Runtime Models for Automatic Reorganization of Multi-Robot Systems

Christopher Zhong
Kansas State University
234 Nichols Hall
Manhattan, Kansas USA 66506
1 (785) 532-6350
czhong@k-state.edu

Scott A. DeLoach
Kansas State University
234 Nichols Hall
Manhattan, Kansas USA 66506
1 (785) 532-6350
sdeloach@k-state.edu

ABSTRACT

This paper presents a reusable framework for developing adaptive multi-robotic systems for heterogeneous robot teams using an organization-based approach. The framework is based on the Organizational Model for Adaptive Computational Systems (OMACS) and the Goal Model for Dynamic Systems (GMoDS). GMoDS is used to capture system-level goals that drive the system. OMACS is an abstract model used to capture the system configuration and allows the team to organize and reorganize without the need for explicit runtime reorganization rules. While OMACS provides an implicit reorganization capability, it also supports policies that can either guide or restrict the resulting organizations thus limiting unexpected or harmful adaptation. We demonstrate our framework by presenting the design and implementation of a multi-robot system for detecting improvised explosive devices. We then highlight the adaptability of the resulting system.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design

Keywords

Self-adaptive systems, runtime models, agent-oriented Software Engineering, Cooperative Robotics

1. INTRODUCTION

The software being produced today is at least an order of magnitude more complex than that being developed a decade ago. Businesses today are demanding applications that operate autonomously, adapt in response to dynamic environments, and interact with other distributed applications in order to provide wide-ranging solutions [17]. To respond appropriately in today's complex environments, software needs to be aware of what it is doing and why in order to take the appropriate steps to achieve its business objectives. There are several instances of these kinds of systems including information systems, service-oriented systems, wireless sensor networks, and multi-robot systems [24]. In each of these types of systems, one key element of adaptivity is the allocation of tasks to appropriate system elements, which has received much attention [3, 11, 13, 14, 23, 28]. However, an equally important aspect of adaptivity is understanding why those tasks need to be performed.

A central feature of these systems is that they are closely tied to their environment and adapt in response to changes in that environment. Such systems are termed *dynamically adaptive systems*[4]. Recently, it has been suggested that *requirements reflection*, or reasoning over system requirements available as runtime objects, is key to developing dynamically adaptive systems [1, 26]. Specifically, [26] proposes three key challenges to requirements reflection: a runtime representation of the requirements, synchronization between the requirements and the architecture, and dealing with uncertainty. During the last several years, we have developed an approach that deals explicitly with the first two of those challenges. We represent requirements as goals at both design time and runtime, and we adapt the system architecture based on the current set of active goals in the system. This paper specifically aims our approach toward cooperative robotics where the system is closely tied to its environment, distributed, and dynamically adaptive.

There are three paradigms generally used to develop multi-robot systems that must adapt autonomously to their environment: (1) bio-inspired, (2) social, and (3) knowledge-based [24]. Each tackles adaptivity using different approaches. Bio-inspired approaches typically assume homogeneous robots [3, 25], thus they are of little interest here. Social approaches are generally role-based [9, 28] or market-based [3, 13]. Role-based approaches use predefined roles to allocate tasks, whereas market-based approaches allow robots to bid on tasks. Knowledge-based approaches [11, 21, 23] explicitly model team member capabilities that are important to the task allocation process. Our approach combines aspects of both social and knowledge-based approaches by using runtime models to capture the system configuration.

In this paper, we propose a framework for multi-robot systems based on runtime models for understanding (1) *what* the system should be doing in terms of system goals and (2) *how* the system

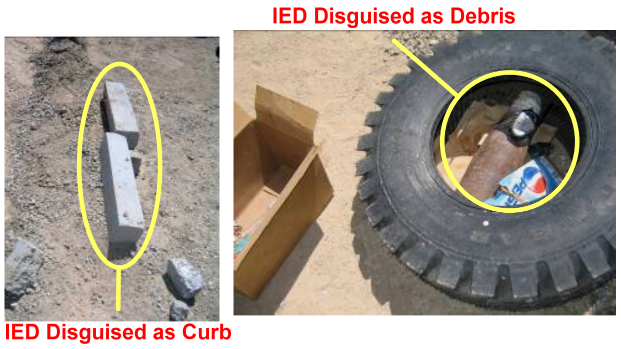


Figure 1. Improvised Explosive Devices (IEDs) [20]

is organized to achieve those goals. The system objectives are captured via the Goal Model for Dynamic Systems (GMoDS), which defines the main goals of the system and the relationships between those goals. The current configuration of the system is captured in the Organizational Model for Adaptive Computational Systems (OMACS). Both models are used in the Organization-Based Agent Architecture (OBAA) to help individual robots collaborate as part of a team in response to dynamic system goals and physical and computational capabilities. Finally, we demonstrate how the robot team uses this knowledge to allocate tasks efficiently and effectively. While we demonstrate our models and architecture on a team of robots, we claim that this set of models can be applied generally to a variety of complex distributed systems as shown in [8, 15].

The contributions of this paper are a real world demonstration of the implementation (via OBAA) of our complete framework of models that capture the goals (via GMoDS) of the system and uses those goals to drive the self-adaptation of the system in terms of the assignment of goals to individual robots (via OMACS).

1.1 Demonstration Scenario

In military situations, routes used for convoys must be constantly monitored for safety, including improvised explosive devices (IEDs), which are easily disguised and hard to spot as shown in Figure 1. The United States military currently uses teleoperated iRobot PackBots [15] to safely identify and disarm IEDs.

To demonstrate our framework, we developed a heterogeneous robot team to monitor and detect IEDs. A human operator defines the overall area to be searched and the team divides the total area into subareas and assigns individual robots to monitor those subareas. In our scenario, the team is assigned to monitor the intersection of two roads on a continuing basis. When suspicious objects are found, the team assigns a robot to investigate. If the investigating robot cannot identify the suspicious object, the team asks the human operator to help identify the object. Once an IED is identified, a robot capable of defusing or disposing of IEDs is assigned to dispose of it.

The remainder of the paper is organized as follows. Section 2 provides an overview of the OMACS and GMoDS models that are central to our approach. Section 3 presents the design of the IED detection system, while Section 4 discusses its implementation using the OBAA architecture. Section 5 evaluates the system's ability to use these models to adapt to changes. Finally, Section 6 provides a general discussion of our approach including current and future work, while Section 7 discussed related work.

2. MODELS

To help systems be proactive as well as reactive to changes in their system objectives and environment, we developed a set of models that allows *these systems to design their own organization at runtime*. In essence, we provide the system with organizational knowledge and let the system design its own organization based on the system goals and current capabilities. While the designer can provide guidance, supplying the system with key organizational information allows it to redesign, or reorganize, itself to match its current situation. This section presents the two key elements of this framework, a model for multiagent organizations called the Organization Model for Adaptive Computational Systems (OMACS) and the Goal Model for Dynamic Systems (GMoDS). OMACS defines the knowledge of a system's organizational structure and capabilities while GMoDS captures the system's overall objectives and defines what type of new objectives can be created in response to events that occur at runtime.

2.1 Organizational Model for Adaptive Computational Systems

The Organization Model for Adaptive Computational Systems (OMACS) [5] is a model that defines the knowledge required to allow a team of agents to reorganize in response to agent failure or changing team goals. As shown in Figure 2, an OMACS organization consists of a set of goals (G), roles (R), agents (A), capabilities (C), and policies (P). This information is used to compute the current set of assignments (Φ) that tell which agents have been assigned to play which roles in the organization in order to achieve organizational goals.

All organizations, even artificial organizations, are formed with some specific objective or goal in mind. This high-level organizational goal is decomposed into specific goals that are assigned to individual departments or persons within the organization. In OMACS, the overall goal of an organization is represented by a set of goals that the organization is trying to achieve. OMACS assumes that each goal in this set can be achieved by a single agent. The relationship between the various goals is not handled directly by OMACS but is entrusted to a goal model, which in our case is GMoDS (see Section 2.2).

Every OMACS organization has a set of heterogeneous agents, which are defined as "computational system instances that inhabit a complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals" [5]. The purpose of OMACS is to help determine the best assignment of agents to goals in order to achieve the overall objectives of the organization.

More specifically, OMACS goals are achieved by agents playing specific roles within the organization. In OMACS, roles are used to capture a set of responsibilities or the expected behavior of an agent playing that role. Generally, a role may be capable of achieving more than one type of goal and each type of goal can be achieved by more than one role. Thus, part of the assignment process includes determining the best role to achieve a specific goal. To support the assignment process, OMACS defines the *achieves* function, which takes as input a goal and a role, and returns a real value in the range [0..1] that reflects how well the

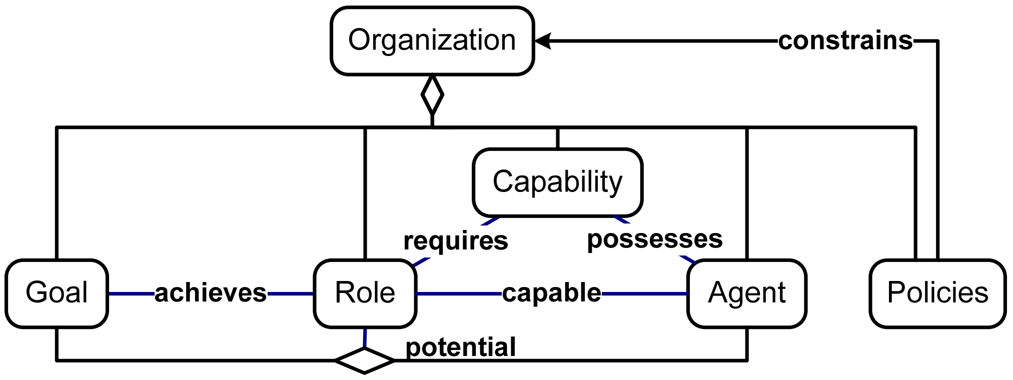


Figure 2. Organization Model

given role achieves the given goal type. A value of 0 means the given role is not able achieve the goal.

However, a role cannot be played by any given agent. Before an agent is allowed to play a given role, it must first meet the requirements of that role. Capabilities are essential in determining what roles agents are capable of playing. Capabilities are used to represent a wide variety of abilities, both soft and hard. Examples of soft abilities include access to resources, communications, and computational algorithms. Hard abilities typically model the abilities of robots such as sensors and effectors. If an agent α has all the required capabilities to play role ρ, the *role capability function* (rcf) is used to compute exactly how well α can play ρ. The rcf takes as input a role and an agent and returns a real value in [0..1]. Thus, the rcf allows organization designers to indicate the importance of specific capabilities to each role. For instance, if all the capabilities required to play a role ρ, 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.

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

The *possesses* function captures an agent's capabilities along with the quality of those capabilities. The possesses function takes as input an agent and a capability, and returns a real value in [0..1], where 0 indicates that an agent does not possess the capability.

In OMACS, agents are assigned roles to achieve goals. How well an agent can achieve a particular goal is captured by the potential function. The *potential* function takes as input a goal, a role, and an agent, and returns a real value in [0..1] indicating how well the agent can play the role to achieve the goal. The potential of an agent to play a specific role in order to achieve a specific goal is defined by multiplying the rcf and achieves functions.

$$\forall a{:}A \; r{:}R \; g{:}G \;\; potential(a,r,g) = achieves(r,g) \times rcf(a,r)$$

In OMACS, every organization is governed by a set of policies. OMACS provides three types of policies: assignment policies (P_Φ), behavioral policies (P_{beh}), and reorganization policies (P_{reorg}). *Assignment policies* provide restrictions on the assignment set such as "an agent can only play one role." *Behavioral policies* specify how the organization should behave when some event occurs, while *reorganization policies* provide heuristics that guide the organization when reorganizing.

To determine the best overall set of assignment for a specific set of goals, OMACS defines the *organization assignment function* (oaf), which determines the effectiveness of a given set of assignments. An *assignment* is a tuple of agent, role, and goal that is placed in Φ. For example, $\langle a,r,g \rangle$ means that agent a has been assigned to play role r to achieve goal g. The oaf returns an organization score of a real value in [0..∞], where the higher the organization score, the better the organization performs. Typically, the oaf is application specific; however, a simple oaf is simply the sum of the potentials from an assignment set.

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

A complete definition of OMACS can be found in [5].

2.2 Goal Model for Dynamic Systems

The Goal Model for Dynamic Systems (GMoDS) is defined in terms of two parts: a specification model and a runtime model. The specification model is defined by the designer and captures the goal types and their relationships. The runtime model contains the goals actually instantiated during execution. A complete definition of GMoDS is contained in [7].

2.2.1 Specification Model

The GMoDS specification model has three main entities: goal classes, event classes, and parameters. A *goal class* defines an observable desired state of the world. An *event class* specifies an observable phenomenon of interest that may occur during system execution. Goal classes and event classes are parameterized to provide a given goal or event instance with specific meaning within the system. For instance, a goal type may specify that some *area* be searched, where *area* is a parameter that is given a specific value when a goal of that class is instantiated.

A goal tree is used in GMoDS to conjunctively or disjunctively decompose parent goal classes into a set of sub-goals, or children. Goal classes without children are leaf goal classes. The GMoDS *goal specification tree* specifies how the goal classes are related to one another. A graphical depiction of a GMoDS specification model is shown in Figure 3, with decomposition denoted by «and» or «or» labels. The goal classes in the goal specification

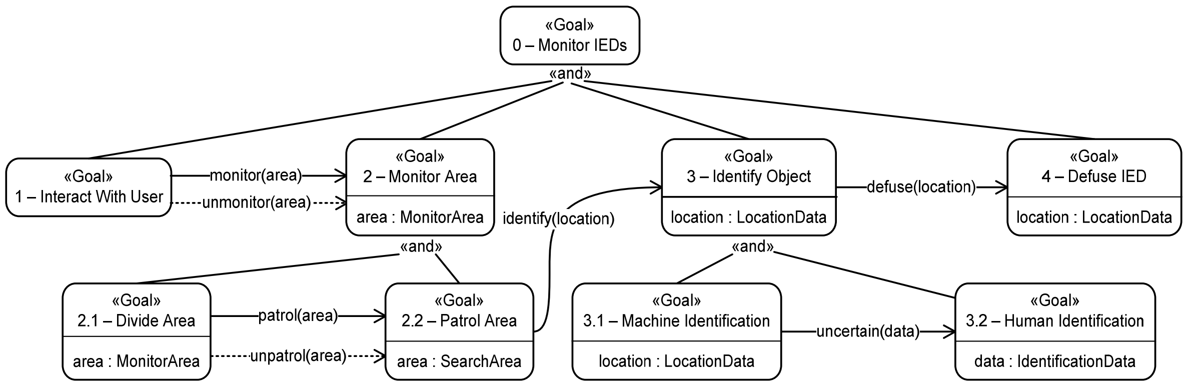


Figure 3. Goal Model

tree are analogous to the specification of classes in an object-oriented language. Classes are designed before hand, and are instantiated at runtime, with each instance having its own set of parameter values. The instances of a goal class are independent of each other. The goal instances are inserted into a goal instance tree at runtime. Each goal instance is achieved independently of every other instance of that goal.

GMoDS uses a set of relations within the tree structure to specify how runtime goals may interact. Because goal instances are created based on the occurrence of specific events, the effect of these events on the goal instance tree must be defined. There are three relations of interest: triggers, negative triggers and precedes. A *triggers* relation between g1 and g2 predicated on event e1 specifies that a new goal instance of class g2 is created when e1 occurs during the pursuit of a goal instance of class g1. Likewise, a *negative trigger* relation from g1 to g2 on event e1 specifies that instances of goal g2 matching the parameter values of e1 should be removed from the goal instance tree when e1 occurs. To bootstrap the goal instance tree, we define an initial event (or initial trigger), e0, which implicitly occurs and adds the initial set of goals (including the root goal) to the goal instance tree. When the system starts, the initial event occurs and the root goal is added to the goal instance tree. Then all children not triggered by some other event are systematically and recursively added to the goal instance tree.

To allow a full or partial ordered execution of goal instances in the system, the designer may specify goal precedence between two goal classes via the precedes relation. Goal *precedence* ensures that no agents may work on a specific goal instance until all goal instances that precede that goal have been achieved. There are several restrictions on goal precedence in the goal specification tree, including restrictions on precedence cycles, mixed trigger/precedence cycles, and a goal preceding its ancestors.

2.2.2 Runtime Model

At runtime, goals are instantiated based on the occurrence of specific events as defined by the GMoDS runtime model. Once instantiated, each instance goal is put in exactly one of six sets: Triggered, Active, Achieved, Removed, Failed or Obviated as shown in Figure 4, where the arrows indicate allowable transitions of goals between sets. Each goal instance is initially placed in the *Triggered* set and stays in that set until it becomes active, failed, obviated, or is removed. The *Active* set includes those goals that have been triggered and are not preceded. Goals in the Active set remain there until they are achieved, failed, obviated, or removed. When an agent achieves a goal, that goal is moved from the Active set into the *Achieved* set. The *Failed* set contains goals that the system can never achieve. The *Obviated* set contains goals that are no longer needed by the system; these goals are not achieved and should not be assigned to any agent. The *Removed* set contains goals that have been removed as the result of a negative trigger. Once in the Removed set, a goal is treated as if it never existed, which means that any precedence/triggers relations related to that goal cease to exist.

The system interacts with the runtime model via two operations: occurred, and initialTrigger. The *initialTrigger* operation creates the initial set of goal instances as defined by the specification tree. The *occurred* operation updates the runtime model based on the occurrence of specific events. There are two types of events of interest: application specific events as defined in the goal specification tree, and general events such as goal achievement or goal failure. The goal sets are modified appropriately based on the event that occurred. Both the initialTrigger and occurred operations return the changes to the Active goal set (which are passed to the OMACS model as changes to the organizational goals). When goals are achieved, the runtime model is updated by moving the appropriate goals into the Achieved set and possibly moving (1) parent goals into the Achieved set, (2) obviated goals to the Obviated set, or (3) goals from the Triggered set into the Active set (if their precedence restrictions have been removed).

3. SYSTEM DESIGN

To design systems for use with OMACS and GMoDS, we developed the *Organization-based Multiagent System Engineering* (O-MaSE) methodology [6], which provides the necessary processes, models, tools, and techniques for designing and implementing OMACS/GMoDS based systems. For our demonstration scenario, the O-MaSE design process yielded four

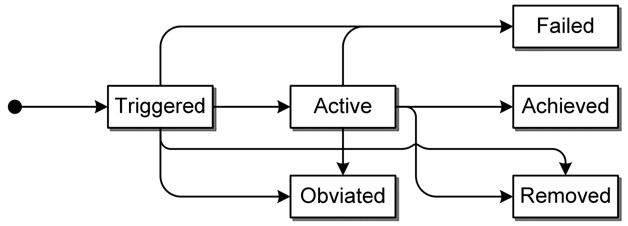


Figure 4. GMoDS Runtime Model

key entities: system goals, roles, capabilities, and agent types. The *GMoDS model* captures the goals of our system. The *roles* specify the behavior and *capabilities* required to achieve the system goals, while the *agent types* define the types of robots in the system based on the capabilities they possess.

3.1 Goals

Figure 3 shows the GMoDS model for the IED detection system. The top-level goal is Monitor IEDs, which has four subgoals: Interact With User, Monitor Area, Identify Object, and Defuse IED. Initially, the subgoal Interact With User is the only goal that exists; the rest of the subgoals are triggered by events.

The Interact with User goal is automatically assigned to an appropriate agent. The Monitor Area goal (and its subgoal Divide Area) is triggered by the monitor event generated by the agent pursuing the Interact With User goal. A monitor event occurs when the human operator specifies an area to monitor for IEDs. Once the Divide Area goal is triggered, it is assigned to an appropriate agent. The agent pursing the Divide Area goal raises the patrol event when it creates a new area to be patrolled. The patrol event causes an instance of the Patrol Area goal to be instantiated and assigned to an agent. When the agent pursuing the Patrol Area goal detects a suspicious object, the identify event is raised causing an Identify Object goal and its subgoal Machine Identification to be instantiated; the Machine Identification goal is then assigned to an agent. If the agent pursuing the Machine Identification goal cannot identify a suspicious object, it raises the uncertain event. The uncertain event causes an instance of the Human Identification goal to be created and ultimately assigned to the human operator for identification. The defuse event can be raised by either the human operator or the agent pursuing the Machine Identification goal when an IED has been identified. The defuse event causes a Defuse IED goal to be instantiated and then assigned to a capable agent.

3.2 Roles

Table 1 shows the system roles, the leaf goals that they can achieve, and the capabilities required to play them. One role has been defined for each of the six leaf goals in Figure 3.

The User Interaction role allows the human operator to input the area to be monitored as well as displaying information about the state of the system such as search areas, agent locations, and the current assignments.

The Area Divider role partitions the overall search area into smaller subareas to be assigned to individual robots. When a subarea is defined, a patrol event is raised causing the creation of a new Patrol Area goal instance.

The Patroller role defines the behavior required to patrol a search area. If a suspicious object is found, the robot raises the identify event and information about the suspicious object is passed to the User Interaction role.

The Machine Identifier role defines how to analyze a suspicious object. If the robot cannot classify an object as inert or an IED, it raises the uncertain event. However, if the object is classified as an IED, the robot raises the defuse event. In any case, the User Interaction role is informed of the result.

Table 1. IED Roles

Role	*Goals Achieved*	*Capabilities Required*
User Interaction	Interact With User	User Interface
Area Divider	Divide Area	Communication Area Division Algorithm
Patroller	Patrol Area	Communication Movement Suspicious Object Detection
Machine Identifier	Machine Identification	Communication Movement Explosive Device Detection
Human Identifier	Human Identification	Communication Human Identification Display
Defuser	Defuse IED	Communication Movement Explosive Device Detection Explosive Device Disposal

The Human Identifier role presents information to the human operator so that the operator can determine if a suspicious object is an IED. If the human operator decides that the object is an IED, the agent raises the defuse event. Otherwise, the suspicious object is classified as inert.

The Defuser role defines how to dispose of an IED, which includes disarming the IED on the spot or moving the IED to a safer location for disarming or detonation. The Defuser role informs the User Interaction role of its status.

3.3 Capabilities

Capabilities are critical to OMACS-based systems as agents are assigned to roles based on the capabilities they possess. Capabilities are defined as a set of *actions*, which can be used to represent logical/physical interactions with the environment or computational processes. Environmental interactions include getting sonar readings, moving a robotic arm, and closing or releasing a gripper [6]. The capabilities used in our IED detection system are described in Table 2.

3.4 Agents

OMACS-based systems define agent types by the capabilities they possess. Table 3 shows the types of agents in the IED system. Note that in Table 3 the *Playable Roles* are derived from the *Capabilities Possessed* and are not hard coded. The Patroller, Identifier, and Defuser agents are all played by robots based on their capabilities, which are defined at an abstract level and are actually based on physical configurations of the robots. For instance, a camera is part of the configuration required for the Suspicious Object Detection and Explosive Device Detection capabilities, while a robotic arm or gripper is required by the Explosive Device Disposal capability. The Laptop Agent requires

Table 2. Capability Definitions

Capability	Description
User Interface	displays monitored area, robot location, location and classification of suspicious objects to operator
Area Division Algorithm	algorithm partitions the monitoring area into smaller search areas is provided by the capability
Communication	used to communicate assignments, events, and application specific information
Movement	enables robots to move and includes collision avoidance
Suspicious Object Detection	algorithm detects objects with certain profile (e.g., IED); uses sensors such as sonars, cameras, explosive detectors
Explosive Device Detection	similar to suspicious object detection but with more accurate classification ability
Human Identification Display	interacts directly with operator; presents information and returns responses to object identification requests
Explosive Device Disposal	allows robots to dispose of IEDs; uses robot gripper to pick up and move an IED to a safe location

the ability to interact with humans using a hand held or laptop computer.

4. IMPLEMENTATION ARCHITECTURE

The robot software was implemented using the *Organization-Based Agent Architecture* (OBAA) [5] shown in Figure 5. Each OBAA agent (robot) has two main components: the *Execution Component* (EC) and the *Control Component* (CC). The EC contains application specific behavior of the robot while the CC is the "brain" of an agent where the organizational knowledge and decisions are made. The CC code is generally domain independent.

The EC consists of the Role Control Component (RCC), the role behavior code, and the software for interfacing to capabilities. The RCC interfaces between the EC and the CC. Assignments from the CC are given to the RCC, which determines how the assignments are carried out. In addition, runtime events are sent to the CC via the RCC. The RCC controls multi-role execution via a modified rate-monotonic scheduling algorithm [18].

The CC contains the adaptive behavior logic for an OMACS-based system, which consists of the Goal Reasoning (GR), Organization Model (OM), Reorganization Algorithm (RA), and Organizational Reasoning (OR) components. The GR implements the GMoDS goal reasoning including goal sequencing, goal instantiation, and goal achievement. The OM maintains the knowledge about the organization such as the current agents, goals, and assignments. The RA computes new assignments when the need arises. The RA can be application-independent but can also incorporate application specific policies when required for optimal or efficient reorganization. For example, the IED system uses a policy to select the agent with the least workload (i.e., the least number of assignments) for new goals. Finally, the OR

Table 3. Agent Types

Agent Types	*Capabilities Possessed*	*Playable Roles*
Laptop Agent	Communication User Interface Area Division Algorithm Human Identification Display	User Interface Area Divider Human Identification
Patroller Agent	Communication Movement Suspicious Object Detection	Patroller
Identifier Agent	Communication Movement Explosive Device Detection	Machine Identifier
Defuser Agent	Communication Movement Explosive Device Detection Explosive Device Disposal	Defuser

integrates the functions of the GR, OM, and RA and acts as an interface to the EC. Assignments are transmitted to the EC and events from the EC are transmitted to the OR. The OR is responsible for supplying the OM with updated information about the agents, assignments, and goals. The OR also decides when to reorganize and whether the reorganization should be full or partial.

The agent OR components work together to keep the system coherent. We are currently using a centralized approach due to the complexities associated with distributed reasoning (see Section 6 for a discussion of current work in the area). In our centralized

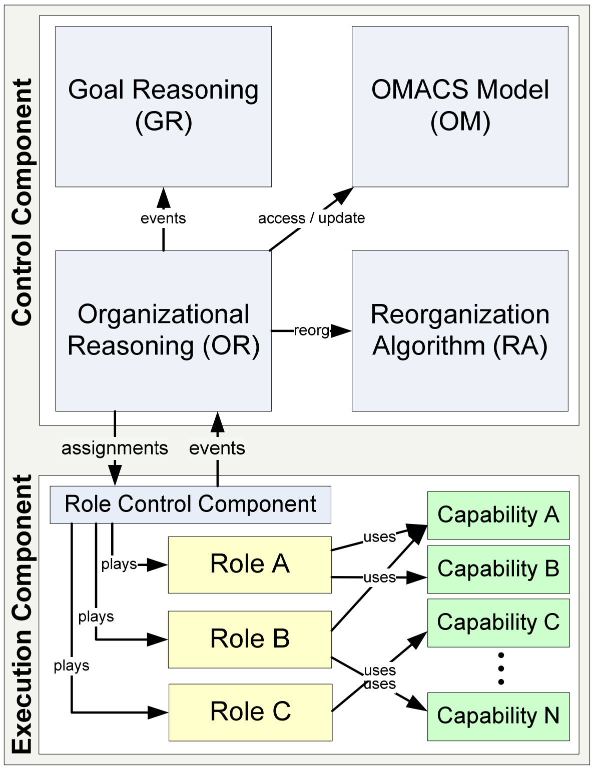


Figure 5. Agent Architecture

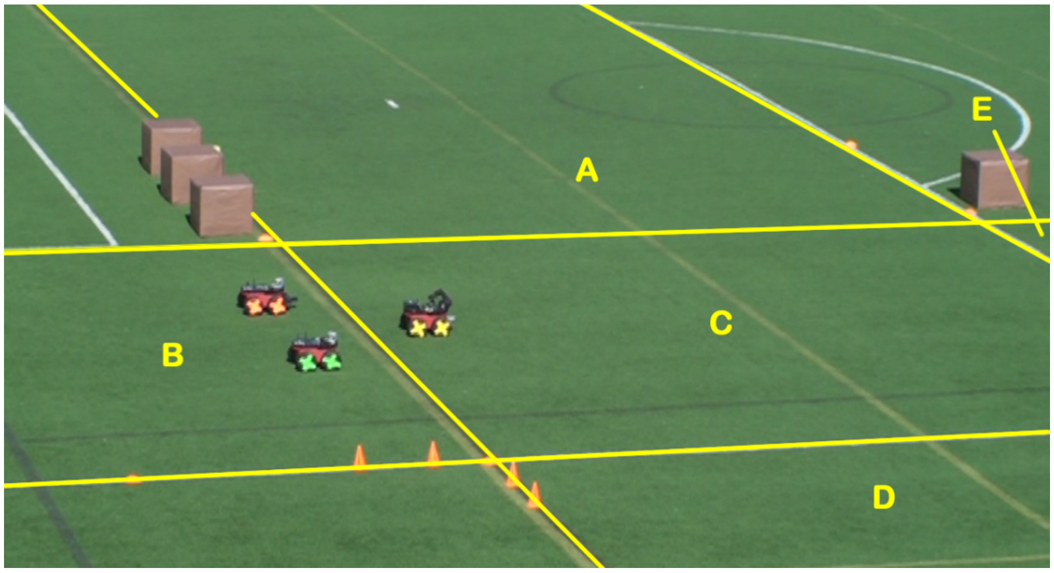


Figure 6. IED Detection System

approach, a team OR 'master' handles all the logic processing such as reorganization, maintaining the OM, and processing events. The OR 'slaves' simply relay information to/from the master to its EC. We are currently working on distributed approaches for the ORs. The beauty of the OBAA architecture is that the CC code can be completely rewritten without affecting the application specific EC.

5. EVALUATION

In this section, we evaluate our system on its ability to adapt to failures, which includes total robot failure or the degradation of a robot's capability. We implemented the system using a heterogeneous team of three Pioneer 3-AT robots and a single laptop agent. The robots are of the Patroller Agent type while the laptop is of the Laptop Agent type.

To test system adaptability, random robots were disabled in one of two ways after the system had assigned their search areas and had started patrolling. First, we simply turned a robot off. The OR constantly polled the Communication capability to detect if a robot was no longer active, in which case the OR removed that robot from the OM knowledge. The OR then called the RA to compute a new set of assignments in which the two remaining robots took on the assignments of the disabled robot. Second, we simulated capability degradation by modifying the *possesses* score for a robot's capability. This caused either (1) the robot to detect that it was unable to proceed and report a failure, or (2) the OR to detect that the robot was unable to continue. In both cases, reorganization occurred and the affected assignments were reassigned to capable robots.

We ran eight experiments. Figure 6 shows a picture of our experimental setup with five search areas (A, B, C, D, and E). We disabled up to two robots in each experiment. Each time the OR successfully reassigned the goals from disabled robots to the remaining robots. In one experiment, robot 1 was initially assigned to patrol area A and B, robot 2 was assigned to patrol area E, and robot 3 was assigned to patrol area C and D. When robot 1 was disabled, robot 2 was assigned area A while robot 3 was assigned area B. Upon disabling robot 2, robot 3 was assigned all five areas (A, B, C, D, and E).

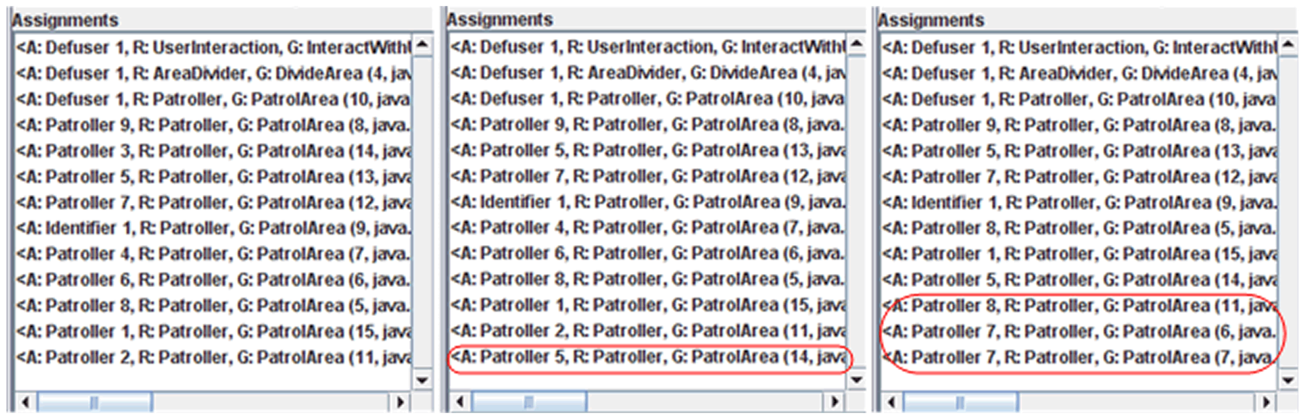
To validate our results for scalability, we also simulated the same scenario. The simulated configuration consisted of 11 robots, 9 of the Patroller Agent (Patroller 1..9) type, 1 of Identifier Agent (Identifier 1) type, and 1 capable of both the Defuser Agent and Laptop Agent types (Defuser 1). With a larger number of agents, we were able to test several permutations. Figure 7 illustrates one example of how the system adapted. Figure 7(a) shows the assignments before any failures. There are 13 assignments; 11 are for patrolling and all 11 Patroller robots are assigned since they all are capable of playing the Patroller role. In Figure 7, an assignment such as <A:Defuser 1, R:Area Divider, G:Divide Area(…)> means that the Defuser 1 robot has been assigned to play the Area Divider role to achieve the Divide Area(…) goal. Figure 7(b) illustrates the reassignment that occurred when Patroller 3 failed. The OR detected the failure and reassigned the most suitable agent Patroller 5. Figure 7(c) illustrates the reassignment that occurred when three additional robots (Patroller 2, Patroller 4, and Patroller 6) failed. Again, the OR detected the failures and new assignments were made. In this case, area 11 was assigned to Patroller 8 while areas 6 and 7 were reassigned to Patroller 7.

Hundreds of combinations were tested and in all cases, the OR adapted appropriately to the failures. The only exception occurred when Defuser 1 failed, since Defuser 1 was the master in our centralized OR implementation.

The adaptive behavior demonstrated is a result of reorganization made possible by the organizational knowledge in OMACs. This knowledge is obtained directly from the O-MaSE models developed and thus the results presented are repeatable.

6. DISCUSSION AND FUTURE WORK

The OMACS and GMoDS models have been successfully used in several types of complex distributed systems including



```
Assignments
<A: Defuser 1, R: UserInteraction, G: InteractWithI
<A: Defuser 1, R: AreaDivider, G: DivideArea (4, jav
<A: Defuser 1, R: Patroller, G: PatrolArea (10, java
<A: Patroller 9, R: Patroller, G: PatrolArea (8, java.
<A: Patroller 3, R: Patroller, G: PatrolArea (14, java
<A: Patroller 5, R: Patroller, G: PatrolArea (13, java
<A: Patroller 7, R: Patroller, G: PatrolArea (12, java
<A: Identifier 1, R: Patroller, G: PatrolArea (9, java.
<A: Patroller 4, R: Patroller, G: PatrolArea (7, java.
<A: Patroller 6, R: Patroller, G: PatrolArea (6, java.
<A: Patroller 8, R: Patroller, G: PatrolArea (5, java.
<A: Patroller 1, R: Patroller, G: PatrolArea (15, java
<A: Patroller 2, R: Patroller, G: PatrolArea (11, java
```

```
Assignments
<A: Defuser 1, R: UserInteraction, G: InteractWithI
<A: Defuser 1, R: AreaDivider, G: DivideArea (4, jav
<A: Defuser 1, R: Patroller, G: PatrolArea (10, java
<A: Patroller 9, R: Patroller, G: PatrolArea (8, java.
<A: Patroller 5, R: Patroller, G: PatrolArea (13, java
<A: Patroller 7, R: Patroller, G: PatrolArea (12, java
<A: Identifier 1, R: Patroller, G: PatrolArea (9, java.
<A: Patroller 4, R: Patroller, G: PatrolArea (7, java.
<A: Patroller 6, R: Patroller, G: PatrolArea (6, java.
<A: Patroller 8, R: Patroller, G: PatrolArea (5, java.
<A: Patroller 1, R: Patroller, G: PatrolArea (15, java
<A: Patroller 2, R: Patroller, G: PatrolArea (11, java
<A: Patroller 5, R: Patroller, G: PatrolArea (14, java
```

```
Assignments
<A: Defuser 1, R: UserInteraction, G: InteractWithI
<A: Defuser 1, R: AreaDivider, G: DivideArea (4, jav
<A: Defuser 1, R: Patroller, G: PatrolArea (10, java
<A: Patroller 9, R: Patroller, G: PatrolArea (8, java.
<A: Patroller 5, R: Patroller, G: PatrolArea (13, java
<A: Patroller 7, R: Patroller, G: PatrolArea (12, java
<A: Identifier 1, R: Patroller, G: PatrolArea (9, java.
<A: Patroller 8, R: Patroller, G: PatrolArea (5, java.
<A: Patroller 1, R: Patroller, G: PatrolArea (15, java
<A: Patroller 5, R: Patroller, G: PatrolArea (14, java
<A: Patroller 8, R: Patroller, G: PatrolArea (11, java
<A: Patroller 7, R: Patroller, G: PatrolArea (6, java.
<A: Patroller 7, R: Patroller, G: PatrolArea (7, java
```

(a) **(b)** **(c)**

Figure 7. Samples of Reassignments

information systems [8], wireless sensor networks [15], cooperative robotics [5, 6], and a variety of multiagent systems [5, 8]. The key to this generality is that we do not try to model the system elements directly as components, but use an abstract representation based on human organizations. Using this abstraction, we have demonstrated the ability to organize and reorganize a system around a set of system level goals. These goals, therefore, are the key to a system that responds appropriately. Thus, GMoDS provides the driving force behind OMACS. By being able to specify system objectives as high-level goals that are decomposed into individual goals, there is a direct tie from the low-level goals or tasks being accomplished to the overall objectives of the system. By including the notion of event triggers, GMoDS provides a model of system objectives that can react to events that occur in the environment or in the system configuration.

While the implementation presented in this paper uses a centralized algorithm, we are currently working on a fully distributed algorithm for the OR. Our approach focuses on keeping information about the OMACS and GMoDS models synchronized, which allows all assignment computation to be done without requiring interaction between the robots. Our distributed implementation will eliminate failure due to a centralized OR and will work in with imperfect communications.

We are also currently using OMACS and GMoDS to develop new forms of supervisory control for robot teams, which we term organizational control. The main concept in *organizational control* is that the operator interacts directly with a *team* instead of individual robots. They key to interacting with a team is the *Team Intelligence* layer as shown in Figure 8, which abstracts individual robots into a team allowing the operator to directly task the team without worrying about how the team decomposes and distributes those commands to individual robots. The Team Intelligence layer is implemented by the collective control components of each robot on the team. The IED system demonstrates *organizational* control in the form of control by design. *Control by design* occurs when the operator is explicitly taken into account during the design of the system. We are also investigating a second form of organizational control called *control by model manipulation* in which the operator is not considered during the system design. Instead, the operator controls the team by manipulating the runtime model data directly.

7. RELATED WORK

Using runtime models of systems has become popular in recent years as an approach to developing self-adaptive systems [2]. In general, various aspects of the system (e.g., architecture) are modeled explicitly and populated and monitored at runtime to help with automatic reconfiguration of the system when required.

While there has been some application of runtime models to single robot systems, we are unaware of the explicit use of runtime models to configure or organize multiple robot systems. In [12], explicit runtime architectural models are applied to manage runtime adaptation within simulated robots. The system uses the model to modify the runtime architecture configuration thus allowing the robots to change their behavior based on specific situations. Other use of runtime models to configure single robots includes [21], where a preliminary version of OMACS was used to overcome sensor/effector loss. In the system, the robot was controlled internally by a multiagent system whose agents were various sensors and effectors; when a sensor or effector failed, the system reorganized to provide the best

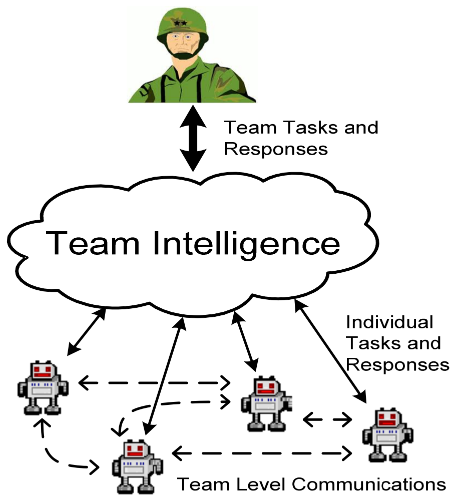


Figure 8. Organizational Control

alternative configuration.

The use of goal-based models to capture requirements has been proposed for several years and most current methods find their roots in KAOS or i* [31, 34]. More recent entries such as Techne [16] and RELAX [33] extend existing approaches to deal with uncertainty and inconsistency. However, to our knowledge there are no models that are used directly for capturing requirements as well as at runtime to drive system behavior and adaptation. There has been several organization models developed to support multiagent systems including OMNI [10], OperA [9] and HarmonIA [32]. However, these models were developed to support open multiagent systems and have not been applied to robotic systems.

8. ACKNOWLEDGMENTS

This work was supported by grants from the US National Science Foundation (0347545) and the Air Force Office of Scientific Research (FA9550-09-1-0108), and a contract from the US Marine Corps Systems Command to M2 Technologies, Inc.

9. REFERENCES

[1] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A. and Letier, E. 2010. Requirements reflection: requirements as runtime entities. *Proc. of the 32nd ACM/IEEE Intl. Conf on Software Engineering - Vol 2* (ICSE '10), ACM, 199-202.

[2] Blair, G. Bencomo, N. and France, R.B. 2009. Models@ run.time, *Computer*, (Oct. 2009), 22-27.

[3] Botelho, S. C., Alami, R.: M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. 1999. *Proceedings of IEEE Int Conference on Robotics and Automation*, pp. 1234-1238.

[4] Cheng, B.H.C., Sawyer, P., Bencomo, N., and Whittle, J. 2009. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. *Proc. of the 12th Intl. Conf. on Model Driven Engineering Languages and Systems*, Springer, 468-483.

[5] DeLoach, S.A. 2009. Organizational model for adaptive complex systems. in Dignum, V. (ed.) *Multi-agent systems: semantics and dynamics of organizational models*. IGI Global.

[6] DeLoach, S.A. and Garcia-Ojeda, J.C. 2010. O-MaSE: a customizable approach to designing and building complex, adaptive multiagent systems. *Int J. of Agent-Oriented Software Engineering*. 4, 3, 244-280.

[7] DeLoach, S.A. and Miller, M. 2010. A goal model for adaptive complex systems. *Int J. of Computational Intelligence: Theory and Practice*. 5, 2.

[8] DeLoach, S.A., Oyenan, W. and Matson, E.T. 2008. A capabilities-based model for adaptive organizations. *J. of Autonomous Agents and Multi-Agent Systems,* 16, 1, 13–56.

[9] Dignum, V. 2004. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Utrecht Univ.

[10] Dignum, V., Vázquez-Salceda, J., Dignum, F. 2004. Omni: introducing social structure, norms and ontologies into agent organizations. *Programming Multi-Agent Systems: Second Intl.* Workshop (ProMAS 2004), LNCS 3346, 181–198, Springer: Berlin, 2004.

[11] Fua C.H., and Ge, S.S. 2005. COBOS: cooperative backoff adaptive scheme for multirobot task allocation. *IEEE Trans. on Robotics,* 21, 6, 1168–1178.

[12] Georgas, J.C., van der Hoek, A., and Taylor, R.N. 2009. Using architectural models to manage and visualize runtime adaptation, *Computer*, 42, 10 (Oct. 2009), 52-60.

[13] Gerkey, B.P, and Matarić, M.J. 2002. Sold! : auction methods for multirobot coordination. *IEEE Trans. on Robotics and Automation*, 18, 5, 758–768.

[14] Gerkey, B.P, and Matarić, M.J. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Intl. J. of Robotics Research*, 23, 9, 939–954.

[15] iRobot PackBot, http://www.irobot.com/sp.cfm?pageid=171.

[16] Jureta, I.J., Borgida, A., Ernst, N.A., Mylopoulos, J. 2010. Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. *Proc of 18th Intl Req. Eng Conf.* 115-124, IEEE.

[17] Kube, C.R., and Zhang, H. 1993. Collective robotics: from social insects to robots. Adaptive Behavior, 2, 2, 189–218.

[18] Liu, J.W.S. 2000. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ.

[19] Luck, M., McBurney, P., Shehory, O. and Willmott, S. 2005. Agent technology: computing as interaction (a roadmap for agent based computing). AgentLink: Southampton, UK.

[20] M2 Technologies. 2007. *M2 and MC IED Awareness: Controlling Robots Teams in Urban Environments.* Presentation.

[21] Matson, E., and DeLoach, S.A. 2004. Enabling intra-robotic capabilities adaptation using an organization-based multiagent system. *Proceedings of the IEEE Intl Conf on Robotics and Automation*. 3, 2135- 2140.

[22] Oyenan, W.H., DeLoach, S.A. and Singh, G. 2010. An organizational design for adaptive sensor networks. *Proceedings of the 2010 IEE/WIC/ACM Intl Conf on Intelligent Agent Technology*, 2, 239-242.

[23] Parker, L.E. 1998. ALLIANCE: an architecture for fault tolerant multirobot cooperation. *IEEE Trans. on Robotics and Automation*, 14, 2, 220–240.

[24] Parker, L.E. 2008. Distributed intelligence: overview of the field and its application in multi-robot systems. *J. of Physical Agents*, 2, 1, 5–14.

[25] Passino, K.M. 2002. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine,* 22, 3, 52–67.

[26] Sawyer, P., Bencomo, N., Whittle, J., Letier, E, Finkelstein, A. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. *Proc of 18th IEEE Intl Requirements Engineering Conf.* 95-103, IEEE.

[27] Simmons, R., Singh, S., Hershberger, D., Ramos, J. and Smith, T. 2001. First results in the coordination of heterogeneous robots for large-scale assembly. In *Experimental Robotics VII*, Lecture Notes in Control and Information Sciences 271, 323–332. Springer, Berlin.

[28] Stone P., and Veloso, M. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110, 2, 241–273.

[29] Tang, F., and Parker, L.E. 2005. ASyMTRe: automated synthesis of multi-robot task solutions through software reconfiguration." In *Proc. of the 2005 IEEE Intl. Conf. on Robotics and Automation*, IEEE, 1501–1508.

[30] Tang, F., and Parker, L.E. 2005. Distributed multi-robot coalitions through ASyMTRe-d. *Proceedings of the Intl. Conf. on Intelligent Robots and Systems*, IEEE, 2606–2613.

[31] van Lamsweerde, A., Letier, E., and Darimont, R. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. on Software Engineering*, 24, 11, 908–926.

[32] Vazquez-Salceda, J., and Dignum, F. 2003. Modelling electronic organizations. in *Multi-agent Systems and Applications III,* LNAI 2691, 584–593, Springer: Berlin.

[33] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.-M. 2009. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. *Proc of 17th Intl Req. Eng Conf*. 79-88, IEEE.

[34] Yu, E.S.K. 1997. Towards modelling and reasoning support for early-phase requirements engineering. *Proc. of the Third Intl. Symp on Req Eng*. 226-235. IEEE.