Engineering Organization-Based Multiagent Systems

Scott A. DeLoach

Multiagent and Cooperative Robotics Laboratory,
Department of Computing and Information Sciences, Kansas State University,
234 Nichols Hall, Manhattan, Kansas 66506, USA
sdeloach@cis.ksu.edu
http://www.cis.ksu.edu/~sdeloach/

Abstract. In this paper, we examine the Multiagent Systems Engineering (MaSE) methodology and its applicability to developing organization-based multiagent systems, which are especially relevant to context aware systems. We discuss the inherent shortcomings of MaSE and then present our approach to modeling the concepts required for organizations including goals, roles, agents, capabilities, and the assignment of agents to roles. Finally, we extend MaSE to allow it to overcome its inherent shortcomings and capture the organizational concepts defined in our organization metamodel.

1 Introduction

Recent trends in multiagent systems are toward the explicit design and use of organizations, which allow heterogeneous agents to work together within well-defined roles to achieve individual and system level goals [8], [19]. When focusing on team goals, organizations allow agents to work together by using individual agents to perform the tasks for which they are best suited. When emphasizing an individual agent's goals, organizations provide the structure and rules that allow agents to find and carry out collaborative tasks with other, previously unknown agents, to the mutual benefit of each agent.

In situations where the nature of the application environment makes teams susceptible to individual failures, these failures can significantly reduce the ability of the team to accomplish its goal. Unfortunately, most multiagent teams are designed to work within a limited set of configurations. Even when the team possesses the ability to accomplish its goal, it may be constrained by its own knowledge of team member's capabilities. In most multiagent methodologies, the system designer analyzes the possible organizational structure and then designs one organization that will suffice for most anticipated scenarios. Unfortunately, in dynamic applications where the environment as well as the agents may change, a designer can rarely account for, or even consider, all possible situations. To overcome these problems, we are investigating techniques that allow multiagent teams to design their own organization at runtime [7]. In essence, we propose to provide the team with the organizational knowledge and let the team define its own organization based on its current context, goals and team capabilities.

In this paper, we present a proposal to extend the Multiagent Systems Engineering (MaSE) methodology toward the analysis and design of multiagent organizations. While

A. Garcia et al. (Eds.): SELMAS 2005, LNCS 3914, pp. 109 – 125, 2006.

MaSE already incorporates many of the required organizational concepts such as goals, roles, laws, and the relations between these entities, it cannot currently be used to completely define a multiagent organization. Most importantly, we must extend MaSE with the notion of capabilities, upon which the definition of roles is based. We also add some specific relationships between these capabilities and existing MaSE concepts.

The remainder of this paper is organized as follows. First, we present a review of relevant background research, including a short description of MaSE and its current weaknesses. Next, we give an overview of our metamodel that describes the elements in multiagent organizations. Finally, we discuss our extensions to MaSE that support the development of multiagent organizations and overcome some of its recognized problems.

2 Background

A recent advance in agent-oriented software engineering has had a significant impact on multiagent development approaches such as MaSE [6], Gaia [18], and MESSAGE [12]. This advancement concerns the separation of the agents from the system framework, or organization [19]. Agents play roles within an organization; however, they are not the organization. The organization defines the social setting in which the agent must exist. An organization includes a structure as well as rules, which constrain valid agent behavior and interaction within the organization.

While these advances are recent, there have been some discussions on how to incorporate them into existing multiagent systems methodologies. For instance, there is a proposal to extend the Gaia to incorporate social laws [19] and organizational concepts [18], while others have proposed implementing the organization as a separate institutional agent [17]. We have even proposed extending MaSE with rules and environmental entities [4], [5].

More recently, new methodologies and approaches have been proposed for building highly adaptive multiagent systems including Adelfe, which follows the AMAS theory [1, 14]. The goal of methods such as Adelfe is to allow designers to build systems that will produce some unknown functionality. This varies from the approach presented here as we are attempting to give the system the ability to adapt while still producing a know function and within certain limitations.

2.1 Multiagent Systems Engineering

MaSE was originally designed to develop general-purpose multiagent systems and has been used to design systems ranging from computer virus immune systems to cooperative robotics systems [6], [7]. Each phase is presented below.

Analysis Phase. The goal of the MaSE analysis phase is to define a set of roles that can be used to achieve the system level goals. This process is captured in three steps: capturing goals, applying use cases, and refining roles.

- Capturing Goals. The first step is to capture the system goals by extracting them from the requirements, which is done by Identifying Goals and Structuring Goals. The purpose of the Identifying Goals is to derive the overall system goal and its

subgoals. This is done by extracting scenarios from the requirements and then identifying scenarios goals. After the goals have been identified, the second step, Structuring Goals, categorizes and structures the goals into a goal tree, which results in a Goal Hierarchy Diagram that represents goals and goal/subgoal relationships.

- Applying Use Cases. In this step, goals are translated into use cases, which capture the previously identified scenarios with a detailed description and set of sequence diagrams. These use cases represent desired system behaviors and event sequences.
- Refining Roles. Refining Roles organizes roles into a Role Model, which describes the roles in the system and the communications between them. Each role is decomposed into a set of tasks, which are designed to achieve the goals for which the role is responsible. These tasks are documented using finite state automata-base Concurrent Task Diagrams. Concurrent tasks consist of a set of states and transitions that represent internal agent reasoning and communications.

Design Phase. The purpose of the design phase is to take roles and tasks and to convert them into a form more amenable to implementation, namely agents and conversations. The MaSE design phase consists of four steps: designing agent classes, developing conversation, assembling agents and deploying the agents.

- Construction of Agent Classes. The first step in the design phase identifies agent classes and their conversations and then documents them in Agent Class Diagrams. The Agent Class Diagram that results from this step is similar to object-oriented class diagrams with two differences: (1) agent classes are defined by the roles instead of attributes and methods and (2) relations between agent classes are conversations.
- Constructing Conversations. Once the agent classes and the conversations are identified, the detailed conversation design is undertaken. Conversations model communications between two agent classes using a pair of finite state automata similar in form and function to concurrent tasks. Each task usually generates multiple conversations, as they require communication with more than one agent class.
- Assembling Agent Classes. Assembling Agent Classes involves defining the agents' internal architecture. MaSE does not assume any particular agent architecture and allows a wide variety of existing and new architectures to be used. The architecture is defined using components similar to those defined in UML.
- Deployment Design. The final design step is to choose the actual configuration of the system, which consists of the number and types of agents in the system and the platforms on which they should be deployed. These decisions are documented in a Deployment Diagram, which is similar to a UML Deployment Diagram.

2.2 MaSE Weaknesses

While MaSE provides many advantages for building multiagent systems, it is not perfect. It is based on a strong top-down software engineering mindset, which makes it difficult to use in some application areas.

1. MaSE fails to provide a mechanism for modeling multiagent system interactions with the environment. While we examined this topic in [5], it has never been fully integrated into MaSE.

2. MaSE also tends to produce multiagent systems with a fixed organization. Agents developed in MaSE tend to play a limited number of roles and have a limited ability to change those roles, regardless of their individual capabilities. As discussed above, a multiagent team should be able to design its own organization at runtime. While MaSE already incorporates many of the required organizational concepts such as goals, roles and the relations between these entities, it cannot currently be used to define a true multiagent organization.
3. MaSE also does not allow the integration of sub-teams into a multiagent system. MaSE multiagent systems are assumed to have only a single layer to which all agents belong. Adding the notion of sub-teams would allow the decomposition of multiagent systems and provide for greater levels of abstraction.
4. The MaSE notion of conversations can also be somewhat bothersome, as it tends to decompose the protocols defined in the analysis phase into small, often extremely simple pieces. When the original protocol involves more than two agents, it often results in conversations with only a single message. This makes comprehending how the individual conversations fit together more difficult.

3 Organization Metamodel

To allow teams of agents to adapt to their environment by determining their own organization at runtime, we have developed a metamodel that describes the knowledge required to define an organization [7], [11]. Given this knowledge, we hypothesize that multiagent teams will be able to organize (and reorganize) themselves to adapt to their dynamic environments.

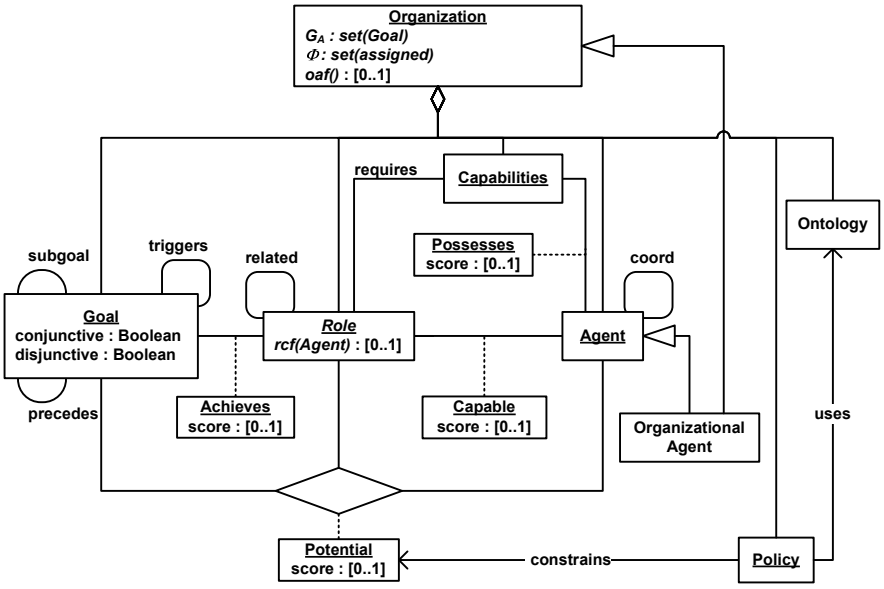

Fig. 1. Artificial Organization Metamodel

From the early days of organization research, organizations have typically been defined as including the concepts of agents who play roles within a structure that defines the relationships between the various roles [2]. We thus begin the foundation for our metamodel by defining what is meant by goals (G), roles (R), and agents (A). We also add four additional entities to our metamodel: capabilities (C), assignments (Φ), policies (P), and an ontology (Σ). Capabilities are central to the process of determining which agents can play which roles and how well they can play them, while policies constrain the assignment of agents to roles thus controlling the allowable states of the organization. The organization ontology supports agent communication and policy definition. A UML-based depiction of our organization metamodel is shown in Fig. 1.

3.1 Goals

Every artificial organization is designed with a specific purpose, which defines the overall function, or goal of the organization. Within our metamodel, each organization has a set of goals, G, that it seeks to achieve in support of a single top-level goal g_o. We define a goal in its normal way as some desired end state. G is derived by decomposing g_o into a tree of subgoals that describe how g_o can be achieved. Following the KAOS goal based requirements modeling approach [16], we allow goals to be decomposed into a set of non-cyclic subgoals using either AND-refinement or OR-refinement, which are denoted via conjunctive and disjunctive predicates. Eventually, g_o is refined into a set of leaf nodes, denoted by G_L, that may be achieved in order to achieve g_o. The active goal set, G_A (where $G_A \subseteq G_L$), is the set of goals that an organization is trying to achieve at the current time.

Goal g_1 *precedes* goal g_2 if g_1 must be achieved before g_2 can be achieved. Essentially, goal precedence allows the organization to work on one part of the goal tree at a time, thus reducing the size of G_A. The *triggers* relation is similar to precedes in that it also restricts goals from being inserted in G_A. However, instead of requiring goal achievement, the triggers relation allows new goals to be inserted into G_A when a specified event occurs.

$$\text{precedes: } G, G \rightarrow \text{Boolean}$$
$$\text{triggers: } G_L, G \rightarrow \text{Boolean}$$

All goals are unachieved when the organization is initialized. Therefore, the initial active goal set, G_{A0}, consists of all goals that have no predecessor goals or that do not require a trigger. However, as goals are achieved or events occur triggering new goal instances, G_A, changes. Essentially, achieved goals are removed from the active goal set and new goals to be achieved are inserted. New goals must be *startable* (all their predecessor goals have been achieved) and, if they are triggered, they must have been triggered by an active goal. We denote a sequence of active goal sets G_A' as G_A' = $[G_{A1}, G_{A2}, \ldots G_{An}]$.

3.2 Roles

Each organization has a set of roles R that it can use to achieve its goals. A role defines an entity that is able to achieve a set of goals within the organization. Each role is responsible for achieving, or helping to achieve or maintain specific system goals.

The *achieves* function describes how well (in a range of 0 to 1) a particular role achieves a specific goal.

$$\text{achieves: } R, G_L \rightarrow 0 \,..\, 1$$

In order to perform a particular role, agents must have a sufficient set of capabilities (which are simply defined as atomic, named entities in our model). Agents possess capabilities, which may include physical capabilities (sensors or actuators) or computational capabilities (data access, knowledge, algorithmic, etc.), and roles require a certain set of capabilities. The set of capabilities required by a particular role is captured using the *requires* predicate.

$$\text{requires: } R, C \rightarrow \text{Boolean}$$

Many times, instead of requiring agents to inherently posses all the required capabilities for a role, we would like to *bestow* the required capabilities on agents to allow them to play that role. While this does not generally work well with hardware agents (robots), with software agents, we are often free to download new algorithms, etc. Our approach to capabilities does not deny this type of role bestowal; it just requires care in defining the capabilities types in the model. Thus if an agent has the appropriate physical capabilities (computational power, communication access, etc.) we can download the specific algorithms and/or knowledge necessary to carry out a role. In many of our current multiagent systems, the algorithm is packaged with the role, not in the individual agents themselves.

To carry out their responsibilities, roles may have to work with other roles within the organization. We capture the basic notion of two roles being related using a *related* predicate, which provides a means of identifying the allowable structure of a given organization.

3.3 Agents

Our metamodel also includes a set of heterogeneous agents, A, within each organization. For our purposes, agents are computational system instances that inhabit a complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals. Agents are assigned specific roles in order to achieve organizational goals. The current set of potential assignments of agents to a role is captured by the *potential* function. The range of the potential function indicates how well an agent can play a role and how well that role can achieve the goal, based on the achieves and capable scores.

$$\text{potential: } A, R, G_L \rightarrow 0 \,..\, 1$$

However, the potential function does not indicate that the actual assignment of agent a to role r to achieve goal g, has been made within the organization. It simply defines a set of possible assignments. To capture the notion of the actual assignments, we define an *assignment set*, Φ, which consists of agent-role-goal tuples, $<a,r,g>$. If $<a,r,g> \in \Phi$, then agent a has been assigned by the organization to play role r in order to achieve goal g. As discussed above, however, only agents with the right set of capabilities may be assigned to a specific role. To capture a given agent's capabilities, we define a *possesses* function, which returns a value in the range of 0 to 1

indicating no (0) capability or excellent (1) capability, which may change with time. Using the capabilities required by a role and capabilities possessed by an agent, we can compute the ability of an agent to play a give role, which we capture in the capable function. Finally, we capture the notion of agents coordinating, to achieve their goals using the *coord* predicate.

$$\text{possesses: } A, C \rightarrow 0 \,..\, 1$$
$$\text{capable: } A, R \rightarrow 0 \,..\, 1$$
$$\text{coord: } A, A \rightarrow \text{Boolean}$$

Organizational agents (OA) are organizations that function as agents within another organization. Thus, organizational agents possess capabilities, may coordinate with other agents, and may be assigned to play roles. They represent an extension to the traditional Agent-Group-Role (AGR) model developed by Ferber [9] and the meta-model proposed by Odell [13]. Organizational agents allow the definition of a hierarchy of organizations, which provides both flexibility and scalability.

3.4 Capabilities

Capabilities are key in determining exactly which agents can be assigned to what roles in the organization. *Capabilities* are atomic entities used to define the abilities of agents in relation to roles. Capabilities can capture *soft* abilities such as the ability to access resources, communicate, migrate, or computational algorithms. They also capture *hard* capabilities such as those of hardware agents such as robots, which include sensors and effectors.

3.5 Policies

Organization *policies* are formally specified rules that describe how an organization may/may not behave in specific situations. In our metamodel, we distinguish between two specific types of policies: *assignment* policies (P_Φ) and *behavioral* policies (P_{beh}). Assignment policies deal with constraints that the assignment set, Φ must satisfy such as "an agent may play one role at a time" or "agents may work on a single goal at a time". Behavioral policies define how agents should behave in solving the problem at hand.

3.6 Ontology

The organization ontology defines the entities within the application domain and their relationships. From these definitions, we extract a set of data types and relationships that allow agents to communicate about application specific information. These domain entities and relationships are also used to help in defining application specific organization policies. We currently use static UML diagrams to define ontological concepts similar to the approach in [3].

3.7 Organization Example

An example of a multiagent team developed using our organization metamodel is shown in Fig. 2. The boxes at the top of the diagram represent goals (A … G), the

circles represent roles (*R1* … *R5*), the pentagons represent capabilities (*C1* … *C5*), and the rounded rectangles are agents (*A1* … *A4*). The arcs on subgoal links denote conjunctive subgoals, whereas undecorated links denote disjunctive subgoals. The arrows between the entities represent the *achieves*, *requires*, and *possesses* functions/relations as defined above. The numbers beside the arrows represent the function value (e.g., possesses(*A1*,*C1*) = 0.5). These *achieves* values are generally assigned at design time and do not change. The *possesses* values, on the other hand, are computed by the individual agents based on their own internal assessment of their capabilities. We also assume we have assignment policies that (1) we only assign a single agent to each goal and (2) only one of the disjunctive goals F or G can be active at any time.

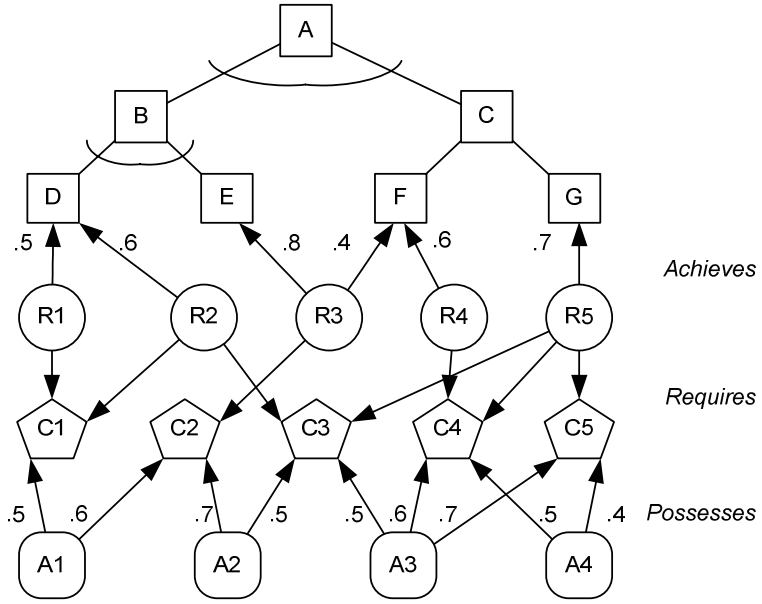

Fig. 2. Organization Example

Therefore, in this example the top-level goal, A, can be achieved by achieving both subgoals B and C, which can be achieved by achieving D and E in addition to either F or G. The *achieves* relation shows that either of the roles R1 or R2 can be used to achieve goal D while only role R3 can be used to achieve goal E. However, role R3 can also be used to achieve goal F, which can also be achieved by role R4. The only role capable of achieving goal G is role R5. The aim is for the organization to assign agents that can play the appropriate roles to achieve specific goals. Because of the disjunctive nature of goal C and the ability to use different roles in achieving individual goals, there is some assignment flexibility built into the system.

Determining which agents should be assigned specific roles in order to achieve particular goals is based on the capabilities the agents currently possess. For instance,

Table 1. Capability Function

	R1	R2	R3	R4	R5
A1	0.5	0	0.6	0	0
A2	0	0	0.7	0	0
A3	0	0	0	0.6	0.6
A4	0	0	0	0.5	0

role R1 requires only capability C1 while R3 requires only capability C2. Therefore, since agent A1 possesses both capabilities C1 and C2, it could be assigned to either role R1 or R3 in order to achieve goals D, E, or F.

In this example, we thus compute the *capable* function value for each agent-role pair as shown in Table 1. For simplicities sake, we average the individual capability score required for each role. Combining the *capable* scores with the *achieves* score, we can easily compute the organizational capability score, *Os*, for any set of assignments that might be made. Based on these computations (keeping in mind our assignment policies), we can see that the maximum organizational capability score, and thus the optimal assignments are as follows:

potential(*A1*,*R1*,*D*) = 0.25
potential(*A2*,*R3*,*E*) = 0.56
potential(*A3*,*R5*,*G*) = 0.42
Os = 1.23

3.8 Exemplar Implementation

Although there is not a single "correct" way to implement our organizational metamodel, we suggest an example to help cement the concepts. Fig. 3 shows the implementation approach used in our current projects. Each *agent* is composed of two components: an Organizational Reasoning component and an Application Reasoning

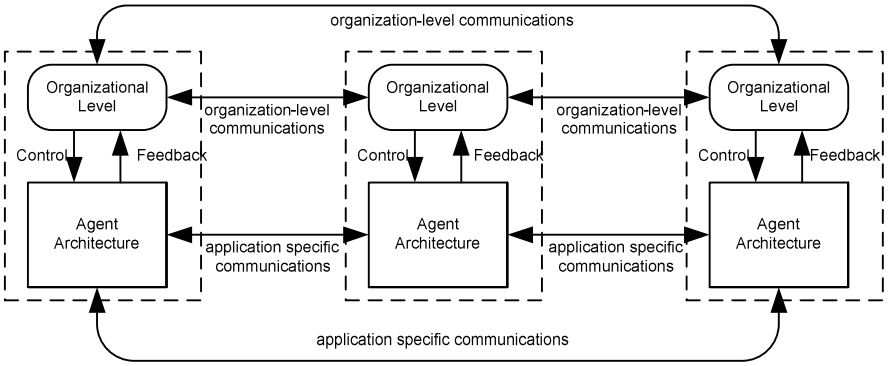

Fig. 3. Example Implementation

component. The Organizational Reasoning component is concerned with computing the current assignment set ϕ based on the active goal set G_A and the feedback received from the Application Reasoning component in regards to goal achievement, goal failure, or the occurrence of triggering events.

The Application Reasoning component accepts its assignment and carries out the tasks necessary to play its assigned roles in pursuit of its assigned goals. The Organizational Reasoning component interacts with the other agent's Organizational Reasoning components to ensure system coherence. Exactly how the coordination is carried out can vary. We have implemented a variety of centralized and distributed approaches to this coordination process; the best approach to this coordination process is domain and application specific. Part of the goal of the architecture presented in Fig. 3 is to be able to provide "plug-and-play" Organizational Reasoning components that can be selected based on application criteria.

4 O-MaSE

To avoid designing static multiagent systems, we have extended MaSE to allow designers to design a multiagent organization, which provides a structure within which the multiagent system may adapt. This extended version of MaSE is called Organization-based MaSE (O-MaSE). A preliminary proposal for the O-MaSE methodology is described below. In general, many of the diagrams used in O-MaSE are variants of the UML class diagrams and use keywords to denote the difference between goals, roles, capabilities, agent classes, etc.

Throughout this section, an Information Flow Monitoring System (IFMS) is used as an example of an organization-based multiagent system. The overall goal of the IFMS is to keep track of the information producers and consumers along with the actual flow of information through a dynamically reconfigurable enterprise information system. The information producers and consumers use a publish/subscribe mechanism that allows consumers to find and subscribe to appropriate information producers. Therefore, the IFMS must keep track of the various information paths between the producers and consumers as well as monitor the actual data flowing along those paths. The IFMS provides data in the form of current paths and information flow statistics to enterprise system operators who monitor the system for problems.

4.1 Requirements

Requirements are translated into system level goals, which are documented in the form of an AND/OR goal tree. Fig. 4 shows the goal tree for the IFMS described above. Given goal precedence relations, it is possible to design goal structures that cannot be achieved, thus we would like to provide the assurance that the top-level goal can be achieved. We have replaced the non-specific MaSE goal tree with a tree with specific AND/OR decompositions to match the organization metamodel.

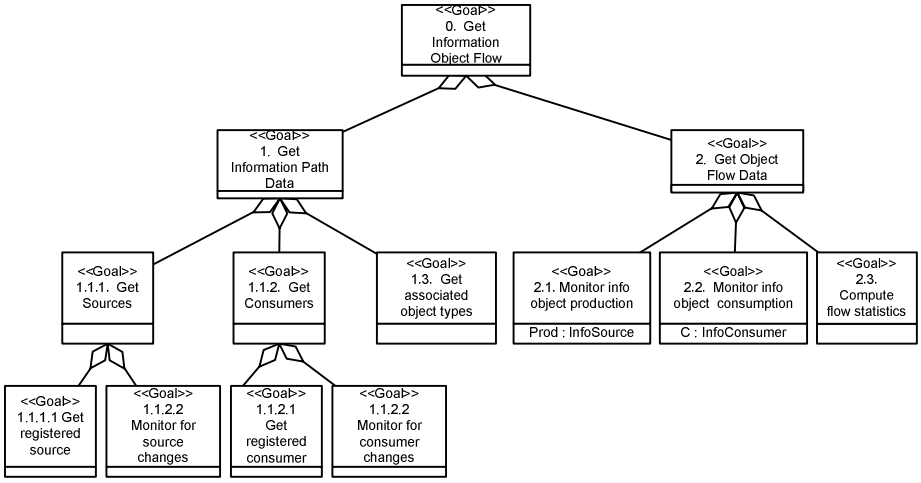

Fig. 4. Goal Hierarchy Diagram

The syntax has also changed in the O-MaSE goal model. We use standard UML class notation with the keyword «Goal». Each goal may be parameterized, with parameters annotated as attributes of the goal class. When goals are instantiated, they are given specific values for these attributes. The aggregation notation is used to denote AND refined goals whereas the generalization notation is used to denote OR refined goals. This notation is somewhat intuitive as AND refined goals require a composition of its subgoals to be achieved. Subgoals of an OR refined parent goal can be thought of as alternative ways to achieve the parent goal, or that they can be substituted in place of the parent goal.

4.2 Analysis

Analysis begins by creating the Organization Model, which defines the organization's interactions with external actors. Generally, there is one organization at the top level (denoted by the «Organization» keyword) and that organization becomes responsible for the top goal in the goal tree. Each organization can achieve goals and provide services, which are further refined via activity diagrams (similar to UML activity diagrams, not included in this paper). The designer can also use sequence diagrams for describing use cases at the system level, similar to the original version of MaSE. Each organization may also include sub-organizations to allow for abstraction during the design process.

While we allow the use of *services* in O-MaSE to help define the activities that agents carry out while performing roles, they do not map directly to the organization metamodel as presented earlier. For the purposes of this paper, we only mention them for completeness, but do not elaborate on them, as their use in defining organizations is not required.

An example of an Organization Model is shown in Fig. 5, where there are three actors making up the system's environment: the *ClientAPI*, the *ServerAPI*, and the *Admin*. The arrows connecting the organization to the actors denote protocols that

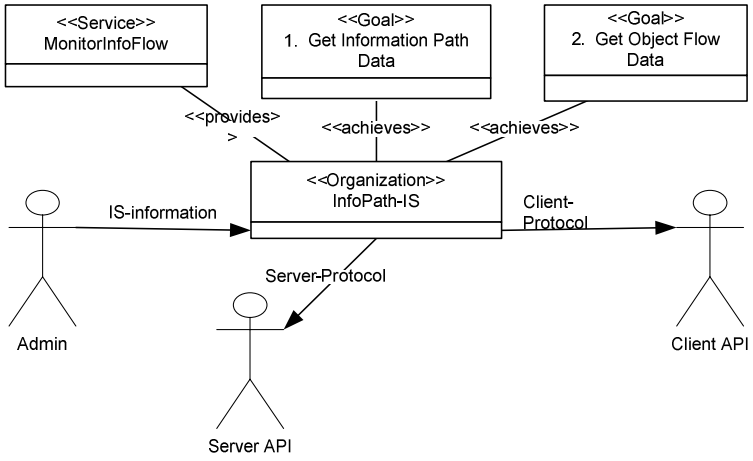

Fig. 5. Organization Model

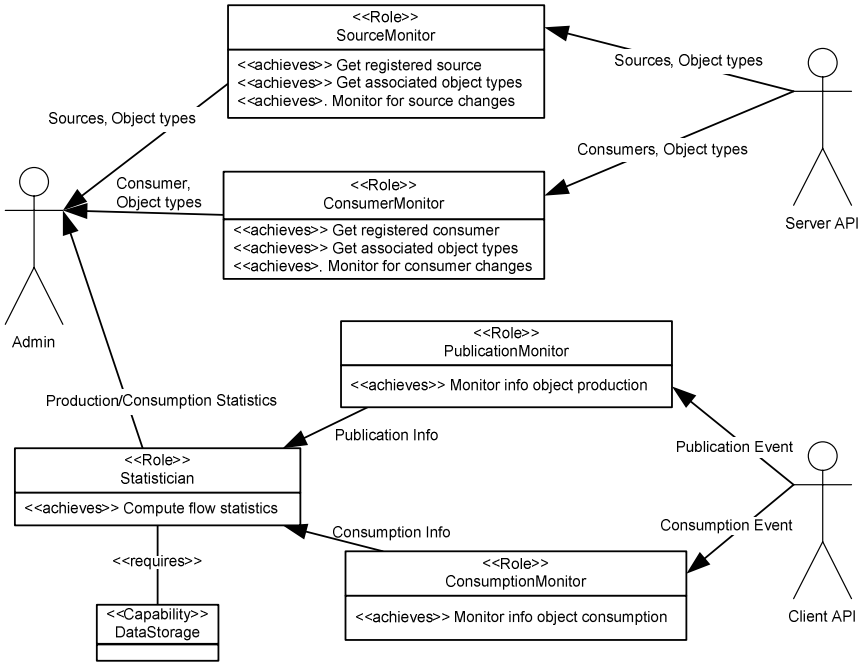

Fig. 6. Role Model

define the agent class's interactions with the environment (these protocols are defined in detail in the high-level design stage). The relations between the organization and the goal and service classes (classes denoted by «Goal» and «Service» keywords) are fixed relation types. An organization provides services while achieving goals. These

relations may be shown via explicit relations between organizations and goals; however, the relations may also be embedded in a class as shown in Fig. 6 (where «achieves» relations are shown embedded within roles).

Next, the organization model is refined into a role model (Fig. 6) that defines the roles in the organization, the services they provide, and the capabilities required to play them. Each role is designed to achieve specific goals from the Goal Model and provide specific activities refined from top-level services in the Organization Model. Again, the arrows between actors and roles and between two roles indicate protocols that are fully defined later in the design stage. The Role Model may also include capabilities (denoted by the «Capability» keyword), which are attached to the roles that require them by the «requires» keyword.

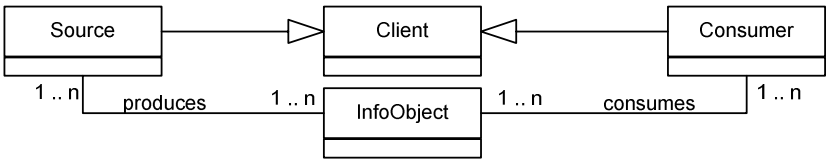

Fig. 7. Domain Model

At this point, O-MaSE differs from MaSE in that O-MaSE does not require the analyst to create concurrent task diagrams to describe the behavior of each role. This task is more appropriately carried out at the low-level design stage. The use of activities, which are refined via activity diagrams, allow the analyst to specify high-level behavior without resorting to low-level details required by concurrent task diagrams.

Throughout the analysis phase, the analyst should also capture and document the ontology that will be used within the system as part of the Domain Model. We have explored the integration of domain models into MaSE in [4]. The Domain Model allows the analyst to model domain entities, their attributes, and their relationships. Fig. 7 shows a simple example of a domain model using standard UML notation to show the relationships between two types of Clients: Source and Consumer. Sources produce *InfoObjects* while Consumers consume *InfoObjects*.

4.3 High-Level Design

The first step in the high-level design is for the designer to use the Role Model and service activity diagrams to define the Agent Class Model as shown in Fig. 8. In the Agent Class Model, agents classes and lower-level organizations are defined by the roles played (which determines the goals they can achieve), capabilities possessed (which determines the roles they can play), or services provided. Fig. 8 shows the use of both explicit and embedded relations. The «plays» and «provides» keywords in the agent classes (denoted by the «Agent» keyword) define which roles instances of the agent class can play as well as the services it can provide. The «possesses» relation between agent classes and capabilities (denoted by the «Capability» keyword) indicates the capabilities possessed by instances of that class of agent.

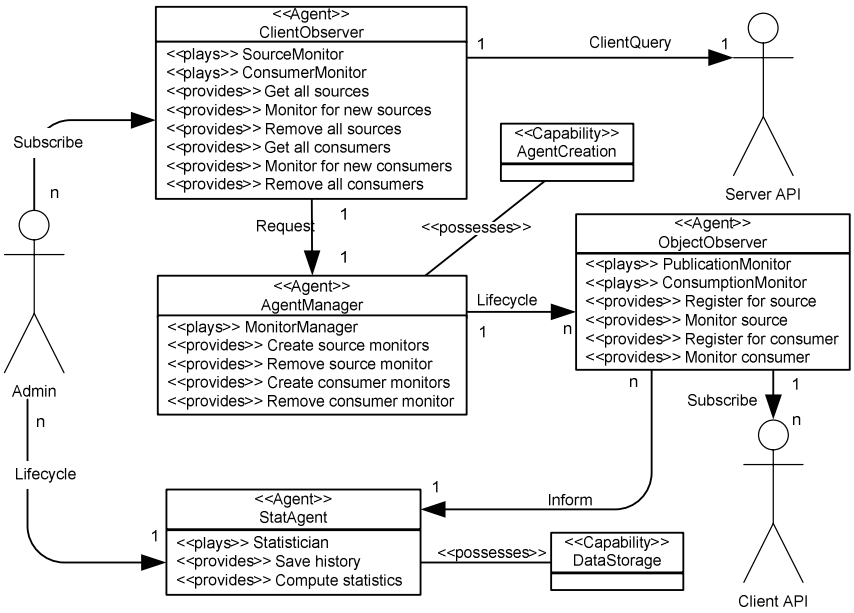

Fig. 8. Agent Class Model

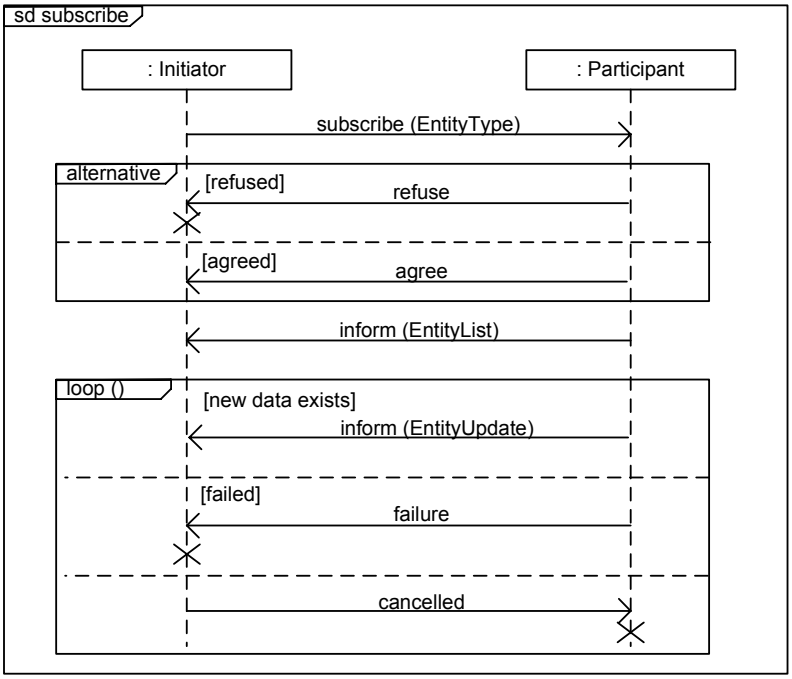

Fig. 9. Protocol Model

One the Agent Class Model is complete, Protocol Models (Fig. 9) are used to define the message-passing protocols between agent classes. These Protocol Models follow the currently proposed AUML protocol diagrams [10], which allow the ability to show alternative and repetitive message structures.

Fig. 9 captures a subscription protocol where the initiator wants to subscribe to information published periodically by the participant. After the initial subscribe message, the participant may either refuse or agree. If the participant refuses, the protocol terminates, which is denoted by the **X** symbol. Assuming the participant agrees, the participant sends an inform message with the current subscription information. The protocol then enters a loop where, typically, the participant sends an inform message with new information. However, the participant may send a failure message or the initiator a cancelled message, both of which end the protocol.

4.4 Low-Level Design

In low-level design, we define agent behavior using an Agent State Model, which is based on finite state automata (Fig. 10). The Agent State Model is similar to the

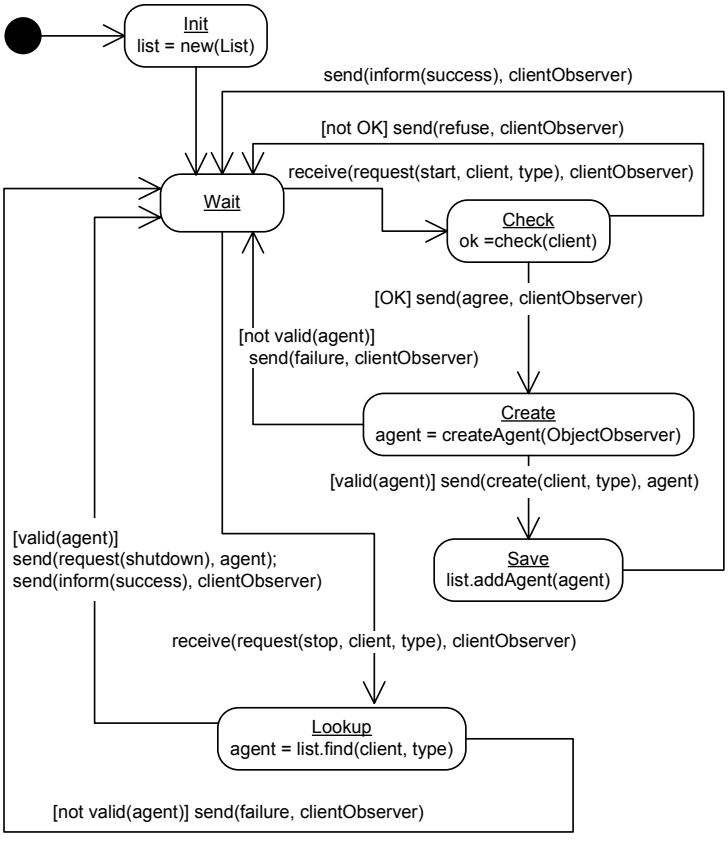

Fig. 10. Agent State Model for AgentManager

original MaSE concurrent task diagrams, as it captures internal behavior and message passing between multiple agents. They feature an explicit send and receive actions to denote sending and receiving messages. The remainder of the syntax and semantics is defined in [6].

5 Conclusions

In this paper, we have discussed the current version of MaSE and some of its shortcomings. With the extension of MaSE to O-MaSE we have dealt with each of these problems. Specifically, we have provided a mechanism for defining the multiagent systems interactions with the environment by adding external actors and defining the interactions protocols between the system and the actors.

Second, we have extended MaSE to capture the organizational concepts identified in our organization metamodel. New concepts include AND/OR refinement of goals, integration of capabilities and the ability to model sub-teams, or sub-organizations. This feature allows designers greater levels of abstractions and directly complements the notion of organizational agents in our organization metamodel.

Finally, we took the notion of concurrent tasks out of the analysis phase and integrated concurrent tasks with conversations into Agent State Models in the low-level design phase. We are currently using O-MaSE and our organization metamodel in several projects including an adaptive Battlefield Information System [7], cooperative robotic teams [11] and a system to monitor and control a large-scale information system.

We are continuing to evolve O-MaSE to provide a flexible methodology that can be used to develop both traditional and organization-based systems. A long term goal is to provide a tailorable methodology that is fully supported by automated tools. We are currently building a new version of agentTool (aT^3) within the Eclipse IDE to support O-MaSE[1]. Future plans include code generation for various platforms as well as integration with the Bogor model checking tool [15] to provide model validation and performance prediction metrics.

References

1. Bernon, C., Camps, V., Gleizes M.P., Picard G. Engineering Adaptive Multi-Agent Systems: the ADELFE Methodology. In B. Henderson-Sellers and P. Giorgini (Eds.), Agent-Oriented Methodologies. Idea Group Pub, June 2005, pp.172-202.
2. Blau, P.M. & Scott, W.R., Formal Organizations, Chandler, San Francisco, CA, 1962, 194-221.
3. Cranefield, S. & Pruvis, M. UML as an Ontology Modelling Language. Proc of the Workshop on Intelligent Information Integration, 1999.
4. DeLoach, S. A. Modeling Organizational Rules in the Multiagent Systems Engineering Methodology. Proc of the 15th Canadian Conference on Artificial Intelligence. 2002.
5. DeLoach, S. A. Analysis and Design of Multiagent Systems Using Hybrid Coordination Media. Proceedings of Software Engineering in Multiagent Systems (SEMAS 2002). 2002.

[1] The current status of O-MaSE and the aT^3 project can be found at the Multiagent and Cooperative Robotics Laboratory web site (http://macr.cis.ksu.edu/).

6. DeLoach, S. A., Wood, M. F. and Sparkman, C. H., "Multiagent Systems Engineering". The International Journal of Software Engineering and Knowledge Engineering, 11(3), pp. 231-258, June 2001.
7. DeLoach, S.A., & Matson, E. An Organizational Model for Designing Adaptive Multi-agent Systems. The AAAI-04 Workshop on Agent Organizations: Theory and Practice (AOTP 2004). 2004.
8. Dignum, V. A Model for Organizational Interaction: Based on Agents, Founded in Logic. PhD thesis, Utrecht University, 2004.
9. Ferber, J., and Gutknecht, O. A meta-model for the analysis and design of organizations in multi-agent systems. In Proceedings of Third International Conference on MultiAgent Systems (ICMAS'98), pages 128-135, IEEE Computer Society, 1998.
10. Huget, M.P., Bauer, B., Odell, J., Levy, R., Turci, P., Cervenka, R., and Zhu, H. http://www.auml.org/. FIPA Modeling: Interaction Diagrams, Working Draft. 2002.
11. Matson, E., DeLoach, S. Capability in Organization Based Multi-agent Systems, Proceedings of the Intelligent and Computer Systems (IS '03) Conference, 2003.
12. MESSAGE: Methodology for Engineering Systems of Software Agents. Deliverable 1. Initial Methodology. July 2000. EURESCOM Project P907-GI.
13. Odell, J., Nodine, M., Levy, R. A Metamodel for Agents, Roles, and Groups. Agent-Oriented Software Engineering V, 5th International Workshop, AOSE 2004. 2004.
14. Picard, G. and Gleizes, M.-P. The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems. In Bergenti, F. and Gleizes, M-P. and Zambonelli, F., editor, Methodologies and Software Engineering for Agent Systems. Kluwer Publishing, 2004.
15. Robby, Dwyer, M.B., & Hatcliff, J. Bogor: An Extensible and Highly-Modular Model Checking Framework, Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003).
16. van Lamsweerde, A., Darimont, R., Letier, E. Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering. 24(11), pp 908-926, 1998.
17. Wagner, G. Agent-Oriented Analysis and Design of Organizational Information Systems. Proceedings of the 4th IEEE International Baltic Workshop on Databases and Information Systems, May 2000.
18. Zambonelli, F., Jennings, N.R., and Wooldridge, M.J. Developing Multiagent Systems: The Gaia Methodology. In AMC Transactions on Software Engineering Methodology 12(3), 317-370, 2003.
19. Zambonelli, F., Jennings, N.R., and Wooldridge, M.J. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. IJSEKE. 11(3) pp. 303-328, June 2001.