Kansas State University
234 Nichols Hall
Manhattan, KS 66506-2302

Phone: (785) 532-6350
Fax: (785) 532-7353
http://macr.cis.ksu.edu/

Technical Report

Designing Adaptive Sensor Networks Using an Organization-based Approach

by

Walamitien H. Oyenan, Scott A. DeLoach and Gurdip Singh

MACR-TR-2010-04

June 1, 2010

Table of Contents

An Organizational Design for Adaptive Sensor Networks

Walamitien H. Oyenan, Scott A. DeLoach and Gurdip Singh

Department of Computing & Information Sciences, Kansas State University

{oyenan, sdeloach, gurdip}@ksu.edu

Abstract

As wireless sensor network applications grow in complexity, ad-hoc techniques for developing this kind of systems are not adequate anymore. Therefore it is crucial that these systems be adaptive and autonomous in order to remain functional even in the face of unreliable communications, dead nodes, and other unexpected failures. We propose to manage a sensor network based on a rigorous multiagent organizational design, which helps separate application logic from low-level sensor implementation details. The organizational design allows designers to specify high-level goals that the systems will try to achieve based on sensor capabilities. We present our design models and agent architecture used to develop a surveillance application in which sensors are used to collaboratively monitor and track all vehicles entering an area. We analyze adaptation mechanisms for this application by providing quantitative results obtained through a simulated system.

1. Introduction

Wireless sensor networks (WSN) have been used in different types of applications to obtain information about the environment. Large-scale deployments of these networks have been used in many diverse fields such as wildlife habitat monitoring [16], traffic monitoring [1] and lighting control [18]. WSN have constrained power and computational resources and often operate unattended in highly dynamic and harsh environments. These conditions along with the inherently distributed nature of these systems make designing WSN applications a very complex and challenging task. Consequently, ad-hoc techniques for developing this kind of systems are not adequate anymore. For instance, when developing a complex WSN application, it would be too complicated to explicitly encode interactions among all the nodes and write error-handling logic for every failure. Therefore, it is crucial that WSN be adaptive and autonomous in order to remain functional even in the face of unreliable communications, dead nodes, and other unexpected failures.

We propose to manage sensor nodes based on a multiagent organizational design, which helps separate application logic from low-level sensor implementation. In fact, organizations facilitate cooperation between heterogeneous sensors and provide guidelines to handle recurrent events

like sensor registration, sensor failure, and capability degradation. Moreover, due to their distributed structure, sensor networks are inherently suitable for multiagent systems approaches [21]. Hence, sensors can be viewed as independent autonomous agents that can collaboratively work and achieve a common goal.

To design organization-based multiagent systems, many framework have been proposed [8]. However, we are basing our work on the Organization Model for Adaptive Computational Systems (OMACS) because of its capability orientation which makes it particularly suitable with heterogonous sensors capabilities. The objective of an organizational design in OMACS is to allow designers to specify high-level goals that the system will try to achieve based on the team capabilities. OMACS defines the organizational concepts that are required in order to provide the agents with the required knowledge to self-organize. In addition to those general organizational concepts, OMACS allows designers to define application events that can cause the system to reorganize. For instance, an application specific event in a surveillance application can be the appearance of a vehicle. In response to this event, the organization would try to find an appropriate goal to pursue.

Moreover, OMACS is supported by a rigorous process defined from the Organization-based Multiagent Engineering. This process has been used successfully to build complex autonomic systems [17]. In this report, we follow a similar design process by defining key design models capturing the important organizational concepts. These models will be implemented into runtime models that will be used as the basis for the adaptive reasoning.

To show how our rigorous organization design can be applied to develop sensor network applications, we focus on using a common sensor networks application, a surveillance application, in which sensors are used to collaboratively monitor and track vehicles entering an area. In our application, sensors are deployed over a large area. As sensors have a limited sensing range, no one sensor can cover the entire area. Hence, agents, which are controlling sensors, need to collaborate in order to provide data covering the entire area of interest. In particular, in order to conserve energy, we would like to provide maximum coverage with the minimum number of agents. In the absence of targets, agents not monitoring must be in a sleep state in order to conserve energy. Once a target is detected, all sensors within a certain

distance from the target must be activated in order to provide the maximum number of measurements that will be used in order to properly locate and identify the target. For this particular application, we will show how the organizational approach allows the system to autonomously adapt to overcome sensor failures and loss of performance due to capability degradation.

Our organizational design is based on two main design models (goal model and role model) and a set of policies, which are presented in Section 2. Then, in Section 3 we introduce our generic agent architecture that is used to develop agents for two different platforms. At runtime, those agents will be assigned to roles to achieve goals. We examine how the decision process takes place at runtime in Section 4 and evaluate how the organization behaves when facing unexpected failures in Section 5.

2. System Design

In general, multiagent organizations are designed using a set of related design models which capture various organizational concepts such as goals, roles, interactions and norms [7]. In this work, we use the organizational concepts defined in the OMACS model. OMACS defines an *organization* as a set of *goals* that need to be accomplished, a set of *roles* that must be played to achieve those goals, a set of *capabilities* required to play those roles, a set of *agents* that are assigned to roles in order to achieve organization goals and a set of *policies* that constrain the possible behaviors of agents in the organization. At runtime, the assignments of agents to play roles to achieve goals represent the key functionality that allows systems to be adaptive.

Our organizational design follows the multiagent systems development process proposed in [17], which is built based on method fragments from the O-MaSE framework [9]. Although the methodology introduced in [17] presents several design models, for the sake of brevity, the sensor network design presented here only focuses on two main models, which are adequate to capture all the organizational concepts aforementioned.

First, we define our organization goals and organize them in a goal model [4]. This goal model consists of AND/OR decompositions of goals along with a trigger relationship that allows goals to be instantiated with particular application-specific parameters. For instance, in our

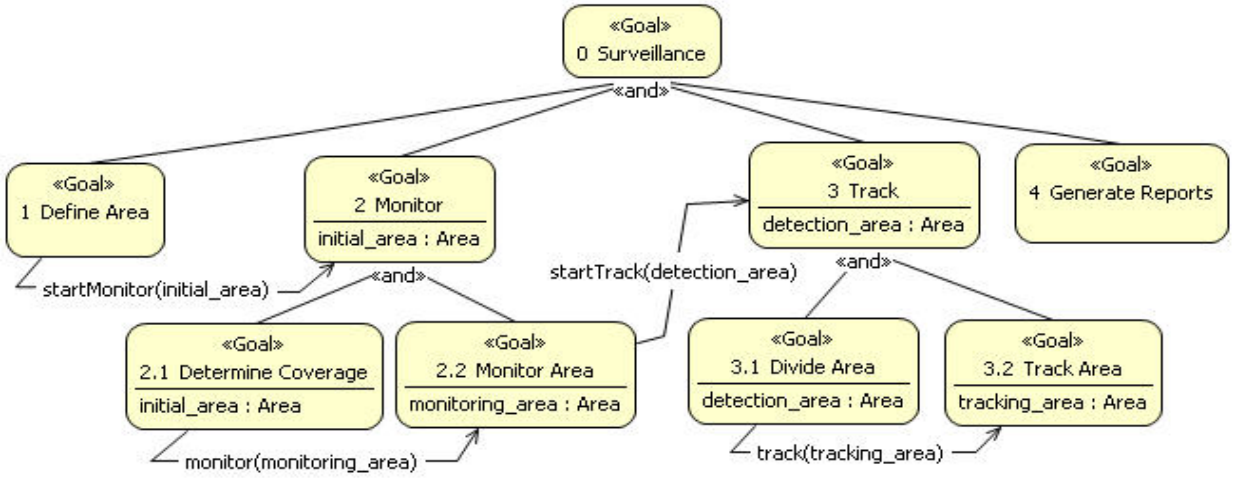

Figure 1. Surveillance Goal Model

surveillance application, monitoring or tracking goals would need to be instantiated for a particular area that will be given as a goal parameter. At runtime, the set of goals changes dynamically as new goals are created or existing ones are achieved.

Then, we identify the organization roles, which represent a high-level description of the behavior required to achieve particular goals. They are organized into a role model, which captures the roles along with their possible interactions (defined as protocols) and their required capabilities.

Finally, agents are designed to assume roles in an organization. They are designed separately and can participate in the organization as long as they have the required capabilities. In the next section, we introduce our agent architecture, which exploits the design models to build organizational knowledge.

Design models are created using agentTool III (aT^3), a multiagent development environment built on the Eclipse platform [3]. aT^3 supports the development and validation of design models that can be automatically translated into platform specific runtime models. More details about the design models and the methodology for which they apply can be found in [9] and [17].

2.1. Goals

We derive the goal model presented in Figure 1 from the requirements of our surveillance system. The goals are conjunctive, meaning that all leaf goals are required to be achieved in

order for the main goal to be achieved. The arrows between goals indicate trigger events and their parameters. The top-level goal is the Surveillance goal. This goal is decomposed into four subgoals: Define Area, Monitor, Track, and Generate Reports. The Monitor and Track goals are further decomposed into Determine Coverage, Monitor Area, Divide Area and Track Area. The organization will actively pursue the leaf goals by assigning them to agents. Triggers between goals are based on events generated by roles during their execution and thereby impose a temporal order in which goals can be activated.

Essentially, once an area is defined via the achievement of the *Define Area* goal, an event is generated, which triggers the instantiation of the goal *Monitor(initial_area)*. As the area parameter might be too large for any single agent, the goal *Determine Coverage* is in charge of dividing it into smaller subareas that can potentially be covered by a single agent. Each of these subareas (*monitoring_area*) is used as a parameter to a *Monitor Area* goal. Each *Monitor Area* goal can in its turn initiate a *Track* goal for a given detection area. Once again, the detection area defined for the track goal might be too vast for a single agent. Hence, the *Divide Area* goal breaks up the detection area, which results in the creation of a *Track Area* goal for each subarea (*tracking_area*) identified. Finally, all application data gathered are aggregated in a user-friendly report by the *Generate Reports* goal.

2.2. Roles, Capabilities and Policies

Next, we define the roles that can achieve the leaf goals identified in the goal model. For each role, we specify a behavior that will be followed by agents enacting this role. Examples of role behaviors (represented as state machines) will be introduced in Section 4. We identified five roles necessary to achieve the goals: User Interface, Coverage Processor, Monitor, Divider and Tracker. In addition, we defined the capabilities that can be required by the roles: Magnetometer (to detect targets), Power (to measure remaining power), GUI (to interact with the user) and Computation (to execute coverage and division algorithms). Table 1 summarizes the main organizational entities of our surveillance organization design by indicating for each role the goals it can achieve and the capabilities it requires.

Finally, to complete the design, we specify some reorganization policies that will guide the system when assigning the goals to agents [11]. These policies typically specify the kind of assignments the system should preferably make or avoid. For instance, we specified a policy requiring that monitoring agents should together provide a 100% coverage of the area of interest. Policies are expressed as conditional statements related to one or more assignments. The policies used in our systems are discussed in Section 5.

3. System Architecture

Once the organization design models are defined, we need to define the agents that will be participating in the organization. In our application, we have two types of agents: *Base Station agents* and *Mote agents*. The system contains one Base Station agent running on a PC platform and several Mote agents running on a Berkeley mote platform, which is a sensor network platform [2].

3.1. Overall Architecture

The overall system architecture is presented in Figure 2. The design models (goal model and role model) and policies obtained from aT^3 are automatically translated into runtime models that are used by the Base Station agent to make decision about the reconfiguration of the organization. We chose to have the entire organizational knowledge in the Base Station agent because it has more computational resources than the motes and it is less prone to failure.

Table 1: Organization Goals, Roles, Capabilities

Role	Goals Achieved	Capabilities Required
User Interface	Define Area, Generate Reports	GUI
Coverage Processor	Determine Coverage	Computation
Monitor	Monitor Area	Magnetometer, Power
Divider	Divide Area	Computation
Tracker	Track Area	Magnetometer, Power

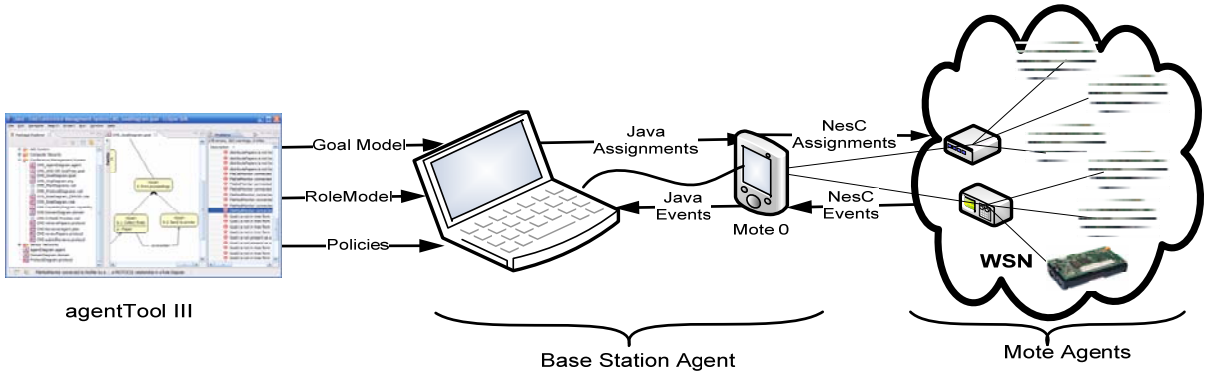

Figure 2. Overall System Architecture

Therefore, in our system, the Base Station is the only agent that possesses the organizational knowledge and decides the next configuration of the organization. Moreover, we assume that all Mote agents are within communication range of the Base Station.

The Base Station agent runs on a laptop with a base station mote (mote 0) attached to it. This mote acts as a gateway and allows the Base Station agent running on a PC to interact with the rest of the agents exclusively running on the mote platform.

Once assignments have been made by the Base Station agent, they are passed on to the proper Mote agents who execute them and return feedback based on events of interest to the organization (goal completion, goal failure, application specific events). Consequently, the Mote agents have limited autonomy as they must agree to play their assigned role and pursue their assigned goal. Nonetheless, they have freedom in the choice of the specific actions necessary to play a role.

3.2. Agent Architecture

The Base Station agent and the Mote agents all participate in the same organization and cooperate to achieve the main organizational goal. Both types of agents are based on the same general architecture, which consists of two main components: the Control Component and the Execution Component (Figure 3). The *Control Component* performs all organization-level reasoning while the *Execution Component* provides all the application specific reasoning. This architecture has been designed to facilitate extensibility and reusability and to provide a clear separation between organization control and application. With this architecture, control

components can be modified to cater for different organization control mechanisms without sacrificing compatibility with the rest of the system.

The Control Component possesses an *organizational knowledge* component that stores all the knowledge about the structure of the organization and a control manager that makes the decisions. The organizational knowledge is created based on design models and updated at runtime as organization events arrive. In addition, agents are added to this knowledge base as they appear and register to participate in the organization. Based on the organizational knowledge, the *control manager* can reason about the state of the entire organization and decide to reconfigure the organization by including/canceling goals or by modifying current assignments. As each Control Component has its own view of the organization, a deliberation process might be needed to reach consensus about the next state of the organization. However, in the system described in this report, we have opted to store the entire organizational knowledge at the Base Station agent who can make decision alone. Hence, all

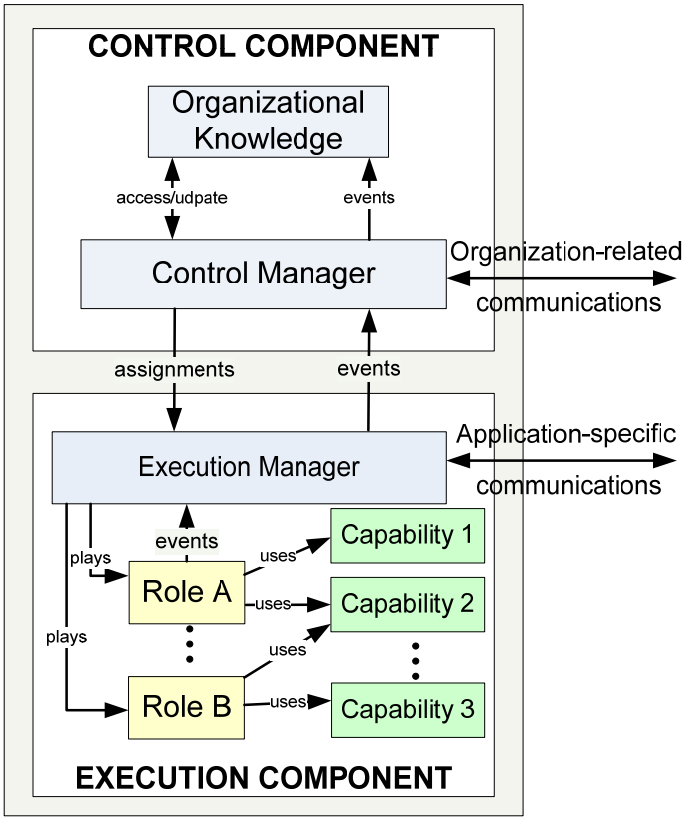

Figure 3. Generic Agent Architecture

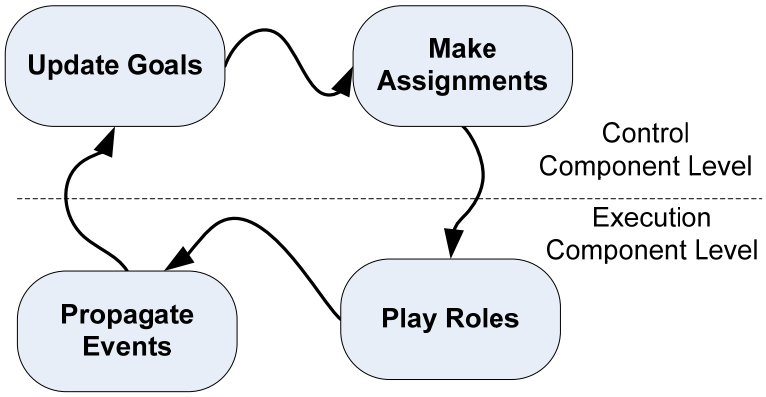

Figure 4. System Phases

events from other agents are passed to the Base Station. Decisions taken by the Base Station agent are passed on to the Control Component of all the agents that are affected, which then forward these assignments to their attached Execution Components.

The Execution Component corresponds to the application specific part of the agent. It is notified by its Control Component about what role to play in the organization. Once it has been assigned a role, the *execution manager* uses its capabilities to execute the plan provided for that role at design time. During role execution, an Execution Component may need to coordinate with other Execution Components in order to exchange application data. In our application, the Mote agents report their sensor data to the Base Station agent (acting as a sink) via their Execution Components.

4. System Implementation

4.1. Runtime Organization

At runtime, the system cycles through four main phases: update goals, make assignments, play roles, propagate events (cf. Figure 4). The *update goals* and *make assignment*s phases are organization-related phases and are performed by agents participating in the organization control, in our case, the Base Station agent. The *play roles* and *propagate events* phases are application-related phases that concern all agents in the organization. Once goals are added in response to organizational events, the Base Station agent assigns agents to play roles to

achieve the newly added goals. Once an agent has been assigned to play a role, it follows the role's plan to achieve its goal and reports all events to the Base Station agent.

To make assignments, the Base Station uses a first-fit greedy algorithm. The assignment algorithm is shown in Figure 5. For each unassigned goal (line 1), we get the first role that can achieve it (line 5) and the first agent that has all the required capabilities to achieve that role (line 8). The assignment thereby produced is checked for policies compliance (line 11). If it fails, the *passPolicies* method removes the roles and agents that caused the assignment to fail and a new assignment is sought. If no assignment exists, the least important policy is deactivated to ensure that the system can progress. In fact, our policies are *guidance policies* [11] that guide the system towards a desired behavior without constraining it. They can be abandoned if they prevent the system from progressing. Policies are arranged by importance order, allowing the system to ignore the least important policies when unable to find an assignment.

4.2. Application

The application consists of sensors arranged in a grid. Each Mote agent is represented by an actual sensor and has the required capabilities to play the Monitor role and the Track role. In addition, the Base Station agent possesses all the required capabilities to play the User Interface, Coverage Processor, and Divider roles. The Mote agents have been implemented in nesC [10], a component-based programming language that is currently used to program the

```
function makeAssignments(activeGoalSet)returns assignmentSet
  1. for each goal g in activeGoalSet.unassigned
  2.   assignment.goal ← g
  3.   do
  4.     for each role r in Knowledge.roles
  5.       if r.achieves(g) then assignment.role ← r; break
  6.     end loop
  7.     for each agent a in Knowledge.agents
  8.       if a.possess(r.requiredCapabilities)
  9.        then assignment.agent ← a; break
  10.         end loop
  11.       until passPolicies(assignment) //can deactivate policy
  12.       assignmentSet.add(assignment)
  13.      end loop
  14.return assignmentSet
```

Figure 5. Assignment Algorithm

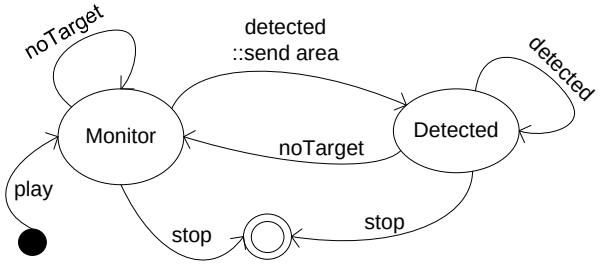

Figure 6. Plan for the Monitor role

Berkeley motes [2] running on the TinyOS [13] operating system. The Base Station agent is implemented in Java.

The Base Station agent gets an area of interest via a user interface (User Interface role) and divides it into subareas such that a unique sensor can cover each subareas (Coverage Processor role). In fact, sensors can only be given areas that fall entirely within their sensing range. This division is made while insuring that the minimum number of sensors cover the entire initial area. Mote agents assigned to the Monitor role execute the role's plan as shown in Figure 6.

For the Monitor role, agents sense the magnetic field at the rate of 1Hz as long as no target is detected (*Monitor* state). Once a target is detected, the agent generates an event to initiate a track over an area equal to twice its sensing radius (*Detected* state). This area has been chosen as a prediction of where the target could soon be located. This event results in the Base Station agent activating the Track Area goal (see Figure 1). The resultant configuration includes the Base Station agent playing the Divider role in order to select a set of subareas that need to be tracked. Subsequently, agents capable of sensing the tracking area are assigned to the Tracking role for a subarea. The plan for the Track role is depicted in Figure 7.

If the tracking agent (the agent playing the tracking role) is on the target trajectory, it eventually detects the target (*Found Target* state). Based on the speed of the target, a tracking agent may have to wait for a certain time before detecting the target (*No Target* state). Taking the target speed into consideration, we have set the wait time such that the tracking agent always detects a target moving in its direction. When the target is lost (*Lost Target* state) or the

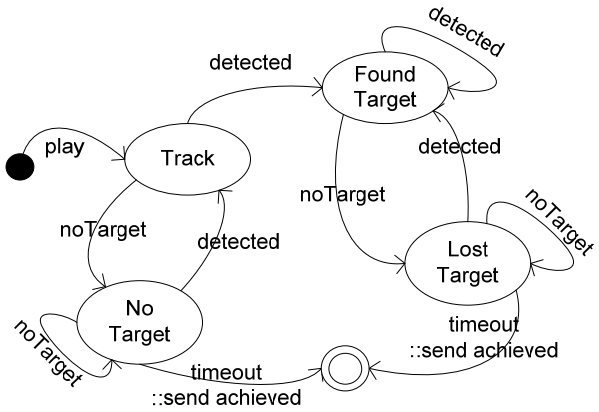

Figure 7. Plan for the Tracker role

target has never been detected (*No Target* state), the agent sends a message to the Base Station agent indicating that it has achieved its role.

Finally, to maintain the sensor topology, sensors periodically send a beacon message indicating that there are still in the network. If a beacon is not received after a certain time, the Base Station considers that the agent has failed and reassigns its roles to other agents. Moreover, to reduce the number of messages sent in the application, the beacon messages are also used to report capability status updates (such as power level of the sensors).

5. Experimental Results

The actual test bed for our experiments consisted of 25 sensors evenly distributed in a 5×5 grid with integer (x, y) coordinates ranging from (0, 0) to (4, 4). The sensors covered an area of 100ft x 100ft. We chose TOSSIM [15] as a simulation environment since it uses nesC and it can emulate the execution of the real code on the motes without the need for deployment. In addition, TOSSIM scripting language, Tython [6], allows us to interact with the simulator environment by inserting failures and by simulating moving targets. We simulated a moving target as a magnetometer source that can linearly affect the magnetometer readings of nearby sensors based on their distance. We set up the detection threshold value such that sensors can detect the moving target for up to 30 ft. We also set the target to move on straight lines at the constant speed of 2ft/s. For all our experiments, we assumed that communication is reliable and that all nodes are within one hop communication range.

When an agent fails while achieving a goal, the organization triggers a reorganization that results in another agent assigned to the failed goal or in another set of alternative goals assigned to the available agents. Hence, our first set of experiments attempted to see how the system recovers from sensors failures. We were interested in the failure of the monitor agents. Such failures are recognized by the organization when monitor agents fail to send beacon messages. When this happened, the organization tried to find another set of monitoring agents that can insure a total coverage of the surveillance area. Initially, the organization tried to find an agent that can cover the same area previously covered by the failed agent. When it was not possible, all the current monitor agents were de-assigned and replaced by a new set of monitor agents capable of covering the area. The adaptation of the system was measured in term of the percentage of the surveillance area covered by monitoring agents after various failure ratios. The coverage was computed as the average observed over 5 runs of 1000 simulation seconds. During the simulation, random failures of monitor agents were introduced at a constant rate (every 50 seconds) between 100 and 900 seconds. In addition, we measured the coverage obtained in a non-adaptive version of the system for which four agents were statically assigned to the monitor role at design time and were not replaced after a failure.

Figure 8 shows the results obtained for our adaptive system and for a non-adaptive version. These results show that even with 6 sensor failures at 350 seconds, our system still provides 100% coverage whereas the non-adaptive version cannot cover the area once all its four static sensors have failed (250 seconds). By the end of the simulation, the adaptive system has lost 70% of its sensors but still covers more than 65% of the area. These large coverage values with few sensors are due to the fact that on average, each sensor can cover 25% of the area. Nevertheless, the organization was able to reorganized in the face of failures and reassign the failed monitoring goals to available sensors in order to continue to insure a maximum coverage of the surveillance area.

Our second set of experiments intended to show how the system adapts to decreasing power levels. Like most surveillance systems, most of the energy is consumed during the surveillance phase, monitoring for potential targets [12]. Thus, the non-adaptive design with static monitoring agents leads to the complete energy depletion in the monitoring agents while the

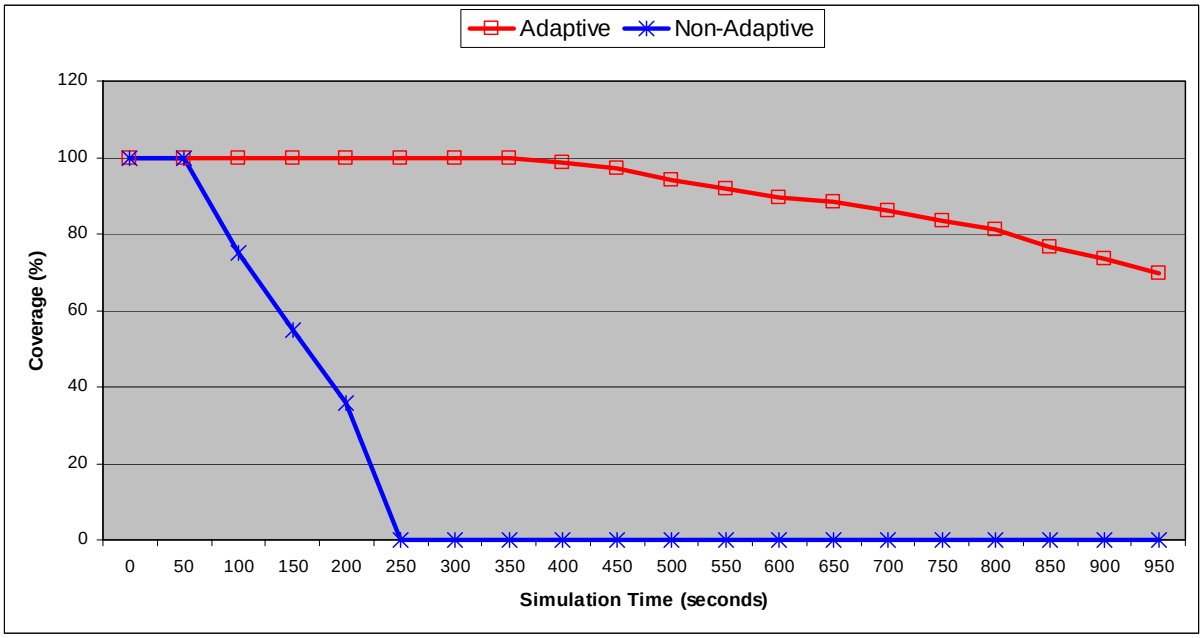

Figure 8. Coverage obtained by injecting a monitor failure every 50 seconds

remaining agents used almost no energy. Hence, we were interested in having the system adapt to maintain a uniform level of energy among all the nodes while trying to always insure a maximum coverage. For that, we introduced several guidance policies aiming at putting a lower bound on the energy required to play a monitor role. This allowed agents to give up the monitoring role when their energy dropped below a certain threshold. We defined three energy policies (EP) and one coverage policy (CP) ordered by least-importance.

- all monitor agents should have less than 20% of their energy used (EP20)
- all monitor agents should have less than 50% of their energy used (EP50)
- all monitor agents should cover 100% of the surveillance area (CP100)

As we explained earlier, the system satisfies all the policies as long as compliant assignments can be made. If no assignment can be found, the system deactivates the least important policy among all the active ones in order to proceed. Therefore, with EP20 active, every time a monitor agent energy usage rose above 20%, there was a reorganization aiming at finding another agent to play the monitor role. When there were no agents with less than 20% energy usage or those with less than 20% energy usage could not cover 100% of the area, the system

deactivated EP20. This process continued until all the policies were deactivated, in which case the system could assign monitoring roles without ensuring maximum coverage.

We compared the energy level of each agent at the end of a simulation with and without the policies. The system running without energy policies kept the same monitor roles throughout the entire simulation, whereas the system with policies behaved as indicated above. We used the number of radio messages sent as a measure of energy consumption, which is reasonable given that radio communication largely dominates energy on the motes [19]. Note that during the monitor or track roles, agents are constantly sending sensor readings and beacon messages back the base station.

Figure 9 shows the energy consumed for both systems. The results were observed over a run of 1000 simulation seconds with 3 targets appearing one at a time along the same path. Globally, there was a target present 20% of the time. In the non-adaptive version, agents 6, 9, 21 and 24 used more energy than any of the other agents did. This is because these agents were playing the monitor role during the entire simulation. On the other hand, we observed that the adaptive version kept the energy level uniform among all agents. In fact, even though both systems used 41% of their global energy, the standard deviation for the adaptive system was 12% whereas the one for the non-adaptive version was 28%. These results support the fact that our adaptive system was able to reorganize in order to maintain a uniform distribution of the

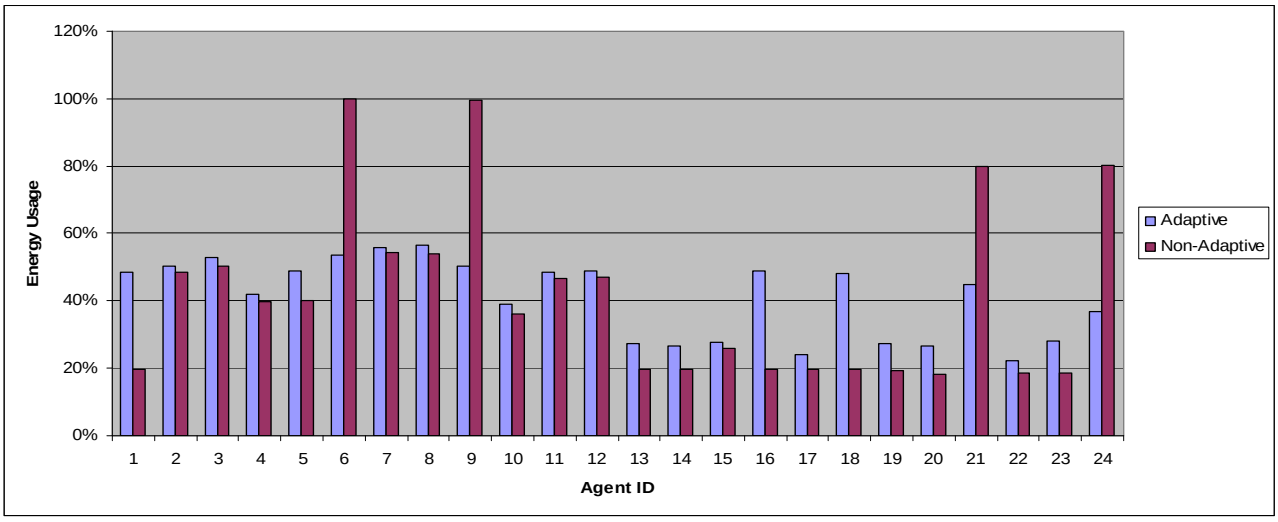

Figure 9. Energy Usage discrepancies between the adaptive and non-adaptive system

energy usage as directed by the policies.

6. Related Work

Several multiagent approaches have been proposed to develop sensor networks. A survey of MAS perspectives for WSN is available in [21]. However, most of the multiagent approaches do not consider an organizational design, which provide a better abstraction for MAS.

Organizational approaches for WSN have been used in [14, 20, 22]. Like ours, these approaches use organization mechanisms to manage sensor nodes. However, these approaches only deal with specific problems and do not specifically tackle the issue of adaptation. In addition, they do not follow a rigorous development process. Our organizational approach provides systems with enough knowledge to be able to self-organize and is supported by the O-MaSE process framework [9].

7. Conclusion

We presented the design and implementation of an adaptive surveillance application that uses wireless sensor nodes to collaboratively monitor and track all vehicles entering an area. Our design was based on a multiagent organizational design paradigm [5] that provides the sensor nodes with the required knowledge to self-organize. We developed designs models that capture important organizational concepts and implemented them into runtime models that were used to support adaptation decisions. The implementation was tested on a simulator in order to demonstrate the adaptive properties gained from developing applications using our organizational design approach. This implementation demonstrated the following adaptive properties:

- *Self-configuration*: The system was able to reconfigure itself when new goals appear in the organization in order to achieve this new goal.
- *Self-healing*: The system was able to reorganize to overcome the loss of a sensor.

8. Acknowledgements

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

9. References

[1] S. Coleri, S.Y. Cheung, and P. Varaiya. *Sensor networks for monitoring traffic*. in Allerton conference on communication, control and computing. 2004.

[2] Crossbow. *Wireless sensor networks (mica motes)*. [cited 2009; Available from: http://www.xbow.com/.

[3] S.A. DeLoach. *The agentTool III Project*. [cited 2008; Available from: http://agenttool.cis.ksu.edu/.

[4] S.A. DeLoach and M. Miller, *A Goal Model for Adaptive Complex Systems.* International Journal of Computational Intelligence: Theory and Practice, 2010. **5**(2).

[5] S.A. DeLoach, W.H. Oyenan, and E. Matson, *A capabilities-based model for adaptive organizations.* Autonomous Agents and Multi-Agent Systems, 2008. **16**(1): p. 13-56.

[6] M. Demmer, et al., *Tython: A Dynamic Simulation Environment for Sensor Networks*, U.o.C. EECS Department, Berkeley. 2005.

[7] V. Dignum, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. 2009: Information Science Reference.

[8] A. Estefania, J. Vicente, and B. Vicente, *Multi-Agent System Development Based on Organizations.* Electronic Notes in Theoretical Comp. Sci., 2006. **150**(3):p.55-71.

[9] J.C. Garcia-Ojeda, et al. *O-MaSE: A Customizable Approach to Developing Multiagent Development Processes*. in 8th International Workshop on Agent Oriented Software Engineering 2007.

[10] D. Gay, et al. *The nesC language: A holistic approach to networked embedded systems*. in PLDI '03: Programming language design and implementation. 2003. California.

[11] S. Harmon, S. DeLoach, and Robby, *Trace-Based Specification of Law and Guidance Policies for Multi-Agent Systems*, in *ESAW: Engineering Societies in the Agents World VIII*. 2008. p. 333-349.

[12] T. He, et al., *VigilNet: An integrated sensor network system for energy-efficient surveillance.* ACM Trans. Sen. Netw., 2006. **2**(1): p. 1-38.

[13] J. Hill, et al., *System architecture directions for networked sensors.* SIGPLAN Notice, 2000. **35**(11): p. 93-104.

[14] B. Horling, et al., *Using autonomy, organizational design and negotiation in a distributed sensor network*, in *Distributed Sensor Networks: A multiagent perspective*, V. Lesser; and Tambe, M. 2003, Kluwer Academic.

[15] P. Levis, et al. *TOSSIM: accurate and scalable simulation of entire TinyOS applications*. in SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems. 2003. Los Angeles, California.

[16] A. Mainwaring, et al., *Wireless sensor networks for habitat monitoring*, in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. 2002, ACM: Atlanta, Georgia, USA.

[17] W.H. Oyenan and S.A. DeLoach, *Towards a Systematic Approach for Designing Autonomic Systems.* Web Intelligence and Agent Systems (WIAS): An International Journal, 2010. **8**(1).

[18] J.S. Sandhu, A.M. Agogino, and A.K. Agogino, *Wireless sensor networks for commercial lighting control: decision making with multi-agent systems.* AAAI workshop on sensor networks, 2004: p. 131-140.

[19] V. Shnayder, et al. *PowerTOSSIM: Efficient Power Simulation for TinyOS Applications*. in ACM Conference on Embedded Networked Sensor Systems (SenSys). 2004.

[20] M. Sims, D. Corkill, and V. Lesser, *Automated organization design for multi-agent systems.* Autonomous Agents and Multi-Agent Systems, 2008. **16**(2): p. 151-185

[21] M. Vinyals, J.A. Rodriguez-Aguilar, and J. Cerquides. *A survey on sensor networks from a multi-agent perspective*. in Proceedings of the 2nd Int. Workshop on Agent Technology for Sensor Networks. 2008. Estoril, Portugal.

[22] H. Zafar, et al., *Using organization knowledge to improve routing performance in wireless multi-agent networks*, in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems* . 2008.