Chapter IV
OMACS a Framework for Adaptive, Complex Systems

Scott A. DeLoach
Kansas State University, USA

ABSTRACT

This chapter introduces a suite of technologies for building complex, adaptive systems. It is based in the multi-agent systems paradigm and uses the Organization Model for Adaptive Computational Systems (OMACS). OMACS defines the knowledge needed about a system's structure and capabilities to allow it to reorganize at runtime in the face of a changing environment and its agent's capabilities. However, the OMACS model is only useful if it is supported by a set of methodologies, techniques, and architectures that allow it to be implemented effectively on a wide variety of systems. To this end, this chapter presents a suite of technologies including (1) the Organization-based Multiagent Systems Engineering (O-MaSE) methodology, (2) a set of policy specification techniques that allow an OMACS system to remain flexible while still providing guidance, and (3) a set of architectures and algorithms used to implement OMACS-based systems. The chapter also includes the presentation of a small OMACS-based system.

INTRODUCTION

Multiagent systems have become popular over the last few years for building complex, adaptive systems in a distributed, heterogeneous setting. The problem is that many multiagent systems are typically designed to work within a limited set of configurations. Even when the system possesses the resources and computational power to accomplish its goal, it may be constrained by its own structure and knowledge of its member's capabilities. To overcome these problems, we have developed a framework that allows the system to design its own organization at runtime. The key component of our framework is the Organization Model for Adaptive Computational Systems (OMACS). OMACS defines the knowledge needed about a system's structure and capabilities to allow it to reorganize at runtime in the face of a changing environment and its agent's capabilities. Thus, as is also pointed out in chapter 3 "Towards an Integral Approach of Organizations in Multi-Agent Systems", the environment plays an critical role in the specification of adaptive organizations. The OMACS model is based on the assumption that agents accept certain limitations on their autonomy; if assigned a role to play in order to achieve a goal, the agents must agree to attempt to play that role in the organization and pursue the assigned goal. The OMACS model also assumes no specific architecture (although one appropriate example is presented later in the chapter). It is assumed that the "organization" reasons based on the knowledge described in OMACS.

That being said, the OMACS model is only useful if it is supported by a set of methodologies, techniques, and architectures that allow it to be implemented effectively on a wide variety of systems. To this end, this chapter also presents a suite of technologies developed to support OMACS-based system development. After discussing the OMACS model, we present the Organization-based Multiagent Systems Engineering (O-MaSE) methodology. O-MaSE is a framework for creating O-MaSE compliant processes that support the development of OMACS-based systems. Next, we discuss an area of vital importance to reorganizing systems, policy specification. Since it is often desirable to constrain how a system can reorganize, we have investigated the notion of system-level policies. In this section we introduce the notion of *guidance policies* that allow us to specify policies that must be followed except when absolutely necessary. Finally, we present an architecture and a set of algorithms that can be used to implement OMACS-based systems.

BACKGROUND

Computational organization theory uses mathematical and computational techniques to study both human and artificial organizations (Carley 1999). While organizational concepts are not exclusive to computational organization theory, results from the field are illuminating. Specifically, they suggest that organizations tend to adapt to increase performance or efficiency, that "the most successful organizations tend to be highly flexible" (Carley 1998), and that the best organizational designs are highly application and situation dependent (Carley, 1995). Recently, the notion of separating the agents populating a multiagent system from the system organization (Zambonelli, Jennings, & Woodridge, 2001) has become well-accepted. While agents play roles *within* the organization, they do not constitute the organization. The organization itself is part of the agent's environment and defines the social setting in which the agent must exist. An organization includes *organizational structures* as well as *policies,* which define the requirements for system creation and operation.

There have been several attempts at formalizing the concepts of teamwork within an organization in the area of multiagent systems. While efforts such as Teamwork (Cohen & Levesque, 1991), Joint Intentions (Jennings, 1995), Shared Plans (Grosz & Kraus, 1996) and Planned Team Activity (Kinny, Ljungberg, Rao, Sonenberg, Tidhar & Werner, 1992), have been proposed and even implemented (Tambe, 1997), they fail to provide straightforward and easily adaptable concepts for wide spread development of such systems. In addition, these approaches require all agents to be capable of sophisticated reasoning about their organization, which tends to get intertwined with reasoning about the application, thus increasing the complexity of the agents.

While there have been several organization models proposed, none have been specifically targeted towards providing a general mechanism that allows the system to reorganize in order to adapt to its environment and changing capabilities. One of the first models of agent organizations was given by Ferber and Gutknecht in the AALAADIN model (Ferber & Gutknecht, 1998) and extended in the AGR model (Ferber, Gutknecht & Michel, 2003).The AALAADIN/AGR model used agents, groups, and roles as its primitive concepts and they are now found in almost all other organization models in one form or another. There have also been other attempts to extend the basic AGR model such as that proposed by Peng and Peng (2005) to provide some behavioral definition of roles. The MOISE+ model greatly extended the notion of an organization model by including three aspects: structural, functional, and deontic (Hübner, Sichman & Boissier, 2002). The structural aspect of MOISE+ is similar to the AGR model, defining the organizational structure via roles, groups, and links. The function aspect describes how goals are achieved by plans and missions while the deontic aspect describes the permissions and obligations of the various roles. One of the most complete organization models is the Organizational Model for Normative Institutions (OMNI) (Dignum, Vázquez-Salceda & Dignum, 2004), which is a framework that caters to open multiagent systems. OMNI allows heterogeneous agents to enter the organization with their own goals, beliefs, and capabilities and does not assume cooperative agents. OMNI combines two previous organization models: OperA (Dignum, 2004) and HarmonIA (Vázquez-Salceda & Dignum, 2003). An excellent overview of existing organizational models and dimensions of organizational modeling is found in chapter 2 "Modelling Dimensions for Multi-Agent Systems Organizations".

While almost all multiagent methodologies have an underlying metamodel that describes their basic modeling concepts, most are not explicitly defined. One exception is the ROADMAP method, whose metamodel is defined in Juan and Sterling (2004). ROADMAP defines a nice clean metamodel that includes the basic modeling concepts of roles, protocols, services, agents, knowledge, and the environment. Likewise, the Multiagent Systems Engineering (MaSE) methodology metamodel was defined in part based on the implementation of agentTool, a tool that supports the MaSE modeling process (DeLoach & Wood, 2001). The MaSE metamodel defines the main modeling concepts of goals, roles, agents, conversations, and tasks. Bernon et. al., combined the metamodels from three well-known methodologies – ADELFE, Gaia, and PASSI – into a common metamodel that they hoped would provide interoperability between the methods (Bernon, Cossentino, Gleizes, Turci & Zambonelli, 2005). While the unified metamodel contains many more concepts than those of single methodologies, the unified metamodel is very complex and it is not clear how many of the concepts are actually related. Based on his experience in trying to combine existing multiagent method fragments using the OPEN process framework, Henderson-Sellers has concluded that a single standard metamodel is required before fragments can be combined successfully on a large scale (Henderson-Sellers, 2005). OMACS provides the foundation for organization-based multiagent metamodel in which the analysis and design concepts are directly related to run-time concepts.

OMACS MODEL

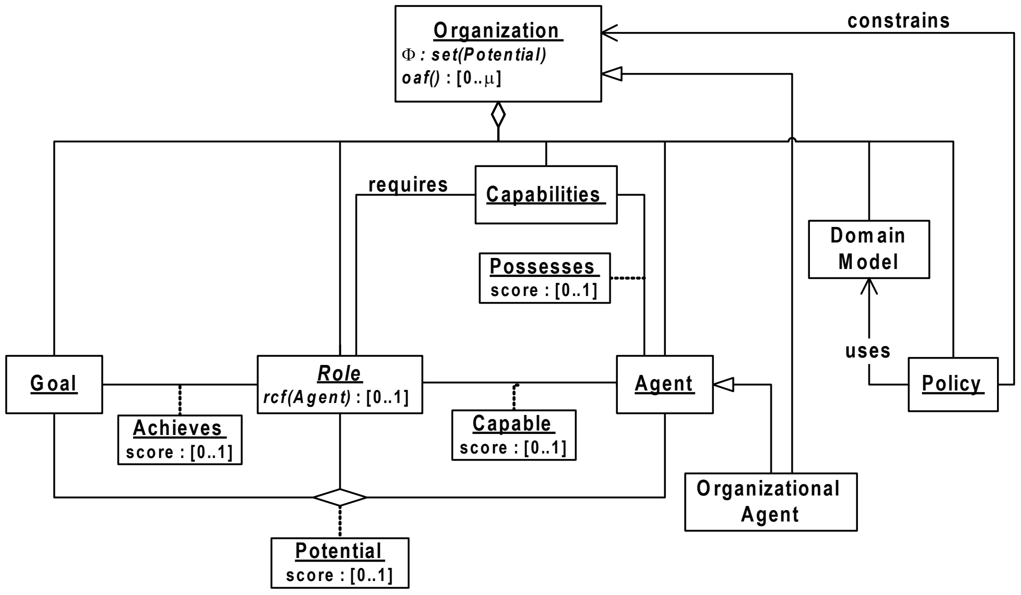
The OMACS model grew from the MaSE metamodel, which was based on the original AGR model (Ferber, Gutknecht & Michel, 2003). We realized that while we could define multiagent systems in terms of agent playing roles in order to achieve system goals, we did not have to limit the flexibility of agents by predefining the roles they could and could not play. Noting that agents could be assigned to roles based on the capabilities required to play various roles and the capabilities possessed by the agents, we figured the agents could adapt their assignments based on the current set of goals required to be achieved by the system. This basic idea led to the OMACS model as defined in DeLoach, Oyenan, and Matson (2007) and shown in Figure 1.

General Organization Definition

OMACS defines an organization as a tuple O=⟨G, R, A, C, Φ, P, Σ, *oaf*, *achieves*, *requires*, *possesses*⟩ where

- G — goals of the organization
- R — set of roles
- A — set of agents
- C — set of capabilities
- Φ — relation over G × R × A defining the current set of agent/role/goal assignments
- P — set of constraints on Φ
- Σ — domain model used to specify environment objects and relationships
- oaf — function P(G × R × A) → [0.. ∞] defining quality of a proposed assignment set

Figure 1. OMACS model



Equation 1.

$$\mathrm{rcf}(a,r) = \begin{cases} \mathit{if} \quad \prod_{c \in requires(r)} possesses(a,c) = 0 & 0 \\ \mathit{else} & \dfrac{\sum_{c \in requires(r)} possesses(a,c)}{|requires(r)|} \end{cases}$$

- achieves — function G × R → [0..1] defining how effective the behavior defined by the role could be in the pursuit of the specified goal
- requires — function R → P(C) defining the set of capabilities required to play a role
- possesses function A × C → [0..1] defining the quality of an agent's capability

OMACS also includes two additional derived functions to help compute potential assignment values: capable and potential.

- capable — function A × R → [0..1] defining how well an agent can play a role (computed based on requires and possesses)
- potential — function A × R × G → [0..1] defining how well an agent can play a role to achieve a goal (computed based on capable and achieves)

Main Elements

The first eight elements in the organization tuple defined above – G, R, A, C, Φ, P, Σ, and *oaf* – constitute the main elements of the OMACS model as depicted in Figure 1. *Goals* are generally defined as a desirable situation (Russell & Norvig, 2003) or the objective of a computational process (van Lamsweerde, Darimont & Letier, 1998). In OMACS, each organization has a set of goals, `G`, that it seeks to achieve. OMACS makes no assumptions about these goals except that they can be assigned and achieved by one, and only one agent playing an appropriate role in the organization (i.e., agents do not share goals).

Each OMACS organization also has a set of roles (`R`) that it can use to achieve its goals. A *role* defines a position within an organization whose behavior is expected to achieve a particular goal or set of goals. These roles are analogous to roles played by actors in a play or by members of a typical corporate structure. Each OMACS role has a *name* and a function that defines how well an agent can play the role based on the capabilities possessed by that agent. This function is termed the role capability function, `rcf : A × R → [0..1]`. The rcf is defined at design time for each role and computed in terms of the capabilities required to play that role. A default rcf (as shown in Equation 1) would assume that all the capabilities required to play a role `r` are equally important, essentially taking the average of all the required possesses values (`possesses(a,c)`) (with the stipulation that none of those possesses scores are 0).

Each OMACS organization also has a set of heterogeneous *agents* (`A`), which are assumed to be computational systems (human or artificial) that sense and act autonomously in a complex dynamic environment in order to realize a set of assigned goals. Thus, we assume that agents exhibit the attributes of autonomy, reactivity, pro-activity, and social ability (Russell & Norvig, 2003). To be part of an OMACS organization, agents must have the ability to communicate with each other (which requires a share ontology and communication language at the organizational level), accept assignments to play roles that match their capabilities, and work to achieve their assigned goals.

OMACS uses the capabilities possessed by an agent to determine whether or not it can play a specific role. In OMACS, *capabilities* are atomic entities used to define a skill or capacity of agents. Capabilities can be used to capture *soft* abilities such as the ability to access certain resources or the knowledge of algorithms, or *hard* abilities such as sensors and effectors that are often associated with hardware agents such as robots. The set of capabilities, `C`, known to an organization is the union of all the capabilities required by roles or possessed by agents in the organization.

$$\forall \text{ c:C } (\exists \text{ a:A possesses(a,c)}>0 \vee \exists \text{ r:R } c\in\text{requires(r)}) \tag{2}$$

An *assignment set* Φ is the set of agent-role-goal tuples ⟨`a,r,g`⟩, that indicate that agent `a` has been assigned to play role `r` in order to achieve goal `g`. Φ is a subset of all the *potential assignments* of agents to play roles to achieve goals. This set of potential assignments is captured by the *potential* function (see equation (8)), which maps each agent-role-goal tuple to a real value ranging from 0 to 1 representing the ability of an agent to play a role in order to achieve a specific goal. If ⟨`a,r,g`⟩∈Φ, then agent `a` has been assigned by the organization to play role `r` in order to achieve goal `g`. The only inherent constraints on Φ is that it must contain only assignments whose potential value is greater than zero (Equation 3) and that only one agent may be assigned to achieve a goal at a time (Equation 4).

$$\Phi \subseteq \{\langle a,r,g\rangle \mid a\in A \wedge r\in R \wedge g\in G \wedge \text{potential}(a,r,g) > 0\} \tag{3}$$

$$\forall \text{ a1,a2:A r1,r2:R g1,g2:G } \langle a1,r1,g\rangle\in\Phi \wedge \langle a2,r2,g\rangle\in\Phi \wedge a1=a2 \tag{4}$$

In order to select the best set of assignments to maximize an organization's ability to achieve its goals, OMACS defines an *organizational assignment function*, or `oaf`, which is a function over the current assignment set, `oaf:` $\Phi \rightarrow$ `0..`∞. As with the `rcf`, the selection of assignments may be application specific. Thus, each organization has its own application specific organization assignment function, `oaf`, which computes the *goodness* of the organization based on Φ. As with the `rcf`, we can define a default `oaf`, which is simply the sum of the potential scores in the current assignment set Φ.

$$oaf = \sum_{<a,r,g>\in\Phi} potential(a,r,g) \tag{5}$$

In general, *policies* are a set of formally specified rules that describe how an organization may or may not behave in particular situations. In OMACS, we distinguish between three specific types of policies: *assignment* policies ($\mathtt{P}_{\Phi}$) *behavioral* policies ($\mathtt{P}_{\mathtt{beh}}$), and *reorganization* policies ($\mathtt{P}_{\mathtt{reorg}}$). A more detailed discussion of policies is given later.

An OMACS *domain model*, Σ, is used to define object types in the environment and the relations between those types. The domain model is based on traditional object oriented class and relation concepts[1]. Specifically, an OMACS domain model includes a set of object classes, each of which is defined by a set of attribute types. The relations between object classes include general purpose associations as well as generalization-specialization and aggregation. Relations may also include multiplicities to constrain the number of object classes participating in any given relation.

Relations and Functions

There are three major relations/functions and two derived functions between the eight main elements that provide the power of the OAMCS model: *achieves*, *requires*, *possesses*, *capable,* and *potential.* The *achieves* function (although somewhat confusingly named) actually defines how effective an agent could be while playing that role in the pursuit of a specific goal. For instance, if one role requires more resources or better capabilities, it can use a different approach and thus yield better results than a second role that requires fewer resources or capabilities. Providing two different roles to achieve the same goal may provide the organization flexibility in deciding how to actually achieve a given goal. The value of

achieves can be predefined by the organization designer or learned before or during system operation (Odell, Nodine & Levy, 2005). Thus, the OMACS `achieves` function formally captures the effectiveness of a role in achieving a specific goal by defining a total function from the R x G to a real value in the range of 0 to 1, `achieves: G × R → [0..1]`. Thus, by definition, a role that cannot be used to achieve a particular goal must have an `achieves` value of 0, while a role that can achieve a goal would have an `achieves` value greater than zero.
The key in assigning agents to roles is determining which agents are capable of playing which roles. In order to play a given role, agents must possess a sufficient set of *capabilities* that allow the agent to carry out the role and achieve its assigned goals. Thus OMACS defines the *requires* function, `requires: R →` **`P`**`(C)`, which allows an organization designer to define the minimal set of capabilities agents must possess in order to play that role.

In order to determine if some agent has the appropriate set of capabilities to play a given role, OMACS defines a similar relation, the `possesses` relation, that captures the capabilities a specific agents actually possesses. The possesses relation is formally captured as a function over agents and capabilities that returns a value in the range of 0 to 1, `possesses: A × C → [0..1]`. The real value returned by the possesses function indicates the quality of each capability possessed by the agent; 0 indicates no capability while a 1 indicates a high quality capability.

Using the capabilities required by a particular role and capabilities possessed by a given agent, we can compute the ability of an agent to play a given role, which we capture in the *capable* function. The capable function returns a value from 0 to 1 based on how well a given agent may play a specific role, `capable: A × R → [0..1]`. Since the capability of an agent, `a`, to play a specific role, `r`, is application and role specific, OMACS provides the `rcf` defined in the previous section that controls how this value is computed. Thus, the *capable score* of an agent playing a particular role is defined via the designer defined `rcf` of each role.

$$\forall\ a{:}A\ r{:}R\ \text{capable}(a,r) = r.\text{rcf}(a) \tag{6}$$

While the *rcf* is user defined, it must conform to one OMACS constraint. To be *capable* of playing a given role in the current organization, an agent must *possess* all the capabilities that are *required* by that role.

$$\forall\ a{:}A,\ r{:}R\ \text{capable}(a,r)>0 \Leftrightarrow \text{requires}(r) \subseteq \{c \mid \text{possesses}(a,c)>0\} \tag{7}$$

The main goal of OMACS is to provide a mechanism to assign goals to agents in such a way that agents cooperate toward achieving some top-level goal. Intuitively, this mechanism should provide a way to assign the best agents to play the best roles in order to achieve these goals. Thus, OMACS has defined a *potential* function that captures the ability of an agent to play a role in order to achieve a specific goal. The potential function maps each agent-role-goal tuple to a real value ranging from 0 to 1, `potential: A × R × G → [0..1]`. Here, a 0 indicates that the agent-role-goal tuple cannot be used to achieve the goal while a non-zero value indicates how well an agent can play a specific role in order to achieve a goal. The potential of agent `a` to play role `r` to achieve goal `g` is defined by combining the capable and achieves functions.

$$\forall\ a{:}A\ r{:}R\ g{:}G\ \text{potential}(a,r,g) = \text{achieves}(r,g) * \text{capable}(a,r) \tag{8}$$

Organizational Agents

Organizational agents (OA) are organizations that function as agents in a higher-level organization. In OMACS, OAs enable hierarchical organizations, providing OMACS with scalability while supporting software engineering principles such as modularity. As agents, OAs may possess capabilities, coordinate with other agents, and be assigned to play roles. OAs represent an extension to the traditional Agent-Group-Role (AGR) model developed by Ferber and Gutknecht (1998) and are similar to concepts in the organizational metamodel proposed by Odell, Nodine, and Levy (2005). While OAs are an integral part of OMACS, they are not discussed further in the chapter and the reader is referred to DeLoach, Oyenan and Matson (2007) for more information.

ORGANIZATION-BASED MULTIAGENT SYSTEMS ENGINEERING (O-MASE)

In order to create OMACS-based systems, we have defined a methodology that allows us to use modern multiagent design approaches. This process is part of the O-MaSE Process Framework (Garcia-Ojeda, DeLoach, Robby, Oyenan & Valenzuela, 2007) as shown in Figure 2. The O-MaSE Process Framework is based on the OPEN Process Framework (OPF) (Firesmith & Henderson-Sellers, 2002) and uses the OPF metamodel in level M2, which defines processes in terms of Work Units (Activities, Tasks, and Techniques), Producers, and Work Products. Level M1 contains the definition of O-MaSE in the form of the O-MaSE metamodel, method fragments, and guidelines. Customized processes are instantiated at the M0 level for specific projects (a *process instance*).

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.

Figure 2. O-MaSE Process Framework (Garcia-Ojeda, et al., 2007)

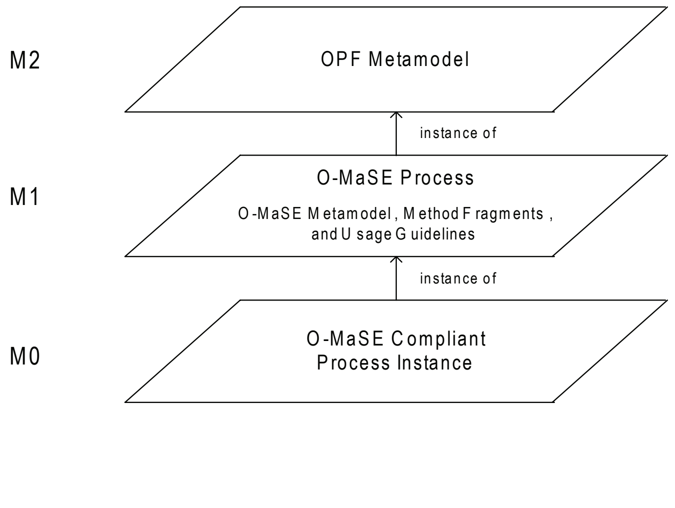
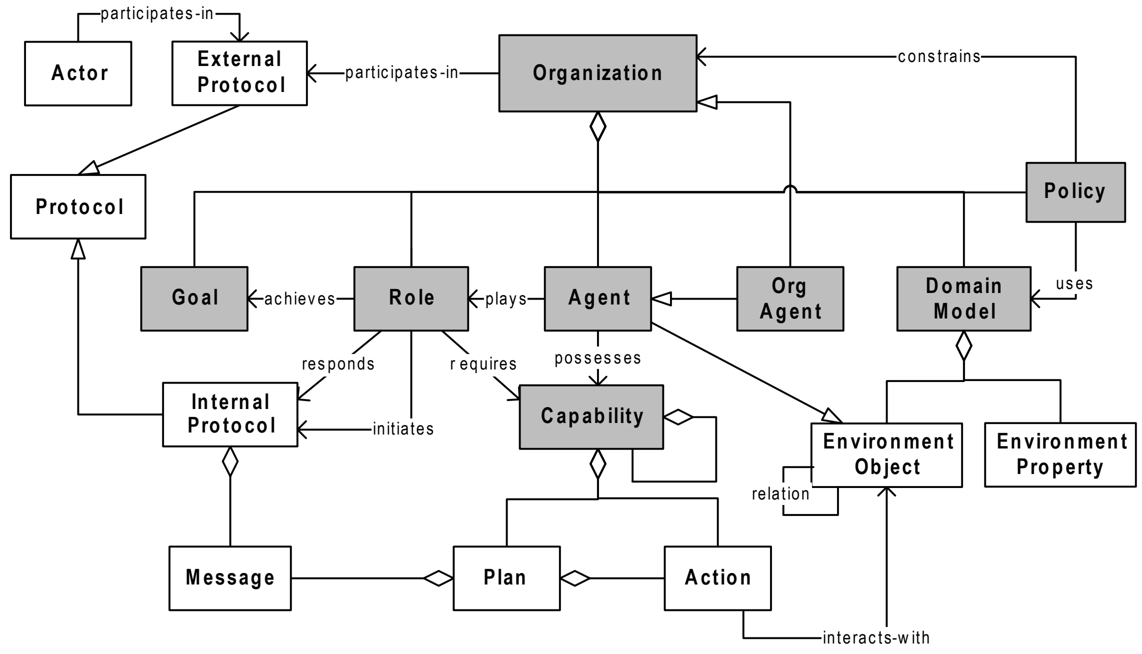

M2
OPF Metamodel
instance of
M1
O-MaSE Process
O-MaSE Metamodel, Method Fragments, and Usage Guidelines
instance of
M0
O-MaSE Compliant Process Instance

Figure 3. O-MaSE Metamodel



The O-MaSE methodology is supported by the agentTool III development environment[2], which is designed as a set of Eclipse plug-ins. agentTool includes a plug-in for each O-MaSE model and the agentTool Process Editor (APE), which was developed to support the design and definition of O-MaSE compliant processes. The APE plug-in is based on the Eclipse Process Framework and provides a process designer the ability to (1) extend O-MaSE with new tasks, models, or usage guidelines and (2) create new process instances by composing tasks, models, and producers from the O-MaSE method fragment library and then verifying that they meet process guidelines.

Metamodel

The metamodel defines the main concepts used in O-MaSE to design multiagent systems. It encapsulates the rules (grammar) of the notation and depicts those graphically using object-oriented concepts such as classes and relationships (Firesmith & Henderson-Sellers, 2002). The O-MaSE metamodel is based on the OMACS metamodel as shown in Figure 3, with the common parts of OMACS and O-MaSE shown in grey. This commonality allows us to easily create O-MaSE compliant process that can be used to develop OMACS systems.

Additional O-MaSE concepts include external actors, protocols, messages, plans, actions, and environment objects and properties. *External actors* are used to represent any agent, system, or user that lies outside the system being designed. *Protocols* define a set of allowable *message* sequences that may exist between external actors and an organization (external protocols) or between agents roles within an organization (internal protocols). *Plans* are a type of agent capability that define a sequence of *actions* or *messages* that can be used to achieve an agent's goal. The O-MaSE domain model consists of

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	

a set of environment objects that describe the objects in the system's domain and a set of environment properties that specify the principles and processes that govern the environment.

Method Fragments

As described above, the OPF metamodel defines Stages, Work Units, Work Products, Producers, and Languages for use in creating tailorable processes. In O-MaSE, the method fragments were derived from an extended version of the MaSE methodology (DeLoach, Wood & Sparkman, 2001). Currently, we have focused our efforts on the analysis and design and thus, we have defined three main activities in O-MaSE (1) requirements engineering, (2) analysis, and (3) design. As shown in Table 1, each Activity is decomposed into a set of Tasks with a corresponding set of Techniques that can be used to accomplish them. Each task also has an associated set of Work Products, Producers, and Languages.

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) (Miller, 2007). 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.

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.

The objective of the *Model Organizational Interfaces* task is to identify the organization's interfaces with external actors. There are three steps in modeling organizational interfaces including (1) identifying each external actor that may interact with the organization (system), (2) identifying how those actors interact with the organization, and (3) identifying the system level goals achieved by the organization.

Guidelines

Guidelines are used to describe how the method fragments can be combined in order to obtain O-MaSE compliant processes and are specified as a set of Work Unit pre and postconditions over Work Products (WP), Producers (P), Work Products states (see Table 2), and Work Product iterations (n) and versions (m). We formally specify guidelines as a tuple ⟨Ινπυτ, *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 that is guaranteed to be true after successfully performing a work unit (if the precondition was true).

Figure 4 shows the guidelines for the Model Goals task. Inputs include the Systems Description (SD), the Systems Requirement Specification (SRS), the Role Description 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 GM work product is obtained.

Figure 4. Model goal task constraints

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>)

POLICIES FOR OMACS-BASED SYSTEMS

Harmon, DeLoach and Robby (2007) define the notion of policies for controlling the adaptiveness of OMACS systems. While this work is similar to the more general specification of institutional norms using concepts from deontic logic (Vázquez-Salceda, et. al. 2004), it is more narrowly focused towards controlling the adaptive processes of closed multiagent organizations such as those supported by OMACS. For our work, we define *policies* as specifications of desired system behavior that can be used to specify either (1) what the system should do or (2) what the system should not do, which essentially equate to liveness and safety properties (Lamport, 1977). These policies are essential in designing adaptive systems that are both predictable and reliable (Robby, DeLoach & Kolesnikov, 2006). In early multiagent research, policies (or laws) were described as *social laws* that must always hold (Shoham and Tennenholtz, 1995); we refer to these as *law policies*. This concept is similar to *regimented norms*, which is one approach to the operationalization of norms that cannot (as opposed to should not) be violated (van der Torre, et. al. 2005). An example of a law policy for a multiagent conference management system might be that no agent (representing a committee member) may review more than five papers as represented in (1).

$$\forall a{:}A, \mathcal{L} : \Box\ (\text{sizeOf}(a.\text{reviews}) \leq 5) \qquad (1)$$

The $\mathcal{L}$ in the policy denotes that this is a law policy and the $\Box$ is the temporal logic operator that declares that this property must always hold. Intuitively, this policy states that no legal run of the system may be in a state where any agent is assigned to review more than five papers-. While the policy is helpful in ensuring no agent overworked, it restricts, possibly unnecessarily, the flexibility of the system to deal with unforeseen situations. Consider the case where more papers are submitted than were expected. If there are not enough agents, the entire system will fail. This is a common problem when using only law policies.

The more traditional use of the term policies in human organizations refers to a normative guidance as opposed to strict laws. Interpreting policies as laws in multiagent systems can lead to inflexibility and poor overall system performance. In contrast, human policies are often suspended to achieve the organization goals. To overcome the inflexibility of law policies, we created a weaker policy type called *guidance policies*, which are defined as policies that the system must adhere to *when possible*. An example of policy (1) converted to a guidance policy is shown in (2).

$$\forall a{:}A, \mathcal{G} : \Box\ (\text{sizeOf}(a.\text{reviews}) \leq 5) \qquad (2)$$

The $\mathcal{G}$: in the policy indicates that it is a guidance policy. Thus, while the specification for both types of policies is identical save the $\mathcal{L}$ or $\mathcal{G}$, the difference lies in the interpretation. Law policies ensure some property is *always* true (e.g. for safety or security), while guidance policies are used when the policy should remain true, when possible. With a guidance policy (2) in effect instead of law policy (1), the system can still achieve its goals when it gets more submissions than expected since it can assign more than five papers to an agent. However, when there are sufficient agents, however, policy (2) ensures that each agent performs five or fewer reviews.

In our research, the phrase *when possible* means that a system may only violate a guidance policy when there is no way the system can achieve its goal without violating the policy. This condition is easy to determine when the system only has to worry about violating a single guidance policy at a time. However, it is often the case that we have multiple guidance policies and the system may be forced to choose between two (or more) guidance policies to violate. To solve this dilemma in a non-random way, we define a partial ordering of guidance policies that allows the system designer to set precedence relationships between guidance policies. This partial order forms a lattice, such that a policy that is a parent of another policy in the lattice, is more-important-than its children. When there is clear precedence relation between the guidance policies being analyzed for violation, the least important policy is violated. However, when there is no clear precedence between policies (due to it only being a partial ordering), then one of the policies is chosen at random. Again, there is similar work related to conflict detection and resolution for norms (Kollingbaum et. al. 2007), which is more complex than our problem of guiding the adaptivity of organizations.

Harmon, DeLoach, and Robby (2007) showed that the guidance policies provide the best of both worlds. When applied in two multiagent systems, they resulted in systems with the same flexibility as similar systems with no policies while providing the improved performance of systems with comparable law policies. Essentially you get the benefit of law policies with none of the drawbacks.

REORGANIZATION ALGORITHMS AND ARCHITECTURES

During the development of several OMACS-based systems, we have developed several reorganization algorithms, most of which have been centralized. These algorithms range from sound and complete total reorganization algorithms to greedy algorithms (Zhong, 2006). As expected, sound and complete total reorganization algorithms are extremely slow, especially when the organization lacks any policies that limit the number of legal combinations of agents, roles, and goals (Zhong & DeLoach, 2006). The greedy algorithms also perform as expected, giving low computational overhead and producing generally adequate organizations. However, greedy algorithms typically must be tailored to the system being developed. We have also looked at learning reorganization algorithms (Kashyap, 2006).

Figure 5. General reorganization algorithm

```
function reorganize(oaf, $G_w$, $A_w$)
    1.   for each $g \in G_w$
    2.     for each role $r \in R$
    3.       if achieves(r,g) > 0 then $m \leftarrow m \cup \langle r,g \rangle$
    4.   ps $\leftarrow P_\Phi$(powerset(m))
    5.   for each agent $a \in A_w$
    6.     for each set $s \in ps$
    7.       if capable(a,s.r) then $pa \leftarrow pa \cup \langle a,s \rangle$
    8.   pas $\leftarrow$ powerset($P_\Phi$(pa))
    9.   for each assignment set i from c
    10.    for each assignment x from pa
    11.      $\Phi \leftarrow \Phi \cup \langle x.a, x.s_i \rangle$
    12.    if $P_\Phi(\Phi)$ is valid
    13.      if oaf($\Phi$) > best.score then best $\leftarrow \langle oaf(\Phi), \Phi \rangle$
    14.  return best.$\Phi$
```

A general purpose reorganization algorithm that produces an optimal solution with OMACS is shown in Figure 5. By *optimal*, we refer to the organization with the highest score as returned by the *oaf*. Therefore, finding the optimal organization score requires going through every potential assignment (every combination of goals, roles, and agents) and computing the organization score for each combination. In the algorithm, G_w refers to the goals that the organization is currently pursuing while A_w refers to the current set of agents that are available to be assigned. Lines 1 – 3 create all valid goal–role pairs from goals in G_w and the roles that are able to achieve that goal. Line 4 creates a powerset of all possible sets of goal–role pairs and then removes invalid sets using the assignment policies. Lines 5 – 7 create all the possible assignments `pa` between the agents from A_w and the goal–role pairs in each set in `ps`. Line 8 removes invalid assignments from `pa` based on the assignment policies and then creates the set of all possible assignment sets, `pas`. Lines 10 – 13 go through each possible assignment set to find the one with the best *oaf* score. Finally, line 14 returns the best (optimal) set of assignments.

Assignment policies can have significant effects on the time complexity of reorganization algorithms. With the simple policy that *agents can only play one role at a time*, algorithmic complexity can be reduced by an order of magnitude as described in Zhong (2006).

While this centralized reorganization algorithm is optimal, its performance makes it impractical in real systems. Therefore, we generally use application specific reorganization algorithms that have been tuned to include application-specific policies. For example, in many cases, we do not always want to do a complete reorganization every time a new goal appears. It is also often the case that we only want to assign a single goal to an agent at a time. These two application specific modifications make most realistic reorganization algorithms run extremely quickly and generally perform close to optimal.

Team Level Architecture

Our current Team-level Architecture is shown in Figure 6. There are two components in each agent: the Control Component (CC) and the Execution Component (EC). The CC performs all organization-level reasoning while the EC provides all the application specific reasoning.

To reason about the team's organizational state, each agent has an instance of the organization model and an appropriate set of reasoning algorithms and coordination protocols to work within their team to organize and reorganize as needed. The model and algorithms perform the organization-level reasoning

Figure 6. Organization-level architecture

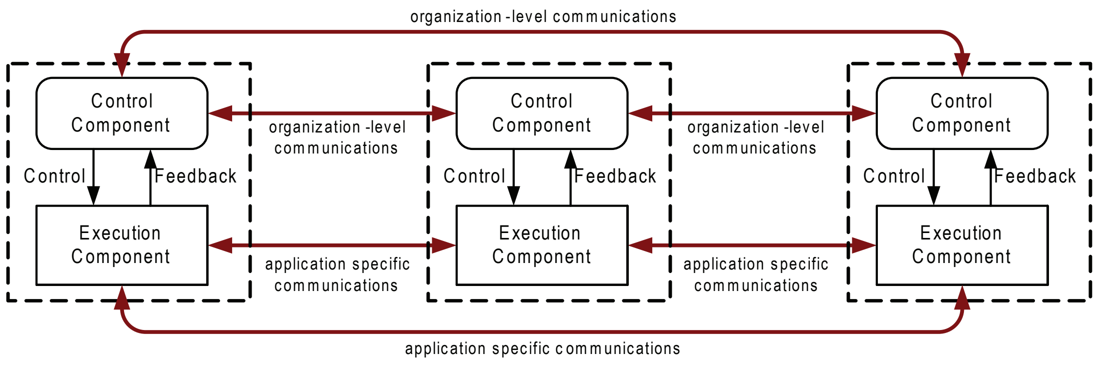
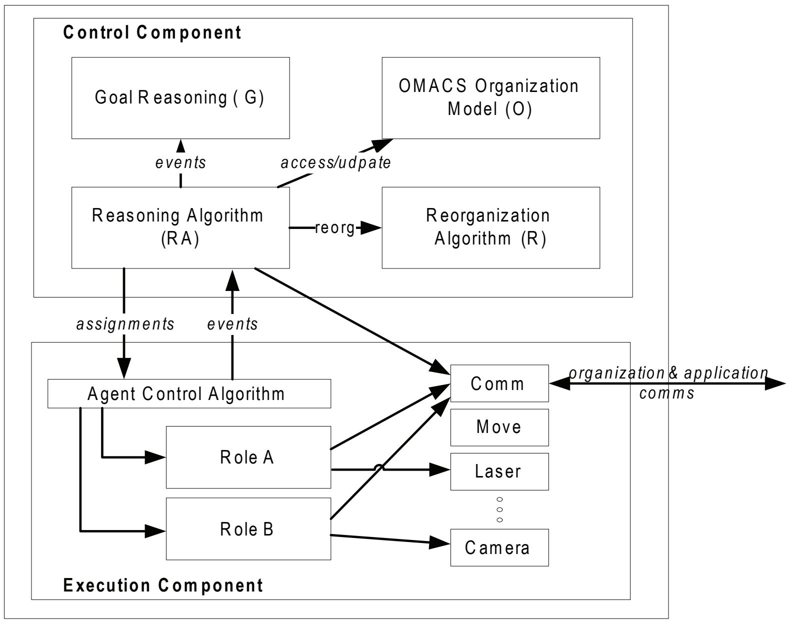

organization-level communications
Control Component
Control
Feedback
Execution Component
organization-level communications
application specific communications
Control Component
Control
Feedback
Execution Component
organization-level communications
application specific communications
Control Component
Control
Feedback
Execution Component
application specific communications

Figure 7. Organization-based agent architecture



via interaction between the individual CCs as shown in Figure 6. In essence, the combined organizational level of the individual agents forms a distributed organizational control mechanism. The agents cooperate to share organizational knowledge and define the appropriate team organization. In this scheme, the CC provides *control* to the agent about the roles it should be playing, the set of goals it is trying to satisfy, and other agents with which it may coordinate. The EC provides *feedback* based on possible events that are of interest to the organization (goal failure, goal completion, or application specific events that may trigger new goals, etc.). The agents also communicate at the application-specific level, which includes the coordination between agents that allows them to carry out their assigned roles.

Organization-Based Agent Architecture

Our current version of the internal architecture of our agents is called the Organization-based Agent (OBA) architecture, as shown in Figure 7. In the *Control Component*, there are four basic components:

Figure 8. Generic OMACS reasoning algorithm

```
loop
   e = getEvent()
   <$G_a$, $G_{additions}$, $G_{deletions}$> = G.occurred(e)
   O.setActiveGoalSet($G_a$)
   O.removeAssignments($G_{deletions}$)
   assignSet = R.reorg(O.A, $G_a$)
   O.addAssignmetns(assignSet)
   sendUpdates(assignSet)
end loop
```

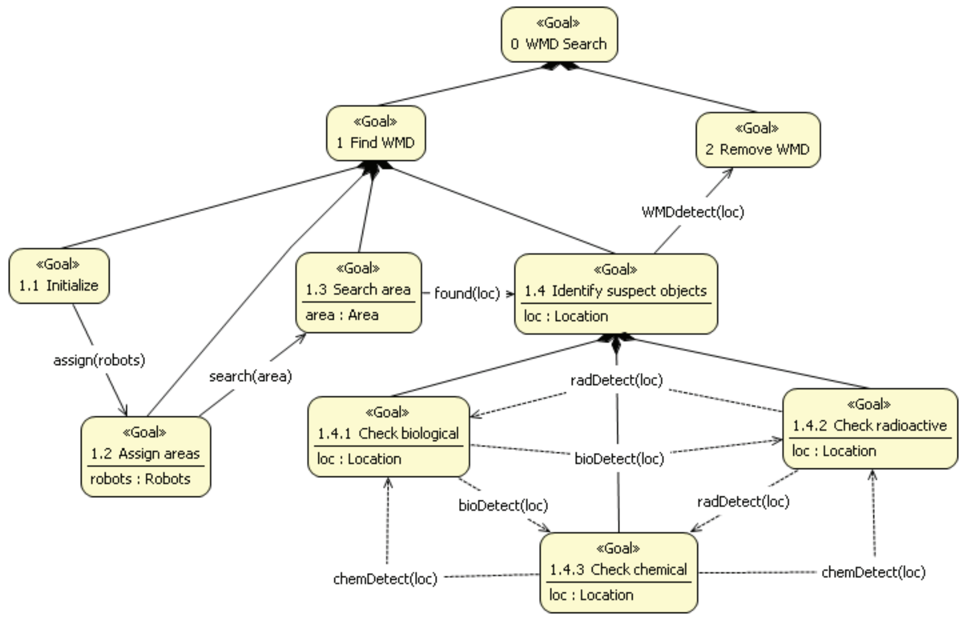
Goal Reasoning, Organization Model, Reorganization Algorithm (RA), and the overall Reasoning Algorithm. The Reasoning Algorithm ties the other three components together. First, the RA gets an event from either another agent or the Application Specific Roles/Behavior component. This event is passed to the Goal Reasoning component, which updates the goal instance tree and returns a tuple that include the new set of active goals, the additions to the active goal set, and the deletion from it. The RA then sets the active goal set in the Organization Model component and removes any assignments related to the goals deleted from the active goal set. If we need to do a total reorganization (via some undetermined process), we compute the new assignment set by calling the Reorganization Algorithm; if not, then we only ask for a reorganization using the unassigned agents and the new goals. The new assignment set is updated in the Organization Model and these assignment updates are sent to either other agents or to the Application Specific Roles/Behavior component.

One of the main design goals of the OBA architecture was enable as much reuse as possible. Since in an OMACS based system, much of the reasoning has standard forms (e.g., goal reasoning, reorganization, etc.), much of the code in the CC is in fact reusable. Knowledge about the OMACS entities can be inserted into standard implementations of both G and O, while a set of standard reorganization algorithms (R) are being developed for plug-and-play compatibility; these algorithms include centralize as well as distributed versions. While much of the EC is application specific, the Agent Control Algorithm can be reused and use of standard capabilities can make the job of developing the application specific roles much simpler.

Reasoning Algorithm

A high-level algorithm for the Reasoning Algorithm (RA) component is shown in Figure 8. Again, this algorithm is often tuned to the application being developed and thus we simply show it as a notional

Figure 9. WMD goal model



algorithm. When RA receives an event (which include organization events such as goal achievement/failure or the modification of agent capabilities as well as application specific events such as goal triggers) from either an external agent or from its Execution Component, it calls the *occurred* method of the Goal Reasoning component in order to get the new active goal set G_a, along with the set of additions to and deletions from G_a. The RA updates the active goal set and assignment set in the OMACS Organization Model. The RA then calls the *reorg* method of the Reorganization Algorithm to get a new set of assignments, which is then updated in the Organization Model. Finally, updates of the assignments are sent to other agents (if this is the centralized reorganization agent) and to its Execution Component.

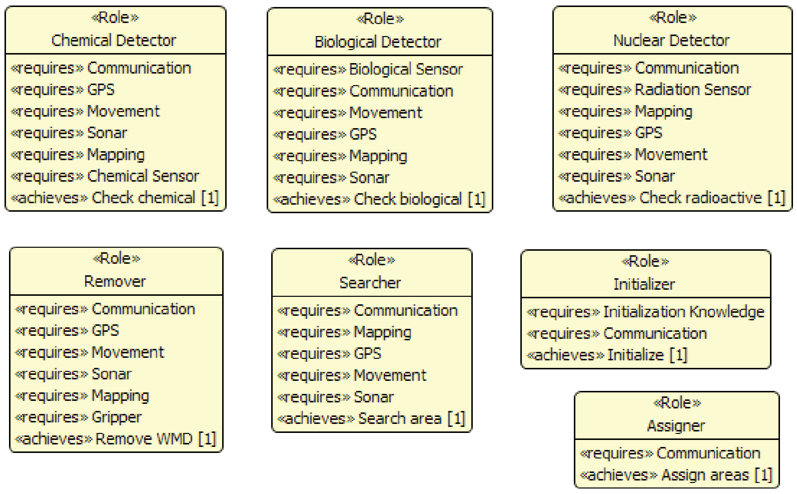
In the Execution Components, the assignments (which include the role to be played to achieve a specific goal) are received by the Agent Control Algorithm, which invokes the appropriate roles and for execution. When a role achieves its assigned goal, fails to achieve its assigned goal, or recognizes an application-specific event that impacts the goal model, the role passes the event to the Agent Control Algorithm, who in turns passes that back to the RA. In the OBA architecture, the various capabilities possessed by an agent are encoded in separate components and invoked directly by the roles.

EXAMPLE

To illustrate the use of O-MaSE to develop OMACS systems, we show the development of a Weapons of Mass Destruction (WMD) Cooperative Robotic Search system. While we do not have space to discuss the creation of the O-MaSE compatible process that we used to develop this system, suffice it to say that we took a very simple approach and developed only four models: a Goal Model, a Role Model, a set of Plan Models, and an Agent Class Model. To completely define the system, we probably should have included Capability-Action Models and a Domain Model as well. The four models we did develop are presented below.

The GMoDS Goal Model for the system is shown in Figure 9. The top goal is decomposed into two subgoals: *Find WMD* and *Remove WMD*. The *Find WMD* goal is decomposed further into four subgoals:

Figure 10. WMD role model



Initialize, *Assign areas*, Search *area*, and *Identify suspect objects*. The solid arrows between goals represent events that occur during achievement of the first goal, which triggers the instantiation of the second goal. Thus, for example, the *Initialize* goal will be assigned to some agent at system initialization. Essentially, each robot must register with the agent performing the *Initialize* goal. Once enough robots register or enough time has elapsed, that robot triggers, via the assign trigger with the list of robots, a new goal, *Assign areas*. A new robot is assigned to achieve the *Assign* areas goal using the list of robots. This robot divides the predefined search area into subareas based on the robots available to search. For each subarea, the robot triggers a new *Search area* goal (parameterized by subarea) via the search event. Each *Search area* goal is then assigned to a robot to be achieved. During the search, if one of the searching robots finds a suspicious object, it triggers a new Iden*t*ify suspect object goal by raising the event found with a specific loc (location) parameter. The *Identify* suspect object goal is actually decomposed into three subgoals, thus, when a found event is raised, the three subgoals – *Check biological*, *Check radioactive*, and *Check chemical* – are actually triggered. Anytime these three goals are triggered, they are assigned to robots with the appropriate sensors to be able to achieve the goals. The dashed arrows between the three subgoals represent *negative triggers*, which actually remove existing goal instances from the system. Thus, if the robot assigned to the goal *Check biological* determines that the object is actually a biological weapon, not only does it raise a *WMDdetect* event with the loc parameter, but it also raises a *bioDetect* event as well. The *bioDetect* event negatively triggers both the *Check radioactive* and *Check chemical* goals whose parameter, loc, matches the event. In essence, it tells the other robots assigned to check out the WMD that they no longer need to check that location. Finally, when a *WMDdetect* event is raised, it triggers a *Remove WMD* goal.

Once the GMoDS Goal Model is defined, a set of roles are identified for achieving the individual leaf goals. A Role Model for the WMD system is shown in Figure 10. Essentially, we have mapped each leaf goal to a single role that is responsible for achieving it as annotated by the «achieves» keyword; the value of the achieves function for the associated role-goal pair is shown in square brackets, which in this case are all 1's as there is only a single role capable of achieving each goal. The «requires» keyword in the role boxes indicate the capabilities that are required to play that role. This maps directly the notion of roles and required capabilities in the OMACS model. In a typical Role Model, it is often the case that there

Figure 11. Searcher plan model

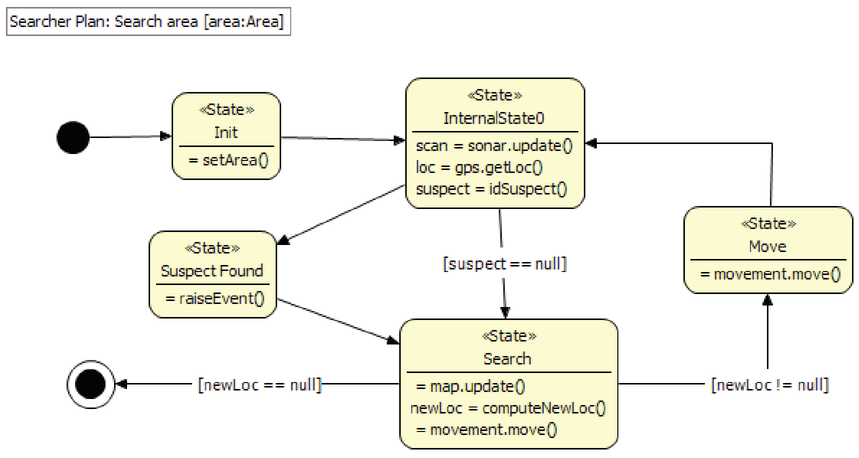
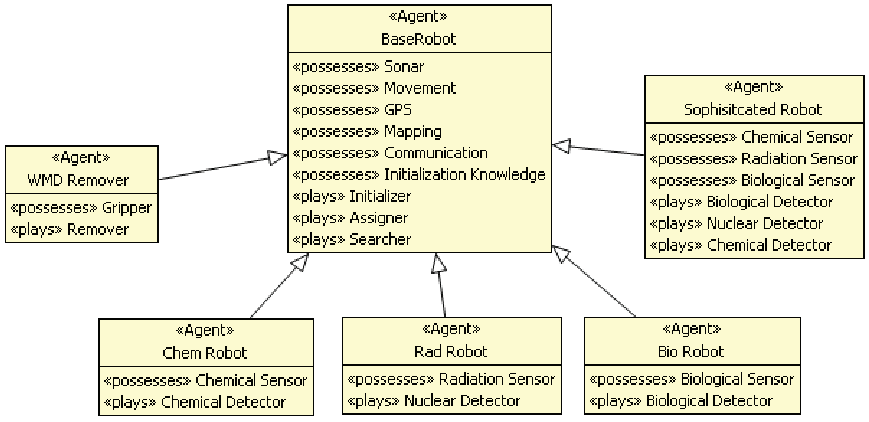


Figure 12. WMD agent class model


«Agent»
BaseRobot
«possesses» Sonar
«possesses» Movement
«possesses» GPS
«possesses» Mapping
«possesses» Communication
«possesses» Initialization Knowledge
«plays» Initializer
«plays» Assigner
«plays» Searcher
«Agent»
Sophisitcated Robot
«possesses» Chemical Sensor
«possesses» Radiation Sensor
«possesses» Biological Sensor
«plays» Biological Detector
«plays» Nuclear Detector
«plays» Chemical Detector
«Agent»
WMD Remover
«possesses» Gripper
«plays» Remover
«Agent»
Chem Robot
«possesses» Chemical Sensor
«plays» Chemical Detector
«Agent»
Rad Robot
«possesses» Radiation Sensor
«plays» Nuclear Detector
«Agent»
Bio Robot
«possesses» Biological Sensor
«plays» Biological Detector

would be directed relations between roles indicating that a protocol, and thus information sharing exists between two roles. However, the lack of explicit protocols between roles is not uncommon in OMACS-based systems as most of the information required for collaboration between roles is embedded in the goals that are instantiated. For instance, in Figure 9, when a search event triggers to *Search area* goal, it carries a parameter that defines the area to be searched. Thus, when an agent is assigned to play the *Searcher* role in order to achieve that goal, the goal is given to the agent and no explicit communication is required. The same thing happens when a suspicious object is found or a WMD is detected.

Once a role and its required capabilities have been identified, plans can be developed that define exactly how the capabilities are used to achieve a specific goal. In O-MaSE, a Plan Model is defined in terms of a finite state machine with sequentially executed actions in each state. The box in the upper left hand corner of Figure 11 shows that the Searcher Plan was developed in order to achieve the *Search area* goal, which has a parameter that specifies the search area. If there were explicit communication required, this communication would be shown on the state transitions as explicit *send* and *receive* actions. The actions called inside the states are either lower level function calls or direct calls to capability actions. Capability actions are denoted by the form `capabilityName.actionName(parameters)`.

After the roles and their required capabilities are identified, the types of agents, or agent classes, must be defined. In a cooperative robotic system, this corresponds to defining the types of robots that will be available along with their capabilities. The Agent Class Model for the WMD system is shown in Figure 12. There is a *BaseRobot* type, which defines a standard robot. By analyzing the capabilities it possesses, we can tell that the *BaseRobot* is capable of playing the *Searcher*, *Initializer*, and *Assigner* roles. The other five agent types inherit the capabilities (and thus the ability to play those three roles) from the *BaseRobot* class. Except for the *Sophisticated Robot* class, the other four specialized agent classes are designed to play a single specific role. However, because the *Sophisticated Robot* has all three types of WMD sensors, it can play *Chemical*, *Biological*, and *Nuclear Detector* roles.

At this point, we have defined all the basic OMACS elements except for policies and a domain model, which are not included due to space limitations. It would also be helpful to use the Capability-Action Model to define the effects of the actions, as used in the plans, on domain model elements. We implemented this system in our Cooperative Robotic Simulator (CROS) to demonstrate the effectiveness of the OMACS and GMoDS models. The simulation worked as expected and showed how the team of robots could effectively assign and reassign robots to play specific roles to achieve goals as they were instantiated based on the events that occurred in the environment. In the next section, we walk through a small part of a scenario to illustrate the details of OMACS operation.

Scenario

To illustrate the basic operation of OMACS, we walk through a simple scenario using the WMD system defined above. For this example, we have four agent types: two *BaseRobots* (BR1, BR2), one *Sophisticated Robot* (SR), one *Chem Robot* (CR), and one *WMD Remover Robot* (RR). We assume each agent is designed using the OBA architecture, although the exact reorganization algorithm used does not need to be specified. For simplicity, the reader may wish to assume that a centralized version of the algorithm is being used where all agents forward events and receive assignments directly from the central decision maker. In this case, the term *the organization* refers to the central decision maker. Since we only have a single role capable of be used to achieve each goal, we assume that the `achieves` value of each role-goal pair is either 0 or 1, depending on whether or not the role can be used to achieve a goal. Also, for simplicity, we assume that capabilities do not degrade and all robots (with a single exception) possess the same quality of capabilities. The exception is that the CR's *Chemical Sensor* capability is better than the SR's *Chemical Sensor* and thus `possesses(CR,Chemical Sensor)=1` while `possesses(SR,Chemical Sensor)=0.5`. Using the default `rcf` as defined in equation (1) and the `potential` function as defined in (8), we see that the potential of all agent, role, goal triples are either 0 or 1 with the exception that `potential(SR, Chemical Detector, Check chemical) =` .917 (the `rcf` averages the possesses of the six required capabilities for the *Chemical Detector* role).

When the system starts, some leader election algorithm decides which robot becomes the central decision maker and all robots in the organization register with it. Then, the *Initialize* goal is instantiated by the organization (as it is not triggered by an application specific event) and placed in the active goal set (G_A). The organization's reorganization algorithm is invoked to decide which robot is assigned the *Initialize* goal. While all robots could play the *Initializer* role in order to achieve the *Initialize* goal (since all have robots have *Initialization Knowledge* and *Communication* capabilities), we assume it is assigned to robot RR and the initial state of the organization is:

G_A = {Initialize}
Φ = {<RR, Initializer, Initialize>}

RR follows the plan associated with the *Initializer* role to determine the number of robots available who are capable of playing the *Searcher* role. Then, RR raises the *assign* event (parameterized with a list of the available robots) to the organization. The organization uses the Goal Reasoning component (G) to determine that an *Assign areas* goal (parameterized with the list of robots) should be instantiated and placed in G_A. At that point, RR has achieved the *Initialize* goal and raises an *achieved* event. At this point, the organization again calls its reorganization algorithm to determine how to assign the *Assign areas* goal. Since the only role capable of being used by an agent to achieve the *Assign areas* goal is the *Assigner* role and all robots are capable of playing it with equal capability, one is chosen at random. We assume the assignment is given to SR and the state of the organization is

G_A = {Assign Areas(robots)}
Φ = {<SR, Assigner, Assign Areas(robots)>}

SR performs the *Assigner* role resulting in the division of the search area into five separate search areas (based on the number of robots that can play the *Searcher* role). The SR raises a *search(area)* event to the organization for each of the five search areas a1, a2, a3, a4, and a5. These five events cause the instantiation of five *Search area* goals, each having a unique parameter based on the unique search areas. At this point, SR has successfully completed playing the *Assigner* role and raises an *achieved* event resulting in the removal of the *Assign areas* goal from G_A and the assignment of SR from Φ. At this point, the organization runs its reorganization algorithm and produces the following set of assignments.

G_A = {Search area(a1), Search area(a2), Search area(a3), Search area(a4), Search area(a5)}
Φ = {<BR1, Searcher, Search area(a1)>, <BR2, Searcher, Search area(a2)>,
<SR, Searcher, Search area(a3)>, <CR, Searcher, Search area(a4)>,
<RR, Searcher, Search area(a5)>}

The robots perform their assignments until a new event occurs; possible events include a goal being achieved, a change in capability, or the discovery of a suspicious object. First, we assume that BR1's sonar fails and it can no longer play the *Searcher* role and thus cannot achieve the *Search area*(a1) goal. BR1 raises a *failed* event for goal *Search area*(a1) and the organizer runs its reorganization algorithm to reassign that goal. Depending on the actual reorganization algorithm in used, a robot could be picked at random, or application specific knowledge such as the physical relationship of the areas to be searched could be used to make the assignment. For our purposes, we assume BR2 is assigned to this goal resulting in the following organizational state.

G_A = {Search area(a1), Search area(a2), Search area(a3), Search area(a4), Search area(a5)}
Φ = {<BR2, Searcher, Search area(a1)>, <BR2, Searcher, Search area(a2)>,
<SR, Searcher, Search area(a3)>, <CR, Searcher, Search area(a4)>,
<RR, Searcher, Search area(a5)>}

Notice that BR2 is assigned to search two areas since we have not specified any policies that restrict the assignment. If we had a policy that a robot could play a single role at a time, then the goal *Search area(a1)* could not be assigned to any robot and it would have to wait until a robot became free. Next, assume BR2 finds a suspicious object, resulting in it raising a *found(loc)* event, where loc represents the x,y location of the object. This event causes the organization to instantiate the three sub-goals of the *Identify suspect object* goal, which are *Check biological*, *Check chemical*, and *Check radioactive*. The only two robots that can achieve these goals are SR and CR. As far as the *Check biological* and *Check radioactive* goals, either of the two robots are equivalent in that their possesses score for five required capabilities (Communication, Movement, GPS, Mapping, Sonar and either Biological or Radiation Sensor) for the associated roles (Biological Detector and Nuclear Detector) yield identical potential values. However, for the *Check radioactive* goal, the difference in the Chemical Sensor capability produces a potential value of .917 for SR and 1.0 for CR. Therefore, while the reorganization algorithm may assigned either SR or CR to the *Check biological* and *Check radioactive* roles, it will always assign CR to play the Chemical Detector role in order to achieve the *Check chemical* goal since its potential value shows that it is the most capable robot. One possible state of the organization is shown below.

G_A = {Search area(a1), Search area(a2), Search area(a3), Search area(a4), Search area(a5)}
Φ = {<BR2, Searcher, Search area(a1)>, <BR2, Searcher, Search area(a2)>,
<SR, Searcher, Search area(a3)>, <CR, Searcher, Search area(a4)>,
<RR, Searcher, Search area(a5)>, <SR, Biological Detector, Check biological>,
<CR, Radioactive Detector, Check radioactive>,
<CR, Chemical Detector, Check chemical>,
}

CONCLUSION

In this chapter, we presented the OMACS model as a metamodel for designing complex, adaptive systems that can reorganize at runtime in the face of a dynamic environment. While the OMACS model itself is a useful tool, we believe that in order to make its application more straightforward, a suite of technologies that support its use must also be available. Specifically, the O-MaSE framework is a key element of this support suite. Using O-MaSE, developers have a set of models and techniques they can use to design the key elements of an OMACS-based system. The fact that O-MaSE was designed to be tailorable to a large number of situations makes its especially useful for OMACS as well as other multiagent approaches.

The use of policies in adaptive systems will be necessary in order to provide some level of control over the system adaptation. Experience has shown that emergent behavior is often detrimental as opposed to being useful. However, the use of strict law policies often has the effect of limiting the flexibility (and thus emergent behavior) too much. We have attempted to overcome this limitation by introducing the formal notion of guidance policies which provides the performance enhancing effects of law policies while not unnecessarily restricting the ability of the system to adapt.

Finally, we presented the OBA architecture, which we use to implement OMACS-based systems. While this is not the only architecture that can be used to implement OMACS-based systems, we have been able to demonstrate its effectiveness on several multiagent and cooperative robotic systems. The goal of the architecture is to provide a modular framework that can reuse as much OMACS-specific code as possible.

FUTURE WORK

There are several areas where additional work is needed. We will highlight three: architecture and algorithm development, software engineering for complex, adaptive systems, and integration of humans into artificial agent teams. While the architectures and algorithms presented in this chapter have been shown to be effective in developing OMACS-based systems, additional work needs to be done toward creating a component-based, plug-and-play agent architecture. While we have used the GMoDS, other goal reasoning models can be used. The same is true for reorganization algorithms. Due to the algorithmic complexity of a general-purpose optimal reassignment algorithm, algorithms that use general or application specific policies to prune the search area would be beneficial. In fact, it is common to have the general policy that an agent may only play one role at a time. Simply embedding this into the reorganization algorithm reduces the algorithm complexity significantly. We are also investigating algorithms that look at local reorganization techniques to reduce the effect of a total reorganization. Having the ability to choose reorganization algorithms from a library would be extremely helpful.

A second area of future work is continued work in the area of software engineering of agent-oriented systems. Specifically, O-MaSE needs to continue its evolution in terms of the number and diversity of models and techniques for designing OMACS-based systems. Because it is impossible to test all possible configurations of OMACS-based systems, developers will need help in trying to get a handle on all the possibly ways the system can reconfigure and how that might affect system performance. We have started working on *predictive metrics*, where we have applied model checking techniques to determine the effectiveness of design and their impact on system adaptivity. In Robby, DeLoach and Kolesnikov (2006), we developed two metrics used to predict system flexibility: Covering Percentage and Coarse Redundancy. Each metric computes the increase or decrease in the flexibility of a design as compared to the maximum flexibility inherent in the system Goal Model. We are currently investigating metrics that tell us about the criticality of individual goals, roles and agents; however, more work needs to be done.

While completely autonomous teams of agents are one possible solution to complex, adaptive systems, we believe that it is much more likely that a mix of humans, hardware agents, and software agents will be necessary to achieve the results that will be expected of future systems. Therefore, there is research needed into how to include human agents into OMACS organizations. When determining the proper human personnel to deploy, many factors are considered such as training, expertise, and current work cycles, etc. While these factors can be looked at as OMACS capabilities, exactly how we might do that is open to speculation. On approach would be to develop specific functions for *human performance shaping factors* that can be employed to determine the team organization and predict human performance degradations. A second area requiring research is to develop interaction techniques to allow humans and artificial agents to work together.

ACKNOWLEDGMENT

This work was supported by grants from the US National Science Foundation grant number 0347545 and by the US Air Force Office of Scientific Research grant number FA9550-06-1-0058.

REFERENCES

Bernon, C., Cossentino, M., Gleizes, M., Turci, P., & Zambonelli, F. (2005). A study of some multi-agent meta-models. In J. Odell, P. Giorgini, J. Müller (Eds.) *Agent-Oriented Software Engineering V: 5th Intl. Workshop*, Lecture Notes in Computer Science 3382, Springer: Berlin, (pp. 62-77).

Carley, K. M. (1995). Computational and mathematical organization theory: perspective and directions. *Computational and Mathematical Organization Theory*, *1*(1), 39-56.

Carley, K. M. (1998). Organizational adaptation. *Annals of operations research*, *75*, 25-47.

Carley, K. M., & Gasser, L. (1999). Computational organization theory. In G. Weiss (ed.), *Multiagent systems: A modern approach to distributed artificial intelligence*. MIT Press: Cambridge, MA.

Cohen, P. R., & Levesque, H. J. (1991). Teamwork. *Nous*, *25*(4), 487-512.

DeLoach, S. A., & Wood, M. F. (2001). Developing multiagent systems with agentTool. In C. Castelfranchi, Y. Lesperance (Eds.). *Intelligent agents vii. Agent theories architectures and languages, 7th international workshop*. Lecture Notes in Computer Science 1986. Springer: Berlin.

DeLoach, S. A., Oyenan, W., & Matson, E. T. (2008). A capabilities-based theory of artificial organizations. *Journal of Autonomous Agents and Multiagent Systems, 16*(1), 13-56.

DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multi-agent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering*, *11*(3), 231-258.

Dignum, V. (2004). *A model for organizational interaction: Based on agents, founded in logic*. PhD thesis, Utrecht University.

Dignum, V. Vázquez-Salceda, J., & Dignum, F. (2004). Omni: Introducing social structure, norms and ontologies into agent organizations. In *Programming multi-agent systems: 2nd international workshop*, Lecture Notes in Computer Science 3346, Springer: Berlin. (pp.181–198).

Ferber, J., & Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems, in *Proceedings of the third international conference on multiagent systems*, IEEE Computer Society: Washington D.C. 128-135.

Ferber, J., & Gutknecht, O., & Michel, F. (2003). From agents to organizations: An organizational view of multi-agent systems. In P. Giorgini, J.P. Muller, J. Odell (Eds.), *Software engineering for multi-agent systems IV: Research issues and practical applications series*, Lecture Notes in Computer Science 2935, Springer: Berlin. 214-230. 10.1007/b95187.

Firesmith, D. G., & Henderson-Sellers, B. (2002). *The OPEN process framework: An introduction*. Harlow, England: Addison-Wesley.

Garcia-Ojeda, J. C., DeLoach, S. A., Robby, Oyenan, W. H., & Valenzuela, J. (2008). O-MaSE: A customizable approach to developing multiagent development processes. In Michael Luck (eds.), *Agent-Oriented Software Engineering VIII: The 8th International Workshop on Agent Oriented Software Engineering (AOSE 2007)*, Lecture Notes in Computer Science 4951, Springer: Berlin, 1-15.

Grosz, B. J., & Kraus, S. (1996). Collaborative plans for complex group action. *Artificial intelligence*, *86*(2), 269-357.

Harmon, S. J., DeLoach, S. A., & Robby. (2007, October). Trace-based specification of law and guidance policies for multiagent systems. *Paper presented at The Eighth Annual International Workshop Engineering Societies in the Agents World*, Athens, Greece.

Henderson-Sellers, B. (2005). Evaluating the Feasibility of Method Engineering for the Creation of Agent-Oriented Methodologies. In M. Pechoucek, P. Petta, L.Varga (eds) *Multi-agent systems and applications iv: 4th international central and eastern European conference on multi-agent systems.* Lecture Notes in Computer Science 3690, Springer: Berlin. (pp. 142-152).

Hübner, J., Sichman, & Boissier, J. O. (2002). MOISE+: Towards a Structural, Functional and Deontic Model for MAS Organization. In *Proceedings of the 1st international joint conference on autonomous agents and multi-agent systems* (pp. 501-502).

Jennings, N. R. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial intelligence*, *75*(2), 195-240.

Juan, T., & Sterling, L. (2004). The roadmap meta-model for intelligent adaptive multi-agent systems in open environments. In *Proceedings 2nd international conference on autonomous agents and multi-agent systems* (pp. 53-68), Lecture Notes in Computer Science 2935, Springer: Berlin.

Kashyap, S. (2006). *Reorganization in multiagent systems*, Master's Thesis, Kansas State University.

Kinny, D., Ljungberg, M., Rao, A. S., Sonenberg, L., Tidhar, E., & Werner, G. E. (1992). Planned team activity. In C. Castelfranchi, E. Werner (eds.), *Artificial social systems - Selected papers from the fourth European workshop on modeling autonomous agents in a multi-agent world* (pp. 226-256), Lecture Notes in Artificial Intelligence 830, Springer: Berlin.

Kollingbaum, M. J., Norman, T. J., Preece, A., & Sleeman, D. (2007). Norm Conflicts and Inconsistencies in Virtual Organisations. In J. G. Carbonell and J. Siekmann (Eds.) *Coordination, Organizations, Institutions, and Norms in Agent Systems II.* Lecture Notes in Computer Science 4386, Springer: Berlin, 245-258.

Lamport, L. (1977). Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, *3*(2), 125-143.

Miller, M. A. (2007). *Goal model for dynamic systems.* Master's Thesis, Kansas State University.

Odell, J., Nodine, M., & Levy, R. (2005). A metamodel for agents, roles, and groups, In J. Odell, P. Giorgini, Müller, J. (Eds.), *Agent-oriented software engineering V: The fifth international workshop* (pp. 78-92). Lecture Notes in Computer Science 3382 Springer: Berlin. 10.1007/b105022**.**

Peng, Z., & Peng, H. (2005) An improved agent/group/role meta-model for building multi-agent systems. In *Proceedings of the 2005 international conference on machine learning and cybernetics* (pp. 287-292).

DeLoach, S. A., & Kolesnikov, V. A. (2006). Using design metrics for predicting system flexibility, in L. Baresi, R. Heckel (eds.), *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science 3922, (pp. 184-198). Springer: Berlin. 10.1007/11693017_15.

Russell, S., & Norvig, P. (2003). *Artificial intelligence a modern approach*. Upper Saddle River, NJ: Pearson Education.

Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, *73*(1-2), 231-252.

Tambe, M. (1997). Towards flexible teamwork. *Journal of AI research*, *7*, 83-124.

van der Torre, L., Hulstijn, J., Dastani, M., Broersen, J. (2005). Specifying Multiagent Organizations. In J. G. Carbonell, J. Siekmann (Eds.) *Proceedings of the Seventh Workshop on Deontic Logic in Computer Science (Deon'2004)*. Lecture Notes in Computer Science 3065, Springer: Berlin, (pp. 243-257).

van Lamsweerde, A., Darimont, R., & Letier, E. (1998). Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, *24*(11), 908-926.

Vázquez-Salceda, J., Aldewereld, H., & Dignum, F. (2004). Implementing norms in multi-agent systems in J. G. Carbonell and J. Siekmann (Eds.) *Multiagent System Technologies*, Lecture Notes in Artificial Intelligence 3187, Springer: Berlin, 313-327.

Vázquez-Salceda, J., & Dignum, F. (2003). Modelling electronic organizations. In V. Marik, J. Muller, M. Pechoucek (eds.), *Multi-agent systems and applications iii* (pp. 584–593). Lecture Notes in Artificial Intelligence 2691, Springer: Berlin, 584-593..

Zambonelli, F., Jennings, N. R., & Wooldridge, M. J. (2001). Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering*, *11*(3), 303-328.

Zhong, C. (2006). *An investigation of reorganization algorithms*. MS Thesis, Kansas State University, 2006.

Zhong, C., & DeLoach, S. A. (2006). An Investigation of Reorganization Algorithms, In Hamid R. Arabnia (Ed.): *Proceedings of the International Conference on Artificial Intelligence*, 2 (pp. 514-517). CSREA Press.

ADDITIONAL READING

Aldewereld, H. Dignum, F. Garcia-Camino, A. Noriega, P. Rodriguez-Aguilar, J.A. and Sierra, C. (2006). Operationalisation of norms for usage in electronic institutions. In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. 223—225.

Artikis A. Sergot M. and Pitt J. (2007). Specifying Norm-Governed Computational Societies. ACM Transactions on Computational Logic (in press).

Bergenti F., Gleizes M.P., & Zambonelli F. (Eds.). (2004). *Methodologies and software engineering for agent systems: The agent-oriented software engineering handbook*. Kluwer Academic Publishers: Norwell, MA.

Bernon C., Cossentino M., & Pavón J. (2005). Agent oriented software engineering. *The Knowledge Engineering Review*. 20, 99–116.

Beydoun G., Gonzalez-Perez C., Henderson-Sellers B., Low G. (2005). 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, 3194. Springer-Verlag: Berlin, 126–142.

Cossentino M., Gaglio S., Henderson-Sellers B., and Seidita V. (2006). A metamodelling-based approach for method fragment comparison. In J. Krogstie, T.A. Halpin, H.A. Proper (Eds.) *Proceedings of the 11th international workshop on exploring modeling methods in systems analysis and design*, Namur University Press: Namur, Belgium, 419-432.

Coutinho, L., Sichman, J., Boissier, O. (2005). Modeling Organization in MAS: A Comparison Of Models, in *Proceedings of the 1st.Workshop on Software Engineering for Agent-Oriented Systems* Uberlândia, Brazil.

DeLoach S.A., Valenzuela J.L. (2007). An agent-environment interaction model. In L. Padgham, F. Zambonelli (Eds): *Agent oriented software engineering VII*. Lecture Notes in Computer Science, 4405. Springer-Verlag, 1-18.

DeLoach, S.A. (2001). Analysis and Design using MaSE and agentTool, In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science* Conference. Oxford, Ohio.

DeLoach, S.A., (2006). Multiagent systems engineering of organization-based multiagent systems. In A. Garcia, et. al. (Eds). *Software engineering for multi-agent systems IV: Research issues and practical applications series*. Lecture Notes in Computer Science 3914, Springer: Berlin, 109-125.

European Institute for Research and Strategic Studies in Telecommunications. (2000). *MESSAGE: Methodology for engineering systems of software agents, EURESCOM project P907-GI*. Heidelberg, Germany: R. Evans, P. Kearney, J. Stark, G. Caire, F.J. Garijo, J.J Gomez Sanz, J. Pavon, F. Leal, P. Chainho, P. Massonet.

Henderson-Sellers, B., Giorgini P. (Eds.). (2005). *Agent-oriented methodologies*. Idea Group: New York.

Horling, B., Lesser, V. (2005). Using ODML to model multi-agent organizations. In *Proceedings of the IEE/WIC/ACM international conference on intelligent agent technology*. 72-80, DOI: 10.1109/IAT.2005.139.

Hübner, J. Sichman, J. & Boissier, O. (2002). MOISE+: Towards a structural, functional and deontic model for MAS organization. In *Proceedings of the 1st international joint conference on autonomous agents and multi-agent systems*. ACM Press: Bologna, Italy. 501-502.

Jennings, N.R., (1993). Commitments and conventions: The foundation of coordination in multiagent systems. *Knowledge Engineering Review*, 8(3), 223-250.

Knottenbelt, J.A. (2001). Policies for Agent Systems. MS Thesis. Department of Computing, Imperial College London.

Matson, E., DeLoach, S.A. (2004). Integrating robotic sensor and effector capabilities with multi-agent organizations. *Proceedings of the international conference on artificial intelligence*. CSREA Press, 481-488.

Matson, E., DeLoach, S.A., (2003). An organization-based adaptive information system for battlefield situational analysis. In *Proceedings of the international conference on integration of knowledge intensive multi-agent systems*. 46-51. DOI: 10.1109/KIMAS.2003.1245020.

Nair, R. Tambe, M. & Marsella, S. (2002). Team formation for reformation. In *Proceedings of the AAAI spring symposium on intelligent distributed and embedded systems*.

Robby, Dwyer, M.B. & Hatcliff, J. (2003). Bogor: An extensible and highly-modular model checking framework. In *Proceedings of the 4th joint meeting of the European software engineering conference and ACM sigsoft symposium on the foundations of software engineering*. ACM: New York. 267 – 276.

Seidita, V., Cossentino, M., Gaglio, S. (2006). A repository of fragments for agent systems design. In F. De Paoli, A. Di Stefano, A. Omicini, C. Santoro (Eds.) *Proceedings of the 7th workshop from objects to agents*. Catania, Italy. 130–137.

Sycara, K. (1998). Multiagent Systems, *AI Magazine*, 19 (2). 79-93.

Turner, R.M. & Turner, E.H. (2001). A two-level, protocol-based approach to controlling autonomous oceanographic sampling networks. *IEEE Journal of Oceanic Engineering*, 26(4), 654-666.

Uszok, A. and Bradshaw, J. and Jeffers, R. and Suri, N. and Hayes, P. and Breedy, M. and Bunch, L. and Johnson, M. and Kulkarni, S. and Lott, J. (2003). KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement. In POLICY 2003: IEEE 4th International Workshop on Policies for Distributed Systems and Networks. IEEE, 93-96.

Wagner, G. (2001). Agent-oriented analysis and design of organisational information systems. In J. Barzdins, A. Caplinskas (Eds.). *Databases and Information Systems*. Kluwer Academic Publishers: Norwell, MA. 111-124.

Vasconcelos W. Kollingbaum M. and Norman T. (2007). Resolving conflict and inconsistency in norm-regulated virtual organizations. In AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM Press : New York, NY, USA. 1-8.

Wooldridge, M. Jennings, N.R., D. Kinny. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*. 3(3), 285-312.

Zambonelli, F. Jennings, N.R., Wooldridge, M.J. (2001). Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini, M.J. Wooldridge (Eds.) *Agent-oriented software engineering – Proceedings of the first international workshop on agent-oriented software engineering*. Lecture Notes in Computer Science 1957, Springer: Berlin. 207-222.

Zambonelli, F., Jennings, N.R., Omicini, A., Wooldridge, M.J. (2001). Agent-oriented software engineering for internet applications. In A. Omicini, F. Zambonelli, M. Klusch, R. Tolksdorf (Eds.) *Coordination of internet agents: models, technologies, and applications*. Springer: Berlin. 326-346.

KEY TERMS

Agent: An entity that perceives and can perform actions upon its environment, which includes humans as well as artificial (hardware or software) entities.

Capability: An atomic entities used to define a skill or capacity of agents.

Goal: A desirable state of the world or the objective of a computational process.

Goal Model for Dynamic Systems (GMoDS): a set of models for capturing system level goals and for using those goals at runtime to allow the system to adapt to dynamic problems and environments.

Organization Model for Adaptive Computational Systems (OMACS): A model that defines the knowledge needed about a system's structure and capabilities to allow it to reorganize at runtime in the face of a changing environment and its agent's capabilities.

Organization-Based Agent (OBA) Architecture: an architecture for implementing agents capable of reasoning within an OMACS organization.

Organization-Based Multiagent Systems Engineering (O-MaSE): A framework for creating compliant processes that support the development of OMACS-based systems

Policy: A specification of desired system behavior that can be used to specify either (1) what the system should do or (2) what the system should not do.

Reorganization: The transition from one organizational state to another.

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