Scott A. DeLoach, Walamitien Oyenan & Eric T. Matson. A Capabilities Based Model for Artificial Organizations. Journal of Autonomous Agents and Multiagent Systems. Volume 16, no. 1, February 2008, pp. 13-56. DOI: 10.1007/s10458-007-9019-4.

A capabilities-based model for adaptive organizations

Scott A. DeLoach, Walamitien H. Oyenan
Department of Computing & Information Sciences
Kansas State University

Eric T. Matson
Department of Computer Science and Engineering
Wright State University

Abstract

Multiagent systems have become popular over the last few years for building complex, adaptive systems in a distributed, heterogeneous setting. Multiagent systems tend to be more robust and, in many cases, more efficient than single monolithic applications. However, unpredictable application environments make multiagent systems susceptible to individual failures that can significantly reduce its ability to accomplish its overall goal. The problem is that 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 are developing a framework that allows *the system to design its own organization at runtime.* This paper presents a key component of that framework, a metamodel for multiagent organizations named the Organization Model for Adaptive Computational Systems. This model defines the requisite knowledge of a system's organizational structure and capabilities that will allow it to reorganize at runtime and enable it to achieve its goals effectively in the face of a changing environment and its agent's capabilities.

Keywords: adaptation, organizations, metamodel, self-organization

1. Introduction

Systems are becoming more complex, in part due to increased customer requirements and the expectation that applications should be seamlessly integrated with other existing, often distributed applications and systems. In addition, there is an increasing demand for these complex systems to exhibit some type of intelligence as well. No longer is it "good enough" to be able to access systems across the internet, but customers require that their systems know how to access data and systems, even in the face of unexpected events or failures.

The goal of our research is to develop a framework for constructing complex, distributed systems that can autonomously adapt to their environment. Multiagent systems have become popular over the last few years for providing the basic notions that are applicable to this problem. A multiagent

system uses groups of self-directed agents working together to achieve a common goal. Such multiagent systems are widely proposed as replacements for sophisticated, complex, and expensive stand-alone systems for similar applications. Multiagent systems tend to be more robust and, in many cases, more efficient (due to their ability to perform parallel actions) than single monolithic applications. In addition, the individual agents tend to be simpler to build, as they are built from a single agent's perspective.

However, unpredictable application environments make multiagent systems susceptible to individual failures that can significantly reduce the ability of the system to accomplish its goal. The problem is that multiagent systems are typically designed to work within a limited set of configurations. Even when the system possesses the resources and computational ability to accomplish its goal, it may be constrained by its own structure and knowledge of its member's capabilities, which may change over time. In most multiagent design methodologies [12, 31, 44], the system designer *analyzes* the possible organizational structure – which determines which roles are required to accomplish which goals and sub-goals – 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 undergo changes, a designer can rarely account for, or even consider, all possible situations. Attempts to overcome these problems include large-scale redundancy using homogenous capabilities and centralized/distributed planning. However, homogenous approaches negate many of the benefits of using a multiagent approach and are not applicable in complex applications where specific capabilities are often needed by only one or two agents. Centralized and distributed planning approaches tend to be brittle and computationally expensive due to their required level of detail (individual actions in most cases).

To overcome these problems, we are developing a framework that allows *a system to design its own organization at runtime*. In essence, we propose to provide the system with organizational knowledge and let the system design its own organization based on the current goals and its current capabilities. While the designer can provide guidance, supplying the system with key organizational information will allow it to redesign, or reorganize, itself to match its scenario. This paper presents a key component of our framework, a metamodel for multiagent organizations named the *Organization Model for Adaptive Computational Systems* (OMACS). OMACS defines the requisite knowledge of a system's organizational structure and capabilities that will allow the system to reorganize at runtime and enable it to achieve its goals in the face of a changing environment and its agent's capabilities.

2. Motivating Examples

In this section, we present three areas where a framework for allowing systems to adapt their configuration at runtime is highly desirable. These areas include cooperative robotics, large scale information systems, and general multiagent systems. The third area, general multiagent systems, is demonstrated with the Conference Management System example that is used periodically for illustrative purposes throughout the remainder of the paper.

2.1 Cooperative Robotic Search and Rescue

Use of our organizational framework is especially applicable to cooperative robotic teams. Because of the amount of hardware devices in robots, the robots can malfunction, either fully or partially. These malfunctions change the robot's capabilities and thus the roles they are suited to playing. In previous work [30], we showed how our organization theoretic model could be applied to the problem of sensor and effector integration in individual robots as well as in robotic teams.

Due to cost and complexity constraints, teams of homogenous robots are not generally envisioned for use in complex and dangerous environments such as toxic waste cleanup, extraterrestrial exploration, or unmanned military operations. Moreover, due to the types of environments these robots will be subjected to, hardware failure is probable and the ability to repair those failures will be limited.

Consider the case where a team of heterogeneous robots is performing a cooperative search and rescue operation. Some robots will have better capabilities for searching due to their enhanced sensor package while some robots may be better suited for rescuing due to their specific effectors such as grippers and robotic arms, each with differing payload limits. However, robots with enhanced rescue abilities can also perform searching, since they must have some type of minimal object detection system in order to perform rescues. Thus at the onset of the search and rescue operation, since the team has not yet found any victims, all the robots are available for searching. Once a victim is found, one of the robots must switch to a rescuer role and attempt to rescue the victim. However, choosing the correct robot to perform the rescue is dependent on many properties of both the victim and their current situation, which may include size of the victim, access to victim, etc.

In addition, the capabilities of the individual robots may change over time. This must be accounted for when organizing the team. For instance, what happens if there are three robots performing the rescue role and one of those robots happens to break down? Should the team get another robot to take its place? Or, should the team continue with its two rescue robots? These are decisions that can only be made within the context of what is best for the team and its current state in terms of the problem being solved. What is needed is a mechanism that the robot team can use to determine the best robot for the job in terms of overall team performance. This mechanism must take into account the current state of the team, which includes the goals being pursued, the available team members, and the team member's capabilities.

2.2 Battlefield Information Systems

The goal of a battlefield information system is to provide the commander with both tactical and strategic intelligence. To accomplish this, various types of sensors are used to detect events and objects of interest. This sensor data is then be combined, or fused, with other sensor data to provide a commander with a more complete picture of the battlefield. Due to the nature of war, there is a high probability that some of these sensors will become disabled. However, when sensors are lost, their information is still required in order to provide the battlefield commander with a complete picture. Thus, the battlefield information system must detect sensor failures and adapt its processing in a timely manner. An example of such a system is one in which air, satellite and ground-based sensors must be monitored to evaluate enemy force deployment and strategy. To operate effectively in this scenario, the battlefield information system must adapt to changes in both the queries from the commander as well as sensor availability.

As an example, assume we have a system with three types of agents as shown in Fig. 1a: data sensor agents, merge agents, and query agents. *Data Sensor Agents* (DS) provide the interface between the hardware sensors and the *Merge Agents* (MA), which fuse data from various sensor types to formulate answers to requests for information of the *Query Agents* (QA). The Query Agent translates, manages and communicates the query to the Merge Agents and returns results to the commander.

When the system receives a query, there is no assurance the sensors required to execute the query are available. If a sensor is damaged, the system may be required to re-evaluate its ability to satisfy the query requirements. If the requirements are not met, the system must reorganize to

produce a new structure that can meet the query goal requirements. Fig. 1a shows the initial layout of the system as set up to answer the query, "Show all tank, troop and helicopter movement within sector". Answering this query requires the minimal capabilities of three sensors and three DS agents to interpret the raw data. SA1 possesses the capability to accept data from ground and air sensors and synthesize it for return to the QA. SA2 accepts and passes data from the satellite via the DS agent.

A problem arises when the Air Sensor that the system is using to answer the query becomes unavailable. The system reacts to this event by reorganizing itself as shown in Fig. 1b. Notice that instead of simply replacing the lost Air Sensor with another one and integrating its data via SA1 as might be expected, the system chose to integrate the Air Sensor via SA2. The answer to why the system chose this particular organization lies in the capabilities of the various sensors and the agents that are combining the data. Even though the new Air Sensor provided similar data to the one that was lost, the system realized that due to its lower quality, combining it with the Satellite data first and then combining it with the Ground Sensor data would yield a better result (either in terms of timeliness or quality) than simply replacing the failed Air Sensor with the new Air Sensor. Analysis of this type requires detailed knowledge about the capabilities of the sensors as well as the agents used to combine the data. The goal of our research is to provide a framework that provides systems with this knowledge and analysis capability.

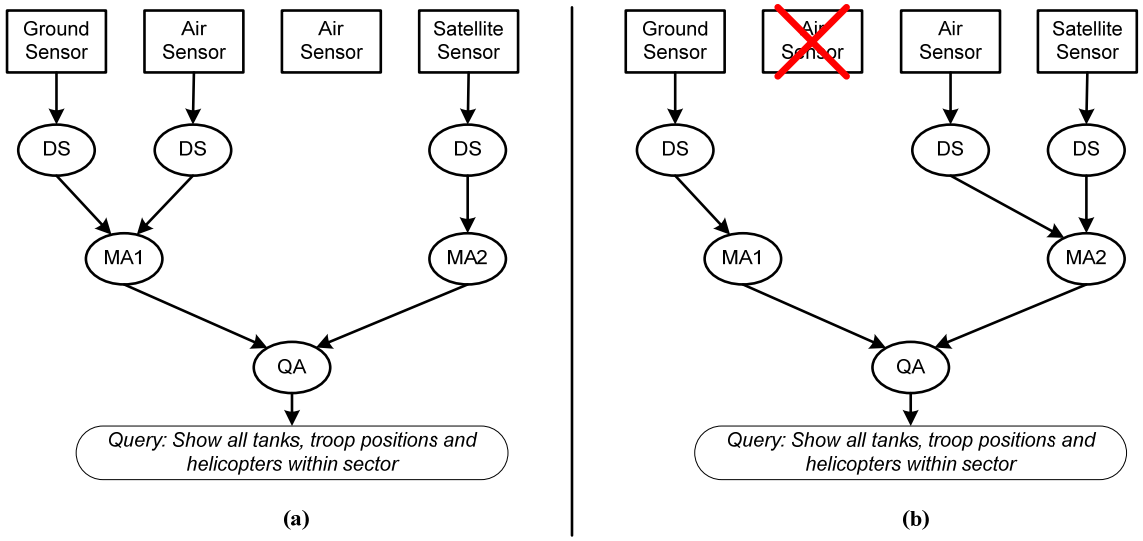


Fig. 1. Original System Organization (a) and Organization after Sensor Failure (b)

2.3 General Purpose Multiagent Systems – The Conference Management System

Throughout this paper, many of the examples will use the conference management example as defined in [46]. The conference management system is a multiagent system supporting the management of various sized international conferences that require the coordination of several individuals and groups. There are five distinct phases in which the system must operate: submission, review, decision, and final paper collection. During the submission phase, authors should be notified of paper receipt and given a paper submission number. After the deadline for submissions has passed, the program committee (PC) has to review the papers by either contacting referees and asking them to review a number of the papers, or reviewing them themselves. After the reviews are complete, a decision on accepting or rejecting each paper must be made. After the decisions are made, authors are notified of the decisions and are asked to

produce a final version of their paper if it was accepted. Finally, all final copies are collected and printed in the conference proceedings. The conference management system consists of an organization whose membership changes during each stage of the process (authors, reviewers, decision makers, review collectors, etc.). Also, since each agent is associated with a particular person, it is not impossible to imagine that the agents could be coerced into displaying opportunistic, and somewhat unattractive, behaviors that would benefit their owner to the detriment of the system as a whole. Such behaviors could include reviewing ones own paper or unfair allocation of work between reviewers, etc.

A model of the system roles and their interactions is shown in Fig 2. In the diagram, boxes denote roles within the system while the UML actor notation is used to represent external entities with which the system must interface. The system starts by having authors submit papers to a paper database (PaperDB) role, which is responsible for collecting the papers, along with their abstracts, and providing copies to reviewers when requested. Once the deadline has past for submissions, the person responsible partitioning the entire set of papers into groups to be reviewed (the Partitioner role) asks the PaperDB role to provide it the abstracts of all papers. The Partitioner partitions the papers and assigns them to a person (the Assigner) who is responsible for finding *n* reviewers for each paper. Once assigned a paper to review, a Reviewer requests the actual paper from the PaperDB, prepares a review, and submits the review to the Review Collector. Once all (or enough) of the reviews are complete, the Decision Maker determines which papers should be accepted and notifies the authors. Authors of accepted papers then submit their final copy to the Finals Collector who forwards them to the Printer for printing.

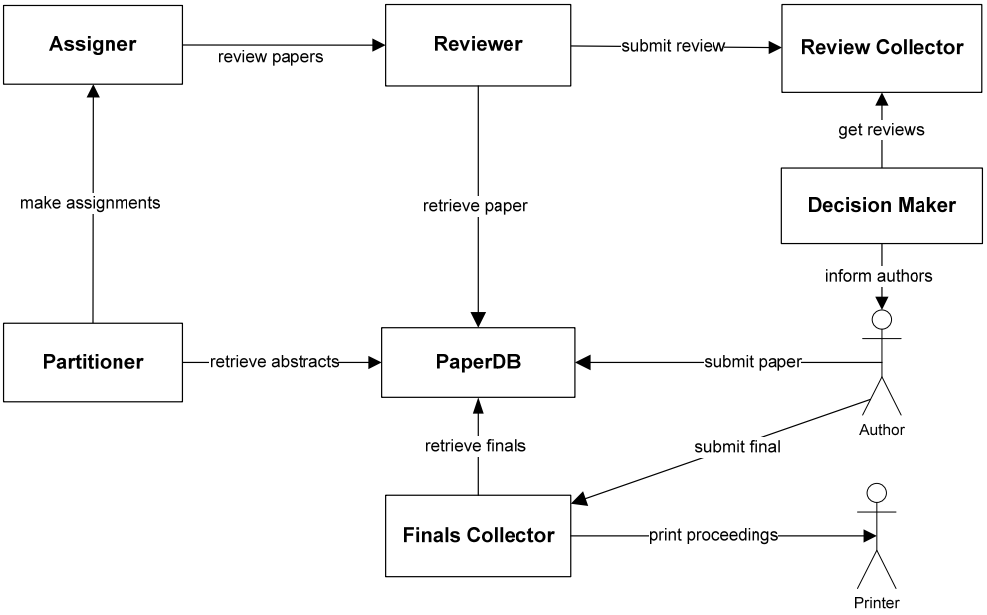


Fig. 2. Conference Management Role Model

In [46], the conference management system is described terms of seven organizational rules. The rules are shown below using the temporal operators as defined in Table 1.

1. $\forall p : \#(\text{reviewer}(p)) \geq 3$
2. $\forall i, p : \text{Plays}(i, \text{reviewer}(p)) \Rightarrow \bigcirc \Box \neg \text{Plays}(i, \text{reviewer}(p))$
3. $\forall i, p : \text{Plays}(i, \text{author}(p)) \Rightarrow \Box \neg \text{Plays}(i, \text{reviewer}(p))$

4. $\forall i, p : Plays(i, author(p)) \Rightarrow \Box \neg Plays(i, collector(p))$
5. $\forall i, p : participate(i, receivePaper(p)) \Rightarrow \Diamond initiate(i, submitReview(p))$
6. $\forall i, p : participate(i, receivePaper(p))$ **B** $initiate(i, submitReview(p))$
7. $\forall p : [submittedReviews(p) > 2]$ **B** $initiate(chair, decision(p))$

The first rule states that there must be at least three reviewers for each paper (# is cardinality) while rule two keeps a reviewer from reviewing the same paper more than once. Rules three and four ensure that a paper author does not review or collect reviews of his or her own paper. The last three rules describe appropriate system operation. Rule five states that if a paper is received, it should eventually be reviewed. Rule six requires that a paper must actually be received before a review can be submitted on it, while rule seven requires that there be at least two reviews before a paper can be accepted or rejected.

Table 1. Temporal Operators

$\bigcirc \varphi$	φ is true next
$\Box \varphi$	φ is always true
$\Diamond \varphi$	φ is eventually true
φ **B** ϕ	φ is true before ϕ is true

3. Organization Metamodel

While most people have an intuitive idea of what an organization is, when asked to define it explicitly, there are large numbers of "correct" answers. From early organizational research we learn that *organizations* have typically been defined as including the concepts of set of *agents* who play *roles* within a structure that defines the *relationships* between the various roles [2]. Thus, we lay the foundation for our model by defining what is meant by goals (G), roles (R), and agents (A). We also add four additional entities: capabilities (C), assignments (Φ), policies (P), and a domain model (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 domain model is a critical component that defines the ontology used to define behavioral policies and to allow agents to communicate effectively. A UML depiction of the organizational metamodel is shown in Fig. 3. Each entity and relationship in the diagram is explained in detail in the following subsections.

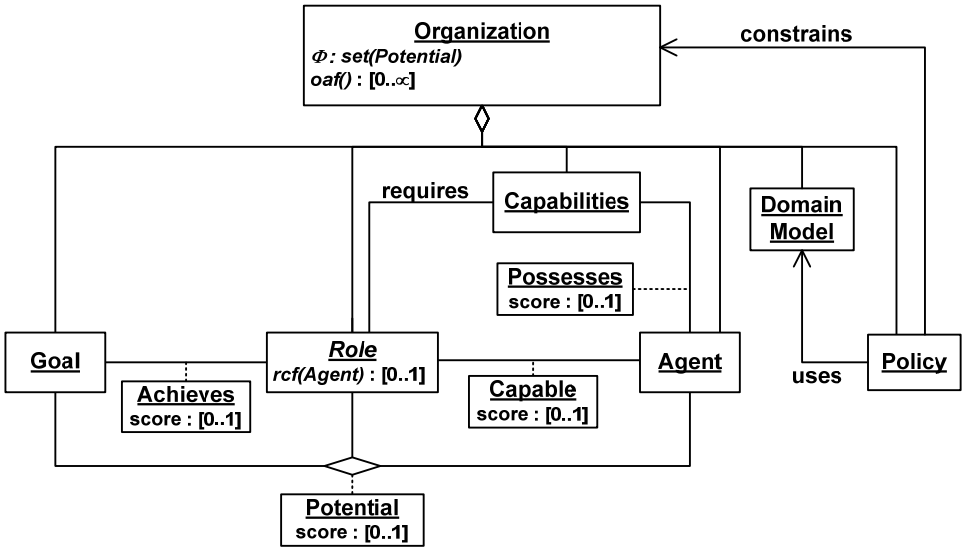


Fig. 3. Organization Model

The key result of this research was the development of a metamodel for artificial organizations, called the Organization Model for Adaptive Computational Systems (OMACS). This model allows MAS developers to define a structure to support specific applications. (Note: In the following definitions, **P**(S) is used to denote the powerset of S.)

3.1 General Organization Definition

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

- G goals of the organization
- R set of roles
- A set of agents
- C set of capabilities
- Φ relation over G × R × A that defines the current set of agent/role/goal assignments
- P set of constraints on Φ
- Σ domain model used to specify objects in the environment, their inter-relationships, and the operations that can be performed upon them
- *oaf* function **P**(G × R × A) → [0.. ∞] that defines the quality of a proposed set of assignments
- *achieves* function G × R → [0..1] that defines how well a role achieves a goal
- *capable* function A × R → [0..1] that defines how well an agent can play a role
- *requires* function R → **P**(C) that defines the set of capabilities required to play a role
- *possesses* function A × C → [0..1] that defines the quality of an agent's capability
- *potential* function A × R × G → [0..1] that defines how well an agent can play a role to achieve a goal

Each of the above components is described below in detail.

3.2 Goals

Artificial organizations are designed with a specific purpose, which defines the overall function of the organization. *Goals* are defined as a desirable situation [37] or the objective of a computational process [41]. Within 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 to individual agents and individual agents have the ability to achieve them independently. (For simplicity, this report refers to a role *achieving* a goal, although there are many types of goals such as goals of achievement, maintenance, etc.).

3.3 Roles

Within OMACS, each organization contains 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. Thus, each role defines a set of responsibilities. Roles are analogous to roles played by actors in a play or by members of a typical corporate structure. A typical corporation has roles such as "president", "vice-president", and "mail clerk". Each role has specific responsibilities, rights and relationships defined in order to help the corporation perform various functions towards achieving its overall goal. Specific people (agents) are assigned to fill those roles and carry out the role's responsibilities using the rights and relationships defined for that role.

OMACS roles consist of a name and a role capability function, *rcf*. Each role, $r \in R$, is a tuple ⟨name, *rcf*⟩ where

- name a string
- *rcf* function $A \rightarrow [0..1]$ that defines how well a given agent can play the role based on the capabilities possessed by the agent

While an agent that is assigned a role may choose to play that role in any way it wishes, OMACS does have certain expectations for agents assigned to roles. First, the agent is expected to play that role in order to achieve a specific goal. Thus, OMACS assumes that a role implies some minimal expected behavior. For instance, it would be assumed that someone playing the "mail clerk" role in a company would pick up mail from the mailroom and eventually deliver that mail to its addressee. This minimal behavior defines the functionality associated with the role. Although an understanding of this behavior is critical to the design and operation of the actual system, it is not critical to the definition of the organization of the system and is not specified further in OMACS.

A role's `rcf` describes the ability of any agent to play a specific role; it is user defined and computed in terms of the capabilities required to play the role. For instance, if all the capabilities required to play a role `r` are equally important, the designer can use the default `rcf` function defined below, which ensures the `rcf` falls in the range [0..1]. If any of the agent's capabilities that are required to play the role are 0, then the result is 0; otherwise, it is simply the average of the possesses values for all the required capabilities.

$$\text{rcf}(a,r) = \begin{cases} \text{if} \quad \prod_{c \in requires(r)} possesses(a,c) = 0 & 0 \\ \text{else} & \dfrac{\sum_{c \in requires(r)} possesses(a,c)}{|requires(r)|} \end{cases} \tag{1}$$

However, simply having the required capabilities may not necessarily be sufficient to determine whether an agent can actually play the role or decide which agent can best play the role. For

instance, in a cooperative robotic system, the Search role may require both *mobility* and *remote sensing* capabilities. However, for a particular application due to the large amount of territory to be covered, the designer might only want to consider robots with a `possesses` value for mobility of 0.5 or greater for assignment to the Searcher role. In addition, some capabilities may be more important to the role than others, thus requiring some kind of weighting system. To capture this on a role-by-role basis, the designer can define a role specific `rcf`, which computes a value in the range of [0..1]. The role capability function allows the role designer to specify how specific capabilities affect the ability of an agent to play that role. OMACS uses the notation `r.rcf(a)` to denote the application of the role capability function for role `r` on agent `a`.

3.4 Agents

OMACS also includes a set of heterogeneous agents (`A`) in each organization. As described by Russell and Norvig, an agent is an entity that perceives and can perform actions upon its environment [37], which includes humans as well as artificial (hardware or software) entities. For our purposes, we define agents as computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals. Thus, we assume that agents exhibit the attributes of autonomy, reactivity, pro-activity, and social ability. *Autonomy* is the ability of agents to control their actions and internal state. *Reactivity* is an agent's ability to perceive its environment and respond to changes in it, whereas *pro-activeness* ensures agents do not simply react to their environment, but that they are able to take the initiative in achieving their goals. Finally, *social ability* allows agents to interact with other agents, and possibly humans, either directly via communication or indirectly through the environment.

Within the organization, agents must have the ability to communicate with each other, accept assignments to play roles that match their capabilities, and work to achieve their assigned goals.

3.5 Capabilities

The set of capabilities, `C`, in an organization is the union of all the capabilities required by roles or possessed by agents in the organization.

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

Capabilities are the key to determining exactly which agents can be assigned to which roles within the organization. *Capabilities* are atomic entities used to define a skill or capacity of agents. Capabilities can be used to capture *soft* abilities such as the access to/control over specific resources, the ability to communicate with other agents, the ability to migrate to a new platform, or the ability to carry out plans to achieve specific goals. Capabilities also capture the notion of *hard* capabilities that are often associated with hardware agents such as robots. These hard capabilities are generally described as sensors, which allow the agent to perceive a real world environment, and effectors, which allow the agent to act upon a real world environment.

3.6 Assignment Set

The *assignment set* Φ is the set of agent-role-goal tuples $\langle a, r, g \rangle$, that indicates 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 Section 3.14), 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. The selection of Φ from the set of potential assignments is defined by the organization's reorganization function as discussed in Section 5.2.

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 (which is specified below in Equation 3) and that only one agent may be assigned to ahieve a goal at a time (Equation 4).

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

$$\forall\ a1,a2:A\ r1,r2:R\ g1,g2:G\ \langle a1,r1,g\rangle \in \Phi \wedge \langle a2,r2,g\rangle \in \Phi \wedge a1=a2 \quad (4)$$

3.7 Policies

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 (P_Φ) *behavioral* policies (P_{beh}), and *reorganization* policies (P_{reorg}).

3.7.1 Assignment Policies

In general, OMACS allows the assignment of any agent a to any role r in order to achieve any goal g, as long as potential(a,r,g)>0. However, in a specific application, there may be additional constraints that the assignment set, Φ must satisfy. These constraints are captured in the form of *assignment policies.* Thus, assignment policies, P_Φ, constrain the assignment set Φ.

In many cases, generic policies such as "an agent may only play one role at a time" or "agents may only work on a single goal at a time" are useful and are shown below.

$$\forall\ a1,a2:A\ r1,r2:R\ g1,g2:G\ \langle a1,r1,g1\rangle \in \Phi \wedge \langle a2,r2,g2\rangle \in \Phi \wedge a1=a2 \Rightarrow r1=r2$$
$$\forall\ a1,a2:A\ r1,r2:R\ g1,g2:G\ \langle a1,r1,g1\rangle \in \Phi \wedge \langle a2,r2,g2\rangle \in \Phi \wedge a1=a2 \Rightarrow g1=g2$$

However, policies are often application specific, such as requiring particular agents to play specific roles or that the correct number of agents are playing specific roles. In an information system application, it might be necessary to ensure that no more than two agents are assigned to play roles that interface to a specific database in order to reduce resource contention. If the specific role that has access to the database is named DBAccess, then we could specify such a policy as (where # is the cardinality operator)

$$\#(\{a \mid \langle a,DBAccess,g\rangle \in \Phi\}) \leq 2$$

The language used to define policies will be implementation specific and will consist of names for the entities and relationships from OMACS (e.g., potential, Φ, etc.) as well as application specific terms such as role, goal, and capability names, which are defined in the organization's domain model, Σ.

To see if an individual assignment ϕ or an assignment set Φ, are legal according to current policies, OMACS defines two legal operations. (**P**(P) refers to the set of all organization policies.)

$$legal: \phi, \mathbf{P}(P) \rightarrow Boolean$$
$$legal: \Phi, \mathbf{P}(P) \rightarrow Boolean$$

The first operation takes a single assignment and determines its legality according to a set of policies while the second operation determines the legality of a set of assignments according to a set of policies. It will often be the case that a single assignment is legal by itself, however, when it is included into a set of assignments, it may become illegal due to policies such as the first two presented in this section.

3.7.2 Behavioral Policies

Behavioral policies P_{beh} define how agents in the organization should behave in relation to one another. For instance, in a conference review system, we would want to describe responsibilities of agent playing specific roles and their relationships to other roles. Although behavioral policies have been identified as part of OMACS, they are not yet fully defined. We are currently investigating the details of their formal language and semantics. However, the following presents a notional overview of how behavioral policies might be used in OMACS.

To refer to an agent playing a particular role, organizational predicates (`achieves`, `capable`, `possesses`, and `potential`) or assignment set membership can be used. Thus, to test whether a particular agent is playing a particular set of roles (e.g., the agent making final decisions cannot be an author of any papers for the conference), we can test for inclusion in Φ as follows.

$$\forall\ a{:}A,\ g1,g2{:}G\ \neg(\langle a,Author,g1\rangle \in \Phi \wedge \langle a,DecisionMaker,g2\rangle \in \Phi)$$

Although it is possible to state some requirements using only concepts from OMACS, other cases require the use of relationships between roles based on system/environment data. For instance, in the conference management system the relationships between roles based on the papers submitted, reviewed, or collected are vital. Thus, we must be able to talk about the data in the system as well, which is defined by the domain model, Σ.

3.7.3 Reorganization Policies

Application specific approaches to reorganization allow the designer to define heuristics to guide the system in its reorganization. For instance, instead of using a generic algorithm, the designer could specify the order in which agents should fill roles. OMACS models these heuristics as a special set of organizational policies called *reorganization* policies, P_{reorg}. Reorganization policies allow the designer to specify default reorganization strategies that are used prior to expensive computational approaches (see Section 5.2). Reorganization can first be attempted using these reorganization policies. If reorganization fails, these policies may be ignored and reorganization attempted using general purpose (and more expensive) approaches. Application specific rules can increase the reasoning efficiency in anticipated scenarios while providing robustness for unknown or uncommon cases[1].

While assignment policies simply restrict possible organizations, reorganization policies are used to direct actions taken during reorganization. An example of a possible application specific reorganization rule is shown below.

$$\langle a1,r1,g1\rangle \in \Phi \wedge \neg capable(a1,r1) \wedge capable(a2,r1) \Rightarrow \langle a2,r1,g1\rangle \in \Phi' \wedge \langle a1,r1,g1\rangle \notin \Phi'$$

[1] An excellent example of this is given in [39] where human intuition led operators to propose reorganization when the automated algorithm deemed it unnecessary. Later analysis showed the operator's proposed reorganization was globally optimal.

Here, Φ' refers to the assignment set after the reorganization occurs. In this case, the rule specifies that if agent `a1` is playing `r1` to achieve goal `g1` and `a1` becomes incapable of playing role `r1`, then if `a2` is capable of playing role `r1`, it should be assigned to goal `g1` and `a1` should be de-assigned.

3.8 Domain Model

The 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 diagrams. They include object classes that each have a set of attribute types. 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.

The domain model Σ is a tuple ⟨O, *Rel*⟩ where

- O set of object types, which consists of public attributes
- *Rel* relation over O × O that defines various relationships between object types

An *object* O is a tuple of ⟨Attributes, C⟩ where

- Attributes set of tuples ⟨name, type, value⟩ defined in the normal manner
- C set of constraints over Attributes

There are three types of relations in Rel,

- Rel_{Agg} a type of *Rel* denoting general aggregation relations between object types
- Rel_{Gen} a subset of *Rel* containing generalization-specialization relations between object types
- Rel_{Ass} a subset of *Rel* containing general associations between object types

The associations in a domain model can be used to define functions for talking about relations between environment object types. For instance, Fig. 4 shows a domain model for a conference review system.

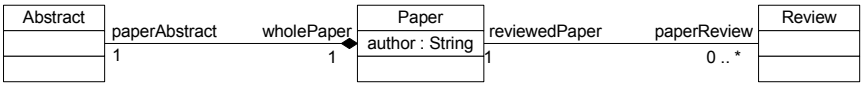


Fig. 4. Conference Management Domain Model

The domain model defines an environment with a set of papers, each with an associated abstract and a set of reviews. Using the relations defined in the model, we can talk about the reviews a paper has received *paperReview(p)* or a paper's abstract *paperAbstract(p)*, etc. Multiplicities may constraint the number of allowable environment objects or the number of objects that may be related. Fig. 4 defines a model where each abstract must have exactly one paper and each paper must have exactly one abstract. It also specifies that a review must be related to a single paper, while a paper may have any number of reviews on it (including none). Thus several organizational constraints can be defined in the domain model itself.

3.9 Organizational Assignment Function

Ideally, an organization will select the best set of assignments to maximize its ability to achieve its goals. 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.

$$\texttt{oaf: } \Phi \rightarrow 0 \texttt{ .. } \infty$$

With the `oaf`, the organization designer can specify how to make assignments based on a variety of organization specific constraints such as the importance of the specific goals or whether the assignment of multiple agents to a given role and goal will improve goal satisfaction. In the absence of an organization-specific organizational assignment function, we often define the `oaf` as the sum the potential scores in the current assignment set Φ.

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

3.10 Achieves

The *achieves* function defines how effective a role is for achieving a specific goal. It can be predefined by the organization designer or learned before or during system operation. Each role is responsible for achieving specific system goals and may actually be able to achieve multiple goals. However, since some roles are better for achieving certain goals than other roles, OMACS must have an approach to determine which roles are preferred for which goals. For instance, if the system had a goal to search an area, it might have multiple roles that could actually achieve the goal. Role A might require an airborne agent while role B might require only a land-mobile agent. As role A could perform the task more quickly, it could be given a higher *achieves* value as opposed to role B to indicate that it is the preferred role in this application. Therefore, OMACS defines an `achieves` function that describes how effective a role is for achieving a specific goal. The `achieves` function is a total function from the cross product of roles and goals to a real value in the range of 0 to 1.

$$\texttt{achieves: R, G} \rightarrow 0 \texttt{ .. } 1$$

Thus a role that cannot achieve a particular goal has an `achieves` value of 0, while any role that can achieve a goal would have an `achieves` value greater than zero. The achieves function is used along with the capable function (defined in Section 3.13) to define the potential of a specific assignment (see Section 3.14).

3.11 Requires

In order to perform a particular role, agents must possess a sufficient set of *capabilities* that allow the agent to carry out the role and achieve its assigned goals. For instance, to play the "president" role, a person would be expected to have knowledge of the corporation's domain, experience in lower-level jobs in similar types of companies, and experience in managing people and resources; an artificial organization is no different. Roles *require* a certain set of capabilities while agents possess a set of capabilities (see Section 3.12).

$$\texttt{requires: R} \rightarrow \mathbf{P}(\texttt{C})$$

All roles require some level of capability, even if it is purely computational or communicative. Therefore, OMACS dictates that all roles *require* at least one capability.

$$\forall\ \texttt{r:R requires(r)} \neq \{\} \tag{6}$$

3.12 Possesses

To be able to play a specific role, an agent must *possess* the capabilities required for that particular role. To capture a given agent's capabilities, we define a `possesses` function, which returns a value in the range of 0 to 1. The `possesses` function defines the quality of each capability that an agent has; 0 represents no capability while a 1 represents a high quality capability.

$$\texttt{possesses: A, C} \rightarrow \texttt{0 .. 1}$$

Because agent capabilities may improve or degrade over time, the output of the `possesses` function is dynamic. Agents may learn and thus (hopefully) improve an agent's capability. However, an agent's capability may also degrade through either hardware failure or loss of access to/competition over a particular resource. As an agent's possesses function changes, the ability of the agent to play specific roles also changes as computed by the role's *rcf* function. If a capability can improve or degrade in more than one dimension (for example, accuracy versus range), the designer must currently convert those dimensions into a single value in the range of 0 to 1. We are actively investigating ways to explicitly model multi-dimension capabilities (see Section 8.1).

3.13 Capable

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.

$$\texttt{capable: A, R} \rightarrow \texttt{0 .. 1}$$

As described above, since the capability of an agent, `a`, to play a specific role, `r`, is application and role specific, OMACS provides a role capability function, `rcf` to compute the capable function for each agent-role pair. Thus, the *capability score* of an agent playing a particular role is defined via the designer defined role capability functions (rcf) for each organizational role.

$$\forall\ \texttt{a:A r:R capable(a,r)=r.rcf(a)} \qquad (7)$$

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.

$$\forall\ \texttt{a:A, r:R capable(a,r)>0} \Leftrightarrow \texttt{requires(r)} \subseteq \texttt{\{c | possesses(a,c)>0\}} \qquad (8)$$

Because it is defined purely in terms of the *rcf*, the *capable* function is actually redundant. However, we believe that the capable function is intuitive and is useful in terms of having a single function that applies to all roles and agents.

3.14 Potential

One of the goals of an organization is to provide a mechanism to distribute goals in such a way that agents work together toward accomplishing the top-level organization goal. As described above, these goals are achieved by assigning agents to specific roles in the organization. However, because the agents in an organization may be heterogeneous, some agents may play a particular role better than others. The *potential* function captures the ability of an agent to play a role in order to achieve a specific goal; it maps each agent-role-goal tuple to a real value ranging from 0 to 1, where 0 indicates that the agent-role-goal tuple cannot be used to achieve the goal. A non-zero value indicates how well an agent can play a role in order to achieve a goal.

$$potential: A, R, G \rightarrow 0 \; .. \; 1$$

The potential of an agent to play a specific role in order to achieve a specific goal is defined by combining the capable and achieves functions.

$$\forall \; a{:}A \; r{:}R \; g{:}G \quad potential(a,r,g) = achieves(r,g) * capable(a,r) \qquad (9)$$

3.15 Organizational Agents

Organizational agents (OA) are organizations that function as agents in a higher-level organization. OAs allow OMACS to represent a hierarchy of organizations, providing OMACS with both flexibility and scalability. 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 [17, 18] and are similar to concepts in the organizational metamodel proposed by Odell [33].

OMACS defines two relationships between the higher-level organization and the OA's internal organization. First, there must be a connection between the role being played in the higher-level organization and the OA's internal `oaf` function. Second, a specific relationship must exist between the OA's internal capabilities and those of the higher-level organization.

Because the role the OA is playing will affect the internal organization of the OA, there must be a way to relate the organizational assignment function of the OA to its role. However, the `oaf` is defined as having no parameters and only has access to the local organizational components (see Section 3.9). Therefore, an OA must extend the definition of an organization by adding a new `oaf` function that allows it to take a parameter that includes a set of roles from the higher-level organization. Thus, an OA is an organization with one extension, a polymorphic `oaf` function that takes as input an assignment set along with a set of roles it has been assigned to play in the higher-level organization. Again, the polymorphic `oaf` function is application specific and must be written to take into account the specific roles the OA can take on in the higher-level organization.

$$oaf : \Phi, R \rightarrow 0 \; .. \; \infty$$

The relationship between the capabilities in an OA and those of the higher-level organization is actually straightforward. Essentially, if a capability belongs to an agent that is part of the OA's internal organization, then those capabilities also exist in the higher-level organization by inclusion. Thus, if an OA, `a`, exists as an agent in an organization, `o`, then the capabilities possessed by `a` in `o` must be equivalent to the entire set of capabilities possessed by the individual agents in `a`'s internal organization. (Dot notation is used to differentiate between the capabilities of the organizations represented by `a` and `o` respectively.)

$$a.C \subseteq o.C \qquad (10)$$

$$\forall \; ag{:}a.A, \; c{:}a.C \quad a.possesses(ag,c) > 0 \Rightarrow o.possesses(a,c) > 0 \qquad (11)$$

Notice that we also stop short of defining the actual possesses score for these capabilities in the higher-level organization. This is because there may be multiple agents in the OA's internal organization with the same capability. Thus, the actual possesses score will be application specific.

Given the definitions above, it is possible for an organization to possess capabilities that are not possessed by individual agents. We are currently investigating the notion of *composed capabilities* that would allow a designer to define higher-level capabilities that consist of several lower level capabilities. For instance, if an organization had individual robots with the

capabilities to Search, Carry, and Communicate, then the composition of those capabilities at the organizational level could result in a Rescue capability for the organization as a whole.

4. Organization Viability

The constraints above define the legality of the organization structure and its instances. However, we are also interested in whether or not an assignment of agents to roles satisfying all the organizational policies exists that *can* allow the system to achieve its goals, which we refer to as *organizational viability*. Although an organization may be structurally valid, there is no guarantee that an instance of that organization exists that *can* achieve its goals. In actuality, we can never guarantee that the system *will* ever achieve all its goals due to the dynamic nature of the environment in which the organization operates. To achieve the organizational goals, the system must have the right mix of agents to play the right roles to achieve those goals. Essentially, a viable organization is a valid organization that has been populated with the right types and numbers of agents so that it might potentially achieve its goals.

> ***Viability*** – an organization, O, is viable if there exists a series of assignments of agents to roles to goals consistent with its policies `P` that can achieve all the goals in `G`.

For an organization to be viable, according to the definition above, it must have the roles and agents to achieve its goals under ideal conditions (no agent failures, etc.). Therefore, we define a viable organization as an organization that is able to show that the organization goals are achievable by some set of assignments of goals, roles, and agents. When a given goal is achievable by a set of assignments, we term that goal *satisfiable*. Therefore, viability refers the satisfiability of all goals in `G`.

Viability does require that the set of assignments used to determine satisfiability is consistent with (or legal) the organizations policies `P`. Thus, to show viability, we must show that an organization's goals are satisfiable using only legal sets of assignments.

To formalize the notion of viability, we need to introduce the notion of a *sequence* of assignment sets Φ^*. First, we can define the notion of a sequence of goal sets, $G'=[G_0, G_1, \ldots G_n]$, where G_i represents the current set of active goals at time *i* and is equivalent to G_{i-1} modified by the removal of goals that were achieved and the addition of new goals based on various events that may occur. Thus, Φ^* is defined based on the sequence of goal sets

$$\Phi^* = [\Phi_0, \Phi_1, \ldots \Phi_n]$$

where Φ_i is a set of assignments corresponding to the goal set G_i.

$$\Phi_i \subseteq \{\langle a,r,g\rangle \mid a \in A \wedge r \in R \wedge g \in G_i \wedge potential(a,r,g)>0\}$$

Finally, we can define viability, viable(O), as a predicate that determines satisfiability of a given organization, `O` where the *viable* predicate is defined as

$$viable(O) = \exists \Phi^* \mid \forall g{:}G\ satisfiable(g,\Phi^*) \wedge legal(\Phi^*,P) \qquad (12)$$

The satisfiability of a goal and the legality of an assignment set are discussed below.

4.1 Satisfiable

Essentially, a goal is satisfiable if we can find a role that can achieve that goal and an agent that can play that role. This assignment must be part of some assignment set in the assignment sequence. Formally, we define *satisfiable* as

$$satisfiable(g,\Phi^*) \Leftrightarrow \exists\ a{:}A,\ r{:}R,\ \Phi_i{:}\Phi \mid achieves(r,g)>0 \wedge capable(a,r)>0$$

$$\wedge\ \Phi_i \in \Phi^* \wedge \langle a,r,g \rangle \in \Phi_i \quad (13)$$

4.2 Legal

We define the *legality* of a sequence of legal assignments, $\Phi^* = [\Phi 1, \Phi 2, \ldots \Phi n]$, in terms of the legality of the individual assignments. To accomplish this, we define an additional legal predicate based on the basic legal predicates defined in Section 3.7.1.

$$\text{legal}: \Phi^*, \mathbf{P}(P) \rightarrow \text{Boolean}$$

Essentially, a sequence of assignments is legal if each assignment set in the sequence is legal.

$$\text{legal}(\Phi^*, P) = \text{legal}(\Phi_1, P) \wedge \text{legal}(\Phi_2, P) \ldots \text{legal}(\Phi_n, P) \quad (14)$$

5. Organization and Reorganization

Each organization has an implicitly defined *organization transition function* that describes how the organization may transition from one organizational state to another over the lifetime of the organization. Since agents in an organization as well as their individual capabilities may change over time, this function cannot be predefined, but must be computed based on the current state, the goal set, G, and the current policies. In our present research with purely autonomous systems, we have only considered reorganization that involves the *state* of the organization. However, we have defined two distinct types of reorganization: *state reorganization*, which only allows the modification of the organization state, and *structure reorganization*, which allows modification of the organization structure (and may require state reorganization to keep the organization consistent). We define the *state* of the organization as the set of agents, A, the `possesses`, `capable`, and `potential` functions, and the assignment set, Φ. However, not all these components may actually be under the control of the organization. For our purposes, we assume that agents may enter or leave organizations or relationships, but that these actions are triggers that cause reorganizations and are not the result of reorganizations. Likewise, `possesses` (and thus `capable` and `potential` as well) is an automatic calculation that determines the possible assignments of agents to roles and goals in the organization. The calculation of `possesses` is the only calculation totally controlled by the agent; the organization can only use this information in deciding how to make assignments. This leaves one element that can be modified via state reorganization: Φ.

5.1 Reorganization Triggers

Various events may occur in the lifecycle of a multiagent system that may require it to reorganize. In general, *reorganization* is initiated when an event occurs that changes the state of the current organization. As we are currently only investigating state-based changes, we have only considered events that change the state of the organization. Thus, we have currently identified two types of events of interest: changes in goals and changes in agents, each of which may cause a change in Φ. We discuss these two situations in detail below.

5.1.1 Goal Set Changes

Any change in G may cause reorganization. There are three basic types of events that can cause a change in G: (1) insertion of a new goal, (2) goal achievement, and (3) goal failure. Each of these is discussed below.

The first situation deals with new goals being added to G. However, we cannot say with certainty that reorganization will occur based on a new goal in G. It is possible that the organization will

choose to forego reorganization for a number of reasons, the most likely being that it has simply chosen not to pursue any new goals added to G at the present time.

The second case deals with goal achievement. When a goal g is achieved, G is changed to reflect that event by (1) removing g from G and (2) possibly adding new goals, which are enabled by the achievement of g, into G. Obviously, the agent assigned to achieve goal g is now free to pursue other goals.

The third instance involves goal failure, which really has two forms: agent-goal failure and goal failure. When a specific agent cannot achieve goal g but g might still be achievable by some other agent, *agent-goal failure* occurs. When agent-goal failure occurs, reorganization must occur to allow the organization to (1) choose another agent to achieve g, (2) not pursue g at the current time, or (3) choose another goal to pursue instead of g. In any of these situations, g is not removed from G since it has not been achieved. In the case where the organization or the environment has changed such that a goal g can never be achieved, then *goal failure* occurs. In this case, g is removed from G and the organization must attempt to assess whether it can still achieve the overall system goals. Reorganization may occur to see if the agent assigned to achieve g can be used elsewhere. In all cases, the selection of the appropriate strategy is left to the organization.

5.1.2 Agent Changes

The second type of change that triggers reorganizations are changes to the set of agents, A, or their individual capabilities. When an agent that is part of Φ is removed from the organization, a reorganization must occur, even if only to remove the agent and its assignment(s) in Φ. Likewise, when an agent that is part of Φ loses a capability that negates its ability to play a role that it is assigned, reorganization must occur as well.

In general, when changes occur in an agent's capability, reorganization may or may not be necessary, based on the agent's capable relation. We have identified four specific types of changes in an agent's capabilities that may indicate a need for reorganization: (1) when an agent gains the ability to play a new role, (2) when an agent loses the ability to play a role, (3) when an agent increases its ability to play a specific role, or (4) when an agent decreases its ability to play a specific role. While case 2 requires reorganization if the agent is currently assigned to play the role for which it no longer has the capability to play, whether or not to reorganize is left up to the organization when the other three cases (along with 2 when the agent is not currently assigned that role) occur.

5.2 Reorganization

Reorganization is the process of changing the assignments of agents to roles to goals as specified in Φ. The organization's oaf function is used to determine the best new Φ; however, total reorganization may not be necessary or efficient. (In the absence of any information or policies, an optimal total reorganization would take on the order of $2^{A \times G \times R}$.)

One approach is to take a *local view*, in which the organization looks at the OMACS state and reorganizes in a locally optimal fashion (i.e. hill climbing). However, when dealing with dynamic environments, it is often desirable to reorganize so that the team can operate more efficiently or effectively in its present situation as well as being adaptable to its changing environment. Thus, we would like to take a long-range or *global* view. Unfortunately, it has been shown that in the general case globally optimal reorganizations are NEXP-complete and, thus impractical for most applications with any time constraints [32]. Therefore, OMACS provides a mechanism for

augmenting the locally optimal algorithm with application specific rules in an attempt to make reasoning more efficient and to enable globally better solutions.

5.2.1 General Purpose Reorganization Examples

For general-purpose reorganization, we have developed several reorganization algorithms that give us a default reorganization capability. When a reorganization trigger occurs, general-purpose reorganization algorithms can be used to find appropriate assignments to achieve the organizations goals, if possible. To compute the best reorganization, an algorithm that simply optimizes the organization's oaf might seem appropriate; however, this approach is short sighted. First, it does not deal with the cost associated with reorganizing and, second, it does not consider the reason reorganizing was initially undertaken. Exploiting reorganizing costs requires a distributed solution since the cost for robots to change roles is not globally known. For instance, if an agent is required to perform a complex computation, any effort toward that computation would be lost if the agent was reassigned to another role/goal. Considering the reason for reorganization may enable less extensive (and less costly) reorganization. If the reason for reorganizing is to fill a single role, then a total reorganization may be a waste of time and resources.

We have developed several reorganization algorithms from sound and complete total reorganization algorithms to greedy algorithms [48, 49]. As expected, the sound and complete total reorganization algorithm is extremely slow, especially when the organization lacks any policies that limit the number of legal combinations of agents, roles, and goals. The greedy algorithms also perform as expected, giving low computational overhead and producing generally adequate organizations. We have also looked at learning reorganization algorithms [27].

A general purpose reorganization algorithm that produces an optimal solution with OMACS is shown in Fig. 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 remove 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.

```
function reorganize(oaf, G_w, A_w)
   1. for each g ∈ G_w
   2.   for each role r ∈ R
   3.     if achieves(r,g) > 0 then m ← m ∪ ⟨r,g⟩
   4. ps ← P_Φ(powerset(m))
   5. for each agent a ∈ A_w
   6.   for each set s ∈ ps
   7.     if capable(a,s.r) then pa ← pa ∪ ⟨a,s⟩
   8. pas ← powerset(P_Φ(pa))
   9. for each assignment set i from c
   10.  for each assignment x from pa
   11.    Φ ← Φ ∪ ⟨x.a,x.s_i⟩
   12.  if P_Φ(Φ) is valid
   13.    if oaf(Φ) > best.score then best ← ⟨oaf(Φ),Φ⟩
   14. return best.Φ
```

Fig. 5. General Reorganization Algorithm

Assignment policies can have significant effects on the time complexity of reorganization algorithms. For example, assume an agent is able to play five roles and each role achieves three goals; without any assignment policies, the agent has $2^{5 \times 3} = 32{,}768$ possible assignments. However, with a simple policy that states that "agents can only play one role at a time", the agent only has 5×3 or 15 possible assignments. If there are four of such agents, the possible assignments are reduced from $32{,}768^4$ or 1,152,921,504,606,846,976 to 15^4 or 50,625. The algorithm in Fig. 5 was run "as is" using no policies as well as with the policy "agents can only play one role at a time". While there are two locations that policy checking occurs, this policy makes an impact in the first check in line 4 only. As expected, the original version of the algorithm with the policy performs better than the version with no polices due to the smaller number of possible assignments. We also modified the algorithm in Fig. 5 by replacing the power set and policy checking functions in line 4 with a custom function that generates only valid assignment sets based on the policy. This version performed much better (an order of magnitude) than both the original and first version on reasonable size organizations as the custom function does not waste time generating unnecessary assignment sets. Details of these results can be found in [48, 49].

5.2.2 Application Specific Approaches

As stated in Section 3.7.3, reorganization policies allow organization designers to specify application-specific rules that can be checked prior to running costly general-purpose reorganization algorithms. If used, a reorganization trigger initiates a two step reorganization process. First, the reorganization policies are checked to see if any of them can be triggered. If so, the rule is applied and the new organization is checked for validity. If no applicable reorganization policies can be found or if they do not result in viable organizations, the general-purpose reorganization algorithm is run to come up with a new organization.

6. Battlefield Information System

To demonstrate the effectiveness of OMACS, we implemented a simulated Battlefield Information System (BIS). The purpose of the BIS is to provide an information system that can adjust its processing algorithms and/or information sources to provide required information at various levels of efficiency and effectiveness [29]. In this system, various types of sensors at different locations are used to detect enemy vehicles. These sensors are subject to failure and

erroneous outputs and typically have a delay in getting the information categorized. When sensor data of interest is available, it is fused with other related information to answer queries from the commander. Queries are generated by a field commander via the system interface. There are two types of queries that can be generated: transient and persistent. Transient queries are executed only once whereas persistent queries are carried out repeatedly until canceled. To be able to overcome the loss of sensors and continue to provide the required information, the Battlefield Information System needs to adapt by replacing the failed sensors and adapting the information processing adequately.

6.1 Organization Design

To implement the BIS, we had to design each of the main entities of the OMACS model. These goals, roles, capabilities and agents are defined in the following paragraphs. To simplify the example, we assume all capabilities required by a role are equally important to that role (thus using the default *rcf* given in Equation 3) and that an agent either possesses a capability or not (possesses(a,c) is either 1 or 0). We also use the default *oaf* function given in Equation 5 and an algorithm similar to the reorganization algorithm in Fig. 5 to compute the best configuration at a particular point in time.

6.1.1 Goals

The main goal of the application is to answer each query presented to the system. From the requirements, we derived a set of goals that the organization must satisfy. Each goal listed below defines a *type* of goal that may be instantiated within the BIS. During the pursuit of specific goals, events may occur that cause the instantiation of new goals. If event E can occur during the pursuit of goal A causing the instantiation of a second goal B, we say that goal A is capable of *triggering* goal B. In operation, this means that event E is recognized by the agent pursing goal A. Once the agent recognizes event E, the agent passes E to the organization, which is responsible for creating a new instance of goal B and eventually assigning an agent to play a role to achieve goal B.

These instantiated goals may be parameterized to allow the goal to take on a context sensitive meaning. For instance, to achieve the Process Query goal (G1 below), it is assigned to an agent who waits for queries to be submitted from the commander. When a query arrives, the agent assigned to achieve G1 forwards this event (the receipt of a query) to the organization, which causes the instantiation of a new goal, Find Sensors (G2). This new instance of G2 is parameterized based on the query received. If a second query arrives, a new instance of the Find Sensor goal is instantiated with the specific query as its parameter.

Goal	Name	Description
G1	*Process Query*	Get the query from the user. There is only one instance of this goal that is created upon system initialization. As described above, this goal is capable of triggering G2.
G2	*Find Sensors ⟨Q: Query⟩*	Find all the sensors in the area of interest for the query Q. This goal is triggered by G1 and is parameterized with the query. After finding a set of sensors that can fulfill the query, it triggers G3 and either G4 or G5.
G3	*Read Sensor ⟨S: Sensor⟩*	Read the data from the sensor S given in parameter. This goal does not trigger any additional goals.

G4	*Merge Diverse ⟨Q: Query, L: ⟨Sensors⟩⟩*	Fuse the data received from the list of different types of sensors L for the area specified by query Q. G4 may trigger G6, G7, G8, G10, or G11. It triggers G9 once a result is ready. If an error occurs during the merging of the data (loss of a sensor, etc.), it can cause a *negative trigger*, which removes all goal instances related to a particular query and triggers a new instance of goal G2 with the same query parameter. This results in a set of new goals based on the current set of available sensors.
G5	*Merge Similar ⟨Q: Query, L: ⟨Sensors⟩⟩*	Fuse the data received from the list of similar sensors L for the area specified by query Q. Sensors in L must all be the same type. G5 has identical triggers and negative triggers as G4.
G6	*Filter Information ⟨Q: Query⟩*	Filter the merged data based on the information required by the query Q.
G7	*Correlate Data ⟨Q: Query⟩*	Compare data with historical data in order to extract persistence information if the query Q is a persistent query.
G8	*Add Information ⟨Q: Query⟩*	Look up additional information if required by the query Q.
G9	*Return Result ⟨Q: Query⟩*	Display the result of the query Q in a user-friendly format.
G10	*Monitor Time Constraints ⟨Q: Query⟩*	Check the validity of the data regarding the time constraint specified by query Q. If the time constraint specified by the query is violated, a *negative trigger* removes all goal instances related to that particular query and triggers a new instance of goal G2 with the same query parameter.
G11	*Monitor Accuracy Constraints ⟨Q: Query⟩*	Check the validity of the data regarding the accuracy constraint specified by query Q. If the accuracy constraint specified by the query is violated, a *negative trigger* removes all goal instances related to that particular query and triggers a new instance of goal G2 with the same query parameter.

Note that goals G10 and G11 are *maintenance goals*. When an agent is assigned a maintenance goal, the agent is responsible for monitoring the maintenance condition and taking action when that condition is violated.

6.1.2 Roles

The roles for the BIS were derived directly from the goals; for each goal we created a role to achieve it. A role achieving a parameterized goal is able to achieve all different parameterized instances of this goal. To simplify the system, we have assigned an achieve score of 1 if the role can achieve a goal, or 0 if it cannot. Following are the roles we have defined for the BIS organization, along with the goals they can achieve. We also describe the behavior of each of those roles.

Role	***Name***	**Achieves**	**Description**
R1	*Query Processor*	G1	Periodically interrogates the GUI to get any new queries entered by the commander. This role generates a *start* event to notify the system that a new query has been entered, which causes the instantiation of a G2 goal.

R2	*Sensors Locator*	G2	Queries the sensor database in order to find all sensors available in the area specified by the parameter of the goal it achieves. Then it executes an algorithm to find the best coverage based on the set of available sensors. For each sensor selected, a *found* event is generated that triggers a new G3 goal, which results in the organization attempting to find an agent capable of reading the selected sensor. After all sensors have been selected, the role generates a *mergeSimilar* or *mergeDiverse* event (based on the types of the sensors selected), which results in the instantiation of a new G4/G5 goal to merge the results coming from the selected sensors.
R3	*Sensor Reader*	G3	Reads the data from the sensor given in the parameter.
R4	*Data Merger Diverse*	G4	Merges the data collected from various sensors covering the area of interest. This role uses a processing algorithm that allows it to merge data coming from sensors of different types. Once the data is fused, it can generate events *filter*, *persistent*, or *addInfo* to trigger G6, G7, or G8 respectively. Depending on the information required by the query, the agent playing this role collaborates with other agents to process the data fused. This coordination allows the agent to formulate adequate answers to the query. Once all the processing is done, a *result* event is generated triggering the G9 goal.
R5	*Data Merger Similar*	G5	Behaves like R4. However, this role uses a processing algorithm that allows it to merge data coming from sensors of the same type efficiently. Thus, this role does not process data from different sources. Role R5 also generates the same events as R4.
R6	*Object Filter*	G6	Filters data based on the type of information required by the query given in parameter. When done, the agent playing R6 returns the data to the appropriate Data Merger agent and terminates. R6 generates no events that trigger any new goals.
R7	*Intelligence Provider*	G8	Looks up additional information about the enemies in the area as specified in the query. When done, the agent playing R7 returns the data to the appropriate Data Merger agent and terminates. R7 generates no events that trigger any new goals.
R8	*Persistence Validator*	G7	Compares new data against historical data in order to extract persistence information. This would be the case if, for example, the BIS was monitoring the entrance of new vehicles in a given area. This role only applies to persistent queries. When done, the agent returns the data to the appropriate Data Merger agent and terminates. R8 generates no events that trigger any new goals.
R9	*Result Interface*	G9	Returns the results of the query to the commander's interface. R9 generates no events that trigger any new goals.
R10	*Time Monitor*	G10	Monitors the organization's ability to return results within the specified time constraint. It communicates the results to the Data Merger agent in charge of the query and triggers a *failure* event if the constraint is violated.
R11	*Accuracy Monitor*	G11	Monitors the organization's ability to return results within the specified accuracy constraint. It communicates the results to the Data Merger agent in charge of the query and triggers a *failure* event if the constraint is violated.

6.1.3 Capabilities

To be valid, each role requires at least one capability. While some capabilities are used to interact with the environment, others allow the agent to carry out specific functional computations within the system. The capabilities identified for the BIS and the roles that require them are listed below.

Cap	Name	Requires	Description
C1	*User Interaction*	R1, R9	Used to interact with the GUI. This capability provides actions to get a query from the commander and to display the result of a query that has been executed.
C2	*Coverage Processing*	R2	Used to compute the optimal set of sensors with the maximum coverage of the area of interest that can satisfy the efficiency and accuracy constraints.
C3	*Sensor Interaction*	R3	Used to interact with the actual sensors on the battlefield. This capability provides an action to query a sensor and read its data.
C4	*Data Merging Diverse*	R4	Provides computational algorithms to merge data coming from diverse type of sensors.
C5	*Data Merging Similar*	R5	Provides fast computational algorithms to merge data coming from similar sources only.
C6	*Data Filtering*	R6	Used to filter out information that is not needed to answer a given query.
C7	*Intelligence Processing*	R7	Used to obtain additional information to answer a query. The additional information comes from existing databases (e.g., the firing range of a vehicle type).
C8	*Correlation Processing*	R8	The ability to correlate data from two successive results of a given query. Correlation exhibits differences in the results of a query obtained at different times.
C9	*DB access*	R2, R7, R8	The ability to access a database including actions to both read from or write to the BIS databases (e.g. sensors database, intelligence database).
C10	*Monitoring*	R10, R11	The ability to check the time and/or the accuracy of a query. The information about accuracy and times of the results are provided by the data sources.
C11	*Coordination*	R3, R4, R5, R6, R7, R8, R10	The ability to communicate with other agents. This capability provides actions to send/receive messages to/from specific agents in the organization. Agents must have this capability to communicate between themselves.

6.1.4 Agents

To be viable, the BIS organization must have the right types of agents capable of playing its organizational roles. To be able to play a specific role, an agent must possess the capabilities required for that particular role. The agent types with their capabilities and the roles they can play (assuming their capabilities do not degrade) are listed below.

Agent	Description	Possesses	Roles
QA	Query Agent	C1	Query Processor, Result Interface
SFA	Sensor Finder Agent	C9, C2	Sensors Locator
DSA	Data Sensor Agent	C3, C11	Sensor Reader
MAD	Merger Agent Diverse	C1, C4, C11	Query Processor, Result Interface, Data Merger Diverse
MAS	Merger Agent Similar	C1, C5, C11	Query Processor, Result Interface, Data Merger Similar

DFA	Data Filter Agent	C6, C11	Object Filter
IA	Intelligence Agent	C9, C7, C11	Intelligence Provider
DCA	Data Correlation Agent	C8, C9, C11	Persistence Validator
MON	Monitor Agent	C10, C11	Time Monitor, Accuracy Monitor

During the actual instantiation of the organization, an agent of each type is created. For the Data Sensor Agent type, each sensor on the battlefield is associated with a unique agent of type DSA. We have designed our system such that all capabilities required by a role are treated as equally important. For this reason, the role capability function for an agent playing that role is 1 if that agent possesses all the capabilities required by the role and 0 if it does not.

6.1.5 Potential Assignments

If we assume a Boolean value for the achieves, requires, and possesses functions, as indicted in Sections 6.1.2, 6.1.3, and 6.1.4 respectively, we can compute the initial potential function for each agent defined in Section 6.1.4 (using Equation 9 with the default *rcf* function for each role as defined in Equation 3). As most of the tuples input to the potential function will result in a zero (0) value, we only show tuples that result in a one (1) value as shown below.

Agent	**Role**	**Goal**	**potential(a,r,g)**
QA	R1 - Query Processor	G1 – Process Query	1
QA	R9 - Result Interface	G9 – Return Result	1
SFA	R2 - Sensors Locator	G2 – Find Sensors	1
DSA	R3 - Sensor Reader	G3 – Read Sensor	1
MAD	R1 - Query Processor	G1 – Process Query	1
MAD	R9 - Result Interface	G9 – Return Result	1
MAD	R4 - Data Merger Diverse	G4 – Merge Diverse	1
MAS	R1 - Query Processor	G1 – Process Query	1
MAS	R9 - Result Interface	G9 – Return Result	1
MAS	R5 - Data Merger Similar	G5 – Merge Similar	1
DFA	R6 - Object Filter	G6 – Filter Information	1
IA	R7 - Intelligence Provider	G7 – Correlate Data	1
DCA	R8 - Persistence Validator	G8 – Add Information	1
MON	R10 - Time Monitor	G10 – Monitor Time Constraints	1
MON	R11 – Accuracy Monitor	G11 – Monitor Accuracy Constraints	1

6.1.6 Organization State Model

Fig. 6 shows a graphical depiction of the BIS OMACS entities and their relations defined above. The boxes at the top of the diagram represent the goals, the circles represent the roles, the pentagons represent capabilities, and the ellipses are agents identified by their types. The arrows between the entities represent the achieves, requires, and possesses functions/relations. Each achieves and possesses arrow has a value of 1.

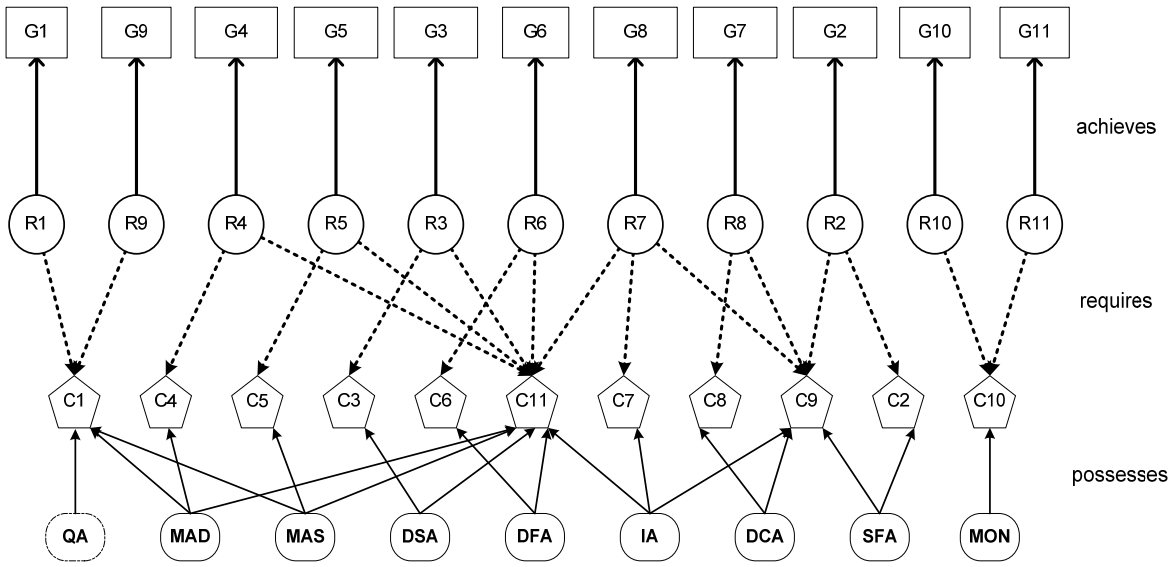


Fig. 6. BIS Organization Overview

6.1.7 Implementation Architecture

The BIS system has been designed using a centralized approach (Fig. 7). This approach allows reusability of various organizational reasoning algorithms by decoupling the organization reasoning part of the application, which can be generic, from the actual BIS system composed of application-specific agents. The system has the following entities:

- Organization Master (OM),
- Agent Reasoning (AR)
- Agent Body (AB)

The Organization Master (OM) is a specialized agent that is in charge of all organization-related tasks; it is not part of the organization and cannot be assigned a role to play. The OM is the only agent that possesses complete organization knowledge and that is able to execute reorganization algorithms. The OM uses its knowledge of the current goals and agents, makes appropriate assignments and sends the assignments to the agents via their Agent Reasoning (AR) component. The OM also receives events and the agent's status from each agent's AR and reorganizes appropriately when needed.

We used a separate OM agent strictly to simplify the implementation of the organizational reasoning and to ease the testing and debugging of the system. There is nothing in OMACS or the application domain that would have precluded us from placing the organizational reasoning of the OM into any one of the BIS agents or distributing the OM reasoning among various BIS agents using more complex distributed organizational reasoning algorithms. Distributed organizational reasoning involves a partial or total distribution of the organization knowledge and OM decision making abilities among all or some of the agents of this system. While a distributed approach would change the AR components of each agent, the agent bodies would be unaffected. Eventually, we plan to have a variety of plug-and-play AR components available for use in our organization-based systems.

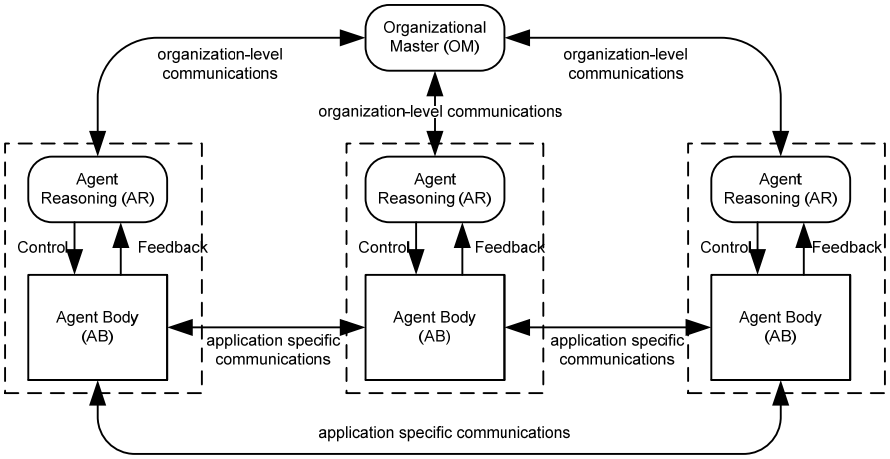


Fig. 7. Centralized Architecture

Each agent is composed of two components: an AR component and an Agent Body (AB) component. In our centralized approach, the Agent Reasoning component of the agent serves as an interface between the OM and the Agent Body. It represents the part of the agent in charge of all organization related tasks. The AR receives assignments from OM and forwards them to the Agent Body. It also reports status/failures of its attached Agent Body to the OM. The Agent Body is the application specific part of the agent as defined in OMACS. It accepts assignments from the AR, plays its assigned roles and reports its status to its reasoning component.

Communication between the OM and the agents is done by message passing via the AR. As the AR is actually part of the agent, the entities AR and AB can communicate directly via methods calls, thereby reducing the communication overheads produced by message passing. This architecture allows us to plug various organization reasoning algorithms into the systems while leaving the agents intact.

6.2 Reorganization Triggers

To adapt to a variety of unpredictable situations, our BIS organization is able to detect changes in the performance of the overall organization and modify its structure accordingly. Many of them are changes within the environment; however, some changes occur within the organization itself (e.g., capability failure or goal completion). Such changes become reorganization triggers when they either cause the organization to be unable to achieve its overall system goal within the time/accuracy constraints given or allow the system to be more efficient or effective in reaching its goal. Specifically, the BIS has four types of reorganization triggers:

- Sensor Failure
- Goal Completion
- Goal Instantiation
- Maintenance Goal Failure

6.2.1 Sensors Failure

Each Data Sensor Agent (DSA) is linked to one physical sensor from the battlefield. The failure of a sensor (e.g., S1) is taken as a goal failure, in this case G3(S1). The corresponding DSA, which is the only agent capable of playing R3 to achieve G3(S1), can no longer achieve its goal. When a sensor involved in a query fails, the Data Merger agent in charge of that query becomes

aware of this goal failure and notifies the OM via its AR component. Then, the system redistributes the sensor reading tasks among all the sensor reader agents still working and capable of covering the area of interest. In some cases, this reorganization process requires the reassignment of the Data Merger agents to ensure optimal performance. When the reorganization is completed, the query is executed again and the results are sent to the user. In a rigid system, the loss of a sensor would mean an irreversible loss of performance in the system.

6.2.2 Goal Completion

The achievement of a goal can free an agent to take on a new role and goal assignment. When this occurs, the organization may make new assignments in order to optimize the performance of the system. In this application, however, unless there are goals that have not been assigned to agent, the agents do not get reassigned until another reorganization trigger occurs.

6.2.3 Goal Instantiation

When an event occurs that triggers the instantiation of a new goal, this goal is entered into the organizations set of goals. The insertion of a new goal requires the BIS to take action to satisfy this new goal. If the organization is able to find an agent-role pair capable of achieving the goal, the agent is assigned to play that role in order to achieve the goal. In some cases, in order to find a valid assignment, the organization has to reassign some agents already playing some roles.

6.2.4 Maintenance goal failure

The user can specify time or accuracy constraints that the query needs to satisfy. To monitor the validity of those constraints, we have defined two maintenance goals: Monitor Time Constraint and Monitor Accuracy Constraint. The agents assigned to achieve those goals monitor for conditions violating the query constraints. If a constraint violation is detected, the assigned agent notifies the OM who tries to reorganize in order to meet the constraint. If the constraint cannot be satisfied, the user is notified and the query is executed with no constraints.

6.3 Example Scenario

To show how the BIS behaves, we present a scenario that exemplifies some of the adaptive behaviors explained above. To show this clearly, we describe the state of the organization after the occurrence of each event that triggers reorganization. The state of the organization is shown by the current organization goals (G) and the current set of assignments (Φ). We assume that the system is only trying to answer one persistent query and omit the query parameter for goals and triggers. The BIS organization answers the following persistent query: "*Show the location and type of all enemy vehicles in the selected area*" (the area selected is defined by a rectangle as shown in Fig. 8. Once the query is entered, it is stored by an external agent (not part of the BIS organization) in charge of the GUI. The screenshot in Fig. 8 shows the simulated battlefield along with the sensors and enemy targets.

In our BIS simulator, there are five different types of vehicles that the system is trying to locate and identify: trucks, halftrack, tank, artillery, and launcher. The accuracy of the sensors describes how accurate they are in describing the actual location of the vehicles as well as the type of vehicle. For this scenario, we have defined two types of sensors: ground sensors and airborne automatic target recognition (ATR) sensors. The ground sensors have a fixed location and provide information about location and type of enemy vehicles with an accuracy of 75%. Ground sensors are also capable of providing requested data within 5 minutes. The airborne ATR sensors are obviously mobile and are also very accurate, providing location and enemy vehicles type information with an accuracy of 95%. Unfortunately, ATR sensors are not very fast; they

typically can only provide their information in 15 minutes. For the specific scenario described below, there are four ground sensors (S1, S2, S3, S4) and one ATR sensor (S5). All sensors have a partial coverage of the area of interest. There are also five enemy vehicles in the scenario as shown in Fig. 8. As stated in Section 5.1.4, we instantiate one agent for each agent type except for the DSA agent type. Agents of type DSA will be named DSA# where # is the sensor number attached to it (for example, DSA1 is the DSA agent attached to S1). All other agent are named after their agent type.

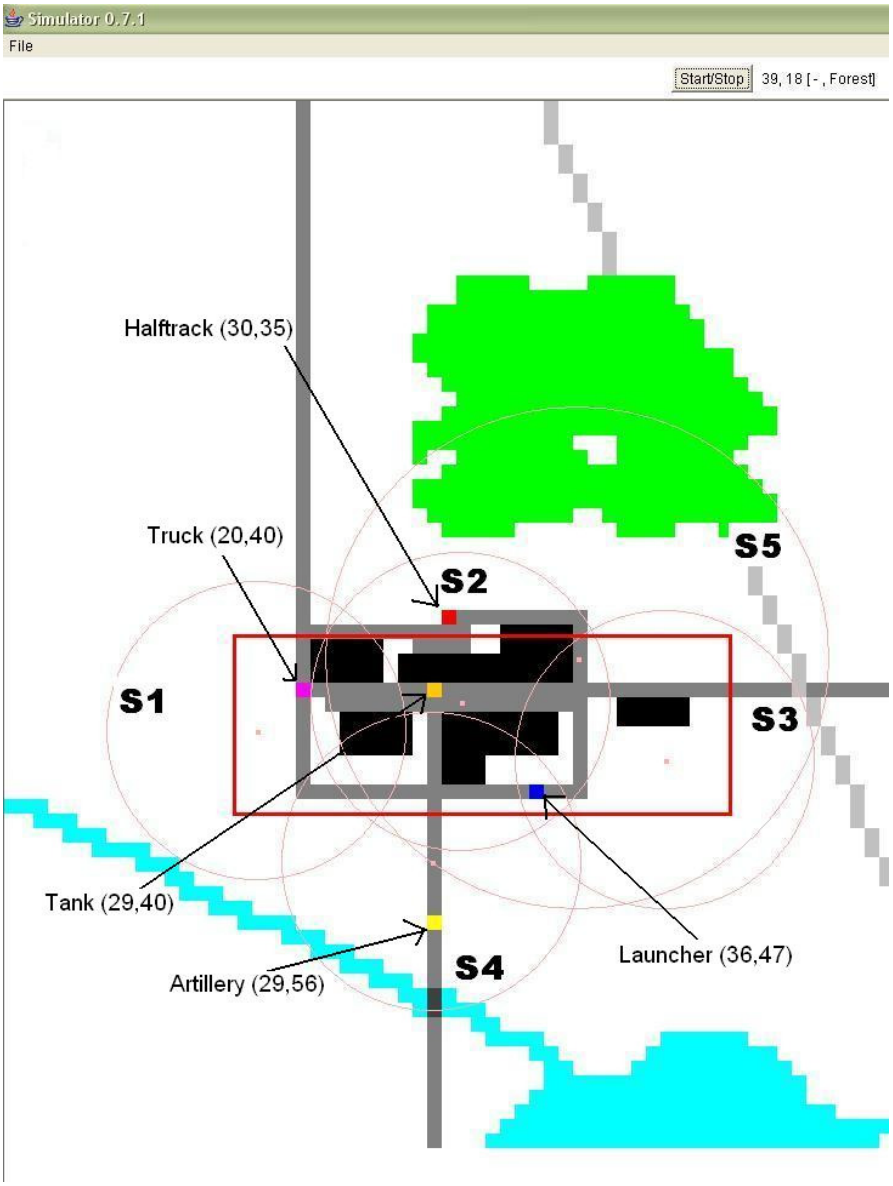
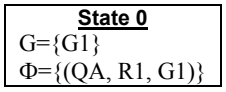


Fig. 8. Battlefield Map

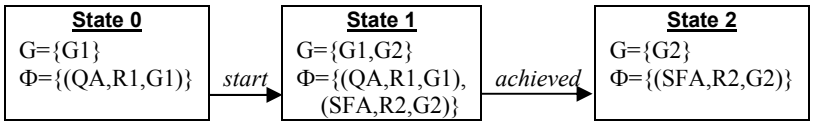
6.3.1 Normal Execution

At system initialization, we assume that all agents available to the organization are registered with the OM. Because all the goals defined in Section 6.1 except G1 are triggered by other goals,

G1 is the only goal that is initially inserted into G.[2] From the potential values given in Section 5.1.5, we can easily see that only the assignment <QA,R1,G1> has a non-zero value and can be used to achieve G1. Therefore, once the initialization process is complete, the OM runs its reorganization algorithm to produce the initial organization and, as expected, assigns the Query Agent (QA) to play role R1 in order to achieve goal G1. Thus, we obtain the following state for the organization:



The retrieval of a query from the GUI by QA is recognized as the *start(query)* event, which is sent to the OM and causes the instantiation of a G2 goal and its insertion into G. At this point, QA also informs the OM that G1 has been *achieved*, which results in G1 being removed from G. (To simplify our example, we assume QA terminates at this point; however, in the real simulation, QA stays active waiting for new queries.) As described in Section 5.1.1, any change to G requires reorganization. Once again, the OM runs its reorganization algorithm, which results in assigning the Sensor Finder Agent (SFA) to play role R2 in order to achieve G2 and the removal of the assignment (QA,R1,G1) from the assignment set Φ. The organizational state transitions after each event are shown below. (As described above, there is actually only one reorganization resulting in a transition from State 0 directly to State 2; however, to clearly illustrate the cause and effect of the various events, we show the reorganization as two separate transitions.)



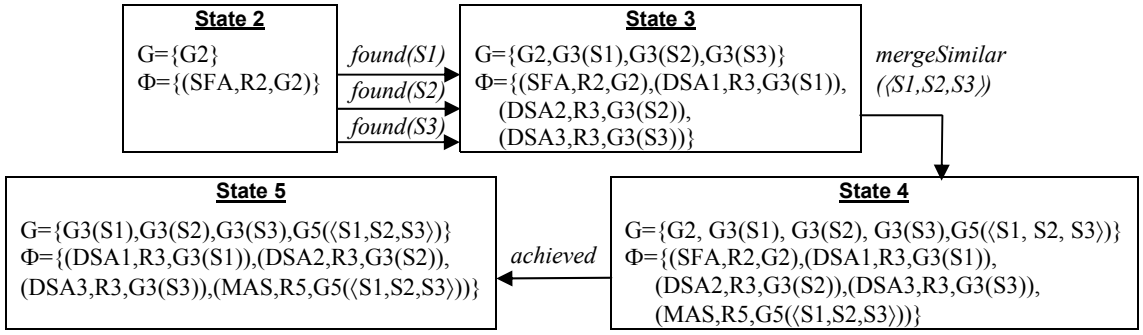
The Sensor Finder Agent (SFA) is responsible for selecting the appropriate sensors for each query. To perform this calculation, it extracts the desired coverage area A and a set of timing and accuracy constraints C from the query, which is a parameter of the goal it is trying to achieve (an instance of goal G2). It also receives the current set of sensors S by querying the sensor database, which is maintained by the OM. The SFA follows the following algorithm to select an *optimal* set of sensors for a particular query.

1. Remove sensor from S that do not cover any part of area A
2. Remove sensors from S that are not capable of meeting constraints C
3. Minimize S by removing sensors whose coverage area is redundant
4. For each sensor in S, generate a found event parameterized by the sensor
5. If all sensors in S are the same type, generate a *mergeSimilar* event parameterized with S
   Otherwise, generate a *mergeDiverse* event parameterized with S

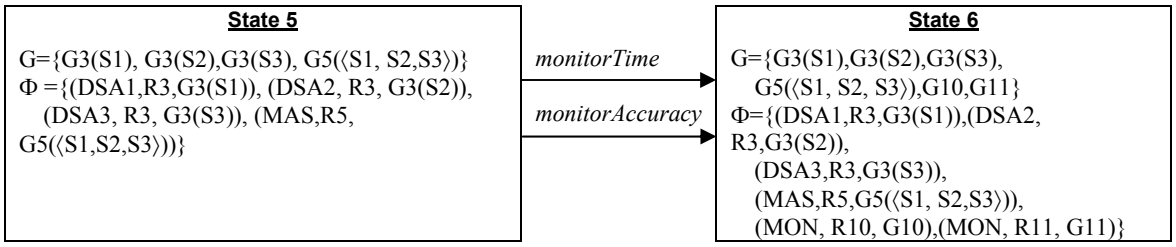
Using its application specific algorithm, the SFA chooses sensors S1, S2, S3 as optimal sensors for the current query, which results in the following events being generated: *found(S1)*, *found(S2)*, *found(S3)*, and *mergeSimilar((S1,S2,S3))*. (Note that the SFA could have also chosen sensors S1, S3 and S5, which is an equally capable set of sensors.) While OMACS does not define where and

[2] While the manipulation of goals is important to this example, it is not part of the OMACS model. For more information on our Goal Model for Adaptive Systems (GMoDS) and reasoning over those goals, see [13].

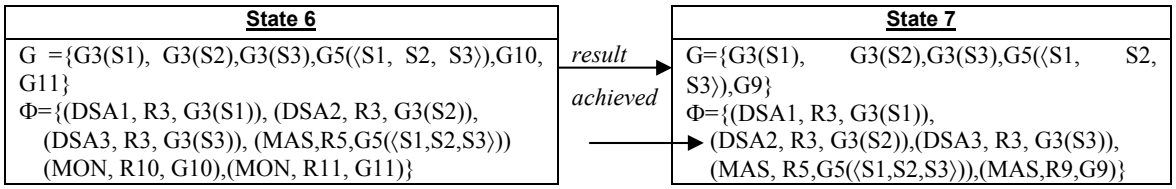
how events are recognized or generated, OMACS roles do effectively define an interface between the application specific algorithms and the events they are expected to generate. In this case, each *found* event causes the OM to instantiate a new G3 goal parameterized with the event parameter (S1, S2, or S3). The three goals, G3(S1), G3(S2), and G3(S3), are inserted into G, which once again requires OM to reorganize and results in the assignments (DSA1,R3,G3(S1)), (DSA2,R3,G3(S2)), and (DSA3,R3,G3(S3)) being inserted into Φ. The *mergeSimilar* event causes the instantiation of goal G5(⟨S1,S2,S3⟩) and its insertion into G, which also triggers a reorganization. In this case, the reorganization algorithm assigns the Merger Agent Similar (MAS) to play role R5 to achieve goal G5(⟨S1,S2,S3⟩). After triggering these events, the SFA informs the OM that goal G2 has been *achieved*, which results in a reorganization for the removal of the goal G2 from G and (SFA,R2,G2) from Φ. These events and their resulting reorganizations are shown below. (Again, it is possible to incorporate all the events described in the previous paragraph into a single reorganization, thus making the system much more efficient.)



Depending on the query, data fusion may be performed after coordination between the MAS and the DSAs in charge of a query; however, for this query, no filtering, correlation, or the addition of data from a database is necessary. Before actually gathering data, the MAS checks the time and accuracy constraints contained in the query, which it accomplishes by triggering *monitorTime* and *monitorAccuracy* events. These events result in the insertion of G10 and G11 in G. During the ensuing reorganization, roles R10 and R11 are selected to achieve G10 and G11 respectively. However, both of these roles can be played by the Monitor Agent (MON) as it has the required capabilities for both roles. Thus, OM assigns MON to play both R10 and R11 to achieve G10 and G11. The transition for these events is shown below.



If neither of the constraints is violated, the MON sends a message to the MAS notifying it that it can proceed, which causes a *result* event. The *result* event causes the instantiation of goal G9, which is inserted into G. The MON tells the OM it has *achieved* goals G10 and G11, which causes the removal of the goals G10 and G11 from G. Due to the changes in G, reorganization by the OM is now necessary, which results in the removal of the assignments (MON, R10, G10) and (MON, R11, G11) from Φ. As the MAS has the capability to interact with the GUI, it is assigned by the OM to play role R9 to achieve goal G9. The result of these events and the reorganization are shown below.



At this point, the MAS sends the result of the query to the GUI. Once the results of the query have been sent, the MAS tells the OM it has *achieved* goal G9, which results in the removal of G9 from G. Because the query is persistent, goals G5, G3(S1), G3(S2), and G3(S3) are not yet achieved and thus remain in G. During the resulting reorganization, the assignment (MAS,R9,G9) is removed from Φ. When an update is required for the query, the MAS will coordinate with the DSA agents to get new data and cycle back to State 5 above. Thus, the final transition for this phase of the persistent query is shown below.

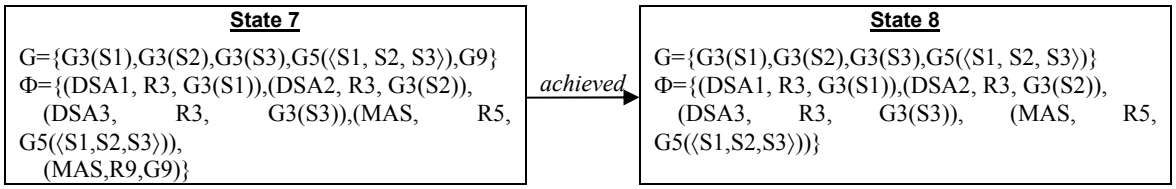


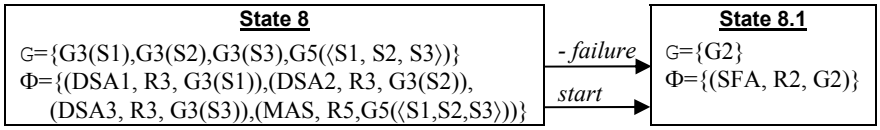
Fig. 9 shows the results obtain from the GUI. The answer for the query covers 100% of the area of interest. The system effectively detected all three targets in the selected area.

- Tank at 29,40
- Truck at 20,40
- Launcher at 36,47

Thus, by designing the BIS using the OMACS model, we were able to implement an organizational reasoning capable of choosing the best assignments to produce an optimal organization that provides the expected results.

6.3.2 Sensor Failure

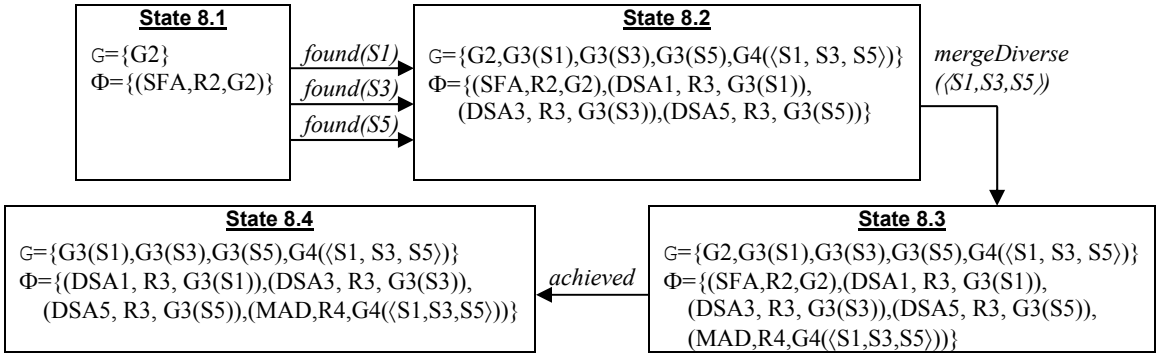
The BIS simulator allows us to fail specific sensors. If we make S2 fail, the attached agent, DSA2, is unable to achieve goal G3(S2). As DSA2 can no longer gather the data, the MAS, which was coordinating with the DSA2, must interrupt its task and generates a negative trigger *failure*. This negative trigger causes all the goals related to that query to be removed from G, resulting in the cancellation of all their current assignments. Thus, goals G3(S1), G3(S2), G3(S3), G5(⟨S1, S2, S3⟩) are all removed from G. The negative trigger *failure* is immediately followed by a *start* event generated by the DSA. The *start* event is parameterized with the initial query and causes the instantiation and insertion of goal G2 in G. Attempting to achieve this new instance of G2 causes the organization to reselect appropriate coordinating agents for the query. The organization is treating the query that it failed to answer due to a loss of sensor as a new query. The BIS then chooses appropriate agents to overcome this loss in order to provide the best results. The state transitions after a sensor failure are shown below.



Taking into account the loss of capability of the DSA for S2, the SFA selects sensors S1, S3, S5 as the new optimal set of sensor for the query. It then triggers the following events: *found(S1)*,

found(S3), *found(S5)*, *mergeDiverse(⟨S1, S3, S5⟩)*. Each *found* event triggers a parameterized goal G3 having the parameter of the trigger. In our case, goals G3(S1), G3(S3), and G3(S5) are triggered.

As the sensors given in parameter for the event *mergeDiverse* are different sensors (S1, S3 are ground sensors whereas S5 is an ATR sensor), this event results in the insertion of the parameterized goal G4(⟨S1, S3, S5⟩) into G. To satisfy the new goal G4, the system chooses role R4, which is played by the Merger Agent Diverse (MAD). When all the events have been triggered, the SFA sends an *achieved* message to the OM. This message results in the removal of the goal G2 from G. During the ensuing reorganization, the (SFA,R2,G2) assignment is removed from Φ and the assignments (DSA1, R3, G3(S1)),(DSA3, R3, G3(S3)), (DSA5, R3, G3(S5)),(MAD,R4,G4(⟨S1,S3,S5⟩)) are added to Φ. The corresponding states of the organization are described below.



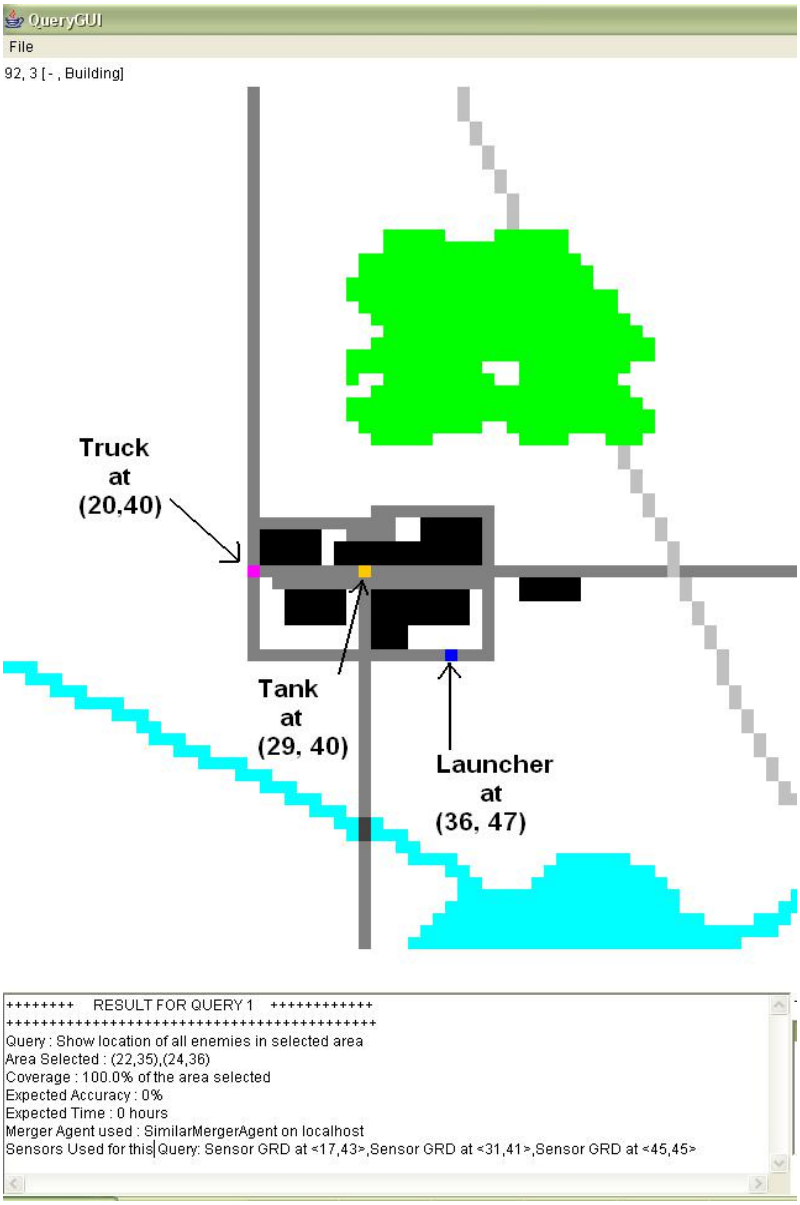


Fig. 9. Results from the GUI for a Normal Execution

The execution then continues as described in the normal execution where State 8.4 would be equivalent to State 5 (see Section 6.3.1). The BIS detects the following enemies.

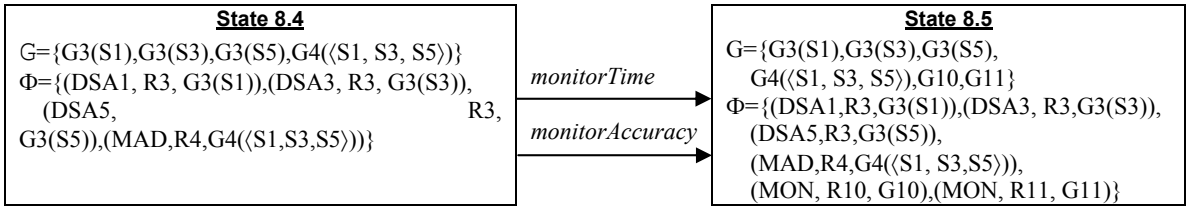
- Tank at 29,40
- Truck at 20,40
- Launcher at 36,47

Therefore, after the loss of S2, DSA2 has been replaced by DSA5 and the BIS organization decided to use the MAD for the merging instead of the MAS in order to insure a better performance. Even though a loss of a sensor used to provide information for the query has occurred, the system was able to reorganize accordingly and maintain the flow of information without the intervention of the user.

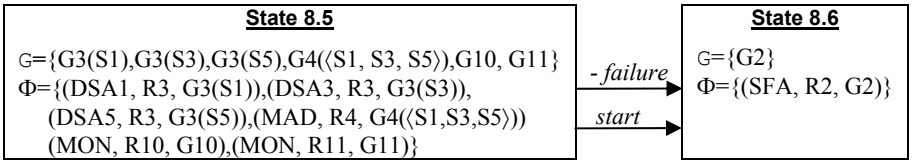
6.3.3 Maintenance Goal Failure

The user interface of the BIS allows the commander to stipulate constraints for the query in terms of desired timeliness or accuracy. For the remainder of our scenario, we assume that the commander has updated the query specifying that the system provide query results within eight minutes.

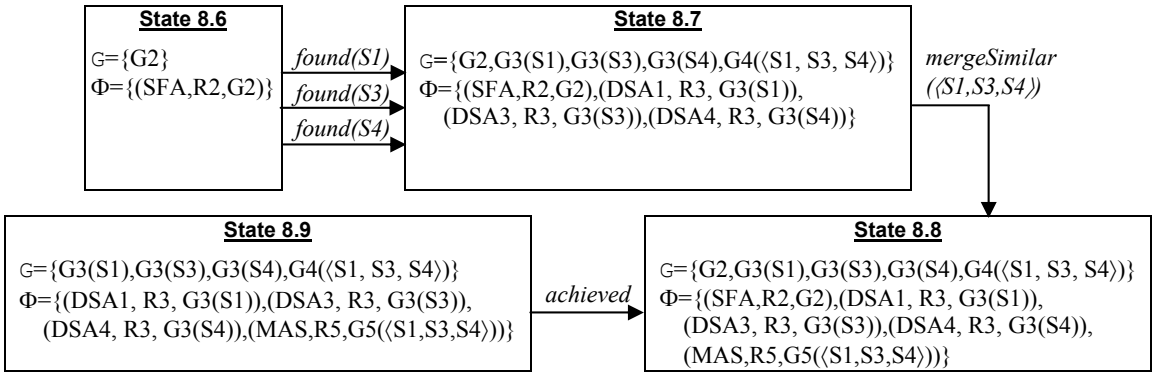
We continue our example with the system currently running the query using DSA1, DSA3, DSA5 and MAD as described above. After the data from the battlefield is refreshed, the MAD triggers *monitorTime* and *monitorAccuracy* event in order to check the query against the time and accuracy constraints that have been updated. These events result in the insertion of G10 and G11 in G. After reorganization, the OM assigns MON agent to play both R10 and R11 to achieve G10 and G11. The transition for these events is shown below



Once the data is sent to the Monitor Agent for checking the constraints as described above, the MON generates a negative trigger *failure* because the query, as executed, does not meet the 8 minutes constraint. In fact, S5, which is an ATR sensor, can only provide data within 15 minutes. Therefore, the maintenance goal G10 fails. The negative trigger causes all the goals related to that query to be removed from G, resulting in the cancellation of all related assignments. Thus, goals G3(S1), G3(S3), G3(S5), G4(⟨S1, S3, S5⟩), G10, and G11 are all removed from G. The negative trigger *failure* is immediately followed by a *start* event generated by the Monitor Agent (MON). This event is parameterized with the initial query and causes the insertion of goal G2 in G. The system then treats the query that failed as a new query and tries to choose the appropriate organization in order to provide result to the query meeting the time constraint defined above. The state transition after the constraint violation is described below.



Taking into account the time constraint for the query, the SFA selects sensors S1, S3, S4 as the new set of sensor for the query because they offer the best coverage of the area of interest while providing data within 5 minutes (due to the fact that they are ground sensors). It then triggers the following events: *found(S1)*, *found(S3)*, *found(S4)*, *mergeSimilar(⟨S1, S3, S4⟩)*. Each *found* event triggers a parameterized goal G3 having the parameter of the trigger. In our case, goal G3(S1), G3(S3), and G3(S4) are triggered. As the sensors given in parameter for the event *mergeSimilar* are all ground sensors, this event results in the insertion of the parameterized goal G5(⟨S1, S3, S4⟩). To satisfy this new goal, the system chooses role R5 which is played by the Merger Agent Similar (MAS). When all the events have been triggered, the SFA notifies the OM that it has *achieved* goal G2, which results in the removal of G2 from G. The corresponding assignment is also removed from the list of current assignments. The corresponding state of the organization is as follow.



The execution then continues as described in the normal execution where State 8.5 would be equivalent to State 5 (Section 5.1). In this case, the BIS detects only the following enemies.

- Truck at 20,40
- Launcher at 36,47

Therefore, due to the new query constraints, the BIS automatically reorganized and replaced DS5 with DS4 to insure the effectiveness of the query with regards to the time constraint. The BIS organization also replaced the MAD by the MAS, which yields a better performance in merging data coming from the new set of sensors. In this scenario, we can see how the BIS has been able to reorganize in order to satisfy a maintenance goal in the system. However, this reorganization process has resulted in a lost of coverage as the Tank located at (29,40) cannot no longer be detected.

6.3.4 Execution Summary

Fig. 10 and Fig. 11 summarize how the BIS organization adapted to overcome sensor failure and to satisfy the time constraint imposed by the commander. The BIS was able to switch its information sources from the set ⟨S1, S2, S3⟩ in Fig. 10a, to ⟨S1, S3, S5⟩ in Fig. 10b, and finally to ⟨S1, S3, S4⟩ in Fig. 11. The system was also able to change its fusing algorithms by assigning agents to play one of the two merging roles available in the organization (R4 and R5).

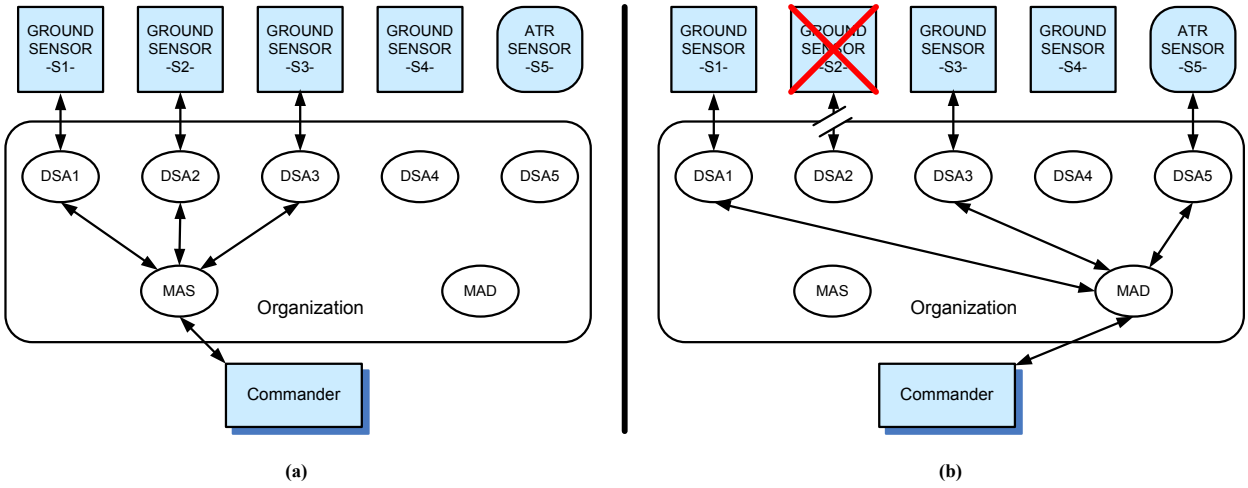


Fig. 10. Normal Execution vs. Execution with Sensor Failure

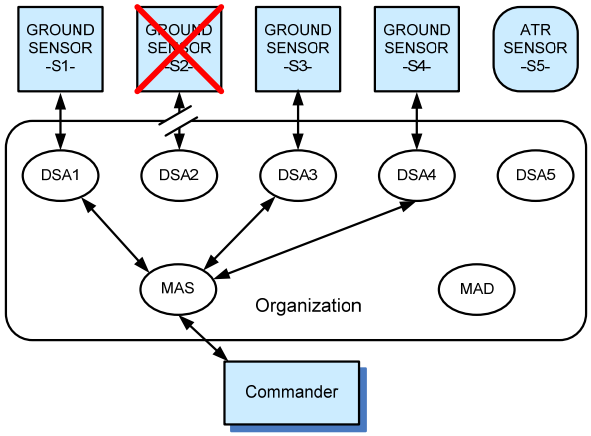


Fig. 11. Execution with Maintenance Goal Failure

6.3.5 Impact of OMACS

While the BIS system described in this section could have been developed using other techniques, the use of the OMACS model provides many advantages over more ad hoc approaches. First, OMACS defines the necessary components that developers must provide in order to have an adaptive system. After defining the system's goals and roles (and their required capabilities), it was fairly straightforward to determine what types of agents were needed and what their required capabilities were. In this case, we used our generic reorganization algorithm along with default oaf and rcf functions to implement the organizational reasoning. While the instantiation and removal of goals (which is not part of OMACS) is important to determining how the environment and problem solving process has changed, it is the ability provided by OMACS to reassign agents based on their current capabilities in response to the changing system goals and problem solving process that allows the system to adapt in ways that may not have been considered at design time. The result of using OMACS on this example is a flexible system that is able to adapt to a variety of changes in the environment or agent capabilities. This adaptivity was achieved without the designer having to consider all the possible ways the system could fail or the appropriate corrective actions.

7. Related Work

Computational organization theory uses mathematical and computational techniques to study both human and artificial organizations [6, 7]. 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" [6], and that the best organizational designs are highly application and situation dependent [5]. It also provides findings about the conditions under which certain organizations work best. For instance, as the number of hierarchical levels in an organization increases, efficiency and effectiveness tends to decrease while decentralized organizations tend to have higher performance. However, hierarchical organizations tend to exhibit higher reliability [6]. These insights seem to suggest that allowing systems to determine their organization at runtime, as we propose, could have positive effects on system performance. On the other hand, too much flexibility can lead to chaotic behavior that is detrimental to system performance [38]. How to permit change while not allowing it to inhibit system performance is a property of the *reorganization algorithm* used [48].

Within the last few years, the notion of separating the agents populating a multiagent system from the system organization [45, 46] 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. These policies include constraints on agent behavior and their interactions. There are separate responsibilities for agents and organizations; the organization, not the agents, should be responsible for setting and enforcing the policies. While these advances are recent, there have been some discussions on how to incorporate them into existing multiagent systems methodologies. For instance, the Gaia multiagent systems methodology has been modified to incorporate the notion of social laws [47]. Other approaches view the organization as a separate *institutional agent* [43]. However, these proposals are not detailed enough to provide guidance on how to use these organizational concepts, leaving designers to translate high-level organizational concepts such as laws or policies into a multiagent design and implementation. The OMACS model provides a foundation upon which a complete organization-based methodology can be constructed. An OMACS-based methodology would provide concrete definitions and relations for organizational entities and could provide a direct mapping onto implementation structures and algorithms.

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 [8, 9], Joint Intentions [23, 24, 25], Shared Plans [19] and Planned Team Activity [28], have been proposed and even implemented [39], 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, which limits the applicability. As shown in our example, only one agent is actually required to understand the entire organizational structure in OMACS although more sophisticated distributed reasoning may be used.

Other closely related work includes the CoDA project at the University of Maine [40]. The CoDA project deals with a team of autonomous underwater vehicles that must self-organize and reorganize using a two level strategy where a meta-level organization designs a task-level organization to carry out system goals. While the CoDA notion of an organization includes agents who plays roles and has an explicit two-layer hierarchy, it is much more limited in its application as it does not include other organizational concepts such as policies and capabilities. In fact, the CoDA organizational model could be considered to contain a subset of the OMACS model and could be implemented using OMACS.

While there have been several organization models proposed over the last few years, 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 [17] and extended in the AGR model [18].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 to provide some behavioral definition of roles [34]. The MOISE+ model greatly extended the notion of an organization model by including three aspects: structural, functional, and deontic [22]. 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. The Organizational Design Modeling Language by Horling and Lesser [21] uses a basic underlying model of organizations in order to perform performance

prediction of the multiagent organization. A more detailed overview of existing organization models is given in [10].

One of the most complete organization models is the Organizational Model for Normative Institutions (OMNI) [16], 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 [15] and HarmonIA [42]. The OMNI framework consists of a Normative Dimension, an Organizational Dimension, and an Ontological Dimension, each of which has an Abstract, Concrete, and Implementation Level. The Abstract Level defines the main objectives of the organization. The Concrete Level refines the definitions of the Abstract Level further by defining the norms and rules of the organization, the roles in the organization, landmarks, and concrete ontological concepts. And finally, the Implementation Level implements the definitions from the Concrete Level. Each of these organization models focus on open systems where cooperation is not necessarily required. In OMACS, once an assignment is made, the organization can be sure all agents will attempt to carry out those assignments and will notify the organization of any events of interest.

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 [26]. ROADMAP defines a nice clean metamodel that includes the basic modeling concepts of roles, protocols, services, agents, knowledge, and the environment. Likewise, the MaSE metamodel was defined in part based on the implementation of agentTool, a tool that supports the MaSE modeling process [11]. 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 [1]. 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 [20]. OMACS provides the foundation for organization-based multiagent metamodel in which the analysis and design concepts are directly related to run-time concepts.

8. Conclusions and Discussion

The OMACS model is unique in that it is focuses on the use of agent capabilities, which may change over time, in order to determine which agents may play the various roles required to achieve the current organizational roles. The efficacy of the model was demonstrated by its application in the Battlefield Information System example given in Section 6. It is also a comprehensive, yet flexible model. While providing a framework for developing adaptive systems, OMACS allows designers to choose how to implement the goals, roles, and agents as well as the organizational reasoning required for determining the current goals, the assignment of agents to roles and goals, and the effect of policies on the organization.

The types of applications that can benefit from the OMACS model are those in which there is a desire for some level of global (organizational) control, but one in which the agents may exhibit a limited form of autonomy. Specifically, the agents must accept the assignment of the goals they should try to achieve and the roles they must attempt to play in order to achieve those goals. However, the details of how an agent plays an assigned role are left up to the agent. In order for the organization to function correctly, an agent is obliged to correctly report the scores for the

capabilities it possesses and the events of interest that occur during the pursuit of its goals. Thus, OMACS defines an interface between the organization and the individual agents that allows the design and implementation of the organization and the agents to be separate and possibly completed by separate developers.

There are several specific contributions of OMACS. First, OMACS defines a metamodel for discussing the sufficient components of adaptive multiagent systems. While there are many models of multiagent systems, OMACS is unique in its focus on the centrality agent capabilities to determining appropriate system configurations. We have used the OMACS metamodels in a number of application areas including multiagent systems, information systems, sensor networks, and cooperative robotics. Second, OMACS provides a level of global control while allowing local autonomous behavior. In many approaches, it is unclear as to how the global goals of the organization are achieved. While the organization selects the goals and roles that an agent must play, the agent is free to carry out those roles as it deems appropriate. Third, OMACS provides a framework for quantifying the effects of the relationships between goals, roles, agents and capabilities. While the actual values and computational formulae used are not prescribed, OMACS states specifically what values and formulae must be defined. Fourth, OMACS provides default computational approach for calculating appropriate configurations while allowing flexibility for application-specific calculations. If we assume all capabilities are equally important to a role (thus using the default *rcf* given in Equation 3) and that an agent either possesses a capability or not (possesses(a,c) is either 1 or 0), we can use the default *oaf* function given in Equation 5 to compute the best configuration at a particular point in time; this is the approach used in the Battlefield Information System example in Section 6. Fifth, OMACS offers a global mechanism for restricting system configurations based on application specific constraints. These constraints may be encoded as policies that can limit possible configurations thus restricting undesirable behavior.

Finally, OMACS builds a foundation upon which a complete software development approach can be created. Due to the flexibility of the OMACS model, a designer will need assistance in determining an appropriate set of goals, roles, agents, capabilities, events, and policies for a proposed application. This will require a methodology to help guide developers as well as a toolset that supports model development, metrics to help quantify design tradeoffs, code generation, and system testing. Instead of defining a "one size fits all" methodology for developing adaptive multiagent systems, the goal is to use *method engineering* [4], which allows developers to create their own methodologies and processes from existing method fragments. However, simply taking method fragments from existing methodologies is naïve and problematic [20]. Although many methodologies use similar terms to describe their approaches, these terms have different semantics, which leads to incompatible fragment use. A better approach, as proposed in [2], is to adopt a standard metamodel upon which all method fragments are defined. We believe that OMACS provides a solid initial metamodel for defining a set of software engineering processes and tools that will actually make our organization-based framework useful in real world applications. The model itself grew out of work related to the development of the Multiagent Systems Engineering (MaSE) methodology [12] and its associated toolset, agentTool [11], for analysis, design, verification, and generation of multiagent systems. While MaSE already captures much of the required knowledge (e.g., goals, roles, agents, and organizational policies), we are extending MaSE to capture additional framework components such as capabilities and relationship scores [14]. This extension of MaSE, called Organization-based MaSE (O-MaSE), is also being defined in terms of method fragments using OMACS as the underlying metamodel.

8.1 Future Work

The OMACS metamodel lays the foundation upon which the rest of our framework will be developed. We are pursuing three veins of research based on the model established in this paper. First, we are formalizing a Goal Model for Dynamic Systems (GMoDS) [13]. In GMoDS, there are two main representations of system goals: the goal specification model, G_{Spec}, and a goal instance model, $G_{Instance}$. The goal specification model is a static representation of system goals that allows the specification of precedence constraints between goals as well as the instantiation of new goals based on events. The goal instance model is a dynamic model used at runtime to define the actual goals generated during system operation.

The second area of research based on this model is to develop a practical set of agent-based architectures and algorithms that make use of OMACS. These architectures and algorithms must address questions such as how best to update the knowledge about the organization, how to recognize reorganization triggers, and how best to reorganize once a reorganization has been triggered. Other questions related to the operation of organizations include how to move agents between higher-level and lower-level organizations as part of the reorganization process, how to split an organization into multiple organizations, or how to merge existing organizations. Finally, we plan to look at structural reorganization, which will most likely introduce the notion of human intervention into the organization. Specifically, we plan to look at human control over the organization structure as well as the assignment of goals to the goal set and agents to roles. We also want to look into goal relaxation, which can be done either by a human or in an automated fashion if the appropriate goal structure is provided in advance.

As discussed in the introduction, the goal of OMACS is to design organizations that can reorganize in ways that its designer would not necessarily be able imagine at design time. However, as OMACS policies can restrict the ways in which the organization may reorganize, it is possible that a set of policies could limit possible system configurations even more than the designer's imagination. Therefore, it is highly desirable that the designer have a set of design-time tools that can help determine the impact of such policies. We are currently developing a set of OMACS design metrics that can help a designer to ensure that policies are not overly restrictive. Our initial set of metrics use model checking techniques to measure the flexibility of organization designs based on the goals, roles, and agents in a system [36]. These metrics allow designers to make design-time tradeoffs between flexibility and computational costs. We are currently extending this initial set of metrics to include the effects of policies.

The current version of OMACS only allows for artificial agents; it does not currently capture humans and their associated capabilities when developing an organization. Thus, we are investigating extending OMACS to incorporate human agents along side artificial agents. We plan to incorporate humans into OMACS based on the roles that humans can play when interacting with artificial agents using standard role types such as supervisor, operator and peer. The incorporation of humans in these roles requires the ability to represent the human capabilities (e.g. expertise) and human performance considerations (e.g. fatigue). As the current OMACS capabilities model is simplistic and does not explicitly model multiple dimensions of capability degradation or enhancement, we are investigating ways to explicitly model multi-dimensional capabilities including parameterized capabilities, compositional capabilities, and capability generalization and specialization.

We are also extending the Multiagent Systems Engineering (MaSE) methodology to allow designers to design a multiagent organization based on the OMACS model. This extended version of MaSE is called Organization-based MaSE (O-MaSE). A preliminary proposal for the O-MaSE methodology is described in [14]. Our goal is to extend MaSE to capture the organizational

concepts identified in OMACS. New concepts include AND/OR refinement of goals, integration of capabilities and the ability to model sub-organizations. 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. Future plans include code generation for various platforms as well as integration with the Bogor model checking tool [35] to provide model validation and performance prediction metrics.

9. Acknowledgements

This research was performed as part of grants provided by the Air Force Office of Scientific Research grant numbers F49620-02-1-0427 and FA9550-06-1-0058 and the National Science Foundation grant number IIS-0347545.

References

1. C. Bernon, M. Cossentino, M. Gleizes, P. Turci, F. Zambonelli, "A Study of Some Multi-Agent Meta-Models," in Agent-Oriented Software Engineering V: 5th Intl. Workshop (AOSE 2004), J. Odell, P. Giorgini, Müller, J. (eds.). LNCS 3382, Springer: Berlin, 2005.
2. G. Beydoun, G. Low, C. Gonzalez-Perez, B. Henderson-Sellers, "Synthesis of a Generic MAS Metamodel," in Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications Series, A. Garcia, et. al. (eds.). LNCS 3914, Springer: Berlin, 2006.
3. P.M. Blau, W.R. Scott, Formal Organizations, Chandler: San Francisco, CA, 1962.
4. S. Brinkkemper, "Method Engineering: Engineering Of Information Systems Development Methods and Tools," Journal of Information and Software Technology, Vol. 38(4), pp. 275-280, 1996.
5. K.M. Carley, "Computational and Mathematical Organization Theory: Perspective and Directions," Computational and Mathematical Organization Theory, Vol. 1(1), pp. 39-56, 1995.
6. K.M. Carley, "Organizational Adaptation," Annals of Operations Research, Vol. 75, pp. 25-47, 1998.
7. K.M. Carley, L. Gasser, "Computational Organization Theory," in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, G. Weiss (ed.). MIT Press: Cambridge, MA, 1999.
8. P. R. Cohen, H. J. Levesque, "Teamwork," Nous, 25(4), pp. 487-512, 1991.
9. P. R. Cohen, H. J. Levesque, "Intention is Choice with Commitment," Artificial Intelligence, Vol. 42(3), pp. 213-261, 1990.
10. L. Coutinho, J. Sichman, O. Boissier, "Modeling Organization in MAS: A Comparison Of Models," in Proc. of the 1st. Workshop on Software Engineering for Agent-Oriented Systems (SEAS'05) Uberlândia, Brazil, October 3, 2005.
11. S.A. DeLoach, "Analysis and Design using MaSE and agentTool," Proc. of the 12th Midwest Artificial Intelligence and Cognitive Science Conf. (MAICS 2001). Oxford, Ohio, March, 2001.
12. S.A. DeLoach, M.F. Wood, C. H. Sparkman, "Multiagent Systems Engineering," The Intl. Journal of Software Engineering and Knowledge Engineering, Vol. 11(3), pp. 231-258, June 2001.
13. S.A. DeLoach, W.H. Oyenan, "An Organizational Model and Dynamic Goal Model for Autonomous, Adaptive Systems," Multiagent & Cooperative Robotics Laboratory Technical Report No. MACR-TR-2006-01. Kansas State University. March 13, 2006.

14. S.A. DeLoach, "Multiagent Systems Engineering of Organization-based Multiagent Systems," in Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications Series, A. Garcia, et. al. (eds.), LNCS 3914, Springer: Berlin, 2006.
15. V. Dignum, "A Model for Organizational Interaction: Based on Agents, Founded in Logic," PhD thesis, Utrecht University, 2004.
16. V. Dignum, J. V´azquez-Salceda, F. Dignum, "Omni: Introducing Social Structure, Norms and Ontologies into Agent Organizations," in Programming Multi-Agent Systems: Second Intl. Workshop (ProMAS 2004), LNCS 3346, pp. 181–198, Springer: Berlin, 2004.
17. J. Ferber, O. Gutknecht, "A Meta-model for the Analysis and Design of Organizations in Multi-agent Systems," in Proc. of 3rd Intl. Conf. on MultiAgent Systems (ICMAS'98), pp. 128-135, 1998.
18. J. Ferber, O. Gutknecht, F. Michel, "From Agents to Organizations: an Organizational View of Multi-agent Systems," in Agent-Oriented Software Engineering IV: 4th Intl. Workshop (AOSE 2003), P. Giorgini, J.P. Muller, J. Odell (eds.), LNCS 2935, pp. 214-230, Springer: Berlin, 2003.
19. B.J. Grosz, S. Kraus, "Collaborative Plans for Complex Group Action," Artificial Intelligence, Vol. 86(2), pp. 269-357, 1996.
20. B. Henderson-Sellers, "Evaluating the Feasibility of Method Engineering for the Creation of Agent-Oriented Methodologies," in Multi-Agent Systems and Applications IV: 4th Intl. Central and Eastern European Conf. on Multi-agent Systems, M. Pechoucek, P. Petta, L.Varga (eds.), pp. 142-152, LNCS 3690, Springer: Berlin, 2005.
21. B. Horling, V. Lesser, "Using ODML to Model Multi-Agent Organizations," in Proc. of the IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology, pp. 72-80, 2005.
22. J. Hübner, J. Sichman, O. Boissier, "MOISE+: Towards a Structural, Functional and Deontic Model for MAS Organization," in Proc. of the 1st Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'02), pp. 501-502, 2002.
23. N.R. Jennings, "Commitments and Conventions: The Foundation of Coordination in Multiagent Systems," Knowledge Engineering Review, Vol. 8(3), pp. 223-250, 1993.
24. N.R. Jennings, "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions," Artificial Intelligence, Vol. 75(2), pp. 195-240, 1995.
25. N.R. Jennings, "Towards a Cooperation Knowledge Level for Collaborative Problem Solving," in Proc. of the 10th European Conf. on Artificial Intelligence, B. Neumann (ed.), pp. 224-228, Vienna, Austria, 1992.
26. T. Juan, L. Sterling, "The ROADMAP Meta-model for Intelligent Adaptive Multi-agent Systems in Open Environments," LNCS 2935, pp. 53-68, Springer: Berlin, 2004.
27. S. Kashyap, "Reorganization in Multiagent Systems," MS Thesis, Kansas State University, 2006 .
28. D. Kinny, M. Ljungberg, A.S. Rao, E. Sonenberg, G. Tidhar, E. Werner, "Planned Team Activity," in Artificial Social Systems - Selected Papers from the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-92), C. Castelfranchi, E. Werner (eds.), pp. 226-256. LNAI 830, Springer: Berlin, 1992.
29. E. Matson, S.A. DeLoach, "An Organization-Based Adaptive Information System for Battlefield Situational Analysis," Proc. of the Intl. Conf. on Integration of Knowledge Intensive Multi-Agent Systems: KIMAS'03: Modeling, Exploration, and Engineering, Boston, MA., Sep 30-Oct 3, 2003.
30. E. Matson, S.A. DeLoach, "Integrating Robotic Sensor and Effector Capabilities with Multi-Agent Organizations," Proc. of the Intl. Conf. on Artificial Intelligence (IC-AI'04), Las Vegas, NV. 2004.

31. "MESSAGE: Methodology for Engineering Systems of Software Agents, Deliverable 1. Initial Methodology," EURESCOM Project P907-GI, July 2000.
32. R. Nair, M. Tambe, S. Marsella, "Team Formation for Reformation," in Proc. of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems, 2002.
33. J. Odell, M. Nodine, R. Levy, "A Metamodel for Agents, Roles, and Groups," in Agent-Oriented Software Engineering V: 5th Intl. Workshop (AOSE 2004), J. Odell, P. Giorgini, Müller, J. (eds.). LNCS 3382, Springer: Berlin, 2005.
34. Z. Peng, H. Heng, "An Improved Agent/Group/Role Meta-Model for Building Multi-Agent Systems," in Proc. of 2005 Intl. Conf. on Machine Learning and Cybernetics, pp. 287-292, 2005.
35. Robby, M.B. Dwyer, J. Hatcliff, "Bogor: An Extensible and Highly-Modular Model Checking Framework," in Proc. of the 4th Joint Meeting of the European Software Engineering Conf. and ACM SIGSOFT Symp. on the Foundations of SW Engineering (ESEC/FSE 2003), pp. 267 – 276, 2003.
36. Robby, S.A. DeLoach, V.A. Kolesnikov. "Using Design Metrics for Predicting System Flexibility," in Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE 2006), L. Baresi, R. Heckel (eds.), LNCS 3922, pp. 184-198, 2006.
37. S. Russell, P. Norvig, Artificial Intelligence a Modern Approach, Pearson Education, 2003.
38. K. Sycara, "Multiagent Systems," AI Magazine, 19 (2), 1998.
39. M. Tambe, "Towards flexible teamwork," Journal of AI Research, Vol. 7, pp. 83-124, 1997.
40. R.M. Turner, E.H. Turner, "A Two-Level, Protocol-Based Approach to Controlling Autonomous Oceanographic Sampling Networks," IEEE Journal. of Oceanic Engineering, Vol. 26(4), pp. 654-666, 2001.
41. A. van Lamsweerde, R. Darimont, E. Letier, "Managing conflicts in goal-driven requirements engineering," IEEE Transactions on Software Engineering, Vol. 24(11), pp. 908-926, 1998.
42. J. Vazquez-Salceda, F. Dignum, "Modelling Electronic Organizations," in Multi-agent Systems and Applications III, V. Marik, J. Muller, M. Pechoucek (eds.), LNAI 2691, pp. 584–593, Springer: Berlin, 2003.
43. G. Wagner, "Agent-oriented analysis and design of organisational information systems." in Databases and information Systems, J. Barzdins & A. Caplinskas (eds.), pp. 111-124, Kluwer Academic Publishers, Norwell, MA, 2001.
44. M. Wooldridge, N.R. Jennings, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," Journal of Autonomous Agents and Multi-Agent Systems," Vol. 3(3), pp. 285-312, 2000.
45. F. Zambonelli, N.R. Jennings, M. Wooldridge, "Organisational Abstractions for the Analysis and design of Multi-agent Systems," in Agent-Oriented Software Engineering-Proc. of the First Intl. Workshop on Agent-Oriented Software Engineering, P. Ciancarini, M. Wooldridge, (eds.), LNCS 1957, pp. 207-222, Springer: Berlin, 2001.
46. F. Zambonelli, N.R. Jennings, M.J. Wooldridge, "Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems," Intl. Journal of Software Engineering and Knowledge Engineering, Vol. 11(3), pp. 303-328, 2001.
47. F. Zambonelli, N.R. Jennings, A. Omicini, M.J. Wooldridge, "Agent-Oriented Software Engineering for Internet Applications," in Coordination of Internet Agents: Models, Technologies, and Applications, A. Omicini and F. Zambonelli and M. Klusch, R. Tolksdorf (eds.), pp. 326-346, Springer-Verlag: Berlin, 2001.
48. C. Zhong, "An Investigation of Reorganization Algorithms," MS Thesis, Kansas State University, 2006.

49. C. Zhong and S.A. DeLoach. “An Investigation of Reorganization Algorithms,” in Proceedings of the International Conference on Artificial Intelligence (IC-AI'2006), CSREA Press, 2006.