O-MaSE: A Customizable Approach to Developing Multiagent Development Processes[1]

Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby,
Walamitien H. Oyenan and Jorge Valenzuela

Department of Computing and Information Sciences, Kansas State University,
234 Nichols Hall, Manhattan, Kansas, USA.
{jgarciao,sdeloach,robby,oyenan,jvalenzu}@ksu.edu

Abstract. This paper describes the Organization-based Multiagent System Engineering (O-MaSE) Process Framework, which helps process engineers define custom multiagent systems development processes. O-MaSE builds off the MaSE methodology and is adapted from the OPEN Process Framework (OPF). OPF implements a Method Engineering approach to process construction. 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 methods 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 operations or tasks that are executed to produce a set of work products, which may include models, documents, or code. The guidelines define how the method fragments are related to one another. The paper also demonstrates two examples of creating custom O-MaSE processes.

1. Introduction

The software industry is facing new challenges. Businesses today are demanding applications that can operate autonomously, can adapt in response to dynamic environments, and can interact with other applications in order to provide comprehensive solutions. Multiagent system (MAS) technology is a promising approach to these new requirements [13]. Its central notion – the intelligent agent – encapsulates all the characteristics (i.e., autonomy, proactive, reactivity, and interactivity) required to fulfill the requirements demanded by these new applications.

In order to develop these autonomous and adaptive systems, novel approaches are needed. In the last several years, many new processes for developing MAS have been proposed [1]; unfortunately, none of these processes have gained widespread industrial acceptance. Reasons for this lack of acceptance include the variety of approaches upon which these processes are based (i.e., object-oriented, requirements engineering, and knowledge engineering) and the lack of Computer Aided Software Engineering (CASE) tools that support the process of software design. There have

[1] This work was supported by grants from the US National Science Foundation (0347545) and the US Air Force Office of Scientific Research (FA9550-06-1-0058)

been some approaches suggested for increasing the change of industry acceptance. For instance, Odell *et al*. suggest presenting new techniques as an incremental extension of known and trusted methods [14], while Bernon *et al*. suggest the integration of existing agent-oriented processes into one highly defined process [3]. Although these suggestions may be helpful in gaining industrial acceptance of agent-oriented techniques, we believe that a more promising way is to provide more flexibility in the approaches offered. The main problem with these approaches is that they do not provide assistance to process engineers on how to extend or tailor these processes. In this vein, Henderson-Sellers suggests the use of method engineering using a well-defined and accepted metamodel in order to allow users to construct and to customize their own processes that fit their particular approaches to systems development [11]. Henderson-Sellers argues that by defining method fragments based on a common underlying metamodel, new custom processes can be created that support user defined goals and preferences.

The goal of this paper is to present an overview of the Organization-based Multiagent System Engineering (O-MaSE) Process Framework. The goal of the O-MaSE Process Framework is to allow process engineers to construct custom agent-oriented processes using a set of method fragments, all of which are based on a common metamodel. To achieve this, we define O-MaSE in terms of a metamodel, a set of method fragments, and a set of guidelines. The O-MaSE *metamodel* defines a set of analysis, design, and implementation concepts and a set of constraints between them. The *method fragments* define how a set of analysis and design products may be created and used within O-MaSE. Finally, *guidelines* define how the method fragment may be combined to create valid O-MaSE processes, which we refer to as O-MaSE compliant processes.

The rest of the paper is organized as follows. Section 2 discusses the background material on O-MaSE. Section 3 presents a brief overview of the O-MaSE Process Framework as defined by the proposed metamodel, method fragments, and guidelines. Section 4 presents examples of two O-MaSE-compliant processes that can be used for developing a simulated cooperative robotic system. Finally, Section 5 concludes and describes future work.

2. Background

One of the major problems faced by agent-oriented software engineering is the failure to achieve a strong industry acceptance. One of the reasons hindering this acceptance is a lack of an accepted process-oriented methodology for developing agent-based systems. An interesting solution to this problem is the use of approaches that allow us to customize processes based on different types of applications and development environments. One technique that provides such a rational approach for the construction of tailored methods is Method Engineering [5].

Method Engineering is an approach by which process engineers construct processes (i.e., methodologies) from a set of method fragments instead of trying to modify a single monolithic, “one-size-fits-all” process. These fragments are generally identified by analyzing these “one-size-fits-all” processes and extracting useful tasks and techniques. The fragments are then redefined in terms of a common metamodel

and are stored in a repository for later use. To create a new process, a process engineer selects appropriate method fragments from the repository and assembles them into a complete process based on project requirements [5].

However, the application of Method Engineering in the development of agent-oriented applications is non-trivial. Specifically, there is no consensus on the common elements of multiagent systems. Thus, it is has been suggested that prior to developing a set of method fragments, a well-defined metamodel of common agent-oriented that are typical of most varieties of MAS (e.g., adaptive, competitive, self-organizing, etc.) should be developed [4].

Fortunately, we can leverage the OPEN Process Framework (OPF), which provides an industry-standard approach for applying Method Engineering to the production of custom processes [9]. The OPF uses an integrated metamodel-based framework that allows designers to select method fragments from a repository and to construct a custom process using identified construction and tailoring guidelines. This metamodel-based framework is supported by a three-layer schema as shown in Fig. 1. The M2 layer includes the OPF metamodel, which is a generic process metamodel defining the types of method fragments that can be used in M1. Thus a process (such as OPEN) can be created in M1 by instantiating method fragments from the M2 metamodel.

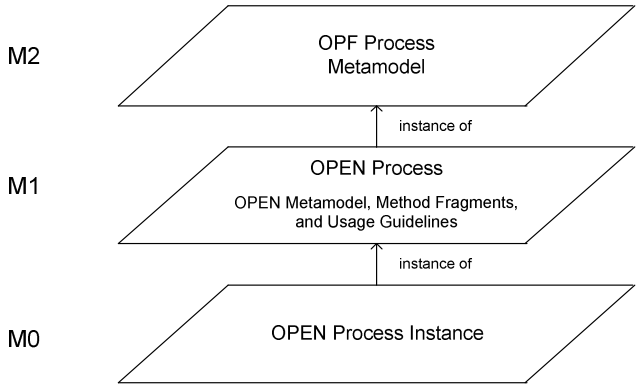

Fig. 1. OPEN Process Framework (adapted from [12])

The OPF metamodel consists of Stages, Work Units (Activities, Tasks, and Techniques), Producers, Work Products, and Languages. A *Stage* is defined as a "formally identified and managed duration within the process or a point in time at which some achievement is recognized" [9, pp. 55]. Stages are used to organize *Work Units*, which are defined as operations that are carried out by a *Producer*. There are three kinds of Work Units in OPF: Activities, Tasks, and Techniques. *Activities* are a collection of Tasks. *Tasks* are small jobs performed by one or more Producers. *Techniques* are detailed approaches to carrying out various Tasks. *Producers* use Techniques to create, evaluate, iterate, and maintain Work Products. *Work Products* are pieces of information or physical entities produced (i.e., application, document, model, diagram, or code) and serve as the inputs to and the outputs of Work Units. Work Products are documented in appropriate *Languages*.

The M1 layer serves as a repository of method fragments instantiated from the M2 metamodel. A set of rules governing the relationship between these concepts (i.e., a process-specific metamodel and a set of reusable method fragments) is also defined in M1. Basically, the process engineer uses the guidelines to extend, to instantiate, and to tailor the predefined method fragments for creating a *custom process* in the M1 layer. These custom processes are then instantiated at the M0 level on specific projects; the actual custom process as enacted on a specific project is termed a *process instance*.

Alternatively, the FIPA (Foundation for Physical Agents) Technical Committee (TC) methodology group[2] is working on defining reusable method fragments in order to allow designers to specify custom agent-oriented processes [17]. Although this approach is quite similar to OPF (they are both based on method engineering), its metamodel is derived from the Object Management Group (OMG) Software Process Engineering Metamodel[3] (SPEM). SPEM is based on three basic *process elements* that encapsulate the main features of any development process: Activities, Process Roles, and Work Products. Development processes are assembled from a set of SPEM *Activities*, which represent tasks that must be done. An Activity is essentially equivalent to an OPF Work Unit and is performed by one or more *Process Roles* (which corresponds to OPF Producers). Process Roles carry out the Activities in order to produce *Work Products* (the same term is used here by SPEM and OPF). A detailed description of this metamodel and a comparison with other method fragment proposals can be found in [6]. The next section focuses on using Method Engineering and the OPF metamodel to specify O-MaSE.

3. O-MaSE Process Framework

In this section, we define the O-MaSE Process Framework as shown in Fig. 2, which is analogous to the OPF from Fig. 1. In fact, we use the OPF metamodel in level M2. Level M1 contains the definition of O-MaSE in the form of the O-MaSE metamodel, method fragments, and guidelines. In the remainder of the section, we present the three components of the O-MaSE contained in the M1. We first describe the O-MaSE metamodel followed by a description of the method fragments obtained. Finally, we discuss the guidelines that govern the construction of O-MaSE compliant processes.

3.1 Metamodel

The O-MaSE metamodel defines the main concepts we use to define multiagent systems. It encapsulates the rules (grammar) of the notation and depicts those graphically using object-oriented concepts such as classes and relationships [9]. The O-MaSE metamodel is based on an organizational approach [7, 8]. As shown in Fig. 3, the *Organization* is composed of five entities: Goals, Roles, Agents, Domain Model, and Policies. A *Goal* defines the overall function of the organization and a

[2] See http://www.fipa.org/activities/methodology.html
[3] See http://www.omg.org/cgi-bin/doc?formal/2005-01-06

Role defines a position within an organization whose behavior is expected to *achieve* a particular goal or set of goals.

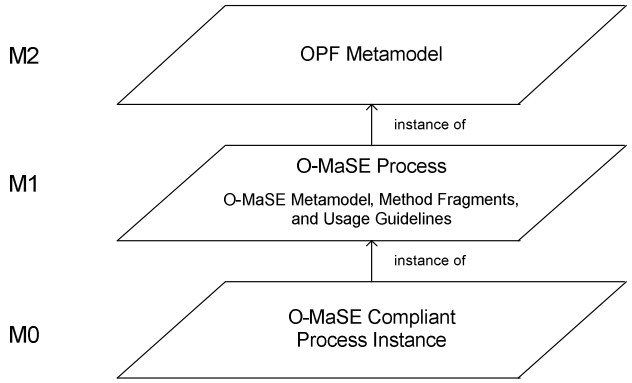

Fig. 2. O-MaSE Process Framework (adapted from [12])

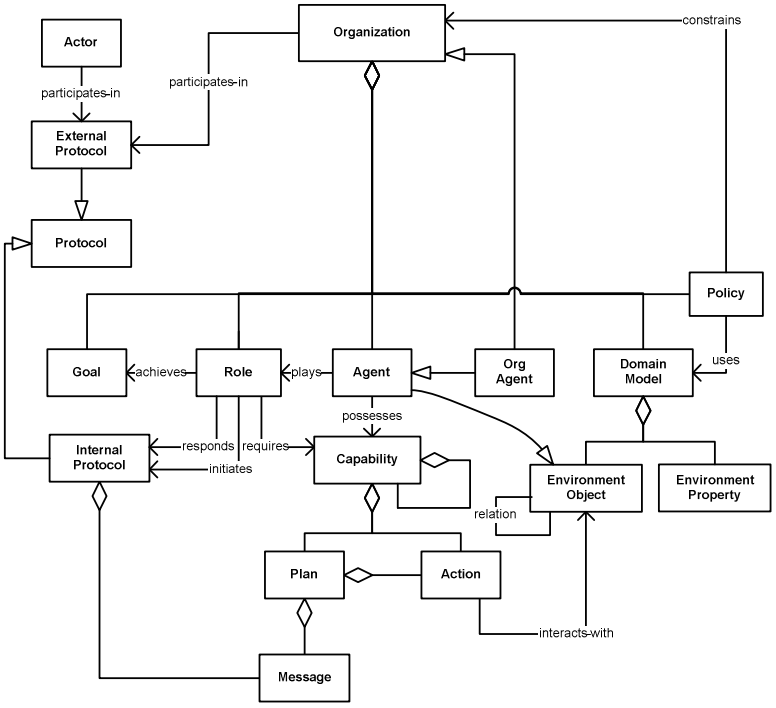

Fig. 3. O-MaSE metamodel (adapted from [8])

Agents are human or artificial (hardware or software) entities that perceive their environment and can perform actions upon it. In order to perceive and to act in an environment, agents possess *Capabilities*, which define the percepts/actions the agents have at their disposal. Capabilities can be soft (i.e., algorithms or plans) or hard (i.e., hardware related actions). *Plans* capture algorithms that agents use to carry out specific tasks, while *Actions* allows agents to perceive or sense objects in the environment. This environment is modeled using the *Domain Model*, which defines the types of objects in the environment and the relations between them. Each organization is governed by rules, which are formally captured as Policies. A *Policy* describes how an organization may or not may behave in a particular situation.

Table 1. O-MaSE Method Fragments

Work Units					
Activity	**Task**	**Technique**	**Work Products**	**Producer**	**Language**
Requirement Engineering	Model Goals	AND/OR Decomposition	AND/OR Goal Tree	Goal Modeler	Natural languages, for textual documents UML, for specific models Agent-UML O-MaSE specific notation Formal Language, for formal specification of properties of the system.
	Goal Refinement	Attribute-Precede-Trigger Analysis	Refined GMoDS		
Analysis	Model Organizational Interfaces	Organizational Modeling	Organization Model	Organizational Modeler	
	Model Roles	Role Modeling	Role Model	Role Modeler	
	Define Roles	Role Description	Role Description Document		
	Model Domain	Traditional UML notation	Domain Model	Domain Expert	
Design	Model Agent Classes	Agent Modeling	Agent Class Model	Agent Class Modeler	
	Model Protocol	Protocol Modeling	Protocol Model	Protocol Modeler	
	Model Plan	Plan Specification	Agent Plan Model	Plan Modeler	
	Model Policies	Policy Specification	Policy Model	Policy Modeler	
	Model Capabilities	Capability Modeling	Capabilities Model	Capabilities Modeler	
	Model Actions	Action Modeling	Action Model	Action Modeler	
	Model Service	Service Modeling	Service Model	Service Modeler	

3.2 Method Fragments

As mentioned above, the OPF metamodel defines Stages, Work Units, Work Products, Producers, and Languages, which are used to construct tailorable processes. In our work, the initial set of method fragments are derived from an extended version of the MaSE methodology [5]. O-MaSE assumes an iterative cycle across all phases with the intent that successive iterations will add detail to the models until a complete design is produced. This nicely fits the OPF's *Iterative, Incremental, Parallel Life*

Cycle model). Our current work focuses on analysis and design. In O-MaSE, we have identified three main activities: (1) requirements engineering, (2) analysis, and (3) design. As shown in Table 1, we decompose each Activity into a set of Tasks and identify a set of Techniques that can be used to accomplish each Task. We also show the different Work Products, Producers, and Languages related to the associated Work Units. Due to the page limitations, we cannot discuss each of these separately[4]. However, to illustrate our basic approach, we describe the details of the requirements engineering activity.

In the Requirement Engineering activity, we seek to translate systems requirement into system level goals by defining two tasks: *Model Goals* and *Goal Refinement*. The first focuses on transforming system requirements into a system level goal tree while the second refines the relationships and attributes for the goals. The goal tree is captured as a Goal Model for Dynamic Systems (GMoDS) [7]. The *Goal Modeler* must be able to: (1) use AND/OR Decomposition and Attribute-Precede-Trigger Analysis (APT) techniques, (2) understand the System Description (SD) or Systems Requirement Specification (SRS), and (3) interact with domain experts and customers. The result of these two tasks are an AND/OR Goal Tree and GMoDS tree.

3.3 Guidelines

Guidelines are used to describe how the method fragments can be combined in order to obtain O-MaSE compliant processes. These guidelines are specified in terms of a set of constraints related to Work Units and Work Products, which are specified as Work Unit preconditions and postconditions. We formally specify these guidelines as a tuple ⟨*Input*, *Output*, *Precondition*, *Postcondition*⟩ where *Input* is a set of Work Products that may be used in performing a work unit, *Output* is a set of Work Products that may be produced from the Work Unit, *Precondition* specifies valid Work Product/Producer states, and *Postcondition* specifies the Work Product State (see Table 1) that is guaranteed to be true after successfully performing a work unit (if the precondition was true). To formally specify pre and postconditions, we use first order predicate logic statements defined over the Work Products (WP) and Producers (P), the Work Products states, and the iteration (*n*) and version (*m*) of the Work Products.

Table 2. Work Product States

No.	*State*	*Definition*
1	inProcess()	True if the work product is in process.
2	completed()	True if the work product has been finished.
3	exists()	exists() = inProcess() ∨ completed()
4	previousIteration()	True if the work product's iteration is any previous one.
5	available()	This state applies to producers and not to work products.

Figs. 4 – 8 illustrate a set of guidelines for a few of the Tasks defined in Table 1. Fig. 4 defines the *Model Goals* task. Inputs to the task may include the *Systems Description* (SD), the *Systems Requirement Specification* (SRS), the *Role Description*

[4] A detailed description of the current set of O-MaSE Tasks, Techniques, Work Products, and Producers can be found at http://macr.cis.ksu.edu/O-MaSE/

Document (RD), or a previous version of the *Goal Model* (GM). Actually, only one of these inputs is required, although as many as are available may be used. The inputs are used by the *Goal Model Producer* (GMP) to identify organization goals. As a result of this task, the Work Product *GM* is obtained.

TASK NAME: Model Goals			
Input	*Output*	*Precondition*	*Postcondition*
SD,SRS, RD,GM	GM	((exists(<SD,n,m>) ∨ exists(<SRS,n,m>) ∨ exists(<RD,n,m>) ∨ previousIteration(<GM>)) ∧ available(GMP)	completed(<GM,n,m>)

Fig. 4. Model Goal Task Constrains

Fig. 5 depicts the task *Goal Refinement*. Generally, this task only requires as input a GM from the Model Goals task and produces a refined GMoDS model.

TASK NAME: Goal Refinement			
Input	*Output*	*Precondition*	*Postcondition*
GM	RG	Completed(<GM,n,m>) ∧ available(GMP)	exists(<RG,n,m>)

Fig. 5. Goal Refinement Task Constrains

Fig. 6 shows the task Model Agent Classes, which requires as input a *Refined Goal Model* (RG), an *Organization Model* (OM), or a *Role Model* (RM). As output an *Agent Class Model* (AC) is obtained. In the task, the *Agent Class Modeler* (ACM) identifies the types of agents in the system. A *Capability Model* (CM) may also be used as input because agents may be defined in terms of capabilities. However, the CM is never sufficient or mandatory and thus is termed as an *optional* input (it is not part of the Precondition). The *Protocol Model* (PrM) may be useful in identifying relationships between agents and thus, it is also optional.

TASK NAME: Model Agents Classes			
Input	*Output*	*Precondition*	*Postcondition*
RG,RM, OM,AC, CM,PrM	AC	(exists(<RG,n,m>) ∨ exists(<RM,n,m>) ∨ exists(<OM,n,m>) ∨ exists(<SM,n,m>) ∨ previousIteration(<AC>)) ∧ available(ACM)	completed(<AC,n,m>)

Fig. 6. Model Agent Classes Task Constrains

The *Model Plan* task is defined in Fig. 7. The inputs can include a RG, RM, or an AC, which allow the *Plan Modeler* (PlM) to define plans used by agents to satisfy organization goals. In addition, a PrM, Action Model (AM), and CM are required as input because such plans may require the interaction with other entities using some defined protocol.

TASK NAME: Model Plan			
Input	*Output*	*Precondition*	*Postcondition*
RG,RM, AC,PrM, AM,CM	PlM	((exists(<RG,n,m>) ∧ exists(<AC,n,m>)) ∨ exists(<PrM,n,m>) ∨ exists(<AM,n,m>) ∨ previousIteration(<PlM>)) ∧ available(PlP)	completed(<PlM,n,m>)

Fig. 7. Model Plans Task Constrains

Finally, the *Model Protocol* task is defined in Fig. 8. To document a PrM, the *Protocol Modeler* (PrP) requires the RM and the AC or a previous iteration of the

PrM. The *Domain Model* (DM), OM, and AM are optional inputs to this task; they define actions that the agent may perform on environment objects, which can also be modeled as interactions.

TASK NAME: Model Protocol			
Input	*Output*	*Precondition*	*Postcondition*
RM,AC, DM,OM AM	PrM	((exists(<RM,n,m>) ∧ exists(<AC,n,m>)) ∨ previousIteration(<PrM>)) ∧ available(PrP)	completed(<PrM,n,m>)

Fig. 8. Model Protocol Task Constrains

4. WMD Search Example

Next, we present two examples of applying the O-MaSE to derive custom processes. We combine O-MaSE method fragments to create a custom process for a Weapon of Mass Destruction (WMD) system in which agents detect and identify WMD in a given area. There are three types of WMD that can be identified: radioactive, chemical, and biological. Once a suspicious object is found, it must be tested to determine the concentration of radioactivity and nerve agents (chemical and biological). If the object is indeed a WMD, it is removed. The mission is successful when the area has been entirely searched and all the WMD have been removed. In the subsequent subsections, we present two custom processes for the WMD Search application.

4.1 Basic O-MaSE Process

The first process we derive is appropriate for a small agent-oriented project in which reactive agents achieve goals that have been assigned at design time. Essentially, the only products required for this type of system are the system goals, agent classes, agent plans, and inter-agent protocols. This type of process leads to a rigid MAS but is very easy and fast to develop. This process may also be suitable for prototyping, where a simple and rapid process is needed.

Fig. 9 shows the result of applying O-MaSE guidelines to the creation of our custom process. (Tasks are represented by rounded rectangles while Work Products are represented by rectangles.) The Work Products associated with the products identified above are included, along with the Tasks required to produce them. (We do not show the Producers to simplify the figure, but we assume the appropriate Producers are available.) Connections between Tasks and Work Products are drawn and the preconditions and postconditions of each Task are verified. Each Task will be discussed below:

Model Goals/Goal Refinement. From the System Description, the Goal Modeler defines a set of system level goals in the form of an AND/OR goal tree. The AND/OR tree is refined into a GMoDS goal tree as shown in Fig. 10. The syntax uses standard UML class notation with the keyword «Goal». The aggregation notation is used to denote AND refined goals (conjunction), whereas the generalization notation is used to denote OR refined goals (disjunction). GMoDS models include the notion of goal

precedence and goal *triggering* [7]. A *precedes* determines which goals must be achieved while a *trigger* relation signifies that a new goal may be instantiated when a specific event occurs during the pursuit of the another goal. Fig. 10 captures a goal-based view of the system operation.

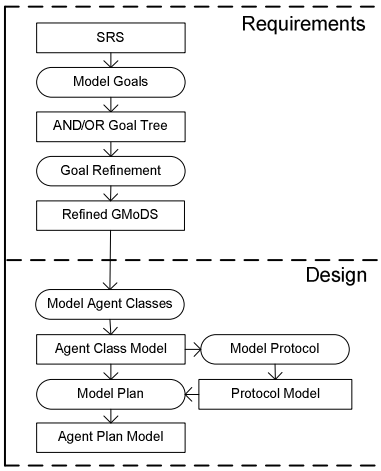

Fig. 9. Basic O-MaSE Process

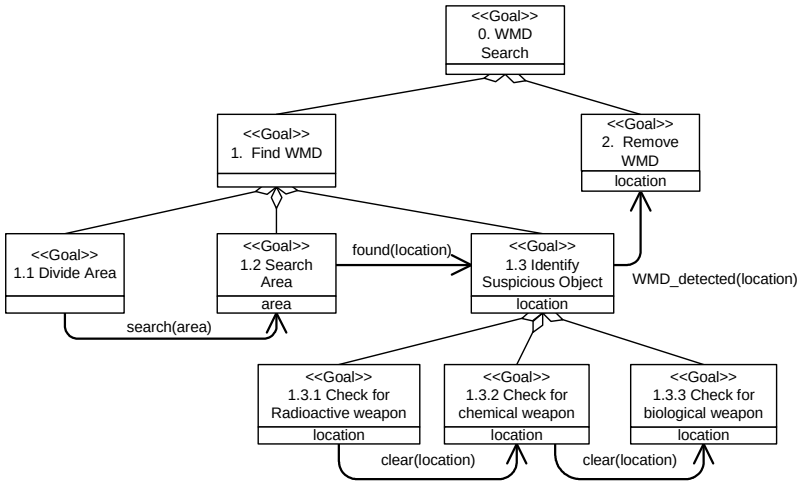

Fig. 10. AND/OR Goal Model

Model Agent Classes. The purpose of this task is to identify the type of agents in the organization and to document them in an Agent Class Model (Fig. 11). In our

example, agents are defined based on the goals they can achieve and the capabilities they possess as specified by the «achieves» and «possesses» keywords in each agent class (denoted by the «Agent» keyword). Protocols between agent classes are identified by arrows from the initiating agent class to the receiving agent class. The details of these protocols are specified later in the Model Protocols task.

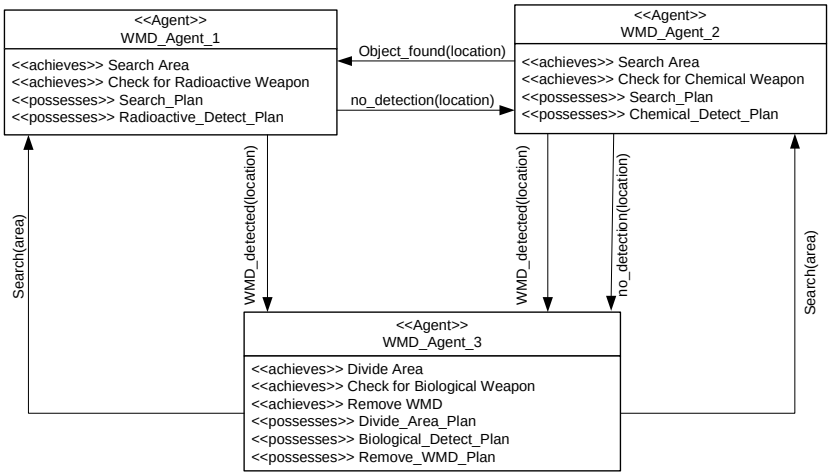

Fig. 11. Agent Class Model

Model Protocol. The Model Protocol task defines the interactions between agents. For example, Fig. 12 captures the *WMD_detected protocol* where WMD_Agent_1, (who is pursuing the *Check for Radioactive Weapon goal*) detects a WMD and notifies WMD_Agent_3 (who is pursuing the *Remove WMD* goal). The notification is done by sending a *detected* message with the *location* as parameter. Upon reception of this message, an acknowledgment is returned.

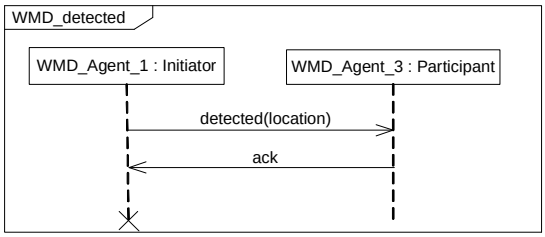

Fig. 12. Protocol Model

Model Plan. The Model Plan task defines plans that agents can follow to satisfy the organization's goals. To model this, we use finite state automata to capture both internal behavior and message passing between agents. Fig. 13 shows the *Radioactive_Detect_Plan* possessed by WMD_Agent_2 to achieve the *Check For*

Radioactive Weapon goal. The plan uses the goal parameter, location, as input. Notice that, a plan produced in this task should correspond to all related protocols.

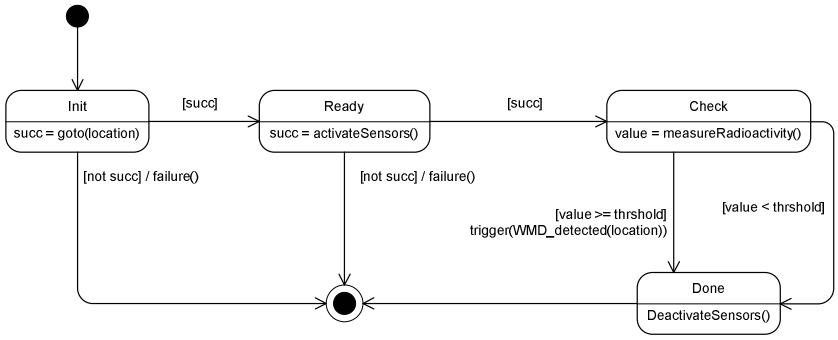

Fig. 13. Plan Model

4.2 Extended O-MaSE Process

To produce a more robust system that adapts to changes and internal failures, it is necessary to have a process that can produce additional information such as roles and policies. *Roles* define behavior that can be assigned to various agents while *policies* guide and constrain overall system behavior. To accommodate such a system, additional Tasks must be introduced into the process to produce a Role Model and a Policy Model. This type of process will allow designer to produce a flexible, adaptive, and autonomous system. Fig. 14 shows the custom process for this example. Below, we briefly discuss the added tasks.

Model Roles. The Model Roles task identifies the roles in the organization and their interactions. Role Modelers focus on defining roles that accomplish one or more goals For example, each role in the Role Model shown in Fig. 15 achieves specific goals from Fig. 10; to do this, each role also requires specific capabilities.

Model Policy. The Model Policy task defines a set of formally specified rules that describe how an organization may or may not behave in particular situations [10]. For example, a policy "An agent may only play one role at a time" can be translated as

$$\forall\ \texttt{a1,a2:agent, r:role} \mid \texttt{a1.plays(r1)} \wedge \texttt{a1.plays(r2)} \rightarrow \texttt{r1=r2}$$

5. Conclusions and Future Work

In this paper we have presented the O-MaSE Process Framework, which allows users to construct custom agent-oriented processes from a set of standard methods fragments. The main advantages of our approach is that: (1) all O-MaSE fragments are based on a common metamodel that ensures the method fragments can be combined in a coherent fashion, (2) each method fragment uses only concepts defined in the metamodel to produce work products that can be used as input to other method fragments; and, (3) the associated guidelines constrain how method fragments may be

combined in order to assemble custom O-MaSE compliant processes that produce an appropriate set of products without producing unnecessary products.

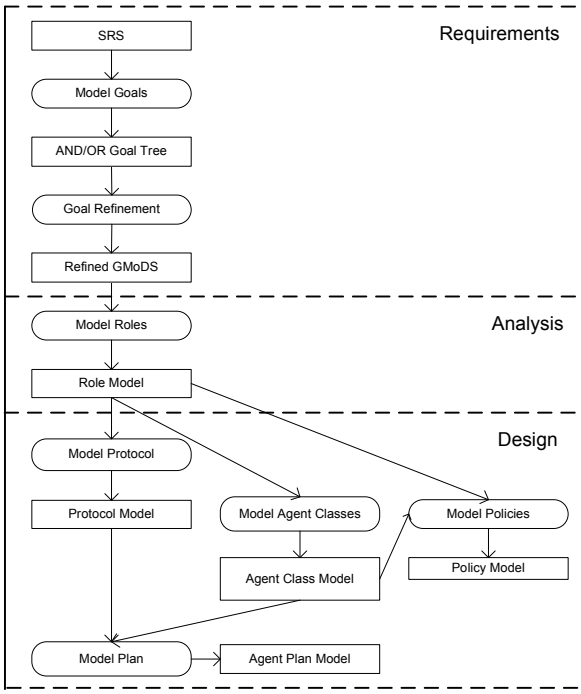

Fig. 14. Extended O-MaSE Process

Although we believe the O-MaSE is headed in the right direction with this approach [11], there is a considerable additional work that must be done in order to create a process amenable to industrial application. First, although the O-MaSE metamodel covers the most basic MAS concepts (i.e., agents, interaction, organization, and interactions), there are other agent-oriented methods and metamodels that deserve further study in order to capture all the main concepts associated with other MAS approaches [2]. We are currently studying several metamodels to determine how to integrate their novel concepts into the O-MaSE metamodel. Second, we are currently working on how to include software metrics into O-MaSE. The aim of these metrics is to predict MAS performance at the analysis and design level [15]. Third, we are continuing to formalize our process guidelines in order to avoid ambiguities between the metamodel and the method fragments used to assembly the agent-oriented applications.

Finally, we are integrating our working into agentTool III (aT^3)[5], which is an analysis and design tool that supports the use of O-MaSE and exists as a plugin for the Eclipse platform[6]. Eventually, we envision adding a module to aT^3 that allows

[5] See http://agenttool.projects.cis.ksu.edu/
[6] See http://www.eclipse.org/

process designers to create and to use custom O-MaSE compliant processes. Future plans for aT^3 also include code generation for various platforms and integration with the Bogor model checking framework for verification and providing predictive metrics [16].

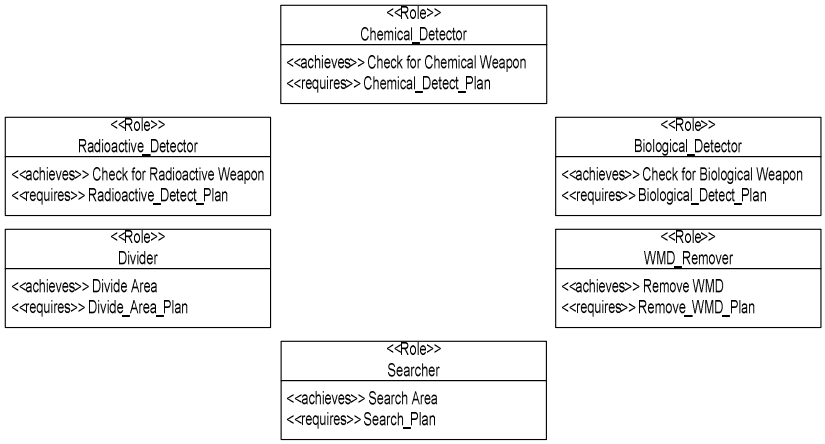

Fig. 15. Role Model

References

1. Bergenti F., Gleizes M.-P., and Zambonelli F. (eds.): Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers (2004).
2. Bernon C., Cossentino M., and Pavón J.: Agent Oriented Software Engineering. The Knowledge Engineering Review. 20(2005) 99–116.
3. Bernon C., Cossentino M., Gleizes M., Turci P., and Zambonelli F.: A study of some multi-agent meta-models. In: Odell, J., Giorgini, P., and Müller, J. (eds.): Agent Oriented Software Engineering V. Lectures Notes in Computer Science. Vol. 3382. Springer-Verlag, Berlin Heidelberg New York (2004) 62–77.
4. Beydoun G., Gonzalez-Perez C., Henderson-Sellers B., Low G.: Developing and Evaluating a Generic Metamodel for MAS Work Products. In: Garcia, A., Choren, R., Lucena, C. Giorgini, P., Holvoet, T., and Romanosky, A. (eds.): Software Engineering for Multi-Agent Systems IV. Lecture Notes in Computer Science, Vol. 3194. Springer-Verlag, Berlin Heideberg New York (2005) 126–142.
5. Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. *Jnl of Information and Software Technology*. 38(4) (1996) 275–280.
6. Cossentino M., Gaglio S., Henderson-Sellers B., and Seidita V.: A metamodelling-based approach for method fragment comparison. In: *Proceedings of the 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 06)*. Luxembourg, June 2006.

7. DeLoach S.A., and Oyenan W. H.: An Organizational Model and Dynamic Goal Model for Autonomous, Adaptive Systems. Multiagent & Cooperative Robotics Laboratory Technical Report No. MACR-TR-2006-01. Kansas State University. March, 2006.
8. DeLoach S.A., and Valenzuela Jorge. L.: An Agent-Environment Interaction Model. In: Padgham, L., and Zambonelli, F. (eds): Agent Oriented Software Engineering VII. Lecture Notes in Computer Science, Vol. 4405. Springer-Verlag, (2007) to appear.
9. Firesmith, D.G., and Henderson-Sellers, B.: The OPEN Process Framework: An Introduction. Addison-Wesley, Harlow–England (2002).
10. Harmon, S.J., DeLoach S.A., and Robby. Guidance and Law Policies in Multiagent Systems. Multiagent & Cooperative Robotics Laboratory Technical Report No. MACR-TR-2007-02. Kansas State University. March, 2007.
11. Henderson-Sellers, B., and Giorgini P. (eds.): Agent-Oriented Methodologies, Idea Group Inc., 2005.
12. Henderson-Sellers, B.: Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework (OPF). Annals of Software Engineering. 14, 1-4 (2002) 341–362.
13. Luck, M., McBurney, P., Shehory, O., and Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), AgentLink (2005).
14. Odell J., Parunak V. D., and Bauer B.: Representing Agent Interactions Protocols in UML. In: Ciancarini, P., and Wooldridge, M. (eds.): Agent Oriented Software Engineering. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag, Berlin Heidelberg New York (2001) 121–140.
15. Robby, DeLoach, S.A., and Kolesnikov, V.A.: Using Design Metrics for Predicting System Flexibility. In: Baresi, L, and Heckel, R (eds.): Fundamental Approaches to Software Engineering. Lectures Notes in Computer Science, Vol. 3922. Springer-Verlag, Berlin Heidelberg New York (2006) 184–198.
16. Robby, Dwyer, M.B., & Hatcliff J.: Bogor: A Flexible Framework for Creating Software Model Checkers. In: *Proceedings of the Testing: Academic & industrial Conference on Practice and Research Techniques*. IEEE Comp Society, Washington, DC, 3–22.
17. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: *Proceedings of the 7th Workshop from Objects to Agents (WOA06).* Catania, Italy (2006) 130–137.