Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT '10). Toronto Canada, September 2010.

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 are no longer adequate. Thus, it is crucial that these systems be adaptive and autonomous to remain functional in the face of unreliable communications, dead nodes, and other unexpected failures. We propose to manage sensor networks based on a rigorous multiagent organizational design, which separates 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.

1. Introduction

Wireless sensor networks (WSN) have been used in many applications to obtain information about the environment. 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 complex and challenging task. Consequently, ad-hoc techniques for developing this kind of systems are no longer adequate.

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 [16]. Hence, sensors can be viewed as autonomous agents that collaborate to achieve a common goal.

To design organization-based multiagent systems, we base our work on the Organization Model for Adaptive Computational Systems (OMACS) [4] as its capability orientation makes it particularly suitable to heterogeneous sensors. OMACS defines the concepts required to provide the agents with the knowledge to self-organize. In addition, OMACS allows designers to define application events that cause the system to reorganize. For instance, if an event of interest is the appearance of a vehicle, the organizational knowledge would define the right goal to pursue in response to it.

Moreover, OMACS is supported by a rigorous process based on Organization-based Multiagent Engineering [6]. In this paper, we follow a similar process to capture the important organizational concepts. These concepts are implemented in runtime models that are the basis for the system's adaptive reasoning.

Our design is based on two O-MaSE design models, the Goal and Role models, and a set of policies, all of which are presented in Section 2. In Section 3 we introduce the system architecture. In Section 4, we examine how agents are assigned to roles and goals at runtime. Finally, we evaluate our approach 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 [5]. In this work, we use the organizational concepts defined in the OMACS model [4]. OMACS defines an *organization* as a set of *goals* that need to be accomplished, a set of *roles* that must be played to achieve these goals, a set of *capabilities* required to play these 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 design follows the multiagent systems development process proposed in [12], which is built based on method fragments from the O-MaSE framework [6]. Although the methodology introduced in [12] 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 that consists of AND/OR decompositions of goals along with a trigger relationship that allows goals to be instantiated with application-specific parameters. At runtime, the organization goals

° 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).

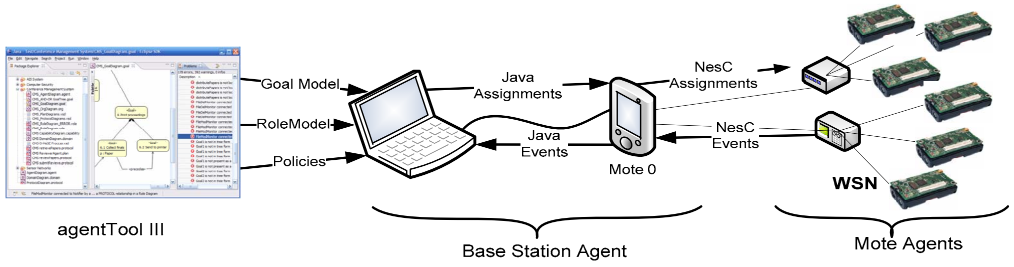

Figure 1. Overall System Architecture

change dynamically as new goals are created or existing goals are achieved. Then, we identify the roles that describe the high-level behavior required to achieve particular goals. These roles are organized into a role model, which captures interactions (protocols) between roles and the capabilities required for each role. Next, we specify reorganization policies to guide the system when assigning the goals to agents [8]. These policies typically specify the types of assignments the system should prefer or avoid. Finally, agents are designed to play roles.

Design models are created using agentTool III (aT^3), a multiagent development environment [2]. aT^3 supports the development and validation of design models that are automatically translated into runtime models.

3. System Architecture

Once the organization design models are defined, we need to define the agents that will be participating in the organization. We have two types of agents: *Base Station agents* and *Mote agents*. In general, a 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 [1].

The overall system architecture is presented in Figure 1. 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. Therefore, in our system, the Base Station is the only agent that possesses the organizational knowledge and decides the next configuration of the organization. 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. Details of the agent architecture are given in [13].

4. System Implementation

At runtime, the system cycles through four main phases: update goals, make assignments, play roles, propagate events. 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, we use a greedy algorithm as shown in Figure 2. For each unassigned goal (line 1), we get the first role that can achieve it (line 5) and the first agent that has the required capabilities (line 8). The assignment produced is checked for policies compliance (line 11). If it fails, the *passPolicies* method removes the assignment that caused it to fail and a new assignment is

```
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)
12.    assignmentSet.add(assignment)
13.  end loop
14.  return assignmentSet
```

Figure 2. Assignment Algorithm

sought. If no valid assignment exists, the least important policy is deactivated and we try again. Our policies are *guidance policies* [8] that guide the system towards a desired behavior without constraining it. They can be abandoned if they prevent the system from progressing. Policies are ordered by importance, allowing the system to ignore the least important policies when needed.

5. Example

To show the application of organizational design to sensor networks, we implemented a familiar surveillance application, where sensors collaboratively monitor and track vehicles entering an area. Sensors have limited sensing range and must collaborate to cover the area of interest. The sensors also have limited energy.

In the application, 25 sensors were arranged in a 5 x 5 grid covering 100 ft^2. Mote agents were implemented in nesC [7] on mote-based sensors [1] running the TinyOS [10]. The Base Station agent was implemented in Java. We used TOSSIM [11] to emulate the mote code execution; TOSSIM also supports inserting mote failures and simulated targets. Targets were implemented as magnetometer sources that affect the magnetometer readings of nearby sensors. We assumed reliable one hop communication. We also developed a non-adaptive version of the system for comparison purposes.

When agents fail, reorganization is triggered resulting in a new agent assigned to the failed agent's goal or in a set of alternative goals being assigned to the available agents. Thus, our first set of experiments attempted to see how the system recovers from failures of agents in charge of covering the surveillance area. When such failures happened, the organization tried to find another set of agents to ensure total area coverage. System adaptation was measured by the percentage of the surveillance area covered by sensors after various failures. The coverage was computed as the average observed over 5 runs of 1000 simulation seconds.

Figure 3 shows the results obtained for the adaptive and non-adaptive systems. With up to 6 sensor failures, the adaptive system provided 100% coverage whereas the non-adaptive system failed to cover the entire area as soon as a sensor failed. By the end of the simulation, the adaptive system lost 70% of its sensors but still covered more than 65% of the area. The organization was able to reorganize in the face of failures and reassign the failed goals to available sensors to continue to get maximum coverage.

Our second set of experiments shows 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 [9]. Having a non-adaptive design with static monitoring agents leads to the complete energy depletion

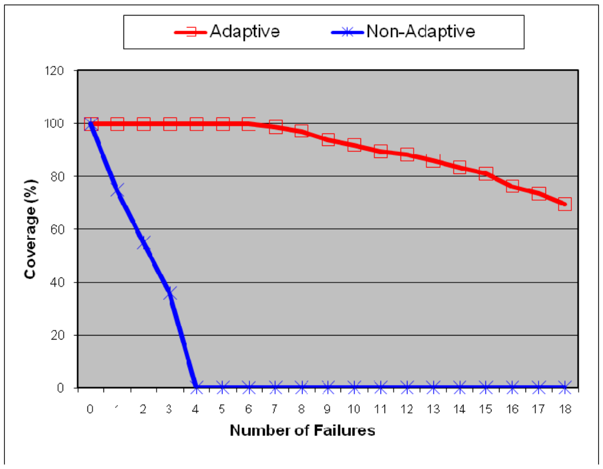

Figure 3. Coverage obtained with agent failures

for monitoring agents, while non monitoring agents use little energy. Hence, we were interested in having the system adapt to maintain a uniform level of energy among sensors while maintaining maximum coverage. To accomplish this, we introduced several guidance policies that allowed agents to give up monitoring when their energy dropped below a given threshold.

We compared the energy level of each agent at the end of a simulation with and without policies. Without policies, the system kept the same monitor agents throughout, whereas the system with policies behaved as indicated above. We used the number of messages to measure energy consumption, which is reasonable given that communication dominates mote energy use [14].

Figure 4 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 other agents. This is due to the fact that 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 to maintain a uniform distribution of the energy used 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 [16]. However, most of the multiagent approaches do not consider an organizational design, which provide a better abstraction for MAS.

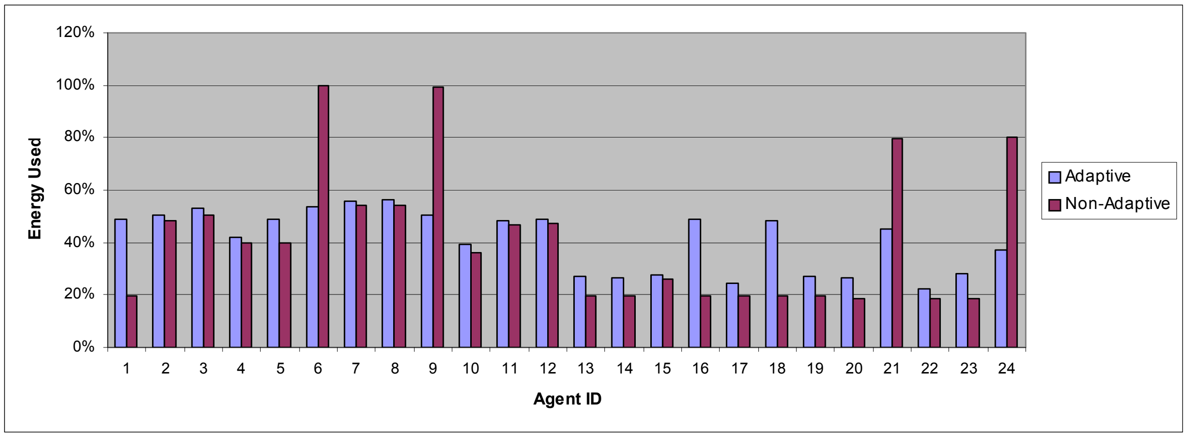

Figure 4. Energy Used discrepancies between the adaptive and non-adaptive system

Organizational approaches for WSN have been used in [15], [17]. 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 [6].

7. Conclusion

We presented 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 [4] that provides the sensor nodes with the required knowledge to self-organize. The implementation was tested on a simulator to demonstrate the adaptive properties gained from developing applications using our organizational design approach. This implementation demonstrated both *self-configuration* and *self-healing* properties.

8. References

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

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

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

[4] 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.

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

[6] 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.

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

[8] 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.

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

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

[11] 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.

[12] 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): p.79-97.

[13] W.H. Oyenan and S.A. DeLoach, and G. Singh. *Designing Adaptive Sensor Networks Using an Organization-based Approach*. MACR Lab Technical Report No. MACR-TR-2010-04. Kansas State University. June, 2010.

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

[15] 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.

[16] M. Vinyals, J.A. Rodriguez-Aguilar, and J. Cerquides, *A survey on sensor networks from a multi-agent perspective*. The Computer Journal, 2010: p. Advance Access, doi:10.1093/comjnl/bxq018.

[17] H. Zafar, et al., *Using organization knowledge to improve routing performance in wireless multi-agent networks*, in 7th intl. joint conference on Autonomous Agents and Multiagent Systems. 2008: Estoril, Portugal.