The O-MaSE Process: a Standard View

Juan C. Garcia-Ojeda[1] and Scott A. DeLoach[2]

[1] Facultad de Ingenieria de Sistemas, Universidad Autonoma de Bucaramanga,
Calle 48 No 39-234, El Jardin. Bucaramanga, Santander - Colombia
jgarciao@unab.edu.co

[2] Kansas State University, 234 Nichols Hall, Manhattan, Kansas USA
sdeloach@k-state.edu

Abstract. Method engineering has widely been proposed as an approach to delivering industrial strength software development processes to spur the adoption of agent-based software in industry. The Foundation for Physical Agents Technical Committee (FIPA-TC) Methodology group is currently attempting to define a template for documenting process fragments. This paper presents our experience in applying the template for the Organization-based Multiagent System Engineering methodology

Keywords: agent-oriented software engineering, method engineering, design process documentation, software processes

1 Introduction

Today's software industry is tasked with building ever more complex software applications. Businesses today are demanding applications that operate autonomously, adapt in response to dynamic environments, and interact with other distributed applications in order to provide wide-ranging solutions [8,10]. Multiagent system (MAS) technology is a promising approach capable of meeting these new demands [10]. Unfortunately, there is a disconnect between the advanced technology being created by the multiagent community and its application in industrial software. The obstacles to industrial adoption have been the focus of several discussions. Jennings, Sycara, and Wooldrige [8] mention two major obstacles to widespread adoption of agent technologies in the industry: (1) the lack of complete processes to help designers to specify, analyze, and design agent-based applications, and (2) the lack of industrial-strength agent-based toolkits. To overcome this situation, several MAS researchers and engineers have suggested the use of method engineering [1,2,9,11]. Method engineering is a discipline in which process engineers construct processes (i.e., methodologies) from a set of existing method fragments.

In a related effort, the Foundation for Physical Agents Technical Committee (FIPA-TC) Methodology group is currently attempting to define reusable method fragments from existing agent-oriented processes [13]. As part of this effort, the group is currently defining a Design Process Documentation Template (DPDT)

specification, which uses SPEM 2.0 as its base [5]. In this paper, we present our experience of applying the DPDT guidelines for the Organization-based Multiagent System Engineering (O-MaSE) methodology. After discussing background material in Section 2, we present a partial definition of O-MaSE following the DPDT in Section 3 followed by a discussion of our experiences with the template in Section 4.

2 Background

Method Engineering is an approach where process engineers construct processes (i.e., methodologies) from a set of method fragments as opposed to modifying or tailoring monolithic, "one-size-fits-all" processes to suit their needs. Method fragments are generally created by extracting useful tasks and techniques from existing processes and redefining them in terms of a common metamodel. The fragments are then stored in a repository for later use. During creation, process engineers select suitable method fragments from the repository and assemble them into complete processes meeting project specific requirements [1].

The Software and Systems Process Engineering Meta-model (SPEM) is "a process engineering meta-model as well as conceptual framework, which can provide the necessary concepts for modeling, documenting, presenting, managing, interchanging, and enacting developments processes" [12]. SPEM distinguishes between reusable *method content* and the way it is applied in actual *processes*. SPEM method content captures and defines the key Tasks, Roles, and Work Products[1] in a software development processes. Essentially, Tasks are performed by Roles, taking a set of Work Products as inputs and producing set of Work Products as its output.

Development processes are assembled into a set of Activities, populated with Tasks and their associated Roles and Work Products. Thus, Activities are aggregates of either basic content or other Activities. SPEM defines three special types of Activities: Phases, Iterations and Processes. Phases are special Activities that take a period of time and end with a major milestone or set of Work Products. Iterations are Activities that group other Activities that are often repeated. Finally, Processes are special Activities that specify the structure of a software development project.

2.2 FIPA Design Process Documentation Template Specification

The *Design Process Documentation Template* specification [5] introduces a set of guidelines whose goal is the identification of the fundamental concepts in the design of agent-oriented design processes independent of the notation (text, icons, diagrams, etc.). This specification follows the situational method engineering approach for reusing fragments from known methods to obtain custom methods suitable for specific development situations. For designing agent-oriented processes, the specification suggests the use of a process documentation template. The template

[1] SPEM 2.0 defines as Method Content with Task Uses, Role Uses, and Work Product Uses as instances of Task Definitions, Role Definitions, and Work Product Definitions in actual processes. This paper refers to both forms as Tasks, Roles or Work Products.

guides process designers to build processes by documenting three main sections: Introduction, Phases of the Process, and Work Product Dependencies. The goal of the *Introduction* is: (i) to introduce the scope and limits of the process, (ii) organize the design process phases according to the selected lifecycle and, (iii) to provide a complete description of the MAS metamodel adopted in the process with the definition of its composing elements. The aim of the *Phases of the Process* is to detail the whole process by decomposing it on the basis of workflows at different levels of granularity (phase-activity-task). Finally, the *Work product dependencies* represent the dependencies between the work products and thus (indirectly) between the activities that produce them. Finally, the template suggests the adoption of SPEM 2.0 as the standard for modeling *some* design process aspects.

3 Applying the DPDT to O-MaSE

In this section, we present a partial definition of O-MaSE using the DPDT. Due to page length considerations, we show selected diagrams with abbreviated descriptions.

3.1 Introduction

O-MaSE 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 framework [7]. 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 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 aT^3 Process Editor (APE) [6] supports the creation of custom O-MaSE compliant processes [6]. APE has five key elements: 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.

The O-MaSE Lifecycle. As O-MaSE was designed as a set of fragments to be assembled by developers to meet their specific requirements, it does not actually commit to any specific set of phases. This is a major difficulty with trying to map O-MaSE to the DPDT. To alleviate this problem, we assume we are following a traditional waterfall approach shown in Figure 1. There are three main Phases: Requirements Analysis, Design, and Implementation, with the main Activities

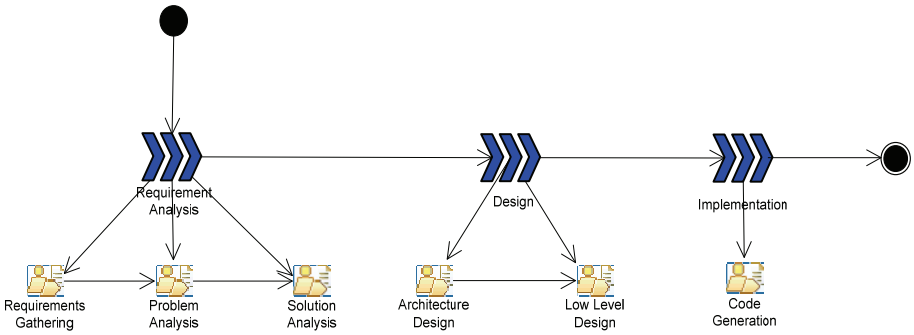

Figure 1. Using Waterfall Phases with O-MaSE

allocated as expected. When actually using O-MaSE, the process designer must define their own set of phases and iterations and then assign Activities and Tasks to those phases and iterations. 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 phase.

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). OMACS captures the knowledge required of a system's organizational structure and capabilities to allow it to organize and reorganize at runtime [3]. The key decision in OMACS-based systems is which agent to assign to which role in order to achieve which goal. As shown in Figure 2, an Organization is composed of six entities: Goals, Roles, Agents, Organizational Agents, a Domain Model, and Policies, which are defined in Table 1.

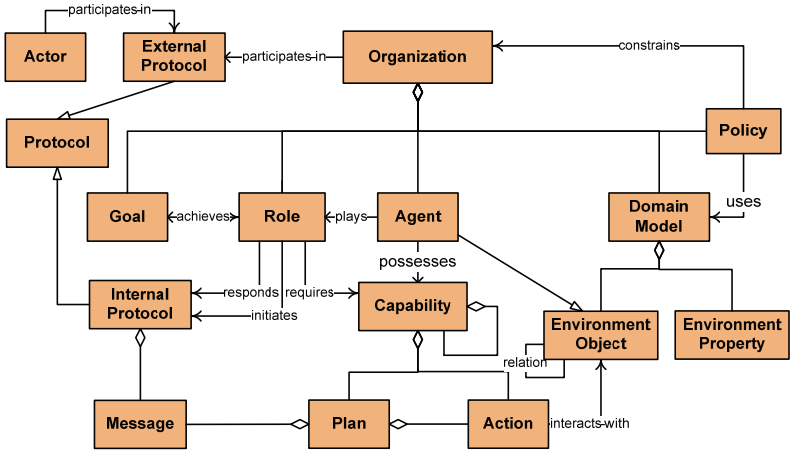

Figure 2. O-MaSE Metamodel

Table 1. Metamodel Entities

Goal	Goals are a desirable state; goals capture organizational objectives
Role	Roles capture behavior that achieves a particular goal or set of goals
Agent	Agents are autonomous entities that perceive and act upon their environment; agents play roles in the organization
Capability	Capabilities capture 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	Policies constrain organization behavior often in the form of liveness and safety properties
Protocol	Protocols define interaction between agents, roles, or external Actors; they may be internal or external
External actor	External Actors exist outside the system and interact with the system
Plan	Plans are abstractions of algorithms used by agents; plans are specified in terms of actions with the environment and messages in protocols

3.2 Phases

As a reminder, the phases presented here are not actually part of the O-MaSE definition, but only included to help define O-MaSE according the DPDT.

Requirements 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 requirements analysis phase is shown in Figure 3.

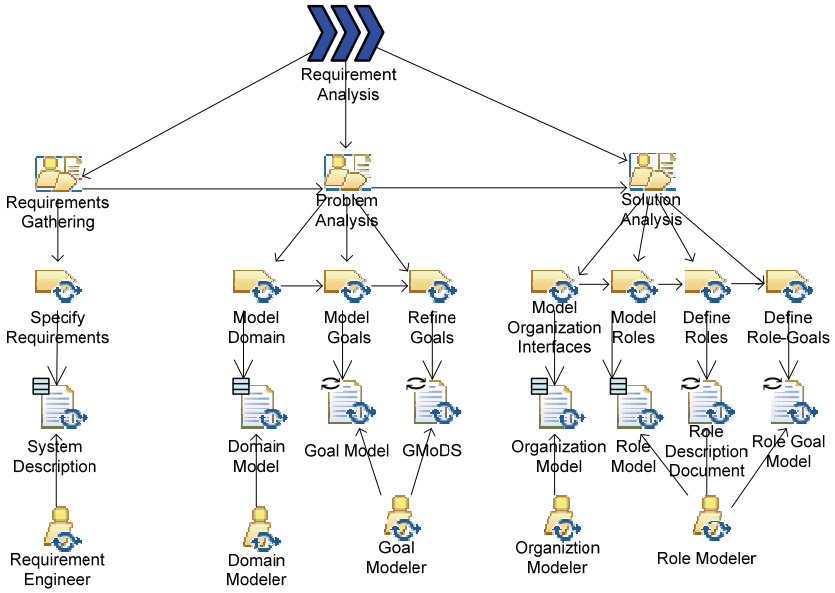

Figure 3. Requirements Analysis Phase

Process Roles. This phase uses five roles: Requirements Engineer, Goal Modeler, Domain Modeler, Organization Modeler, and Role Modeler. The *Requirements 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 be able to collaborate with stakeholders, domain modelers, and project managers. 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 must be proficient in GMoDS AND/OR Decomposition and ATP Analysis [4]. 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. The *Organization Modeler* is responsible for documenting the Organization Model. Thus, the Organization Modeler must understand the system requirements, Goal Model, and the Domain Model and be skilled in organizational modeling techniques. 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.

Activity Details. In the Requirements Analysis phase, there are three activities: Requirements Gathering, Problem Analysis, and Solution Analysis. Requirements 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 non-functional requirements, which specify traits of the software such as performance quality, and usability. Problem Analysis captures the purpose of the product and documents the environment in which it will be deployed. O-MaSE captures this information in a Goal Model, which captures the purpose of the product, and a Domain Model that captures the environment in which the product exits. Finally, 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.

Work product kinds. There are six work products produced in the Requirements Analysis phase: System Description Specification, Goal Model, GModS Model, Domain Model, Organization Model, and Role Model as defined in Table 2.

Table 2. Requirements Analysis Work Products

Name	**Description**	**Work Product Kind**
System Description Specification	describes the technical requirements for a particular agent-oriented software	Textual
Goal Model	captures the purpose of the organization as a goal tree; includes goal attributes, precedence relationships and triggering relationships	Behavioral
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	Behavioral

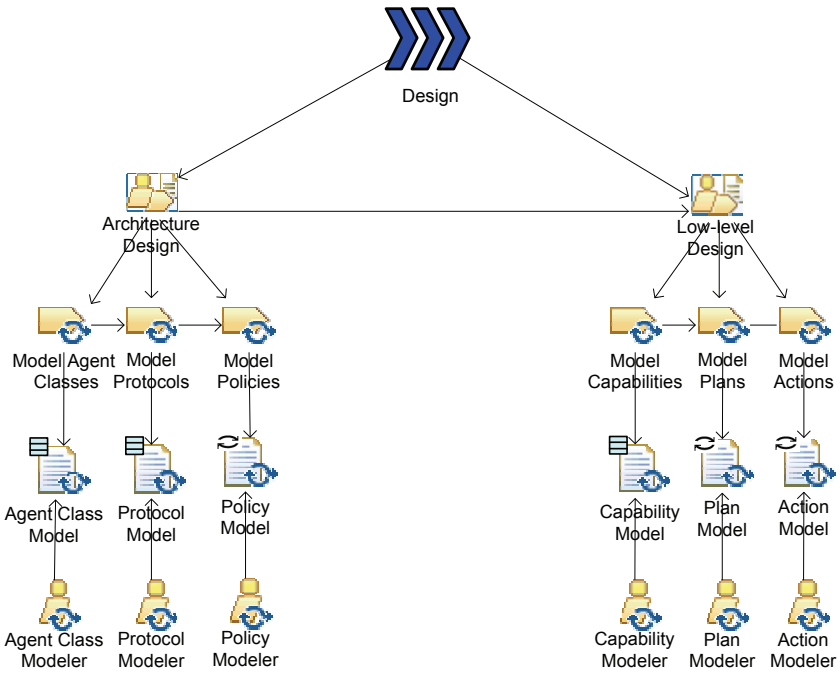

Figure 4. Design Phase

Design. The design phase consists of two activities: Architecture Design and Low Level Design. Once the goals, the environment, the 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 (see Figure 4).

Process Roles. There are six roles in the design phase: Agent Class Modeler, Protocol Modeler, Policy Modeler, Capability Modeler, Plan Modeler, and Action 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. The *Protocol Modeler* designs the protocols required between agents, roles, and external actors and requires protocol modeling skills. The *Policy Modeler* is responsible for designing the policies that govern the organization. The *Capability Modeler* is responsible for defining the Capability Model and requires modeling skills and O-MaSE Capability Model specification knowledge. 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. Finally, the *Action Modeler* documents the Action Model, which requires the ability to specify appropriate pre- and post-conditions for capability actions.

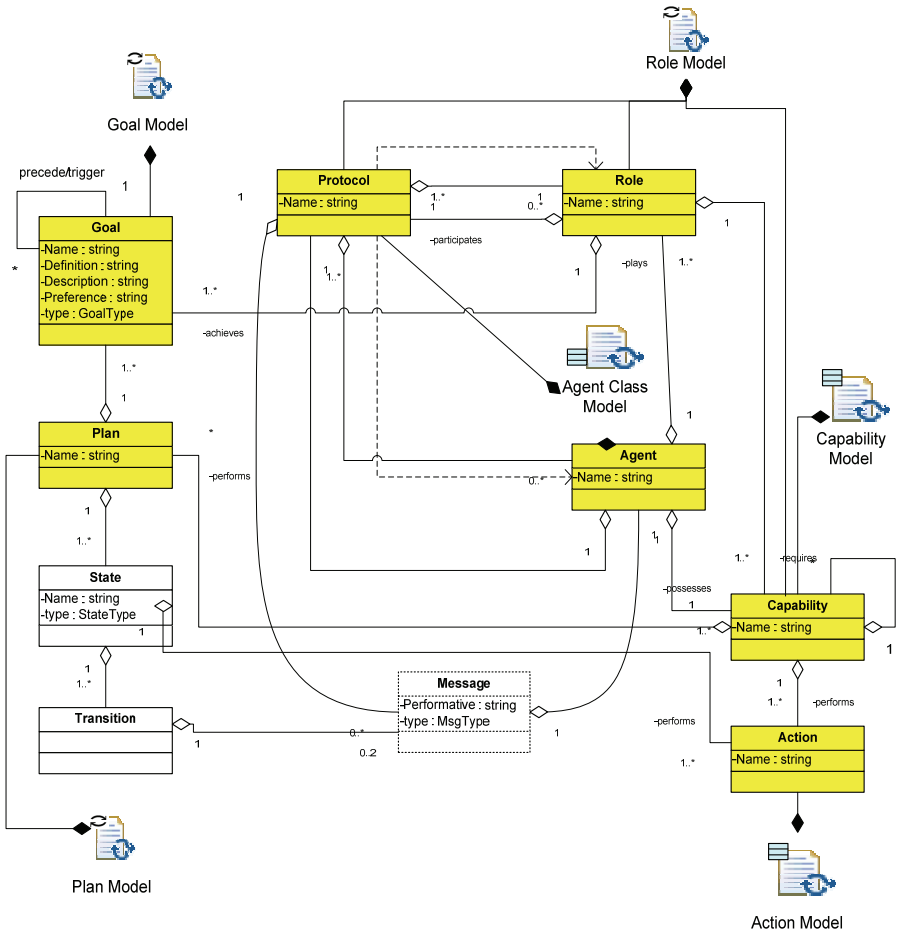

Figure 5. Relationship among several Work Products and the Metamodel

Activity Details. 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. The *Model Agent Classes* task identifies the types of agents that may participate in the organization. Agent classes may be defined to play specific roles, or they may be defined in terms of capabilities, which implicitly define the types of roles that may be played. The *Model Protocols* task defines the details of the interactions between agents or roles. The Protocol Model produced defines the types of messages sent between the two entities. Finally, the *Model Policie*s task defines a set of formally specified rules that describe how an organization may or may not behave in particular situations. During the organization design, the Policy Modeler captures the desired system properties and documents them in a formal or informal notation.

Table 3. Requirements Analysis Work Products

Name	Description	Work Product 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.	Structural

In the Low-level design we focus on the capabilities possessed by, actions performed by, and plans followed by agents. 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). Finally, 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. Figure 5 shows the relationship between some work products (i.e., Goal Model, Role Model, Agent Class Model, Capability Model, Plan Model, and Action Model) and the entities used to design a typical system. Notice for instance, that the Goal Model defines goals; the Capability Model defines capabilities, while the Role Model uses those goals and capabilities to define roles and protocols in the Role Model. Likewise, the Plan Model defines plans in terms of actions defined by the Action Model.

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 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.

Table 4. Work Product Dependencies

Work product	*Work product Kind*	*Dependency*
System Description	Structural	None
Goal Model	Behavioral	System Description
GMoDS	Behavioral	System Description, Goal Model
Domain Model	Structural	System Description
Organization Model	Structural	System Description
Role Model	Structural	GMoDS, Organization Model
Role Description Document	Behavioral	Role Model
Role-Goal Model	Behavioral	GMoDs, Role Model
Agent Class Model	Structural	Organization Model, GMoDS
Protocol Model	Structural	Role Model, Agent Class Model
Plan Model	Behavioral	Role Model, Agent Class Model, GMoDS
Policy Model	Behavioral	Agent Class Model, Role Description Document, GMoDS
Capability Model	Structural	Domain Model, Agent Class Model, Role Model
Action Model	Behavioral	Domain Model, Capability Model
Code	Composite	Capability Model, Action Model

3.3 Workproduct Dependencies

Table 4 shows the dependencies between the different work products in O-MaSE. These dependencies characterize different pieces of information or physical entities 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). Also, each work product is specified in terms of the kind of model/information/data documented. For instance, a structural work product is used to model static aspects of the system. In turn, a behavioral work product is used to model dynamic aspects of the system. Finally, a composite work product is used to model both static and dynamic aspects of the system (for further details on the different work product kind's see [13]). Figure 6 presents the dependencies between the various O-MaSE work products.

4 Conclusions and Future Work

In this paper, we presented a part of the O-MaSE documentation produced by following the DPDT specification. To be able to fit into the DPDT template, we had to select an example set of phases, which we did base on a simple waterfall approach. Then, we proceeded to document the different phases of our simple process in terms of the different activities, tasks, roles, and work products.

Based on our experience, we do not believe that requiring the process to be defined in terms of phases is necessarily the best approach. While we were able to use a simple waterfall model and describe our activities and tasks as if they were all be used

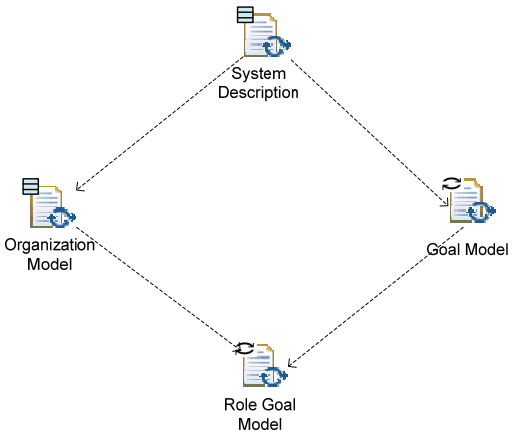

Figure 6. Requirement Analysis Phase´s Work Product Dependencies

in a straightforward, sequential manner, this might not always be the case. Specifically, it is unclear how to document activities and tasks that might take differing approaches and thus would likely be incompatible within the same process. In addition, forcing O-MaSE into any predefined set of phases masks the flexibility of the general approach proposed in O-MaSE.

We also believe it to be the case that the DPDT was defined assuming that fragments would be defined at the Activity level. However, when we create custom O-MaSE compliant processes, we generally use fragments at the Task level. Since Tasks are broken down only within an Activity and exist as rows in a table, there is not a natural mechanism for referring to them other than to call them by name and refer to the Activity in which they are defined. This breakdown also makes it difficult to document tasks that can be used in more than one Activity. For example, we have a Model Protocols task that is nominally defined in the Architecture Design activity. However, we can also model protocols within the Solution Analysis activity as well. Actually, you can Model Protocols in five ways: between organizations and external actors, between external actors and roles, between external actors and agents, between roles and roles, and between agents. We could make five copies of the Model Protocols task and specify the Work Product inputs slightly differently in each case; however, this seems redundant. In our original O-MaSE definition, we have one task called Model Protocols that has several optional Work Product inputs.

Although we believe the DPDT specification is headed in the right direction by supporting the construction of custom agent-based processes, there is considerable work to do before the DPDT will make an impact on industrial acceptance. While the DPDT will allow fragments to be documented in a common format, this is not useful unless tools for creating, maintaining, and transforming fragments are developed. While APE is an initial step in this direction, additional effort should be pursued to further support industrial acceptance.

Specifically, taking the DPDT as a starting point, research should be performed to extend this work to (i) develop qualitative and quantitative methods for helping process designers in creating custom processes based on the functional, non-

functional, and general architectures of new systems; (ii) formalize process guidelines in order to avoid ambiguities between the metamodel and the method fragments used to assembly agent-oriented applications, (iii) provide a set of guidelines on how to integrate different agent-oriented metamodels.

References

1. Brinkkemper, S. Method engineering: engineering of information systems development methods and tools. Information and Software Technology. 38(4) 1996, pp 275-280.
2. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardization to research. *Intl Jnl of Agent-Oriented Software Engineering,* 1(1), 91–121, 2007.
3. DeLoach, S.A., Oyenan, W., Matson, E.T. A capabilities based model for artificial organizations. *Autonomous Agents and Multiagent Systems.* 16(1), 13-56, 2008.
4. DeLoach, S.A., and Miller, M. A Goal Model for Adaptive Complex Systems. International Journal of Computational Intelligence: Theory and Practice. Volume 5, no. 2, 2010. (in press).
5. FIPA Design Process Documentation and Fragmentation Working Group. Design Process Documentation Template (08-06-2010). http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/.
6. Garcia-Ojeda, J.C., DeLoach, S.A. Robby. agentTool process editor: supporting the design of tailored agent-based processes. In *Proc. of the 2009 ACM Symp on Applied Computing,* ACM: New York, 2009.
7. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyenan, W.H.,Valenzuela, J. O-MaSE: a customizable approach to developing multiagent development processes. In M. Luck (ed.), *Agent-Oriented Software Engineering VIII: The 8th Intl Workshop on Agent Oriented Software Engineering (AOSE 2007)*, LNCS 4951, 1-15, Springer: Berlin.
8. Jennings, N. R., Sycara, K., Wooldridge, M. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems* 1(1) 7-38, 1998.
9. Low, G., Beydoun, G., Henderson-Sellers, B., Gonzalez-Perez, C. Towards method engineering for multi-agent systems: a validation of a generic MAS metamodel. In *10th Pacific Rim Intl Conf on Multi-Agent Systems, PRIMA 2007, Bangkok, Nov 21-23, 2007.* A. Ghose, G. Governatori, R. Sadananda, Eds. LNAI 5044. Springer: Berlin, 255-267, 2009.
10. Luck, M. McBurney, P. Shehory, O. Willmott, S. *Agent technology: a roadmap for agent based computing*. AgentLink, Southampton, UK, 2005.
11. Molesini, A., Denti, E., Nardini, E., Omicini, A. Situated process engineering for integrating processes from methodologies to infrastructures. In *Proc. of the 2009 ACM Symposium on Applied Computing,* ACM: New York, 699-706, 2009.
12. OMG (2008) "Software & Systems Process Engineering Meta-Model Specification", v2.0. Formal/2008-04-01, http://www.omg.org/docs/formal/08-04-01.pdf.
13. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In *Proc. of the 7th Workshop from Objects to Agents* (WOA 2006), Catania, Italy, pp. 130–137, 2006.