Design and Evaluation of a Multiagent Autonomic Information System[°]

Walamitien H. Oyenan and Scott A. DeLoach
Department of Computing & Information Sciences, Kansas State University
{oyenan, sdeloach}@ksu.edu

Abstract

The goal of an autonomic system is to self-manage itself and adjust its actions in the face of environmental changes. In this paper, we adopt a multiagent approach to developing an Autonomic Information System. The aim of this Autonomic Information System (AIS) is to provide an information system that can adjust its processing algorithms and/or information sources to provide required information at various levels of efficiency and effectiveness. Our approach to developing autonomic multiagent systems is based on the Organization Model for Adaptive Computational Systems. We describe the design of one particular autonomic system, the AIS, and illustrate how this system fulfills certain desired autonomic properties. We also evaluate the performance of our autonomic system by comparing it to a non-autonomic system.

1. Introduction

The goal of autonomic computing is to create new systems that are able to manage themselves. This requires such systems to have the ability to self-configure, self-optimize, self protect, and self-heal [4, 5, 6, 10]. They must adapt to environmental changes and strive to improve their performance over time [5]. As systems become increasingly complex, they are expected to handle this complexity on their own. Therefore, it is crucial that they exhibit autonomic behavior. In this paper, we adopt a multiagent approach to developing an Autonomic Information System as agents are autonomous and map naturally to the autonomic computing principles.

The goal of our Autonomic Information System is to provide an information system that can adjust its processing algorithms and/or information sources to provide required information at various levels of efficiency and effectiveness. In this system, various types of sensors at different locations are used to detect enemy vehicles. These sensors are subject to failure and erroneous outputs and typically have a delay in getting the information categorized. When sensor data of interest is available, it is fused with other related information to answer queries from the commander. A field commander uses the system interface to generate two types of queries: transient and persistent. *Transient queries* are executed only once whereas *persistent queries* are carried out repeatedly until canceled. To overcome the loss of sensors and continue to provide the required information, the Autonomic Information System (AIS) needs to adapt by replacing the failed sensors and adapting the information processing adequately without the intervention of the user.

The purpose of this paper is to design an autonomic system based on the Organization Model for Adaptive Computational System (OMACS), which provides the system with enough knowledge to be able to self-organize. OMACS is used to model the system we want to develop. Once the system has been modeled, we use our proposed generic agent architecture to create a Multiagent Autonomic System. We demonstrate one particular autonomic system, the AIS, and illustrate how it realizes autonomic behavior.

The remainder of this paper is organized as follows. First, we present other work related to autonomic multiagent systems in Section 2. Next, we present our AIS multiagent organization in Section 3 and 4 followed by the system architecture in Section 5. In Section 6, we show the autonomic properties of the AIS via a scenario. A performance evaluation of the system is given in Section 7 while Section 8 concludes and discusses future work.

2. Related Work

There have been several multiagent approaches used to build autonomic systems. Bigus et al. [12] describes a set of agent component libraries that can be used to build autonomic systems, which extend the ABLE platform by adding external agents to manage and control the system. In [15], Tesauro et al. presents an architecture in which agents have predefined responsibilities which allow the system to exhibits certain autonomic behaviors. In [13], Kumar and Cohen describe an adaptive agent architecture in which broker agents share the same knowledge of the system and are thereby aware of any agent failure. Sterritt and Bustard [14], propose design template in which agents monitor the system using heartbeat signals.

All these works are similar to ours in the sense that agents have system-level knowledge that allows them to manage themselves in unpredictable environments. However, in our approach, instead of embedding this system-level knowledge in the architecture, we design

[°] 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).

systems based on the underlying OMACS model, which allows us to reuse the agent architecture and systematically develop autonomic applications. Actually, we are also in the process of developing an autonomic sensor network application using the ideas outlined in this paper. In addition, we have developed the Organization-based Multiagent Systems Engineering methodology [2] to support the development of OMACS-based systems.

3. Overview of OMACS

OMACS [2] is a metamodel for agent organizations. It defines the required organizational structure that allows multiagent teams to autonomously reconfigure at runtime, thus enabling them to cope with unpredictable situations in a dynamic environment. Specifically, OMACS specifies the type of knowledge required for a multiagent system to know itself and be able to reason about its own status. Hence, multiagent teams are not limited by a predefined set of configurations and can have the appropriate information about their team, enabling them to reconfigure in order to achieve their team goals more efficiently and effectively. During the design of an OMACS-based system, the designer only provides high-level guidance about the organization, which will then be able to self-configure based on the current goals and team capabilities. These characteristics make OMACS ideal for designing autonomic multiagent systems. OMACS defines an organization as a set of goals (G) that the team is attempting to accomplish, a set of roles (R) that must be played to achieve those goals, a set of capabilities (C) required to play those roles, and a set of agents (A) who are assigned to roles in order to achieve organization goals. At runtime, the assignments of agents to play roles to achieve goals represent the key functionality that allows the system to be autonomic.

4. The AIS Organization

Our Multiagent Autonomic System is an organization-based multiagent system [1] built upon OMACS. Hence, to implement the system, we need to design the goals, roles, capabilities, and agent types defined in OMACS.

4.1. Goals

Goals are a high level description of what the system is supposed to be doing [9]. We use a goal model to represent system-level goals; this model includes the goal definitions, goal decomposition, and the relationship between the goals and their subgoals, which are either conjunctive or disjunctive [7]. Typically, each organization has a top-level goal that is decomposed into refined subgoals. Eventually, this top-level goal is refined into a set of leaf goals that will be actually pursued by the organization. The set of all organizational goals is denoted as G. The *active goal set*, G_a, is the current set of

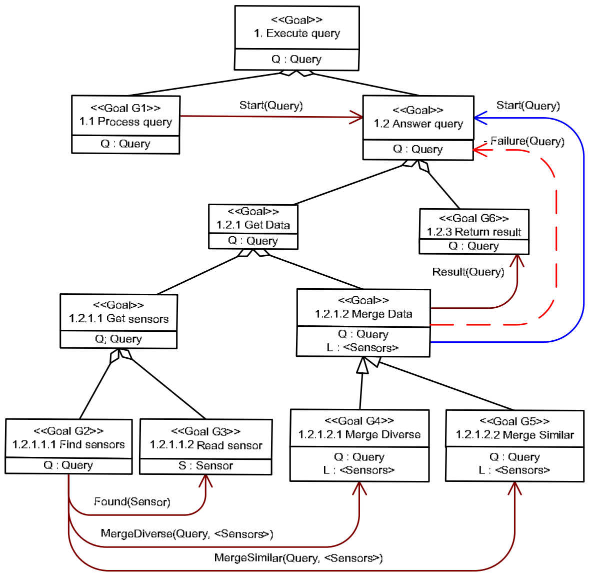

Figure 1. AIS Goal Model

goals that an organization is trying to achieve. G_a changes as new goals are inserted or existing goals are achieved.

To capture the dynamic nature of OMACS-based systems, there is a time-based relationship that exists between goals. We say goal g1 precedes goal g2, if g1 must be satisfied before g2 can become active. This allows the organization to work on one part of the goal tree at a time. During the pursuit of specific goals, events may occur that cause the instantiation of new goals. Instantiated goals may be parameterized to allow a context sensitive meaning. If an event can occur during pursuit of goal g1 that instantiates goal g2, we say g1 triggers g2.

The main goal of the application is to answer each query presented to the system. From the requirements, we have derived the goal tree structure presented in Figure 1. The boxes represent the goals and their parameters. The arrows indicate triggers and their parameters. The dashed arrows represent *negative triggers*, which allow for the cancellation of a goal and all its subgoals. As the queries are not predefined, most of the goals in the systems will be triggered based on the query. These parameterized goals will be instantiated and associated to specific queries. Conjunctive sub-goals are connected to their parents by a diamond shaped connector (◊) while disjunctive sub-goals are connected to their parent by a triangle shaped connector (Δ). The leaf goals that the organization must achieve are goal G1 to goal G6.

4.2. Roles

Roles are a high level description of how to achieve some particular goals [3]. In OMACS, each organization has a set of roles that it can use to achieve its goals. The *achieves* function, which associates a score between 0 and 1 to each <goal, role> pair, tells how well that particular role can be used to achieve that goal (1 being the maximum score). In addition, each role requires a set of capabilities, which are inherent to particular agents. Agents must possess all the required capabilities to be considered as a potential candidate to assume that role.

For each goal in this organization, we have created a role that can achieve it. Following are the roles we have defined for the AIS organization, along with the goals they can achieve and a description of their behavior.

R1 – Query Processor (G1): Periodically gets new queries from the user. The role triggers an event to notify the system that a new query has been entered.

R2 – Sensors Locator (G2): Query the sensor database in order to find all sensors available in the area specified by the parameter of the goal it achieves. Then it executes an algorithm to find the best coverage based on the set of available sensors. For each sensor selected, an event is triggered (event '*found(S:Sensor)*'). This event will result in the organization attempting to find an agent capable of reading the selected sensor. After all sensors have been selected, the role triggers an event (event '*mergeSimilar*' or '*mergeDiverse*') to notify the organization that it has found all sensors capable of providing data for the query.

R3 – Sensor Reader (G3): Read the data from the sensor given in parameter of the Read Sensor goal. This role interacts with the battlefield simulator in order to get the appropriate data. The data will be sent to any agents (typically a data merger agent) interested in those data.

R4 – Data Merger Diverse (G4): Merge the data collected from various sensors covering the area of interest. This role uses a processing algorithm that allows it to merge data coming from sensors of different type.

R5 – Data Merger Similar (G5): Behave like the previous role. However, as oppose to the previous role, this role uses a processing algorithm that allows it to efficiently merge data coming from sensors of the same type. Thus, this role will not be able to process data from different sources.

R6 – Result Interface (G6): Return the results of the query to the GUI for displaying to the user.

4.3. Capabilities

In OMACS, capabilities are fundamental in determining which agents can be assigned to what roles in the organization [8]. Agent may possess two types of capabilities: hardware capabilities like actuator or effectors, and software capabilities like computational algorithms or resources usage.

The capabilities identified for the AIS and the roles that require them are listed below.

C1 – User Interaction (R1, R6): Used to interact with the GUI.

C2 – Coverage Processing (R2): Used to compute the optimal set of sensors that has the maximum coverage of the area of interest and that can satisfy the efficiency and accuracy constraints.

C3 – Sensor Interaction <sensor> (R3): Used to interact with the actual sensors on the battlefield.

C4 – Data Merging Diverse (R4): Provide computational algorithms to merge data coming from diverse type of sensors.

C5 – Data Merging Similar (R5): Provide fast computational *algorithms* to merge data coming from similar sources only.

C6 – Coordination (R3, R4, R5, R6): Provide the ability to communicate with other agents.

For simplicity, we designed our system so all capabilities equally are important in accomplishing a role.

4.4. Agents

The agents represent the autonomic elements of the system. Within an OMACS organization, agents have the ability to communicate with each other, accept assignments to play roles that match their capabilities, and work to achieve their assigned goals. Each agent is responsible for managing its own state and its interactions with the environment and with other agents. Once the system provides goals and roles, the agents are expected to determine themselves what behavior is necessary to fulfill them. For that, a plan on how to play a role needs to be provided at design time either by the role or the agent designer. In OMACS, a tuple <a,r,g> represents the assignment of agent a to play role r in order to achieve goal g. The assignment set Φ represents the set of all the current assignments in the organization. To capture a given agent's capabilities, OMACS defines a possesses function, which maps each <agent, capability> pair to a value between 0 and 1, describing the quality of the capability possessed by an agent (1 representing the maximum quality).

We define a set of agent types capable of playing at least one role in the organization. To simplify the application, we assume that an agent either possesses a capability or not (possesses score is either 1 or 0). The agent types and their capabilities are listed below.

QA – Query Agent (C1)
SFA – Sensor Finder Agent (C2)
DSA – Data Sensor Agent (C3, C6)
MAD – Merger Agent Diverse (C1, C4, C6)
MAS – Merger Agent Similar (C1, C5, C6)

5. Generic Agent Architecture

In this section, we present the generic agent architecture of the AIS agents, which represent the autonomic elements of our system [6]. As Figure 2 shows, an AIS agent typically consists of two components: the *Execution Component* (EC) and the *Control Component* (CC).

The EC represents the non-autonomic part of the agent. It corresponds to the application specific part of the agent. It is notified by its CC about what role to play in

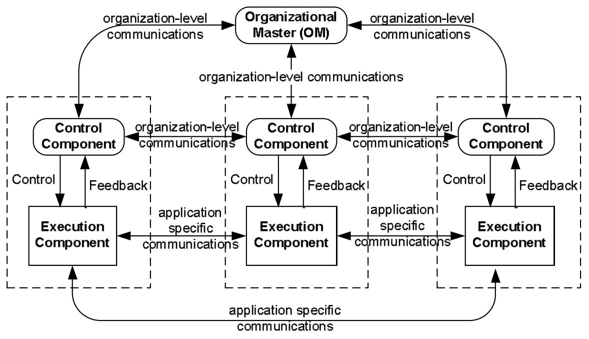
organization-level communications
Organizational Master (OM)
organization-level communications
organization-level communications
Control Component
organization-level communications
Control Component
organization-level communications
Control Component
Control
Feedback
Control
Feedback
Control
Feedback
Execution Component
application specific communications
Execution Component
application specific communications
Execution Component
application specific communications

Figure 2. Generic Agent Architecture

the organization. Once it has been assigned a role, the EC plays that role according to a predefined plan provided at design time either by the role or the agent designer.

The CC represents the autonomic part of the agent. In general, the more sophisticated this component, the more autonomy the system will display. Typically, the CC is an intelligent component in charge of all organization related tasks. Depending on design strategies, it can have a partial or total knowledge of the organization structure. We have design a fixed communication interface between the CC and the EC. This gives us the flexibility to plug several different CCs designs into our application without having to modify other application-specific components (the ECs). Therefore, Control Components are generic and can be reused for any other autonomic applications as long as the communication interface is respected.

In general, a CC operates based on its knowledge and the information collected from other agents via their CCs. It can decide to reconfigure the organization by including or canceling goals in the organization, or by modifying the current assignments. This reconfiguration process can be distributed or centralized. A distributed reconfiguration would involve a deliberation process between all the CCs in order to reach a consensus about the next state of the organization. However, in our current implementation, we have opted for a centralized approach in which all CCs report to one particular CC that will then have all the knowledge to make appropriate decisions. To differentiate with other CCs, we call this particular CC the Organization Master (OM). Therefore, the OM possesses all the organizational knowledge and is in charge of all the organization-related tasks (Figure 2). The OM reasons about the current state of the organization and once it reaches a decision about a new configuration, it notifies all the CCs that are affected by this reconfiguration. This autonomous reasoning is based on the underlying OMACS architecture and results in a reconfiguration which is fundamental in achieving the autonomic properties described in the following section.

6. Autonomic properties of the AIS

In this section, we present a scenario that exemplifies some of the autonomic behaviors of the AIS. To adapt to a variety of unpredictable situations, our AIS organization is able to detect changes in the performance of the overall organization (self-monitoring) and modify its structure accordingly (self-adjusting). Many of them are changes within the environment; however, some changes occur within the organization itself (e.g., capability failure or goal completion). Hence, the AIS is not only aware of its environment but it is also aware of its own state (self-aware). These self-* properties of the AIS are facilitated by our use of OMACS, which provides all the necessary knowledge for a self-managing system.

6.1. AIS Scenario

To demonstrate the AIS system, we use a simulated battlefield with sensors and enemy targets. In our battlefield simulator, there are five different types of vehicles that the system is trying to locate and identify: truck, halftrack, tank, artillery, and launcher. For the specific scenario described in this paper, we have defined two types of sensors: ground sensors and airborne automatic target recognition (ATR) sensors. The screenshot in Figure 3 shows the simulated battlefield along with the sensors and enemy targets. There are four ground sensors (S1, S2, S3, S4) and one ATR sensor (S5). There are also five enemy vehicles. We assume that the system is only trying to answer one persistent query and

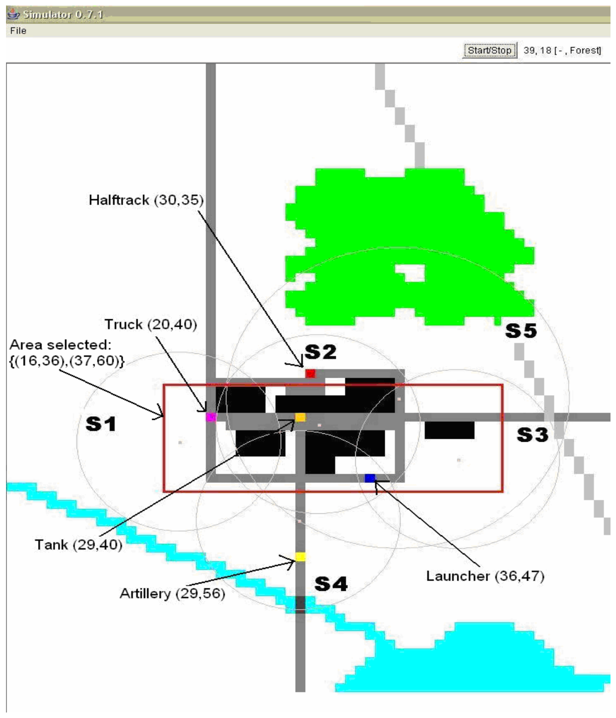

Figure 3. Battlefield Map

omit the query parameter for goals and triggers. The persistent query is: "*Show the location and type of all enemy vehicles in the selected area*" (the area selected is defined by a rectangle in Figure 3).

6.2. Initialization

For this scenario, we have created one agent for each agent type except for the agent type DSA for which one DSA agent has been created for each sensor in the battlefield. Agents are named after their types and DSA agents are numbered to match their sensor number. At initialization, all the goals that have no predecessors and do not require any triggers are inserted in G_a (the active goal set) and thereby pursued by the organization. Based on the goal model (Figure 1), only goal G1 is active initially. Once G1 is active, the OM chooses the best role to achieve G1. R1 will be chosen to achieve G1 because the pair <G1,R1> has the highest achieves score. Then the organization chooses the Query Agent QA to play R1. This choice is motivated by the fact that A1 possess all the required capabilities to play R1. Figure 4 shows the successive states of the organization after the occurrence of events. States contain the active goal set (G_a) and the set of assignments (Φ). The initial assignment we have just described corresponds to State 1 in Figure 4.

Once the QA retrieves the query from the GUI, it triggers an event *start(Query)*. This trigger results in the activation of G2. Upon this activation, the system must reconfigure itself to achieve this new active goal. By taking the best role and agent to achieve goal G2, the Sensor Finder Agent (SFA) is assigned to play role R2 to achieve goal G2 (State 2).

When the query has been retrieved, the QA agent terminates by sending an *achieved* message to the OM, which causes the goal and its related assignment to be removed from G_a and Φ. Next, the SFA agent chooses sensors S1, S2, S3 for the query as those sensors maximize the area of interest coverage. The following events are then triggered: *found(S1)*, *found (S2)*, *found(S3)*. Each *found* event instantiates a parameterized goal G3 having the parameter of the trigger. In our case, goals G3(S1), G3(S2), and G3(S3) become active, which again requires a reconfiguration with (DSA1,R3,G3(S1)), (DSA2,R3,G3(S2)), and (DSA3,R3,G3(S3)) being inserted into Φ (State 3).

As all the sensors found by the SFA agent are the same type, an event *mergeSimilar(<S1,S2,S3>)* is triggered, which results in the activation of the parameterized goal G5(<S1,S2,S3>). After computing the best assignment, the assignment of the Merger Agent Similar (MAS) to play role R5 to achieve G5 is chosen (State 4).

Once all the events have been triggered, the SFA agent notifies the OM that it has successfully completed its role by sending an *achieved* message. At this point, the MAS agent starts getting data from the DSA agents and merges

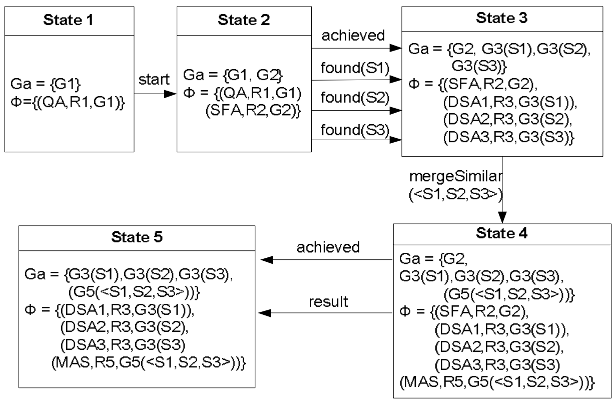

Figure 4. Organization States

them to extract the necessary information. When the results are ready, the MAS agent will then trigger a *result* event, which will result in the activation of G6. As the MAS agent has the capability to interact with the GUI, it plays role R6 to achieve goal G6 and sends the results to the GUI (State 5). When a query update is required, the MAS agent coordinates with DSA agents to get new data.

The results report a coverage of a 100% of the area of interest and the system effectively detected all three targets in the selected area: Tank at 29,40, Truck at 20,40 and Launcher at 36,47.

This scenario shows an important part of the autonomic system: self-configuration. The system is able to reconfigure itself when new goals appear in the organization. Every newly activated goal in the organization requires the AIS to take action in order to achieve this new goal. Our autonomic system is also capable of self-optimizing in the case of a goal completion. The achievement of a goal can free an agent to take on a new role and goal assignment. When this occurs, the organization may make new assignments in order to optimize the performance of the system.

6.3. Sensors Failure

The AIS simulator allows us to fail specific sensors. If we make S2 fail, the corresponding DSA agent (DSA2), which is the only agent capable of playing R3 to achieve G3(S2), can no longer achieve its goal. As a result, the MAS agent, which was coordinating with the DSA2 agent to gather the data, interrupts its task and generates a negative trigger *failure* and a trigger *start(query)*. The negative trigger causes all the goals related to that query to be removed, resulting in the cancellation of all their current assignments. Thus, goals G3(S1), G3(S2), G3(S3), G5(⟨S1, S2, S3⟩) are all removed. The *start(query)* event causes the activation of a new instance of goal G2 that will be achieved by the SFA agent playing role R2. Taking into account the loss of capability of the DSA2 agent, the SFA agent selects sensors S1, S3, S5 as the new optimal set of sensors for the query. The SFA agent then triggers the following events: *found(S1)*, *found(S3)*, *found(S5)* which result in the activation of goal G3(S1), G3(S3), and G3(S5). As this set of sensors contains sensors of different types (S1, S3 are ground sensors whereas S5 is an ATR sensor), the SFA agent triggers an event *mergeDiverse(<S1, S3, S5>)*, which results in the activation of goal G4(⟨S1, S3, S5⟩).

To achieve this new goal, the system chooses role R4, which is played by the Merger Agent Diverse (MAD). Then, the SFA sends an *achieved* message to the OM and terminates. The system then continues its execution as described in the previous section, except that the merger in charge of the query is now the MAD agent. As the coverage provided by the new set of sensors is also 100%, the AIS detects all the enemies in the area of interest: Tank at 29,40, Truck at 20,40 and Launcher at 36,47.

Therefore, by effectively detecting the sensor that failed and replacing it, the AIS has demonstrated its self-healing capability. The failure triggered a reconfiguration of the system to allow sensor reading tasks to be redistributed among all the DSA agents still operational and capable of covering the area of interest. Through this reconfiguration, the DSA5 agent has replaced the DSA2 agent that failed. In addition, during this scenario, an illustration of the AIS self-optimizing behavior has also occurred when the AIS organization has decided to replace the MAS agent, which was merging the data prior to the failure, by the MAD agent in order to insure a better performance. Hence, even though a loss of a sensor used to provide information for the query has occurred, the AIS was able to reconfigure itself and maintain the flow of information without the intervention of the user.

7. Experimental Results

To evaluate the performance of our autonomic system, we performed 100 persistent queries executed 20 times each. We selected the number of sensors failures which were injected in the system almost simultaneously two minutes after system initialization.

Our first sets of experiments attempted to evaluate the system robustness. We created a battlefield containing 40 sensors at fixed locations and 100 merger agents. The sensors were placed such that each point in the area of interest was covered by at least two sensors. For each run, we submitted 100 persistent queries and the experiment ran until each query had been executed 20 times. Experiments were run on both our autonomic system and a non-autonomic system that we simulated. Basically, once the queries were submitted, both systems chose the optimal set of Data Sensor Agent (DSA) to provide maximum coverage. However, when a DSA failed, the non-autonomic system ignored the failure and continued to provide results whereas the autonomic system reconfigured. The robustness of the system was measured

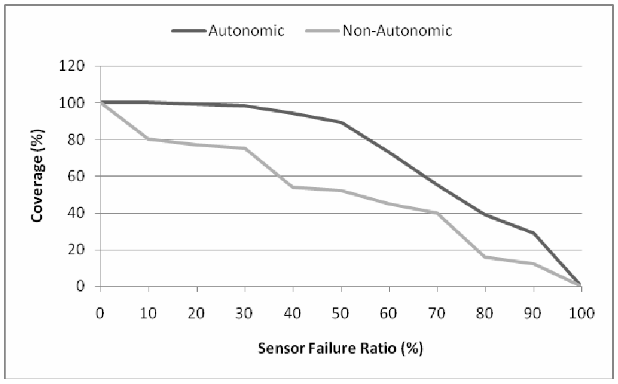

Figure 5 : Comparison of area coverage

in terms of the average coverage obtained at the end of the experiment.

As expected, the results, as shown in Figure 5, clearly indicate that for any failure rate, the autonomic system was significantly more robust. Observe that after 30% failure rate, the autonomic system was still able to provide a coverage of 100% whereas this coverage had dropped to 75% for the non-autonomic system. However, when too many sensors were lost (around 90%), the coverage provided by both system was very close as the autonomic system could not find any other sensors to replace the failed sensors. On average, the autonomic system achieved 25% more coverage than its counterpart.

In our second set of experiments, we were interested in characterizing the cost of reconfiguration in term of the number of messages sent by the agents. As in the first experiment, we used 40 data sensor agents and 100 merger agents trying to answer 100 persistent queries. The results in Figure 6 show that for the non-autonomic system, the number of messages decreased as the failure rate increased. This is due to the fact that as more sensors fail, the system interacts with less data sensor agents resulting in fewer messages. For the autonomic system, we expected a monotonically increasing function due to the messages involved in an increasing number of reconfigurations. However, the number of messages only

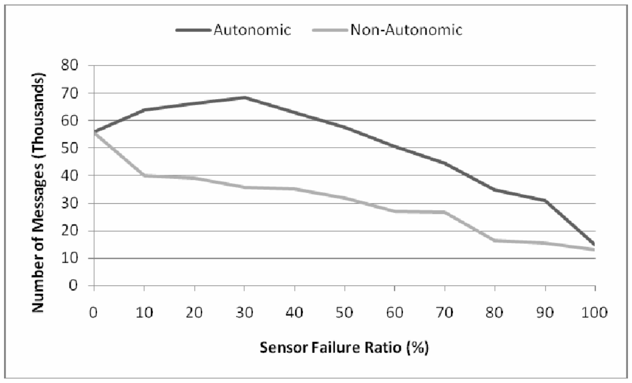

Figure 6 : Comparison of number of messages sent

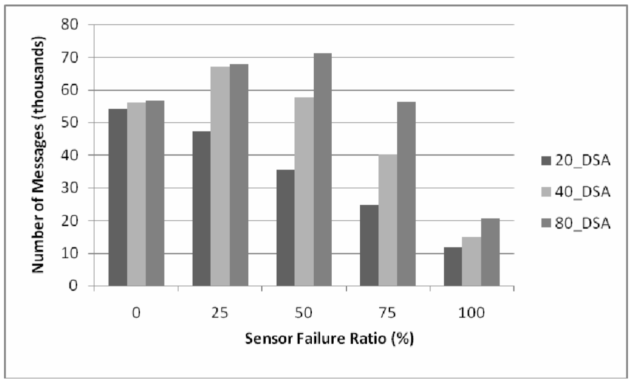

Figure 7 : Impact of data sensor agents numbers

increased until we reached a failure rate of 30% and then decreased. These results suggest that before the 30% failure rate, the number of reconfiguration messages was larger than the number of messages generated by sensors prior to their failure. As more sensors are lost (30% failure rate and larger), the increase in messages due to reconfiguration is completely hindered by the reduction in messages due to the decreased number of sensor interactions, globally resulting in less messages sent in the system. Overall, the autonomic system generated approximately 50% more messages.

Our final set of experiments attempted to confirm the influence of the number of data sensor agents on system performance. We ran the previous experiments with 3 different numbers of data sensor agents: 20, 30 and 80. For each set, we measured the number of messages sent at different failure rates. The results are shown in Figure 7. We note that there is a direct correlation between the number of data sensor agents and the number of message sent. However, as the number of data sensor agents doubles, the number of messages only increases by a constant factor. This is an important result which ensures an effective scalability of our system.

8. Conclusions and Future Work

We have described a multiagent approach for building an Autonomic Information System. Our approach takes advantage of OMACS, which fits well in the autonomic computing perspective and allows autonomic systems to achieve many self-* properties. Through a specific scenario, we have illustrated the self-configuring, self-optimizing, self-healing and self-protecting properties of our autonomic information system. As a result of these autonomic properties, the AIS can reason about system goals, recover from failures, recognize and prevent undesirable behaviors and optimize performance without disrupting the flow of information and requiring the intervention of the user. Finally, we have shown experimentally that our autonomic information system was more robust than a non-autonomic version.

In future work, we are investigating ways to allow a distributed reconfiguration, as oppose to the centralized approach of this paper. This would allow all the agents to participate in the reconfiguration process, resulting in a more robust autonomic system.

9. References

[1] P.M Blau, and W.R Scott, “Formal Organizations”, Chandler, San Francisco, CA, 1962, 194-22.

[2] S.A. DeLoach, W.H. Oyenan, “An Organizational Model and Dynamic Goal Model for Autonomous, Adaptive Systems,” Multiagent & Cooperative Robotics Laboratory TR MACR-TR-2006-01. Kansas State Univ. March 2006.

[3] J. Ferber, O. Gutknecht, C.M. Jonker, J.P. Müller, and J. Treur, “Organization Models and Behavioral Requirements Specification for Multi-Agent Systems”, *Proc. of the 10th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Lecture Notes in AI, Springer, 2002.

[4] A.G. Ganek and T. A. Corbi, “The dawning of the autonomic computing era”, *IBM Systems Journal*, Vol 42, No 1, 2003, pp. 5-18

[5] P. Horn, “Autonomic computing: IBM perspective on the state of information technology”, *IBM T.J. Watson Labs*, NY, 2001 (http://www.research.ibm.com/autonomic/)

[6] J. O. Kephart and D. M. Chess, “The vision of autonomic computing”, *Computer*, 36(1):41–50, 2003.

[7] A. van Lamsweerde, R. Darimont, E. Letier, “Managing conflicts in goal-driven requirements engineering”, *IEEE Trans on Software Engineering*. 24(11), pp 908-926, 1998.

[8] E. Matson, S. DeLoach, “Capability in Organization Based Multi-agent Systems”, *Proceedings of the Intelligent and Computer Systems Conference*, 2003.

[9] Russell, S. and Norvig, P. “Artificial Intelligence a Modern Approach”, *2nd Ed. Pearson Education, Inc*., 2003.

[10] R. Sterritt, “Towards Autonomic Computing: Effective Event Management”, *Proceedings of the 27th Annual IEEE/NASA Software Engineering Workshop*, Greenbelt, MD, Dec. 2002. pp 40-47.

[11] C. Zhong, “An Investigation of Reorganization Algorithms,” MS Thesis, Kansas State University, 2006.

[12] J. P. Bigus et al., “ABLE: A toolkit for building multiagent autonomic systems”, *IBM Sys Jnl*, Vol. 41, No. 3, 2002.

[13] S. Kumar, P. Cohen, “Towards a Fault-Tolerant Multi-Agent System Architecture”, *Proceedings of the Fourth International Conference on Autonomous Agents*, Barcelona, Spain, 2000.

[14] R. Sterritt, D. Bustard, “Towards an Autonomic Computing Environment”, *DEXA Workshops*, 2003

[15] G. Tesauro, et. al., “A Multi-Agent Systems Approach to Autonomic Computing”, *AAMAS*, 2004