The O-MASE Methodology

Scott A. DeLoach and Juan C. Garcia-Ojeda

Abstract

Today's software industry is tasked with building evermore complex software applications, and multiagent system technology is a promising approach capable of meeting these new demands. Unfortunately, multiagent systems have not been widely adopted in industry for reasons that include lack of industrial strength methods and tools to support multiagent development. Method engineering, an approach to constructing processes from a set of existing method fragments, has been suggested as a solution to this problem. This chapter presents the Organization-based Multiagent Software Engineering (O-MaSE) methodology framework, which integrates a set of concrete technologies aimed at facilitating industrial acceptance. Specifically, O-MaSE is a customizable agent-oriented methodology based on consistent, well-defined concepts supported by plug-ins to an industrial strength development environment, agentTool III. O-MaSE is defined, and demonstrations of customizing O-MaSE for the CMS problem as well applying the customized process to the CMS design are presented.

1 Introduction

Organization-based Multiagent Software Engineering (O-MaSE) [4] is a new approach in the analysis and design of agent-based systems, being designed from the start as a set of method fragments to be used in a method engineering

S.A. DeLoach (✉)
Kansas State University, 234 Nichols Hall, Manhattan, KS 66506, USA
e-mail: sdeloach@ksu.edu; sdeloach@k-state.edu

J.C. Garcia-Ojeda
Facultad de Ingenieria de Sistemas, Universidad Autonoma de Bucaramanga, Avenida 42 No 48-11, El Jardin. Bucaramanga, Santander, Colombia
e-mail: jgarciao@unab.edu.co

M. Cossentino et al. (eds.), *Handbook on Agent-Oriented Design Processes*,
DOI 10.1007/978-3-642-39975-6_9,

framework [1,2,12]. The goal of O-MaSE is to allow designers to create customized agent-oriented software development processes. O-MaSE consists of three basic structures: (1) a metamodel, (2) a set of method fragments, and (3) a set of guidelines. The O-MaSE metamodel defines the key concepts needed to design and implement multiagent systems. The method fragments are tasks that are executed to produce a set of work products, which may include models, documentation, or code. The guidelines define how the method fragments are related to one another.

The aT3 Process Editor (APE) shown in Fig. 1 is a tool that supports the creation of custom O-MaSE-compliant processes [10]. APE is part of the agentTool III toolset, which provides tool support to developing multiagent systems using O-MaSE [11]. There are five key elements of APE: a Method Fragment Library, the Process Editor, a set of Task Constraints, a Process Consistency checker, and a Process Management tool. The Library is a repository of O-MaSE method fragments, which can be extended by APE users. The Process Editor allows users to create and maintain O-MaSE-compliant processes. The Task Constraints view helps process engineers specify Process Construction Guidelines to constrain how tasks can be assembled, while the Process Consistency mechanism verifies the consistency of custom processes against those constraints. Finally, the Process Management tool provides a way to measure project progress using the custom process.

O-MaSE also provides a set of Method Construction Guidelines that states how O-MaSE method fragments may be combined to form O-MaSE-compliant processes. Table 1 shows the Method Construction Guidelines for the O-MaSE Tasks. These Method Construction Guidelines are defined in terms of a precondition and post-condition. The precondition specifies the set of Work Products that must be available prior to the Task being undertaken while the post-conditions specify the Work Products produced by the task. For example, for the Model Goals task, either a Requirements Spec must be available or a Goal Model/GMoDS and a Role Model must be available. The Requirements Spec is used when the Model Goals task is used to model system-level goals, while the Goal Model/GMoDS and Role Model are used when the task is used to model role-level goals. Disjunctive preconditions generally specify alternative ways the Task can be used. However, it does not limit what information can be used in the definition of a model. For instance, the Model Domain task only requires a Requirements Spec as input; however, that does not mean that other Work Products such as Goal Models cannot be used in the Task. This additional information is generally documented in the individual task definitions.
Useful references related to O-MaSE include the following:

- Scott A. DeLoach and Juan Carlos Garcia-Ojeda. O-MaSE: a customizable approach to designing and building complex, adaptive multiagent systems. International Journal of Agent-Oriented Software Engineering. Volume 4, no. 3, 2010, pp. 244–280.
- Juan C. Garcia-Ojeda, Scott A. DeLoach, and Robby. agentTool Process Editor: Supporting the Design of Tailored Agent-based Processes. Proceedings of the 24th Annual ACM Symposium on Applied Computing to be held at the Hilton Hawaiian Village Beach Resort and Spa Waikiki Beach, Honolulu, Hawaii, USA. March 8–12, 2009.

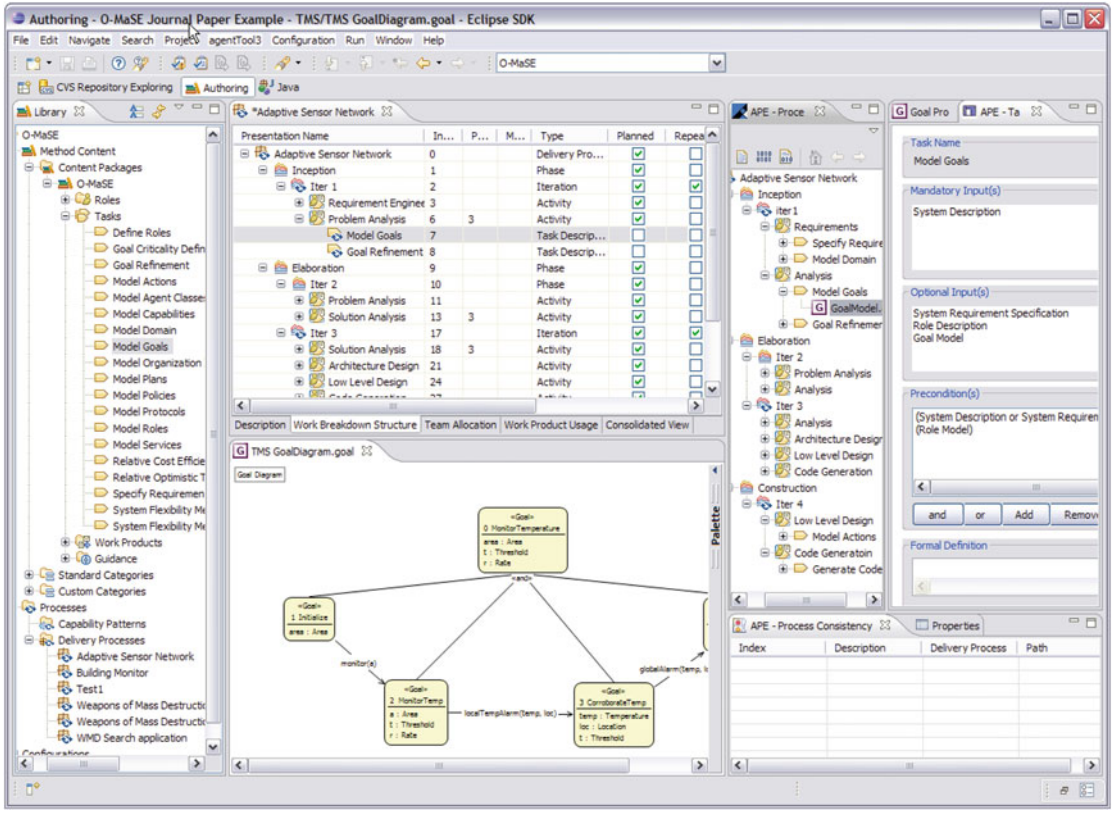

Fig. 1 agentTool III Process Editor

Table 1 Method construction guidelines

Task	Pre-condition	Post-condition
Requirements Specification	True	Requirements Spec
Model Goals	Requirements Spec ∨ ((Goal Model ∨ GMoDS) ∧ Role Model)	Goal Model
Refine Goals	Goal Model	GMoDS
Model Domain	Requirements Spec	Domain Model
Model Organization Interfaces	Requirements Spec ∧ GMoDS	Organization Model
Model Roles	GMoDS ∧ Organization Model	Role Model
Define Roles	Role Model	Role Description
Model Agent Classes	GMoDS ∨ Role Model ∨ Organization Model	Agent Class Model
Model Protocols	Role Model ∨ Agent Class Model	Protocol Model
Model Policies	GMoDS ∨ Organization Model ∨ Role Description ∨ Agent Class Model	Policy Model
Model Plans	(GMoDS ∧ Role Model) ∨ (GMoDS ∧ Agent Class Model)	Plan Model
Model Capabilities	Role Model ∧ Agent Class Model ∨ Domain Model	Capability Model
Model Actions	Capability Model ∧ Domain Model	Action Model
Code Generation	(Plan Model ∨ Protocol Model) ∧ (Capability Model ∨ Action Model)	Source Code

- Scott DeLoach, Lin Padgham, Anna Perini, Angelo Susi, and John Thangarajah. Using Three AOSE Toolkits to Develop a Sample Design. International Journal of Agent Oriented Software Engineering. Volume 3, no. 4, 2009, 2009, pp 416–476.
- Scott A. DeLoach. Organizational Model for Adaptive Complex Systems. in Virginia Dignum (ed.) Multi-Agent Systems: Semantics and Dynamics of Organizational Models. IGI Global: Hershey, PA. ISBN: 1-60566-256-9 (March 2009). This chapter copyright 2008, IGI Global, www.igi-pub.com. Posted by permission of the publisher.
- Lin Padgham, Michael Winikoff, Scott DeLoach, and Massimo Cossentino. A Unified Graphical Notation for AOSE. Proceedings of the 9th International Workshop on Agent Oriented Software Engineering, Estoril Portugal, May 2008.
- Scott A. DeLoach. Developing a Multiagent Conference Management System Using the O-MaSE Process Framework. Proceedings of the 8th International Workshop on Agent Oriented Software Engineering, May 14, 2007, Honolulu, Hawaii.
- Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenan and Jorge Valenzuela. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. Proceedings of the 8th International Workshop on Agent Oriented Software Engineering, Honolulu HI, May 2007.

Table 2 O-MaSE overview

Entity	Task	Work Product	Role
Requirements Gathering	Requirements Specification	Requirements Spec	Requirements Engineer
Problem Analysis	Model Goals	Goal Model	Goal Modeler
	Refine Goals		
	Model Domain	Domain Model	Domain Modeler
Solution Analysis	Model Organization Interfaces	Organization Model	Organization Modeler
	Model Roles	Role Model	Role Modeler
	Define Roles	Role Description Document	
	Define Role Goals	Role Goal Model	
Architecture Design	Model Agent Classes	Agent Class Model	Agent Class Modeler
	Model Protocols	Protocol Model	Protocol Modeler
	Model Policies	Policy Model	Policy Modeler
Low Level Design	Model Plans	Agent Plan Model	Plan Modeler
	Model Capabilities	Capabilities Model	Capabilities Modeler
	Model Actions	Action Model	Action Modeler
Code Generation	Generate Code	Source code	Programmer

- Scott A. DeLoach and Jorge L. Valenzuela. An Agent-Environment Interaction Model. in L. Padgham and F. Zambonelli (Eds.): AOSE 2006, LNCS 4405, pp. 1–18, 2007. Springer-Verlag, Berlin Heidelberg 2007.
- Scott A. DeLoach. Multiagent Systems Engineering of Organization-based Multiagent Systems. 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05). May 15–16, 2005, St. Louis, MO. Springer LNCS Vol 3914, Apr 2006, pp 109–125.

1.1 The O-MaSE Life Cycle

O-MaSE was designed from scratch as a set of fragments that could be assembled by developers to meet the specific requirements of their project. While SPEM [15] uses Phases to organize the various Activities of a development method, O-MaSE makes no commitments to a predefined set of Phases. Instead, O-MaSE explicitly defines Activities and Tasks (see an overview in Table 2) and allows method engineers to organize Activities in different ways based on project need. For instance, O-MaSE has been used to support modern iterative, incremental approaches as well as much simpler waterfall-based approaches. The fact that O-MaSE does not commit to any specific set of phases causes a minor problem when trying to map O-MaSE directly to the DPDT. To alleviate this problem, we assume that we follow a traditional waterfall approach when describing O-MaSE as shown in Fig. 2. As shown, there are three main Phases: Requirement Analysis, Design, and Implementation, with

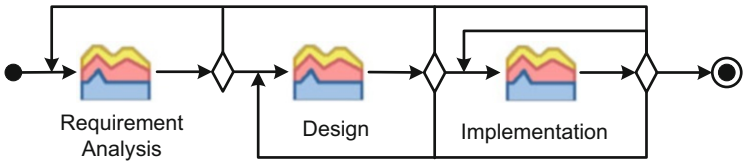

Fig. 2 Using waterfall phases with O-MaSE

the main Activities allocated as shown. When using O-MaSE on a real project, the process designer is free to define their own set of phases and iterations and to assign Activities and Tasks to those phases and iterations as appropriate. As this will be unique for each system being developed, there are no hard-and-fast rules on what activities should be placed in which phases.

1.2 The O-MaSE Metamodel

The O-MaSE metamodel defines the main concepts and relationships used to define multiagent systems. The O-MaSE metamodel is based on an organizational approach and includes notions that allow for hierarchical, holonic, and team-based decomposition of organizations. The O-MaSE metamodel was derived from the Organization Model for Adaptive Computational Systems (OMACS), which captures the knowledge required of a system's organizational structure and capabilities to allow it to organize and reorganize at runtime [5]. The key decision in OMACS-based systems is which agent to assign to which role in order to achieve which goal. As shown in Fig. 3, an Organization is composed of six entity types: Goals, Roles, Agents, Organizational Agents, a Domain Model, and Policies. Each of these entities is discussed below, and a concise definition is given in Table 3.

While a variety of subtle interpretations of goals exist in the artificial intelligence and agent communities, O-MaSE defines a Goal as an objective of the organization, which is generally described in terms of some desired state of the world. A Role defines a position within an organization whose behavior is expected to achieve a particular goal or set of goals. (Due to the naming conflict between O-MaSE Roles and SPEM roles, the term *method role* is used to refer to SPEM roles throughout the remainder of this chapter.) Agents are assigned to play those roles and perform the behavior expected of those roles. Agents are autonomous entities that can perceive and act upon their environment [19]. To carry out perception and action, an agent possesses a set of capabilities. Capabilities can be used to capture soft abilities (i.e., algorithms) or hard abilities (i.e., physical sensors or effectors). An agent that possesses all the capabilities required to play a role may be assigned that role in the organization. Capabilities can be defined as (1) a set of sub-capabilities, (2) a set of actions that may interact with the environment, or (3) a plan that uses actions in specific ways.

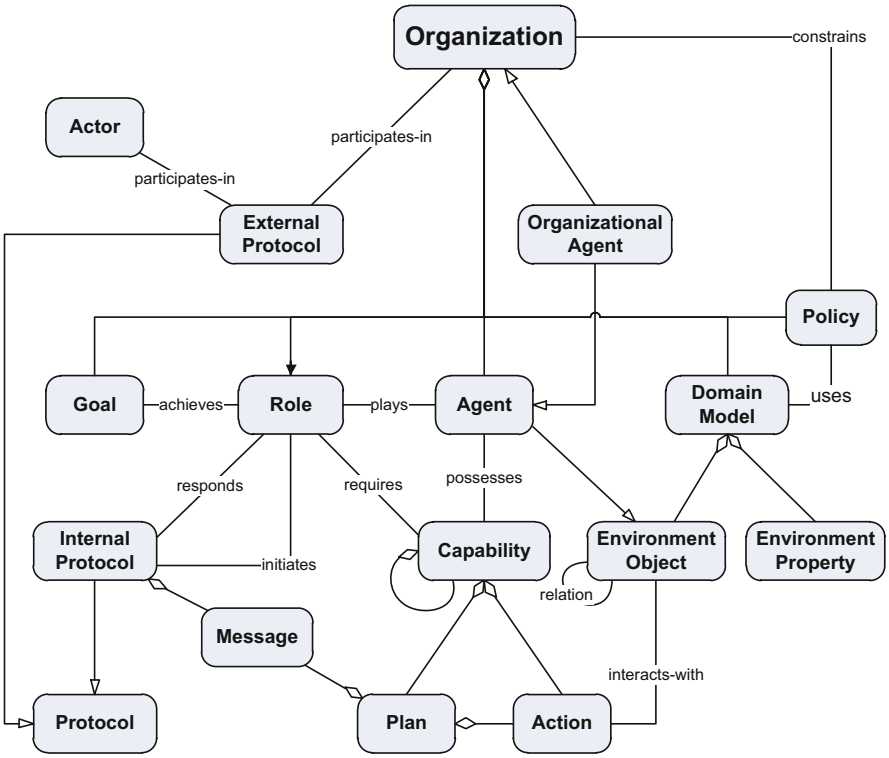

Fig. 3 O-MaSE metamodel

Table 3 Metamodel entities

Entity	Definition
Goal	A desirable state; goals capture organizational objectives
Role	Capture behavior that achieves a particular goal or set of goals
Agent	Autonomous entities that perceive and act upon their environment; agents play roles in the organization
Organizational Agent	A sub-organization that functions as an agent in a higher-level organization
Capability	Soft abilities (algorithms) or hard abilities of agents
Domain model	Captures the environment including objects and general properties describing how objects behave and interact
Policy	Constrain organization behavior often in the form of liveness and safety properties
Protocol	Define interaction between agents, roles, or external Actors; they may be internal or external
Actor	Actors that exist outside the system and interact with the system
Plan	Abstractions of algorithms used by agents; plans are specified in terms of actions with the environment and messages in protocols

Organizational Agents (OAs) are organizations that act as agents in a higher level organization and thus capture the notion of organizational hierarchy. As agents, OAs may possess capabilities, coordinate with other agents, and be assigned to play roles. OAs are similar to the notion of non-atomic holons in the ASPECS methodology [3]. Therefore, OAs represent an extension to the traditional Agent–Group–Role (AGR) model [8, 9] and the organizational metamodel proposed by Odell et al. in [17].

The Domain Model is used to capture the key elements of the environment in which agents will operate. These elements are captured as Domain Object Types from the environment, which includes agents, and the relationships between those object types. It can also be used to capture general Environment Properties that describe how the objects behave and interact [6]. A designer may use entities defined in the O-MaSE model (goals, roles, agents, etc.) along with entities defined in the Domain Model to specify organizational Policies to constrain how an organization may behave in a particular situation. Policies are often used to specify liveness and safety properties of the system being designed.

Protocols define interactions between roles or between the organization and external Actors. Protocols are generally defined as patterns of communication between such entities [16]. A protocol can be of two types, External or Internal. External Protocols specify interactions between the organization and external actors (i.e., humans or other software applications), while Internal Protocols specify interactions between agents playing specific roles in the organization. Either messages or actions can be used to define protocols. Messages are typically used for communications; however, actions may be used to modify the environment as a means of communication [14].

2 Phases

The first step in using O-MaSE to define a system is to define an O-MaSE compliant process. There may be several ways to define an O-MaSE compliant process; however, the simplest approach is to perform a bottom-up analysis of the work products required to produce the desired system. In a bottom-up approach, the key decision is what type of system we need to develop and what are the final work products that are needed to support the implementation of such a system. From there we work backwards to determine which other work products are required to produce the final work products. The example given here for the CMS is an appropriate O-MaSE compliant process [12].

Ultimately, the CMS is a centralized system with which a variety of humans interact. The roles of the system and humans are well defined, and, outside of major system shutdown, there is little chance of failure that would require that various agents might need to be reassigned goals in order for the system to work efficiently and robustly. Therefore, there is no requirement for an autonomously adaptive system such as produced by OMACS [7]. Thus, the definition of individual capabilities of the roles and agents is not required. Therefore, we can implement the system by defining a set of agent classes, the protocols between those classes, a set of

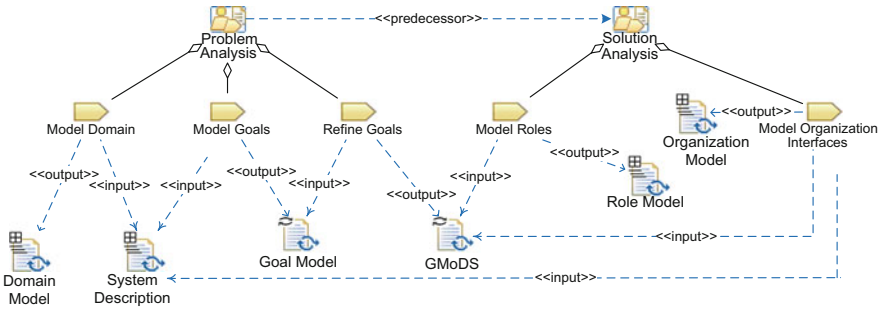

Fig. 4 CMS O-MaSE-compliant process—analysis phase

policies to constrain agent behavior and interaction, and a set of plans to implement the agent behavior. This information is provided by the Agent Class Model, Protocol Model, Policy Model, and Plan Model, which are defined using the Model Agent Classes, Model Protocols, Model Policies, and Model Plans tasks.

In order to provide the appropriate inputs to the tasks, we need to define a set of roles and goals for the system. Also, to allow us to define the parameters of the goal model, the protocols between the agents, and the policies and plans, we need to have a valid domain model. Thus, the work products we need to create to support the design phase include a Domain Model, a Goal Model, a Goal Model for Dynamic Systems (GMoDS) model, an Organizational Model, and a Role Model. These work products are defined using the Model Domain, Model Goals, Refine Goals, Model Organizational Interfaces, and Model Roles tasks.

The final step is to define the phases and possible iterations for our process. Since this is a fairly simple system, we choose a simple waterfall approach. The input to the process is the system specification of the CMS, while the output is the design models discussed above. We do not consider an implementation phase as the system can be implemented in a number of ways depending on where and how the final system is to be used.

Thus, the final process chosen for designing the CMS is a basic waterfall approach as shown in Fig. 2. However, because the system does not require adaptivity in terms of assigning agents to roles, we have simplified the Requirement Analysis and Design phases as shown in Figs. 4 and 5. Requirement Analysis begins by using the existing system requirements to define a Domain Model in the Model Domain task. Next, we define the basic Goal Model and refining it into a GMoDS Goal Model via the Model Goals and Goal Refinement tasks. Once the Goal Model is complete, the GMoDS Goal Model is used to create the initial Role Model.

The Design Phase begins by creating an Agent Class Model based on the Role Model and GMoDS model created during the Requirement Analysis phase. The details of the protocols identified in the Agent Class Model are further refined into several Protocol 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 Role

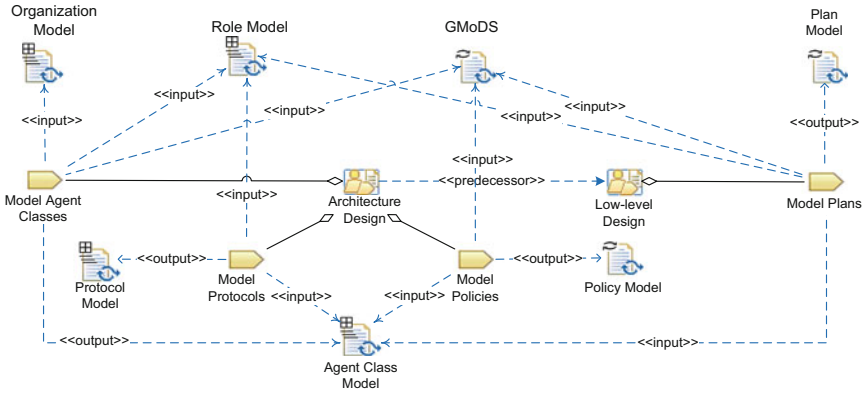

Fig. 5 CMS O-MaSE-compliant process—design phase

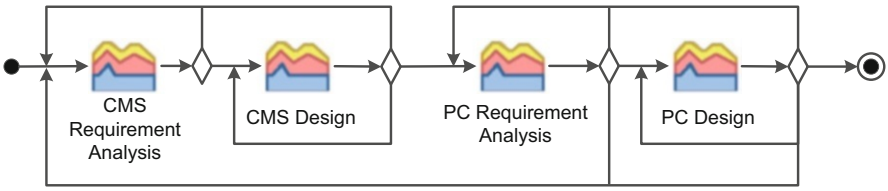

Fig. 6 O-MaSE-compliant process for CMS and PC organizations

Model as it also identifies protocols. Next, we model the policies that the agents and their protocols must adhere to. Finally, the plans of the agents are defined in the Model Plans task and produce a set of plans that implement the agent behavior.

However, to manage the complexity of the Conference Management System (CMS) provided earlier in the book, we decided to make use of the decomposition and abstraction mechanisms available in O-MaSE. Specifically, we decided that since the Program Committee (PC) (including the PC Chair, Vice-Chairs, and PC Members) operated as a single entity in relation to the other system actors (Authors, Publishers, and Reviewers), we would treat the PC as a separate entity in the design process and use the O-MaSE notion of an Organizational Agent to capture the PC. Thus, at the top-level description of the system, the PC is a single entity that is further decomposed in terms of its own organization. This actually requires a slight modification to the waterfall model. In actuality, this approach simply requires one iteration of the Requirement Analysis and Design phases for the CMS organization and a second iteration for the PC organization. However, to clarify the situation, we show an extended version of the CMS process in Fig. 6.

Again, as a reminder, the phases used to define O-MaSE as presented below are not actually part of the O-MaSE definition but only included to help define O-MaSE according to the DPDT. The process shown in Figs. 6, 4, and 5 is actually a subset

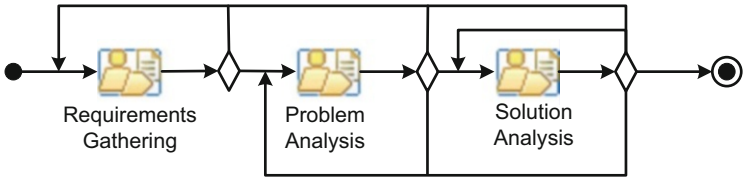

Fig. 7 Requirement Analysis phase flow of activities

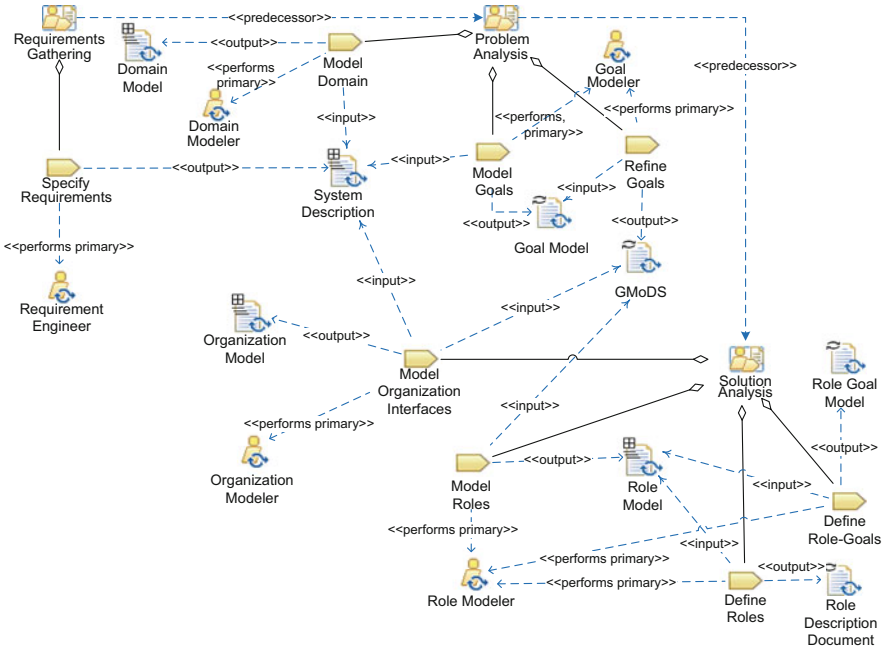

Fig. 8 Requirement Analysis phase in terms of activities and work products

of the phases, tasks, and activities discussed below, and examples of work products produced from this process are presented where appropriate.

2.1 Requirement Analysis

In traditional software engineering practice, the requirement analysis phase attempts to define and validate requirements for a new or modified software product, taking into account the views of all major stakeholders. A generic example of an O-MaSE requirement analysis phase is shown in Fig. 7 in terms of process flow and Fig. 8 in terms of process roles, work products, and tasks.

2.1.1 Process Roles

This phase uses five roles: Requirement Engineer, Goal Modeler, Domain Modeler, Organization Modeler, and Role Modeler.

Requirement Engineer

The Requirement Engineer captures and validates the requirements of the system. Thus, the person in this role must be able to think abstractly, work at high levels of abstraction, and collaborate with stakeholders, domain modelers, and project managers.

Goal Modeler

The Goal Modeler is responsible for the generation of the GMoDS goal model. Thus, Goal Modeler must understand the system description/SRS, be able to interact openly with various domain experts and customers, and be proficient in GMoDS AND/OR Decomposition and ATP Analysis [5].

Domain Modeler

The Domain Modeler captures the key concepts and vocabulary in the current and envisioned environment of the system, helping to further refine and validate requirements.

Organization Modeler

The Organization Modeler is responsible for documenting the Organization Model. Thus, the Organization Modeler must understand the system requirements, Goal Model, and Domain Model and be skilled in organizational modeling techniques.

Role Modeler

The Role Modeler creates the Role Model and the Role Description work products, which requires knowledge of the role model specification and a general knowledge of the system.

2.1.2 Activity Details

In the Requirement Analysis phase, there are three activities: Requirement Gathering, Problem Analysis, and Solution Analysis.

Requirement Gathering

Requirement Gathering is the process of identifying software requirements from a variety of sources. Typically, requirements are either functional requirements, which define the functions required by the software, or nonfunctional requirements, which specify traits of the software such as performance quality and usability. An overview of the Requirement Gathering tasks and work products used is shown in Fig. 9.

Problem Analysis

Problem Analysis captures the purpose of the product and documents the environment in which it will be deployed using three tasks: Model Domain, Model Goals,

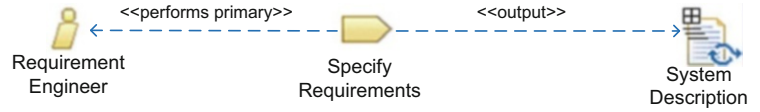

Fig. 9 Requirement Gathering activity diagram

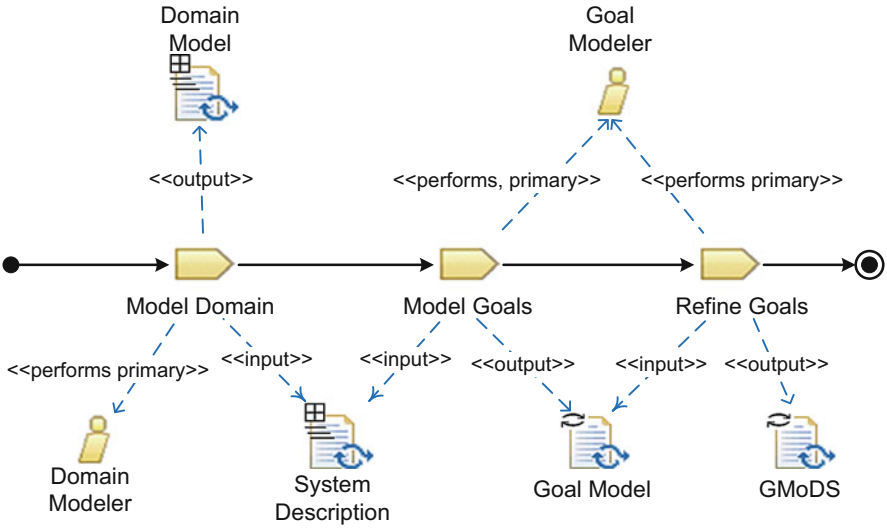

Fig. 10 Problem Analysis activity diagram

and Refine Goals. The Model Domain task captures the object types, relationships, and behaviors within the domain in which system will operate. The Domain Model captures the environment as a set of Object Types and Agents that are situated in the environment. Object types are defined by a name and a set of attributes. In O-MaSE, domain object types are similar to object classes rather than instances. The Model Goals task transforms the initial system requirements into a set of structured goals for the system. The deliverable of the Model Goals task is an initial Goal Model. The Refine Goals task captures the dynamic aspects of the Goal Model and further defines each goal using a technique called Attribute–Precede–Trigger Analysis. The result is a refined version of the Goal Model called a GMoDS goal model [5]. An overview of the Problem Analysis tasks and work products used is shown in Fig. 10.

Solution Analysis

Solution Analysis defines the required system behavior based on the goal and domain models. The end result is a set of roles and interactions in the Organization Model. Solution Analysis is decomposed into four tasks: Model Organizational Interfaces, Model Roles, and either Define Roles or Define Role Goals. The Model Organization Interfaces task identifies the organization's interfaces with external entities, which can include other agents, organizations, or external actors.

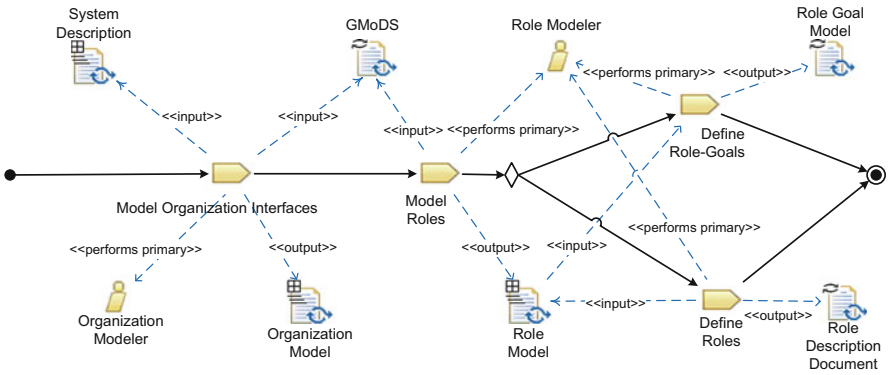

Fig. 11 Solution Analysis activity diagram

The Model Roles task identifies all the roles in the organization as well as their interactions with each other and with external actors, resulting in a Role Model. The goal of role modeling is to assign each leaf goal from the organization Goal Model to a specific role and to identify interactions between roles as well as with external actors. Interactions with external actors should be consistent with the Organization Model. The internal behavior of roles can be defined either through the Define Roles or Define Role Goal tasks and both types of definitions can be used within the same system. In the Define Roles task, the designer specifies the capabilities required by a role, the goals the role is able to achieve, constraints associated with the role, and the plan(s) that implements the role, which are defined via the Model Plan task as described below. In the Define Role Goals task, role behavior is defined by a role-level Goal Model. The top-level goal in a role-level Goal Model is the leaf goal from the organization that is to be achieved by the role. An overview of the Solution Analysis tasks and work products used is shown in Fig. 11.

2.1.3 Work Products

The Requirement Analysis phase produces eight work products. One is a pure text document, the System Description. The other seven work products are models that create various elements in the O-MaSE metamodel. The relationships between these models and the O-MaSE metamodel are documented in Fig. 12. Each model is defined in terms of elements from the O-MaSE metamodel, which are represented with UML class icons. Within each model, each metamodel element may be Defined, reFined, Quoted, Related, or Relationship Quoted.

Work Product Kinds

There are six possible work products produced in the Requirement Analysis phase: System Description Specification, Goal Model, GMoDS Model, Domain Model, Organization Model, and Role Model as defined in Table 4. Examples of most of these models are provided in the sections below. Each work product is specified in

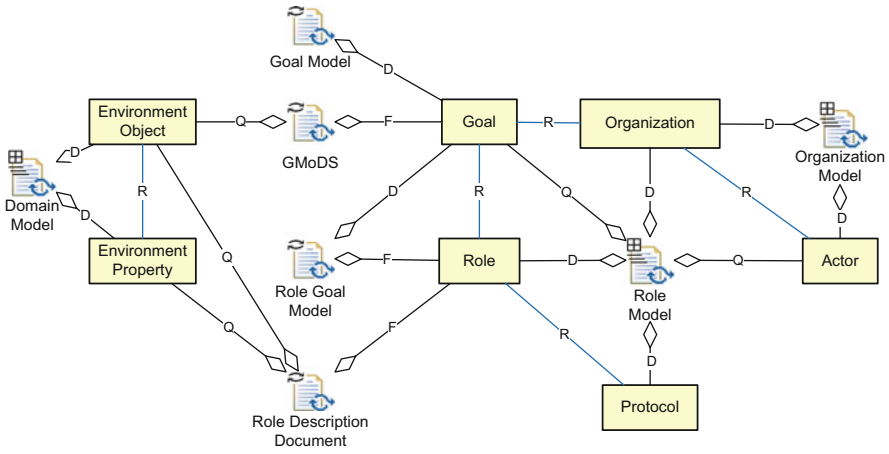

Fig. 12 Requirement Analysis document structure

Table 4 Requirement Analysis work products

Name	Description	Kind
System Description Specification	describes the technical requirements for a particular agent-oriented software	Structural
Goal Model	captures the purpose of the organization as a goal tree; includes goal attributes, precedence and triggering relationships	Behavioral
Domain Model	defines the language that can be used when defining the operation of the system	Structural
Organizational Model	documents the interaction between the organization and the external actors	Structural
Role Model	depicts organization roles, the goals they achieve and interactions between roles/external actors	Structural

terms of the kind of model, information, or data documented. A *structural* work product is used to model static aspects of the system, a *behavioral* work product is used to model dynamic aspects of the system, and a *composite* work product is used to model both static and dynamic aspects of the system. For further details on the differences between types of work products see [20].

System Description

There are many ways to capture and categorize requirements for use in systems. O-MaSE assumes that either traditional or multiagent-focused requirement gathering techniques are sufficient and thus does not stipulate a specific document structure.

Goal Model

The top-level CMS GMoDS goal model is shown in Fig. 13. (The initial version of the Goal Model from the Model Goals task is simply the GMoDS model with

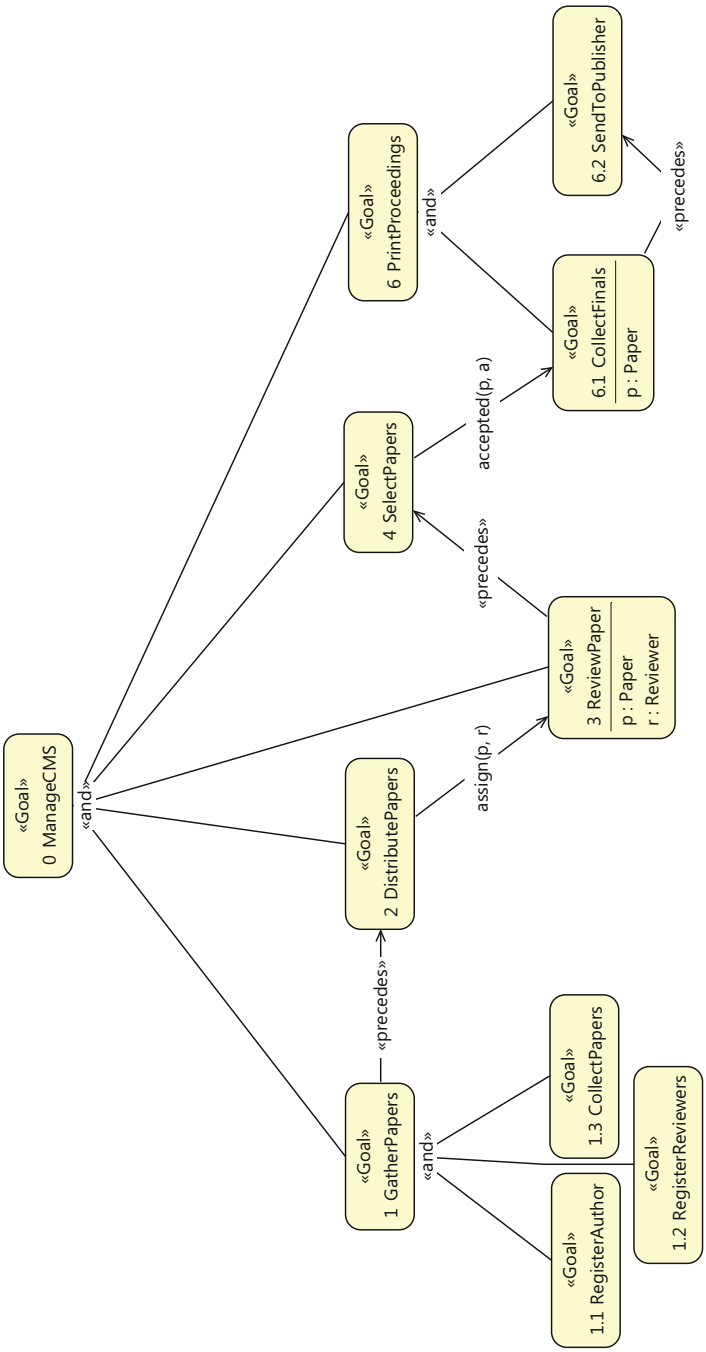

Fig. 13 Top-level GMoDS Goal Model for conference management system

the precede and trigger relations removed.) The overall goal of ManageCMS is broken down into five sub-goals: GatherPapers, DistributePapers, ReviewPaper, SelectPapers, and PrintProceedings. The GatherPapers goals is further decomposed into two sub-goals, RegisterAuthor and CollectPapers, while the PrintProceedings goal is also decomposed into the CollectFinals and SendToPublisher sub-goals. GMoDS allows new goal instances to be created via an *event trigger*, which is denoted by an arrow between two goals labeled by an event signature such as assign(p,r) on the arrow between the DistributePapers and ReviewPaper goals. In this case, it means that when an assign even occurs during pursuit of the DistributePapers goal, a new ReviewPaper goal is created. We use this to create a ReviewPaper goal for each reviewer assigned to review a given paper. Similarly, the accepted event during pursuit of the SelectPapers goal will create a new CollectFinals goal to collect each paper accepted for the proceedings.

The GMoDS precedence relation (denoted by an arrow labeled with «precedes») is used in the goal model to ensure proper sequence of actions in the system. Thus, since the papers must be gathered before they can be distributed for review, the GatherPapers goal precedes the DistributePapers goal (i.e., GatherPapers must be achieved before DistributePapers can begin to be pursued). Likewise, all the reviews must be performed (ReviewPapers goal) before papers can be selected (SelectPapers) for the conference, and all CollectFinals goals must be achieved before the SendToPublisher goal is attempted.

The PC organization is actually designed to achieve three top level goals: SendToPublisher, DistributePapers, and SelectPapers goals. The DistributePapers and SelectPapers goals are further decomposed as shown in Fig. 14 while the SendToPublisher is not.

The DistributePapers goal is decomposed in the AssignPapers and Disseminate-Papers goals. While the description of the CMS provides for several ways to assign papers, there is only a single goal that drives that assignment process. How the PC actually achieves the assignment goal is defined by the process they use. Therefore, this aspect of the CMS definition should be captured as a process, which in the case of O-MaSE would be captured as variations to a plan.

The SelectPapers goal is decomposed into several sub-goals: CollectReviews, MakeDecision, and InformAuthors. Since all the reviews should be collected prior to making a decision, a precedence relation exists between the CollectReviews and MakeDecision goals. As a decision is made on each paper, a *declined* or an *accepted* event triggers either an InformDeclined or an InformAccepted goal for that paper. In addition, an *accepted* event triggers a CollectFinals goal in the CMS goal model as shown in Fig. 13.

Domain Model

The Domain Model is an essential part of problem analysis and is very important to the O-MaSE approach in general. The Domain Model defines the language that can be used by designer to ensure that everyone is talking about this same thing. It is also essential in formally defining the operation of the system from the attributes in the goal model to the information passed via system protocols to the policies of the system. The Domain Model for the CMS is shown in Fig. 15.

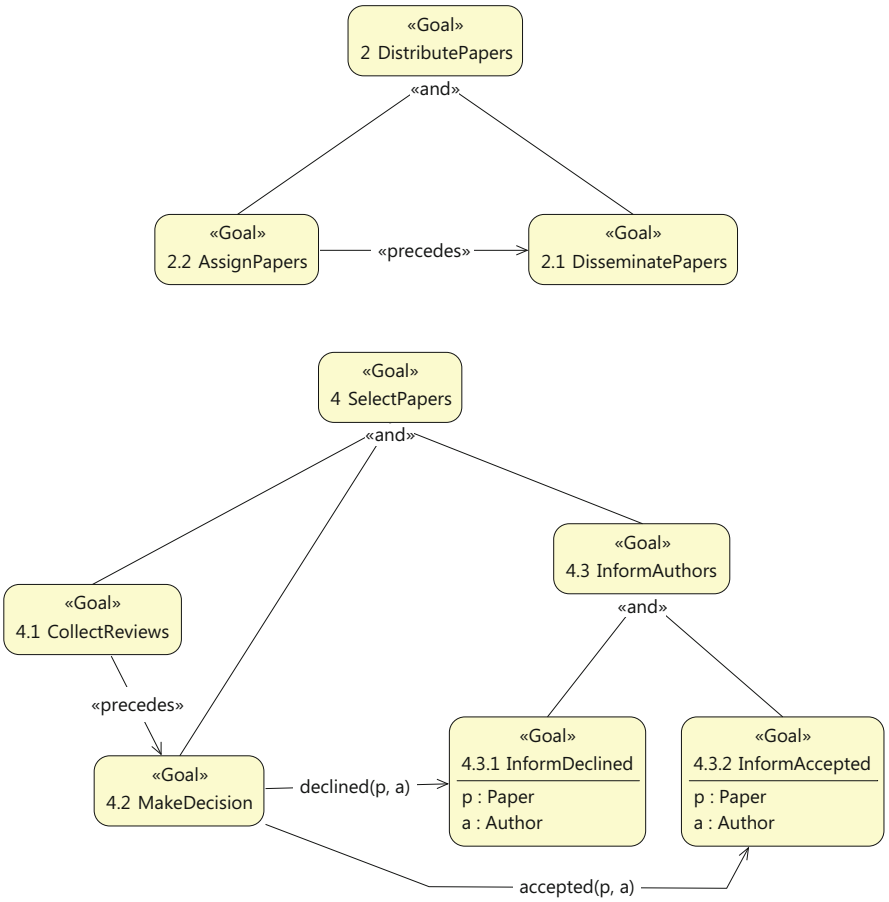

Fig. 14 Program committee Goal Model

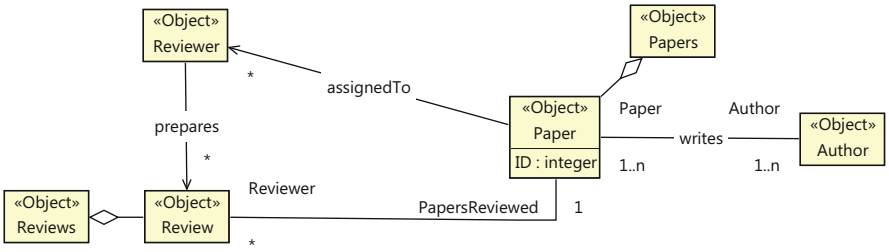

Fig. 15 Domain Model for conference management system

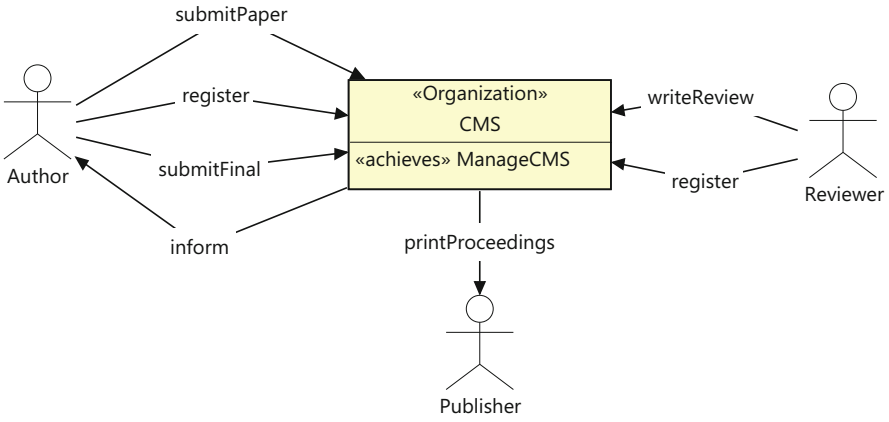

Fig. 16 Organization Model for conference management system

There are four main objects of interest in the CMS domain: Author, Paper, Reviewer, and Review. Each of these and their relationships are shown explicitly in the Domain Model. In addition, an aggregation of individual papers (Papers) and reviews (Reviews) is also defined. The model uses standard UML notation to define the multiplicities that are allowed between Objects. For instance, an Author must have at least one Author, and to be valid, an Author must have at least one Paper. It also shows that a Reviewer prepares a Review, and each Review has exactly one Paper that it can be written over. In addition, the Domain Model allows the definition of Object attributes. As shown, each Paper object has an ID.

Organizational Model

The Organization Model for the CMS is shown in Fig. 16. Here, the decision to make the PC a sub-organization shows up in the absence of the PC as an external actor. As shown, since the PC is a sub-organization, we are essentially considering it to be part of the system at this level. The three external actors shown, Author, Publisher, and Reviewer, all interact with the system through the given protocols. As the organization is refined into a Role and Agent Class Models, these external actors and protocols should show up in a consistent manner.

Role Model

The role model for the top-level CMS is shown in Fig. 17. There are six roles defined to carry out the goals defined in Fig. 13. Each role is designed to achieve a single goal as denoted by the «achieves» attribute in each role. Two exceptions are the PaperCollector role, which is designed to collect the initial and final copies of the papers, and the Registrar role, which is designed to register both authors and reviewers.

The role model also shows the external actors that interact with each of the roles. (As discussed above, the PC will actually be a sub-organization and thus the PC

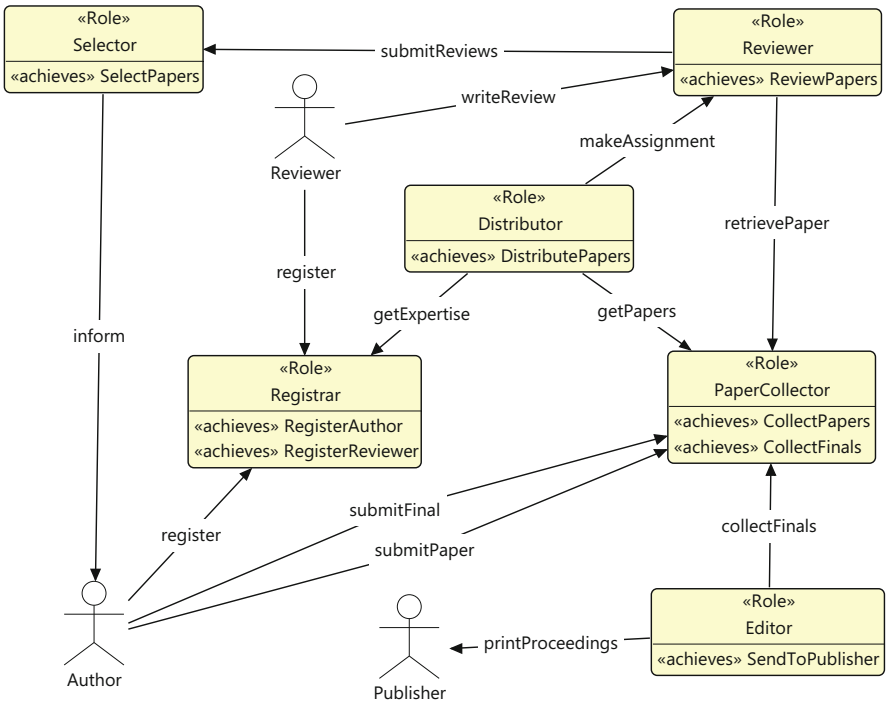

Fig. 17 Role Model for conference management system

Chair, Vice-Chairs, and PC Member external actors show up at a lower level of the design shown later.) The arrows between roles and between actors and roles represent protocols that support the passing of information. The direction of the arrow denotes who initiates the protocol and not the flow of information itself, which may occur in both directions. Each protocol is defined by a Protocol Model that shows the details of the messages and data passed. However, by studying both the goal model and role model, one can begin to understand the overall flow of the system.

The system begins with Author and Reviewer actors registering with the Registrar role and the Authors submitting papers to the PaperCollector role. Next, the Distributor role (which will eventually performed by the PC) gets the papers from the PaperCollector and the reviewer expertise from the Registrar and then assigns those papers to the reviewers. After all the Reviewers have submitted their reviews to the Selector, the Selector decides which papers are accepted and declined and informs the appropriate Authors. Once all the Authors have submitted the final version of their paper to the PaperCollector, the Editor takes those papers and sends them to the Publisher for publication.

The Role Model for the PC organization is shown in Fig. 18. Since the PC organization is designed to achieve three different CMS goals, the role model must

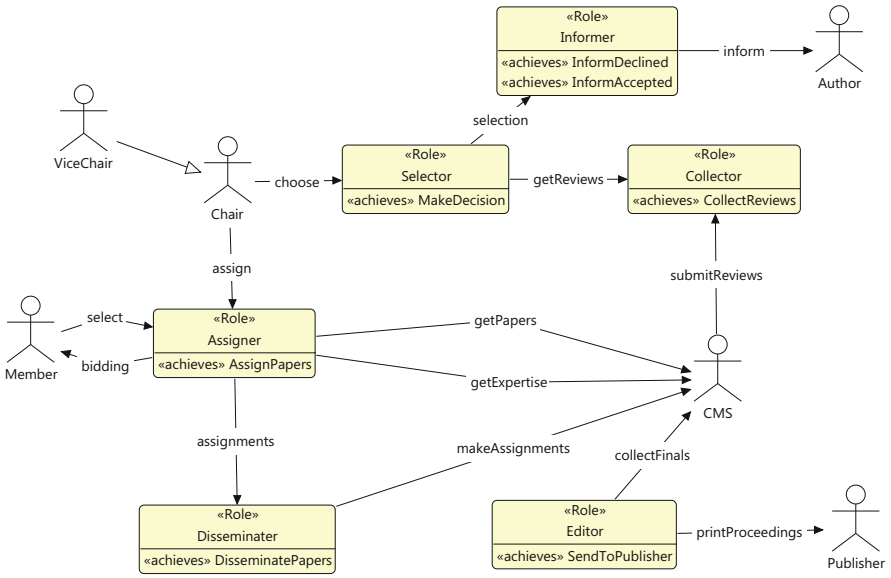

Fig. 18 Program Committee Role Model

accommodate all three. Notice that the actors of the PC organization include the PC Chair, Vice-chairs, and Members. In addition, the CMS is modeled as an external actor since its functionality lies outside the PC organization. As shown in the CMS Agent Class Model in Fig. 17, the PC organization must play the Editor, Distributor, and Selector roles defined in the CMS Role Model (Fig. 17).

The key to the correct decomposition of the PC organization is ensuring that the interfaces defined in the CMS Agent Class and Role models are consistently implemented in the PC organization. The easiest example of this is the CMS Editor role, which is implemented directly as a single Editor role in the PC organization. Notice that the protocols from the Editor to the Publisher and PaperCollector in the CMS Role Model are implemented as protocols to the CMS and Publisher actors in the PC Role Model. Thus, the CMS's PaperCollector role is captured as part of the CMS actor in the PC Role Model.

The CMS Distributor role is implemented as two separate roles in the PC organization: Assigner and Disseminater. The Assigner role is used to encapsulate the various approaches to assigning papers to PC Members as defined in the CMS description. While the approaches are not defined in the Role Model, the protocols required for the various approaches are. For instance, the PC Chair can assign them directly and thus assign protocol from the PC Chair to the Assigner role. The PC Members can also be involved in selecting papers or can be part of a bidding process; these options require separate protocols: select and bidding. The process can also be carried out automatically by the Assigner role. Once the assignments have been made the Assigner role sends the assignments to the Disseminater role

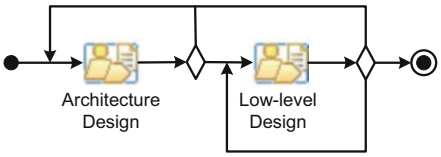

Fig. 19 Design-Phase flow of activities

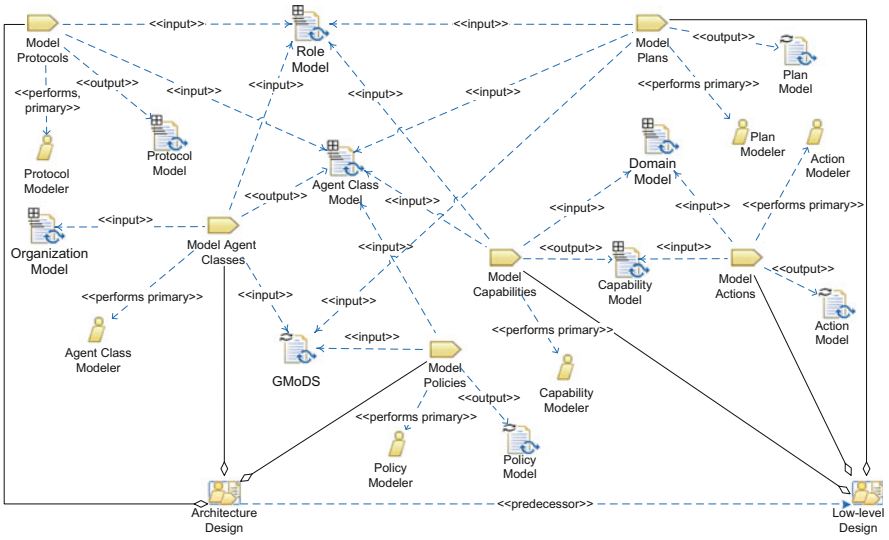

Fig. 20 Design Phase in terms of activities and work products

who is responsible for making assignments. These assignments are sent to the CMS actor, which includes the Reviewer role as defined in the CMS Role Model. We assume that if PC Members are Reviewers, they are registered in the system as both and are thus included as both an external PC Member actor and a CMS Reviewer.

2.2 Design

The design phase consists of two activities: Architecture Design and Low-Level Design. Once the goals, environment, behavior, and interactions of the system are known, Architecture Design is used to create a high-level description of the main system components and their interactions. This high-level description is then used to drive Low-Level Design, where the detailed specification of the internal agent behavior is defined. This low-level specification is then used to implement the individual agents during the Implementation phase. A generic example of an O-MaSE design phase is shown in Fig. 19 in terms of process flow and Fig. 20 in terms of process roles, work products, and tasks.

2.2.1 Process Roles

There are six roles in the design phase: Agent Class Modeler, Protocol Modeler, Policy Modeler, Capability Modeler, Plan Modeler, and Action Modeler.

Agent Class Modeler

The Agent Class Modeler is responsible for creating the Agent Class Model and requires general modeling skills and knowledge of the O-MaSE Agent Class Model specification.

Protocol Modeler

The Protocol Modeler designs the protocols required between agents, roles, and external actors and requires protocol modeling skills.

Policy Modeler

The Policy Modeler is responsible for designing the policies that govern the organization.

Capability Modeler

The Capability Modeler is responsible for defining the Capability Model and requires modeling skills and O-MaSE Capability Model specification knowledge.

Plan Modeler

The Plan Modeler designs the plans necessary to play a role; required skills include understanding of Finite State Automata and O-MaSE Plan Model specification knowledge.

Action Modeler

The Action Modeler documents the Action Model, which requires the ability to specify appropriate pre- and post-conditions for capability actions.

2.2.2 Activity Details

The Design phase has two activities: Architecture Design and Low-level Design. In the Architecture Design we focus on documenting the different agents, protocols, and policies using three tasks: Model Agent Classes, Model Protocols, and Model Policies. In the low-level design we focus on the capabilities possessed by, actions performed by, and plans followed by agents. The tasks of low-level design include Model Capabilities, Model Plans, and Model Actions.

Architecture Design

Architecture Design consists of three tasks as shown in Fig. 21. The Model Agent Classes task identifies the types of agents in the organization and the protocols between them. Agent classes may be defined by the roles they play or the capabilities they possess, which implicitly defines the roles they can play. Thus, an Agent Class provides a template for a type of agent in the system.

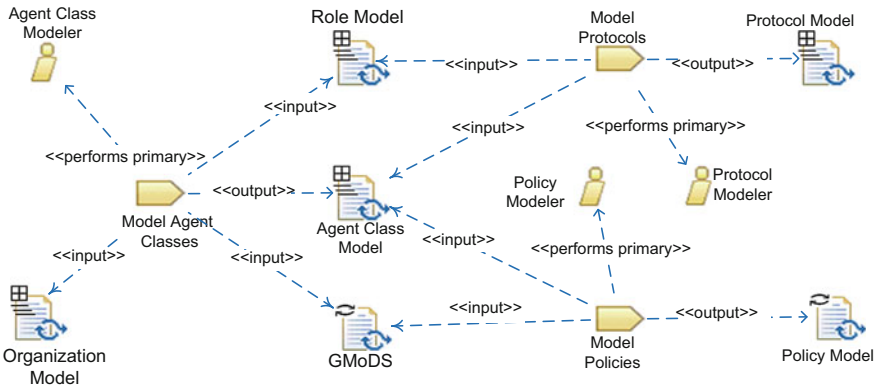

Fig. 21 Architecture Design activity diagram

The Model Protocols task specifies the details of the interactions between agents or roles. Since protocols can be specified in Organization Models, Role Models, and Agent Class Models, the method engineer may decide which set of protocols to define. If the Role Model protocols are defined via Protocol Models, agent classes playing those roles should inherit those protocols. The Protocol Model defines the types of messages sent between the two entities and is similar to UML interaction models [18].

The Model Policies task defines a set of rules that describe how an organization should behave. In general, policies are used to restrict agent behavior and may be enforced at design time or at runtime. How policies are enforced is a critical decision that affects the way the Policy Model is used during development. If there is no runtime mechanism designed or provided by the runtime environment, designs and implementations must be evaluated to ensure that they conform to the policies.

Low-Level Design

Low-level Design consists of three tasks as shown in Fig. 22. The Model Capabilities task defines the internal structure of the capabilities possessed by agents in the organization, which may be modeled as an Action or a Plan. An action is an atomic functionality possessed by an Agent and defined using the Model Actions task. A plan is an algorithmic definition of a capability and is defined using the Model Plans task.

The Model Plans task captures how an agent can achieve a specific type of goal using a set of actions specified as a Plan Model (a Finite State Machine). The Model Actions task defines the low-level actions used by agents to perform plans and achieve goals. Actions are typically defined as a function with a signature and a set of pre- and post-conditions. In some cases, actions may be modeled by providing detailed algorithmic information.

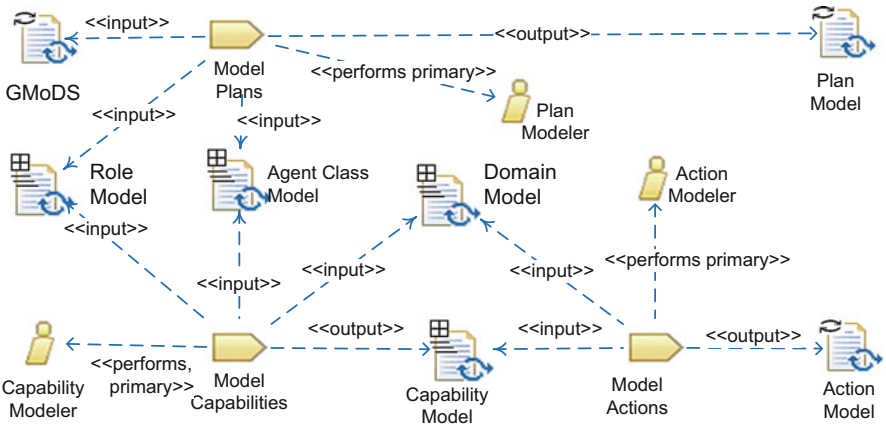

Fig. 22 Low-level Design activity diagram

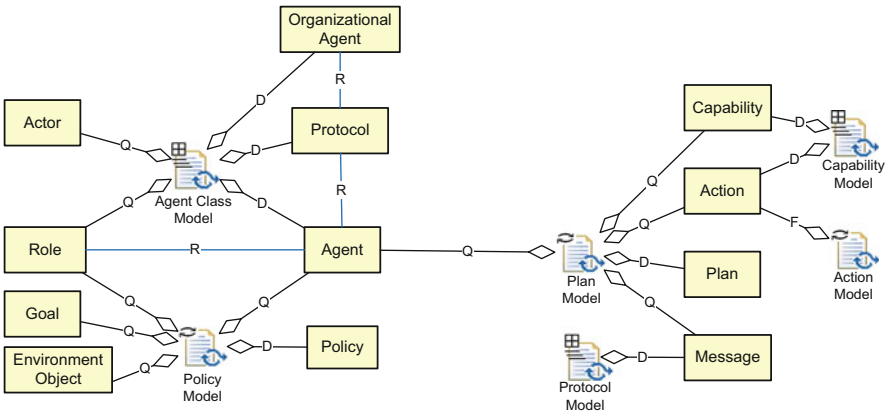

Fig. 23 Design document structure

2.2.3 Work Products

The Design phase produces six work products, each of which is a model that creates various elements in the O-MaSE metamodel. The relationships between these models and the O-MaSE metamodel are documented in Fig. 23. Each model is defined in terms of elements from the O-MaSE metamodel, which are represented with UML class icons. Within each model, each metamodel element may be Defined, reFined, Quoted, Related, or Relationship Quoted.

Work Product Kinds

There are six work products produced in the Design phase: Agent Class Model, Protocol Model, Policy model, Capability Model, Plan Model, and Action Model as defined in Table 5.

Table 5 Design work products

Name	Description	Kind
Agent Class Model	defines the agent classes and sub-organizations that will populate the organization.	Structural
Protocol Model	represents the different relations/interaction between external actors and agents/roles.	Structural
Policy Model	describes all the rules/constraints of the system	Behavioral
Capability Model	defines the internal structure of the capabilities possessed by agents in the organization.	Structural
Plan Model	captures how an agent can achieve a specific type of goal using a set of actions (which includes sending and receiving messages).	Behavioral
Action Model	defines the low-level actions used by agents to perform plans and achieve goals.	Behavioral

Agent Class Model

The Agent Class Model is shown in Fig. 24. There are three agents and one sub-organization (an organizational agent) defined to implement the six roles defined in the role model. The Database agent plays the PaperCollector role, the Registration agent plays the Registrar role, and the Reviewer agent plays the Reviewer role. The protocols defined in the role model are each inherited by the agent class model based on the roles assigned to the various agents.

A unique aspect of this design is the use of an organizational agent to capture the PC. In this design, the PC organization plays the Distributor, Selector, and Editor roles within the CMS. A further decomposition of this organizational agent is given below.

The Agent Class Model for the PC organization is shown in Fig. 25. The six roles from the PC Role Model result in four separate agents in the PC organization. Two agents simply implement single roles: the ReviewCollector agent plays the Collector role, while the Editor agent plays the Editor role. However, four other roles are combined into two agents. The reason for combining these roles is the fact that in both cases, there were two roles that communicated directly with each other and operated in a basically sequential manner. Therefore, the Disseminater and Assigner roles were combined into the PaperAssigner agent, while the Informer and Selector roles were combined into the PaperSelector role. The protocols and external agents were inherited directly from the Role Model, and no new protocols or external agents were added.

Protocol Model

The Model Protocols activity defines the internal details of each protocol identified in the Role and Agent Class models. At this point, all of the protocols in the CMS or the PC organization are modeled. One of the more interesting examples of a Protocol Model is shown in Fig. 26, which shows the *bidding* protocol between the Member actor and the Assigner agent. In reality, this protocol would likely be more

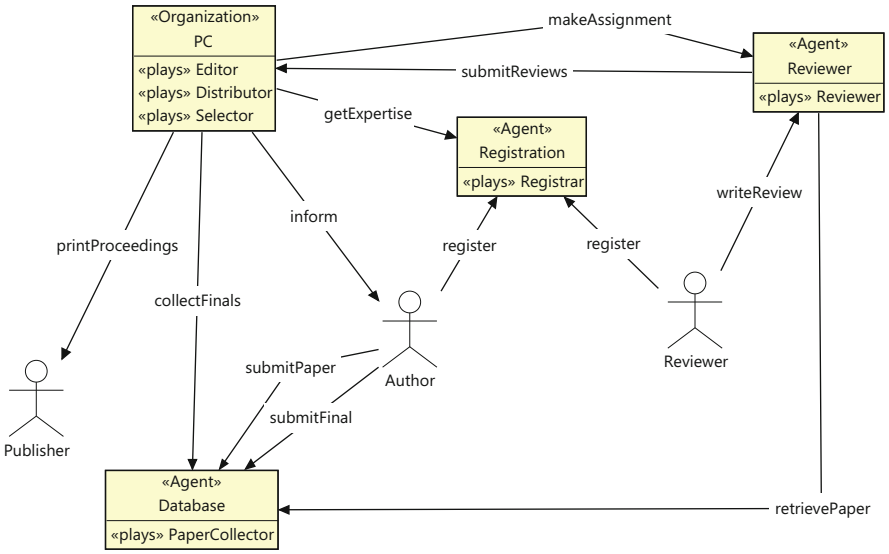

Fig. 24 Agent Class Model for conference management system

complex, possibly allowing the member to change bids or to protest an assignment; however, for the purposes of this example, we will take a simple approach. When the PC distributes papers via a bidding process, this protocol is used between the Assigner role and each Member of the PC committee. The protocol only captures the interactions between the Assigner and one Member—this protocol is repeated for each member. First, the PC Assigner role sends the *callForBids* message to the Member with a list of available papers. Next, the Member decides what bid to place on each of the papers in the set of papers received. For each paper the Member would like to bid on, a *bid* message is sent back to the Assigner. When the Member has completed bidding on papers, the Member sends a *done* message to the Assigner. Once the Assigner receives all the bids from all the Members, the Assigner decides the final assignments for each Member and a *assignment* message is sent to the Member and the protocol is complete.

Policy Model

As defined in [13], we use a language that includes temporal formula with quantification. For simplicity, we limit our examples to first-order predicate logic in this example. The language used to specify policies comes from entities defined in the various models, for example, objects defined in the Domain Model, the Roles defined in the Role Model, and the Agents defined in the Agent Class Model. While most of the interesting policies apply the PC organization, we do want each paper to have a unique ID. Thus, a policy to ensure that each paper has a unique ID can be stated as

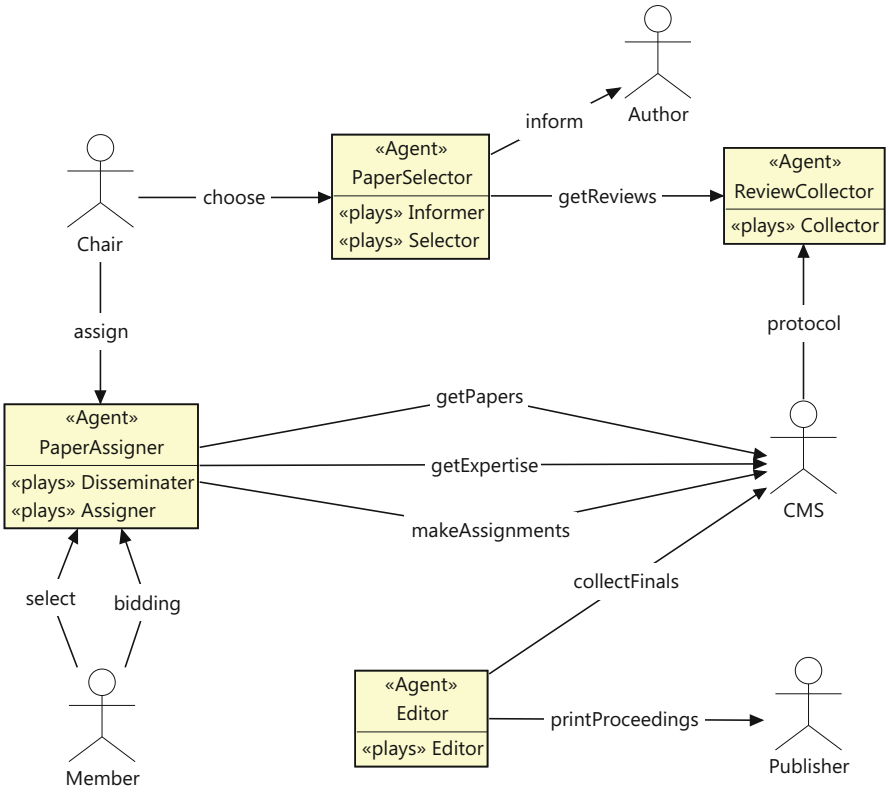

Fig. 25 Program Committee Agent Class Model

$$\forall p1, p2 : \mathit{Paper}\ p1.\mathit{ID} = p2.\mathit{ID} \Rightarrow p1 = p2$$

The CMS requires that a PC Member or Reviewer may not see or infer information about their own submissions. Essentially, this requires that PC Members would not be able to see any reviews or decisions made about their papers except through the normal inform protocol with Authors. We assume here that the design of the system only allows a PC Member (not the Chair) to see submitted reviews through the submittReview protocol with the Collector role. We also assume that the Member can only see reviews related to papers that the member has submitted a review for. Thus to specify that a member cannot review their own paper and that a Member may not view reviews related to their papers, we can specify the following policies:

$$\forall p : \mathit{Paper}, m : \mathit{Member}, a : \mathit{Author}\ a = m \wedge \mathit{writes}(a, p) \Rightarrow m\neg \in \mathit{assignedTo}(p, m)$$

$$\forall p : \mathit{Paper}, m : \mathit{Member}\ \mathit{submitReview}(m, p) \Rightarrow m \in \mathit{assignedTo}(p, m)$$

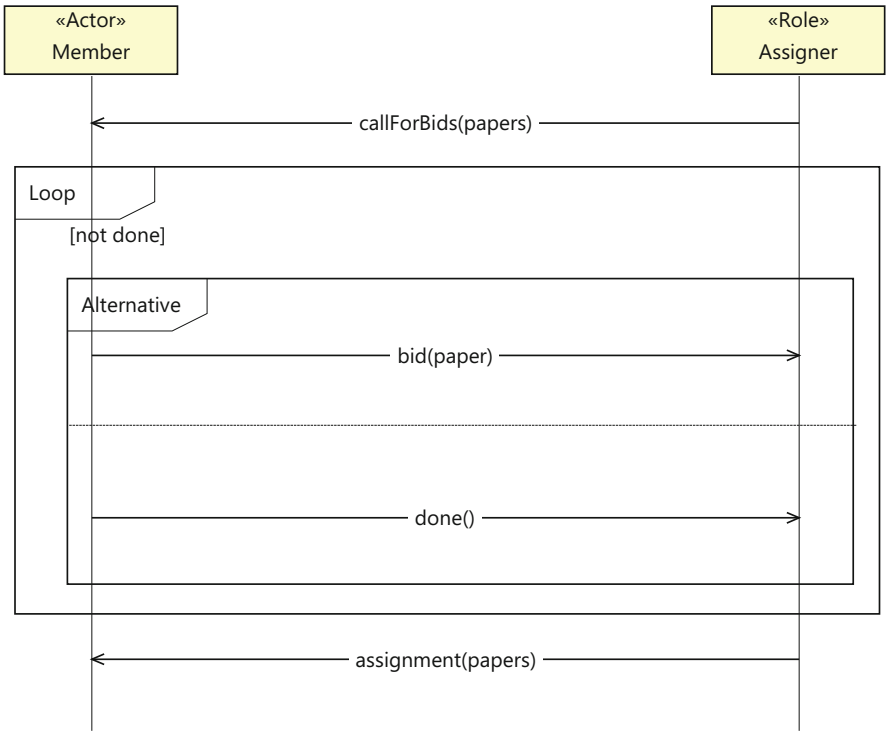

Fig. 26 Program Committee bidding protocol

The first policy states that if a Member is also an Author and writes a paper, that Member cannot be in the set of Reviewers in the assignedTo association with that paper. The second policy states that if a Member submits a review of a paper, then that Member must be in the set of assigned reviewers in the assignedTo association with that paper.

Capability Model

Capability Model captures the internal structure of the capabilities possessed by agents in the organization. Each capability may be modeled as an Action or a Plan. An action is an atomic functionality possessed by an Agent and defined using an Action Model as described in section "Action Model". A plan is an algorithmic definition (defined via a state machine) of a capability that uses actions and implements protocols. Each plan is defined using a Plan Model as presented in section "Plan Model".

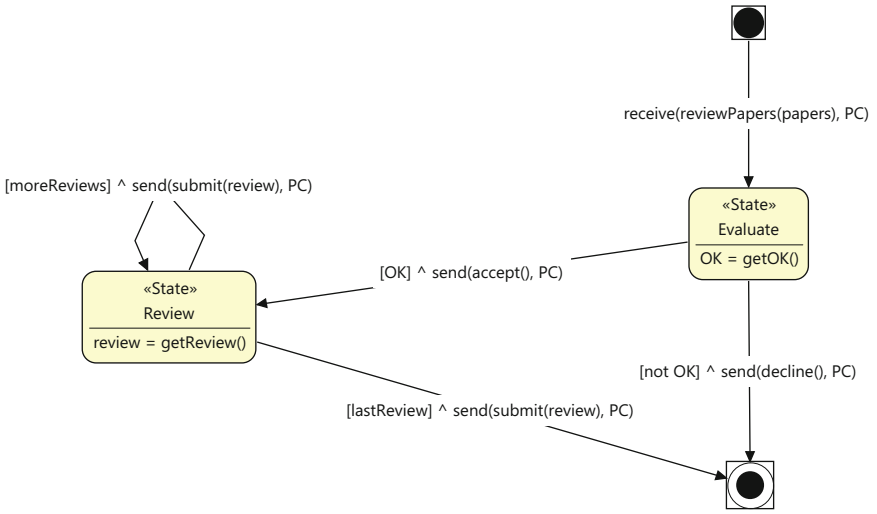

Fig. 27 Program Committee bidding protocol

Plan Model

Typically, a plan is required for each type of goal that an agent can achieve. Thus, since agents are defined by the roles they play, one must look at the goals that can be achieved by each role the agent can play. We illustrate this process with the Reviewer agent. Since the Agent Class Model in Fig. 24 defines that Reviewer agent can only play the Reviewer role, we only need to look at the goals that can be achieved by the Reviewer role. In the Role Model of Fig. 17 we can see that the Reviewer role is designed to only achieve the ReviewPapers goal. Therefore, we only need to define a single plan to fully define the behavior of the Reviewer agent.

The ReviewPapers plan for the Review agent is shown in Fig. 27. It is defined by a simple-state machine that starts when the agent receives a *reviewPapers* message from the PC organization (denoted by the label on the transition from the start state to the Evaluate state). The Reviewer has the right to accept or reject the papers presented. If accepted, the Reviewer sends an *accept* message to the PC and enters the Review state. Here the Reviewer agent waits for the actual human reviewer to enter reviews. As each review is received, the Reviewer sends the review to the PC via a *review* message. When all reviews have been received, the Review plans end.

Action Model

The Action Model defines the low-level actions used by agents to perform plans and achieve goals. Actions belong to capabilities possessed by agents. Actions are typically defined as a function with a signature and a set of pre- and post-conditions. In some cases, actions may be modeled by providing detailed algorithmic information. If using automatic code generation techniques, this information is generally captured as a function or an operation in the language being generated. In either case, the Action Model is usually just a textual document.

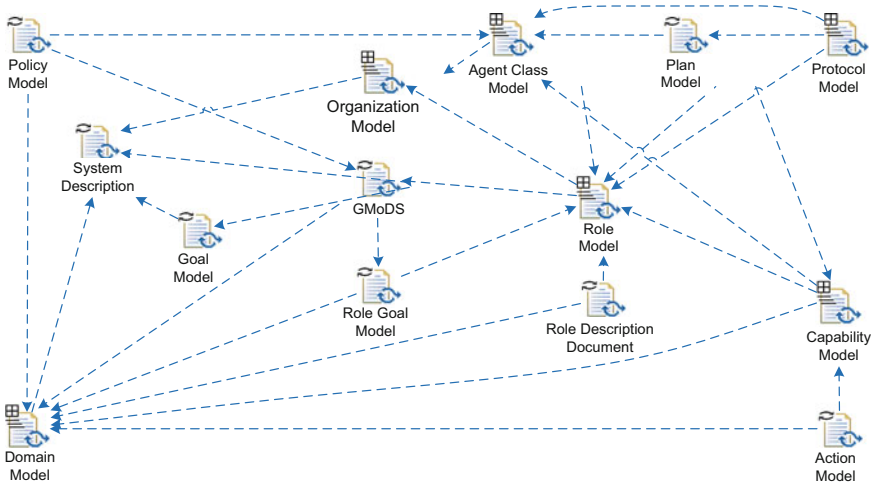

Fig. 28 Work Product Dependencies

2.3 Implementation

Finally, the design is translated to code. The purpose of this phase is to take all the design models created during the design and convert them into code that correctly implements the models. Obviously, there are numerous approaches to code generation based on the runtime platform and implementation language chosen. In this phase there is a single Role, the Programmer who is responsible for writing code based on the various models produced during the Design phase. The output of the Generate Code task is the source code of the application. While not currently covered in the process, system creation ends with testing, evaluation, and deployment of the systems.

3 Work Product Dependencies

Figure 28 identifies the dependencies between all the work products in O-MaSE. These dependencies characterize different pieces of information produced during the different stages of the development process and serve as inputs to and outputs of work units (i.e., either activities or tasks).

References

1. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Inf. Softw. Technol. **38**, 275–280 (1996)
2. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardization to research. Int. J. Agent Oriented Softw. Eng. **1**, 91–121 (2007)

3. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: ASPECS: an agent-oriented software process for engineering complex systems. J. Auton. Agent Multi Agent Syst. **20**, 260–304 (2009)
4. DeLoach, S.A., Garcia-Ojeda, J.C.: O-MaSE: a customizable approach to designing and building complex, adaptive multiagent systems. Int. J. Agent Oriented Softw. Eng. **4**, 244–280 (2010)
5. DeLoach, S.A., Miller, M.: A goal model for adaptive complex systems. Int. J. Comput. Intell. Theory Pract. **5**, 83–92 (2010)
6. DeLoach, S.A., Valenzuela Jorge, L.: An agent-environment interaction model. In: Padgham, L., Zambonelli, F. (eds.) Agent-Oriented Software Engineering VII: 7th International Workshop, AOSE 2006. Lecture Notes in Computer Science, vol. 4405, pp. 1–18. Springer, Heidelberg (2006)
7. DeLoach, S.A., Oyenan, W., Matson, E.T.: A capabilities based model for artificial organizations. Auton. Agent Multi Agent Syst. **16**, 13–56 (2008)
8. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Proceedings of the 3rd International Conference on Multi Agent Systems, pp. 128–135. IEEE Computer Society, Washington (1998)
9. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Muller, J.P., Odell, J. (eds.) Agent-Oriented Software Engineering IV. Lecture Notes in Computer Science, vol. 2935, pp. 214–230. Springer, Berlin (2003)
10. Garcia-Ojeda, J.C., DeLoach, S.A.: Robby: agentTool process editor: supporting the design of tailored agent-based processes. In: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09), pp. 707–714. ACM, New York (2009)
11. Garcia-Ojeda, J.C., DeLoach, S.A.: Robby: agentTool III: from process definition to code generation. In: Decker, K., Sichman, J., Sierra, G., Castelfranchi, C. (eds.) Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), vol. 2, pp. 1393–1394. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2009)
12. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyenan, W.H., Valenzuela, J.: O-MaSE: a customizable approach to developing multiagent development processes. In: Luck, M., Padgham, L. (eds.) Proceedings of the 8th International Conference on Agent-oriented Software Engineering VIII (AOSE'07). Lecture Notes in Computer Science, vol. 4951, pp. 1–15. Springer, Berlin (2007)
13. Harmon, S., DeLoach, S.A., Robby: trace-based specification of law and guidance policies for multiagent systems. In: Artikis, A., O'Hare, G.M., Stathis, K., Vouros, G. (eds.) Engineering Societies in the Agents World VIII. Lecture Notes in Artificial Intelligence, vol. 4995, pp. 333–349. Springer, Berlin (2008)
14. Holland, O., Melhuish, C.: Sigmergy, self-organization, and sorting in collective robotics. Artif. Life **5**, 173–202 (1999)
15. Object Management Group. Software and systems process engineering meta-Model specification, v2.0. Object Management Group. http://www.omg.org/spec/SPEM/2.0/PDF (2008). Accessed 14 May 2012
16. Odell, J., Parunak, H., Bauer, B.: Representing agent interaction protocols in UML. In: Wooldridge, M.J., Ciancarini, P. (eds.) First International Workshop, AOSE 2000 on Agent-oriented Software Engineering, pp. 121–140. Springer, New York (2001)
17. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. In: Odell, J. Giorgini, P., Muller, J. (eds.) Proceedings of the 5th International Conference on Agent-Oriented Software Engineering (AOSE'04). Lecture Notes in Computer Science, vol. 3382, pp. 78–92. Springer, Berlin (2005)
18. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Upper Saddle River (2004)

19. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Upper Saddle River (2002)
20. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: Proceedings of the 7th Workshop from Objects to Agents (WOA 2006), pp. 130–137 (2006)