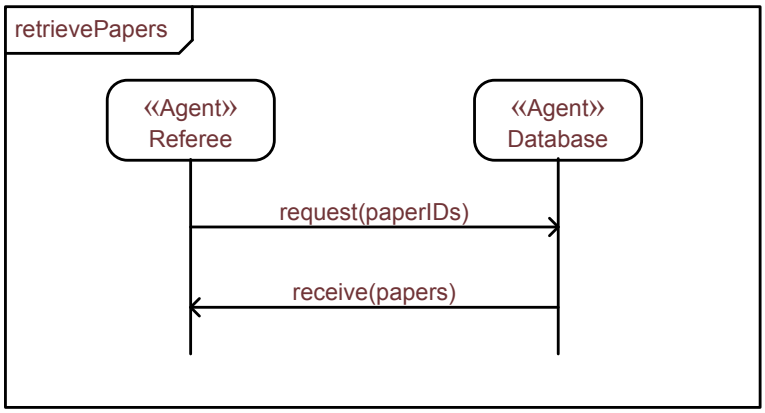

Fig. 9. CMS retrievePapers Protocol Model

4.6 Agent Plan Models

The last design models developed in our O-MaSE compliant process are the Agent Plan Models. Basically, a plan represents a means by which agents can satisfy a goal in the organization, thus a plan can be viewed as an algorithm for achieving a specific goal. Again, because there are four different agents defined in the Agent Class Model, there should be at least four Agent Plan Models developed, one for each agent. Depending on the internal architecture chosen for each agent, we could develop multiple Agent Plan Models for each agent. This might be the case when we wanted a unique plan for each role an agent could play or if we could choose between multiple plans to achieve the same goal. In either case, the agent architecture would be responsible for selecting the appropriate plans and interleaving their execution if required.

O-MaSE plans are modeled using a finite state automata to specify a single thread of control that defines the behavior that the agent should exhibit. As such, each plan has a `start state` and an `end state`. All messages are sent and received on state transitions. For the Plan Model, the syntax of the transitions is `[ guard ] receive(message,sender) / send (message,receiver)`. The `guard` defines a boolean condition that determines if the transition is enabled. The `receive(message,sender)` is a message that is received from the `sender` agent that enables the transition, while the `send(message,receiver)` is a message sent to the `receiver` agent when the transition occurs. Messages are specified in the form $performative(p_1...p_n)$, where the $performative$ is the

name of the message and $p_1...p_n$ are the parameters of the message. Each part of the transition is optional and a null transition may exist between two states.

Each state has a (possibly empty) set of actions that are executed sequentially once the state is entered. Each action is represented in the form of a function that returns a value. These actions may represent internal computations of the agent or be part of interactions with objects in the environment. Transitions out of a state are not enabled until all actions have returned their values. The parameters to the actions, the action return values, and all parameters in messages in the plan are considered variables within a single name space, thus a parameter `X` of a message is the same as the return value of an action `X`.

Figure 10 shows the Plan Model for the Reviewer agent. The plan starts upon receipt of a `reviewpapers` message from the `PCmember` agent. Immediately upon receipt of the message, the agent sends a `request` message to the `Database` agent to get the papers identified by the list of paper identifiers, *paperIDs*, and moves into the `Wait` state. When the `Database` returns a list of the *papers* requested, the plan moves into the `Evaluate` state where it interacts with its associated `Reviewer` via the `getOK` action. If the `Review` does not agree to review the set of papers, a `decline` message is sent to the `PCmember` agent and the plan ends. However, if the `Review` does agree to review the set of papers, an `accept` message is sent to the `PCmember` agent and the plan moves to the `Review` state. In the `Review` state, the plan interacts with the `Reviewer` via the `getSelectedPaper` and `getReview` actions. Every time a review is completed, the *review* is submitted to the `PCchair` agent via a `submit` message and the the list of *papers* is reduced in size. Once the *papers* list is empty, the plan moves into the `Done` state and immediately sends a `done` message to the `PCchair` agent.

As the Agent Plan Model implements the protocols identified in the Agent Class Model and defined in the Protocol Models, it is critical that a Plan Model be consistent with all Protocol Models that it is required to implement. Thus, by looking at Figure 6, we can see that `Referee` agent must implement the `reviewPapers`, `getOK`, `writeReview`, `retrievePapers` and `submitReview` protocols. While the `getOK` and `writeReview` protocols interact with the `Reviewer` actor and are implemented as action, we can analyze the `reviewPapers`, `retrievePapers` and `submitReview` protocols with the `Referee` Agent Plan Model to verify that they are indeed consistent.

5 Conclusion and Future Work

This paper has presented an example of an O-MaSE complaint process for the Conference Management System example. After defining a custom process for this example, we demonstrated how to step through each of the tasks to create a set of models that eventually resulted in a set of Agent Plan Model that describe how to implement the set of agents required for the system. While the example showed several of the main O-MaSE tasks and models, it did not include several potentially powerful concepts supported by O-MaSE as defined in [4]. Specifically, we did not use the Define Roles, Model Domain, or Model

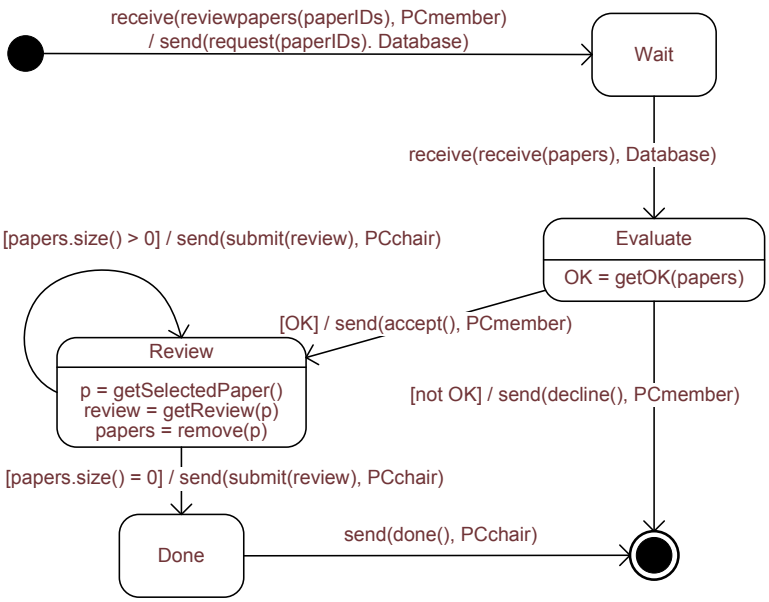

Fig. 10. CMS Reviewer Agent Plan Model

Policies, tasks. We also did not do detailed design that would have included the tasks of Model Capabilities and Model Actions.

The approach we used also reflected a traditional approach to designing agent systems. That is, we defined a set of roles and then designed specific agent classes to perform those roles. While this approach is straightforward and is very useful for the CMS system, it does not reflect the power of the O-MaSE approach for modeling highly adaptive systems. By incorporating the notion of capabilities within O-MaSE, we allow designers to design much more flexible and adaptive multiagent systems. Instead of assigning roles directly to specific agent classes, we define roles in terms of the capabilities required to carry out that role and agent classes in terms of the capabilities they posses. This capability-based role definition allows the system to reorganize when agents gain or lose capabilities to select the best agent to play a specific role. Role behavior is then specified in terms of a Role Plan Model that uses its required capabilities to implement the actions in the plan. By specifying roles and agents in terms of capabilities required or possessed, the assignment of roles to specific agents can be delayed until runtime. When a goal is instantiated that requires a specific role, the agent that has all the required capabilities of the role can be selected to play that role to achieve a specific goal.

We are continuing to add new tasks and models to O-MaSE to allow it to be even more flexible and useful. We are currently encoding O-MaSE for integration into the Eclipse Process Framework (EPF) tool [7], which allows designers to pick and choose method fragments to create custom processes. While the EPF

supports the main concepts we need to define O-MaSE, we are looking into extending EPF to allow us to formally verify that the custom process created is actually O-MaSE compliant.

We are also developing agentTool III (aT3) [2] to support O-MaSE modeling. aT3 is being developed by an Eclipse plugin and will be available at the aT3 web site. The Goal Model, Organization Model, Role Model, and Agent Class Model were all created using a prototype of aT3.

6 References

1. DeLoach, S.A. Engineering Organization-based Multiagent Systems. LNCS Vol. 3914, Springer, (2006) 109-125.
2. DeLoach, S.A. Multiagent & Cooperative Robotics Laboratory. "agentTool III Home Page," http://agenttool.projects.cis.ksu.edu/ (April 2007).
3. DeLoach, S.A., Mark F. Wood and Clint H. Sparkman, Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, 11(3) (2001) 231-258.
4. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyenan, W.H., and Valenzuela, J. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. to appear at the 8th International Workshop on Agent Oriented Software Engineering, Honolulu HI, May 2007.
5. Huget, M. and Odell, J. Representing Agent Interaction Protocols with Agent UML. In Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3. International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, 1244-1245. 2004.
6. Miller, M. A Goal Model for Dynamic Systems. Master's Thesis, Dept. of Computing and Information Sciences, Kansas State University, 2007.
7. The Eclipse Foundation. "Eclipse Process Framework Project Home Page," http://www.eclipse.org/epf/ (April 2007).
8. Zambonelli, F., Jennings, N.R., and Wooldridge, M.J. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. International Journal of Software Engineering and Knowledge Engineering. Volume 11, Number 3, June 2001. Pages 303-328.