Computing and Informatics, Vol. 37, 2018, 1363–1385, doi: 10.4149/cai_2018_6_1363

EXPLOITING THE USE OF COOPERATION IN SELF-ORGANIZING RELIABLE MULTIAGENT SYSTEMS

Sebnem Bora

Department of Computer Engineering
Ege University
35100 Izmir, Turkey
e-mail: sebnem.bora@ege.edu.tr

Abstract. In this paper, a novel and cooperative approach is exploited introducing a self-organizing engine to achieve high reliability and availability in multiagent systems. The Adaptive Multiagent Systems theory is applied to design adaptive groups of agents in order to build reliable multiagent systems. According to this theory, adaptiveness is achieved via the cooperative behaviors of agents and their ability to change the communication links autonomously. In this approach, there is not a centralized control mechanism in the multiagent system and there is no need of global knowledge of the system to achieve reliability. This approach was implemented to demonstrate its performance gain in a set of experiments performed under different operating conditions. The experimental results illustrate the effectiveness of this approach.

Keywords: Adaptive systems, availability, autonomous agents, redundancy, software reliability

Mathematics Subject Classification 2010: 68T42, 68M15

1 INTRODUCTION

Today, technology has become an integral part in the majority of our lives. From smart phones, to laptops to tablets – we are heavily connected to networks and systems; computers perform critical tasks in many areas of our lives every millisecond.

High reliability and availability are of the utmost importance to the majority of these computer systems. For example, such systems include nuclear reactor facilities, global air traffic control systems, and banking systems.

In order to build reliable systems, the programming is quite complex since such systems are usually software intensive. A promising approach to deal with complexity is to provide robustness, autonomy, and adaptation to a system by applying self-organizing algorithms. These algorithms are defined based on self-organization mechanisms inspired by nature. Ant and bee swarms, flocks of birds, and school of fish, the human immune system are typical examples of natural systems that exhibit properties inherent to self-organization. Swarms provide inspiration for mobile networks systems management [1], such as load-balancing and routing [2]. Solutions for the distribution of tasks to the available nodes and distribution of data between nodes in computational grids vary from techniques inspired by bee foraging behavior [3] to select the algorithms for executing small pieces of data, to business and market inspired techniques for dynamically changing the task assignment [4, 5]. The use of mobile agents makes us aware of network conditions. Mobile agents [6] can be supportive for intrusion detection and intrusion response in large-scale network infrastructures by following the behavior of the human immune system [7] and the ant foraging behavior [8]. Insect colonies based models are exploited in agent-based software for manufacturing control [9, 10]. PROSA is a representative example of the ant-like approach, where agents mimic the ants' behaviors [11, 12].

Reliability and availability are achieved via redundancy, i.e., duplication of critical functions exists in the system so that application software can reconfigure and maintain (continue to perform) their tasks in the presence of faults in the system. One technique that creates redundancy in a multiagent system (MAS) is adding extra computers or agents. In order to improve reliability and ensure the availability of the MAS organization, critical agents for the system's operation are replicated into groups. This paper presents a novel replication approach that includes resilience, self-organization, and adaptation as its primary properties. It employs a technique grounded in the Adaptive Multiagent Systems (AMAS) theory; therefore, replicating agents into groups and sharing limited resources among them are achieved without a central control or global knowledge about the MAS organization. These properties result from some simple behaviors of agents. The AMAS theory conceptualizes the design of an adaptive multiagent system and provides agents with adaptive capabilities [13]. According to this theory, adaptiveness is based on cooperative behaviors meaning that an agent seeks to help the most troubled agent in the system while achieving its goal. When agents encounter similar problems simultaneously, the program computes a degree of criticality in order to understand potential impacts of these problems on achieving its goal. Considering this criticality, the agent can determine what the most cooperative action should be taken.

The AMAS theory has been applied in various application domains such as dynamic ontologies [14], aircraft design [15], simulation of the functional behavior of a yeast cell [16], crisis management [17], bioprocesses control [18], ambient systems [19], product design [20], maritime surveillance [21, 22], and self-organizing

biological neural networks [23, 24, 25]. In this study, the AMAS theory is applied to the adaptive replication to effectively improve reliability in MAS organizations. Applying the AMAS theory to the model, in conjuncture with the software, creates a more reliable and adaptive MAS distribution because the system is lacking a centralized control. A key element that sets this study apart from previous research is that the adapting replica groups are achieved in a self-organized way without a centralized control. Agents applying this approach use local information and behave in cooperative ways to help the troubled neighbors. In previous studies, agents were usually associated with an adaptive replication manager which dynamically computed the criticalities of agents [26, 27, 28, 29, 30, 31, 32, 33, 34, 35]. The quantities that define the criticalities of agents are used for calculations to share the limited resources between the replica groups. The adaptive replication manager uses an observation service to collect global data about agents in the organization in order to calculate agents' criticalities.

In this study, the AMAS theory is adopted in the reliable MAS where each agent observes whether there is an increase or a decrease in its criticality. If there is an increase, it decides to increase the number of its replicas. However, during the decision process, the leader exploits only its local information when making a decision about the change in the agent's criticality.

This research thoroughly explains how the AMAS theory is applied to MAS in order to enhance reliability and availability. This approach is the first one that uses the AMAS theory for managing adaptive replication (not only in multiagent systems, but in general). A case study chosen to illustrate the AMAS theory is exemplified in the design of a library system which includes cooperative agents with local goals and partial views of their environment. In our conclusion some experimental results are presented to exhibit the effectiveness of this approach. The remaining sections of this paper are organized as follows. Section 2 explores related works; Section 3 provides background information detailing the AMAS theory; Section 4 introduces the agent-based model; Section 5 illustrates the experimental model developed for the study, presents data and provides the analysis and discusses the approach; and Section 6 summarizes the study.

2 RELATED WORK

There is a large number of multiagent platforms; however just a few offer reliability. Those multiagent platforms provide useful solutions of the problem of reliability in MAS. Since the approach presented in this study is a replication-based approach, scholarly articles and case studies associated with this topic published in computer science literature have been investigated.

In order to increase reliability in MAS, Fedoruk and Deters implemented transparent replication via proxies [36]. Their approach is a static replication approach which disregards the idea of changing replication techniques at run time. Thus, replication is only realized by a programmer before an application starts.

Guessoum et al. presented an adaptive multiagent architecture that was implemented with the DIMA [26] platform and DarX middleware [27, 28]. In DarX, software components can either be replicated or not, and it is possible to change the replication strategy at run time. DarX middleware must be integrated into any multiagent organization in order to improve reliability in the organization.

DimaX [29] is a fault-tolerant multiagent development platform that is the integration of DarX into DIMA. DimaX is founded on system level (DarX middleware); application level (agents); and monitoring level. In DimaX, the criticality of agents is calculated by using an interdependence graph. At the monitoring level, the control of replication is monitored by using an observation service [30, 31, 32].

Bora and Dikenelli proposed an approach which provided flexibility to multiagent organizations in terms of fault tolerance because the fault tolerance policies were implemented as reusable plan structures. Thus, whenever an agent needed to be made fault-tolerant, the action was performed by sending a request to that agent [33]. Bora and Dikenelli introduced a self-adaptive replication approach to exploit a feedback control loop and a proportional controller within a replication infrastructure [34, 35]. This approach was used to examine the criticality of specific agents. Moreover, Bora and Dikenelli presented a replication approach based on role concept for multiagent systems. They defined a "fault tolerant" role that is responsible for replicating instances of critical roles, coordination between critical role instances and satisfying all replication-based fault tolerance requirements [37].

In this study, the AMAS theory is adopted in the reliable MAS where each leader agent observes whether there is an increase or a decrease in its criticality. The leader exploits only its local information when it decides to change the agent's criticality. This approach is the first one that uses the AMAS theory for managing adaptive replication (not only in multiagent systems but in general). Further, the self-organizing replication approach in this study provided higher performance when compared to other self-adaptive replication approaches [30, 32, 35, 28], replica groups' monitoring costs not present in this state of the art increased due to the observation mechanisms and the need of global information when the self-adaptive replication approach was applied.

3 AMAS THEORY

The Adaptive Multi-Agent System (AMAS) theory was developed to act as the engine for any system to self-adapt itself to any changes encountered in a dynamic environment. It explains the cooperative relationship between the system's internal operations and its functional adequacy, thus ensuring that the cooperative system carries out the appropriate task it was designed for. It was proved that there is at least one cooperative internal medium system that accomplishes an equivalent function of any functionally adequate system in the same environment. In a cooperative internal medium system, the components of a system are always collaborating. This secures a symbiotic relationship and provides protection from Non Cooperative

Situations (NCS) which may be harmful for their cooperative situations. In regard to MAS, NCS represent situations that are against the cooperative social behaviors of an agent; thus, when identifying an NCS, an agent alters its relationships with other agents in order to return to a cooperative state [13].

If a multiagent system adopts AMAS, an agent can have two types of behavior:

1. nominal behavior; and
2. cooperative behavior,

which is divided into tuning, reorganization, and evolution. An agent with the cooperative behavior attempts to assist other agents; consequently, this agent has to detect and repair the NCS and avoid producing a new NCS [16].

A multiagent system applying AMAS includes agents that try to decrease the criticality of troubled agents by exhibiting local and cooperative behaviors. These behaviors maintain the system's ability to produce an adequate global function. Each agent calculates its own criticality, a function that calculates whether a signal is activated for a NCS or not by examining the criticality value and a threshold [22]. If the agent decides that a NCS has occurred, it exploits the situation by adapting itself to this new situation triggering tuning and reorganization behaviors. Tuning behaviors modify the parameters computed by nominal behaviors. If it fails to solve the NCS, it triggers the reorganization behavior by sending messages (feedback) and/or it informs the cooperative agents that, in turn, eliminate the NCS and transmit feedback.

Reorganization behaviors modify the agent's interaction with its environment or with other agents. If an agent cannot execute its nominal behavior because of NCS, experiences a failed tuning behavior and/or sends requests that cannot be served, then it may create a new communication link with a new agent. If the reorganization behaviors fail to solve the NCS, the evolution behaviors resolve the NCS by creating new agents or removing agents [16, 24].

While applying the AMAS theory, a programmer defines the agents of the system, the agents' nominal behaviors, any NCS the agents may encounter, and cooperative behaviors to overcome each NCS. In the following section, the nominal behaviors of reliable agents are explained in detail.

4 SELF-ORGANIZING RELIABLE AGENTS

The self-organizing reliable MAS consists of three variables:

1. the domain agents (Agent-0 to Agent-N) and their replicas;
2. the environment agents and their replicas; and
3. the environment.

Agents and their replicas run on computers and are grouped according to either active or passive replication approaches [38, 39]. The methods of building consistency in groups vary in active or passive replication approaches; nevertheless, in

both approaches, one replica is designated as the leader and is responsible for providing responses. If a leader fails, any replica can become a leader. Domain agents (Agent-0 to Agent-N) are leaders of their groups.

In addition to replication techniques, the replication degree, which provides the level of redundancy and assigns the number of replicas within a group, is essential in achieving redundancy in MAS while using replication policies. In static replication, both the replication degree and approach are set by a programmer during initialization. In adaptive replication, the group leader primarily decides on its replication degree based on system resources and its criticality. The agent's criticality is a numerical quantity that indicates its importance in the system. However, there is not just one general description and/or definition attached to an agent's criticality. It is the programmer who decides, after careful consideration, the agent's varying properties. In general, this process incorporates an observation service that transparently monitors the agents' behaviors as well as the availability of resources, and adaptively reconfigures the system according to the agents' criticalities according to data evaluation collected by the observation service [26, 27, 28, 29, 30, 31, 32, 33, 34, 35].

According to this design, when the leader performs a self-organizing replication, the group leader will either increase and/or decrease the replication degree without a central control (without an observation service or a replication manager) based on global knowledge of the environment. The leader collects data related to its communication activities and calculates the change in the value of its criticality. The value of an agent's criticality is determined by a quantity that indicates whether the importance of an agent to the system increases or not. If the leader agent's criticality increases, a request message is transmitted to the environment agent which provides resources (hosts in the environment) to agents for replication. Next, a new replica of that particular agent is created. On the other hand, if there is a gradual decrease in the agent's criticality, the replica must be removed after a certain period.

4.1 Nominal Behaviors of Reliable Agents

In this section, reliable agents' nominal behaviors are briefly described. These behaviors are implemented to form the infrastructure for achieving reliability and availability in a goal-oriented MAS architecture [40].

In a reliable self-organizing MAS, each agent performs its actions without a global function (goal) for the system; hence, a dependable (reliable and available) multiagent system emerges. Each reliable agent achieves specific goals, such as Group Communication, Failure Detection, Election of a New Leader, Recovery from Failure, and Adaptive Replication (shown in Figure 1), while being cooperative. In order to achieve these identified goals, their plans and the reusable services are realized. These services and plans are described in more detail in [41].

In order to improve replication in a self-organizing manner, the replication plan was modified and Self-Organizing Replication, Evolution, and Reorganization plans were created. To replicate new replicas, the leader executes the Replication plan

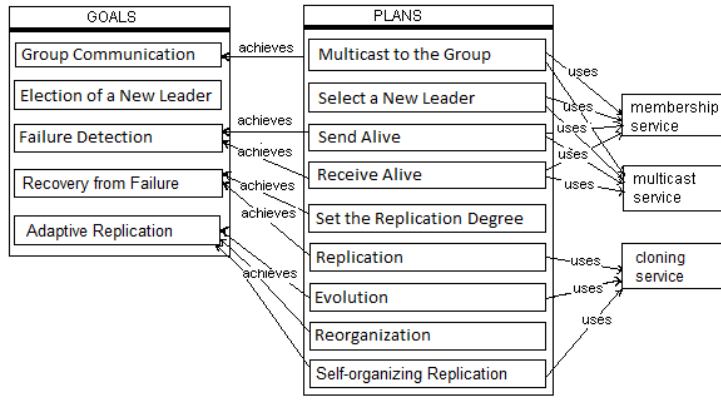

Figure 1. Goals and plans for the system's reliability

as many times as the number of replicas needed to be replicated, and asks the environment agent to locate a suitable host where new replicas can be placed.

For self-organizing replication, the critical agents must be identified. Each leader agent determines whether the change in the criticality occurs or not. The number of messages received by each agent, the degree of system's reliability, and the agent's role criticality are the example of certain metrics needed to calculate the agents' criticalities. In order to achieve the Adaptive Replication goal, the Self-Organizing Replication plan is executed.

4.2 Applying the AMAS Approach for Reliability

In order to achieve reliability in MAS, critical agents periodically monitor their criticalities. This data indicates the potential impact of the agent's failure in the MAS. We must consider two cases to ensure reliability in multiagent systems. Firstly, multiagent systems must have static organization structures so that critical agents can be identified and replication performed by the programmer before run time.

Secondly, if the agent's criticality cannot be determined before run time due to the multiagent systems' dynamic organization structures, critical agents can be dynamically evaluated at run time. In addition, metrics can be used for dynamically estimating and updating agents' criticalities in the MAS. Within the MAS organization, a role is defined as an abstract characterization of social agent's behavior in a specific domain and has a different operational impact in the MAS organization. The concept of a role represents the importance of an agent in an organization, and its dependencies from other agents, i.e., those affecting their criticalities in the multiagent organization.

The agent's dependency on another specific agent is yet another metric that indicates the agent's criticality. If for some reason, a critical agent that relies on an-

other agent fails, then goals will be difficult to achieve. The dependency on an agent is obtained from the number and performatives of the messages received. Messages, which are considered in terms of an agent's criticality in this work, contain performatives such as request, request-whenever, query-if, query-ref, and subscribe [37].

In this research, the change in the value of an agent's criticality is calculated by using precise formulas. If there is an increase in the value of agent's criticality, it signals the agent has become more critical than before and needs to have new replicas. The change in the agent's criticality in the current period, $\Delta Critic(t)$, is given below:

$$\Delta Critic(t) = Activity(t) - Activity(t-1) + C_{reliable}(t). \tag{1}$$

- $C_{reliable}(t)$: Is the contribution of the reliability of the system to the criticality in the current period. The $C_{reliable}$ is the difference between the number of failures in a group and half the number of replicas in the group in terms of a time interval. If the value of $C_{reliable}$ is larger than 0, then the contribution of the $C_{reliable}$ to the criticality is added. If the number of failures for a replica group is less than half the number of replicas, there is no point worrying about the system's reliability. In this case, the contribution of the reliability of the system is set to zero.
- $Activity(t)$: Is the degree of an agent's activity in the current period. $Activity(t-1)$ is determined for the last period and stored in the data structure. The value of the agent's role criticality and the ratio of the change in the number of messages sent to an agent for a specific role relative to the average number of messages over a number of periods are used for calculations of $Activity(t)$ as in the following equation.

$$Activity(t) = a * role(t) + b * Ratio\_req(t) \tag{2}$$

- a, b: Coefficients for contributions of the weight of the role and the average number of requests to the $Activity(t)$.
- $role(t)$: The value corresponding to the weight of the role in the current period. The weights of the roles are explicitly defined in the role ontology before the program starts.
- $Ratio\_req(t)$: The ratio of the change in the number of messages sent to an agent for a specific role to the average number of messages over a number of periods.

If $\Delta Critic(t) < 0$, it specifies the agent's criticality has decreased; however, if $\Delta Critic(t) > 0$, it signals the agent's criticality has increased. Next, the number of replicas of the group must be increased by 1. The agent asks the environment agent to provide a resource to create a new replica.

Following the calculations that determine the agent's criticality value, the leader of a replica group executes a *Self-Organizing Replication* plan in which a self-organizing replication is performed. The *Self-Organizing Replication* plan is given in Figure 2.

Algorithm 1 Self-organizing Replication

1. Calculate the Change in the Criticality, ΔCritic
2. **if** ΔCritic<0 $freq_{dec} \leftarrow freq_{dec} + 1$ **endif**
3. **if** ΔCritic>0 send("request","need a new replica", Environment Agent)
4. **if** Environment agent cannot provide any resource
5. $freq_a \leftarrow freq_a + 1$ and
6. **if** $freq_a > thresh_a$ send("request","need a new replica", Neighbour Agents)
7. **else** FILE← Agent's internal state as a serialized object and
8. Send FILE via RMI to the Environment Agent
9. $freq_a \leftarrow 0$
10. **endif**
11. **endif**

Figure 2. Algorithm for self-organizing replication

The first task of the *Self-Organizing Replication* plan is to determine the change in the agent's criticality, $\Delta Critic(t)$, by using the number of requests received over several periods, the number of failed replicas, and the weight of the role the agent has played (line 1 in Algorithm 1 in Figure 2).

If $\Delta Critic(t)$ is a positive integer, then a message containing a copy request is prepared and sent to the environment agent by using the send method (line 3 in Algorithm 1 in Figure 2). When the environment agent receives this message, it attempts to find a suitable host where new replicas will be placed. After an available host's address is obtained, the environment agent sends a request message to the agent to be replicated in order to convey the agent's internal state. As previously stated, the agent's internal state is serialized and written to a text file (line 7 in Algorithm 1 in Figure 2). Several Remote Method Invocation (RMI) messages are then sent to the environment agent to transfer both the agent's knowledge and internal state (line 8 in Algorithm 1). Thus, both sets of data are needed to commence the replication process.

The remote host's cloning server provides storage for the replicated agent. Multiple messages using RMI are sent to the cloning server by the environment agent to transfer the agent's knowledge and internal state received from the critical agent to be replicated. Upon receiving the RMI messages from the environment agent, the cloning server creates a new replica by using the contents of RMI messages. Next, the cloning server places the unserialized agent's state, libraries and source code to the selected paths and executes the agent's source code.

When the new replica is launched, it has identical state as the leader and contains the last view of the group. It then multicasts a JOIN message informing the other replicas that it has joined the group; subsequently, the other replicas register the new replica to their membership lists.

When $\Delta Critic(t)$ is a negative integer, it demonstrates the agent's criticality has decreased in the current period. If this occurs, the $freq_{dec}$ variable is increased by 1 so that it is possible to determine the number of periods in which the agent has experienced low levels of criticality (line 2 in Algorithm 1).

4.3 Identification of Non-Cooperative Situations

The proposed self-organizing replication architecture, in which replica agents can either be inserted or removed, is subject to NCS. Each NCS is identified by analyzing problematic stages of reliable multiagent systems.

Two kinds of NCS are identified in this work, the first being Unable to create a new replica. It occurs when the environment does not provide a resource to an agent that needs to replicate a new replica. The second NCS is Bad Message Density.

4.3.1 NCS: Unable to Create Replica

If an environment agent does not provide a resource to an $Agent_i$ that needs to replicate, it will send a message informing that there are no available resources. If an agent receives this message, it increases $freq_a$, which is termed a complain variable (see line 5 in Algorithm 1).

During the next period, the agent's criticality might increase. If so, it will then send the environment agent a new request for creating a new replica. The environment agent must find available resources for replication. If it fails, it informs the agent by sending a message containing *No available resources*. In this case, the Unable to create a replica NCS occurs and $freq_a$ is increased by 1 (line 5 in Algorithm 1). Concurrently, the environment agent waits to receive a message containing *Available resource:IP*. When it receives the message, it is able to provide a resource to $Agent_i$ and initialize the *Create a Replica* plan [41] in order to replicate a new replica of $Agent_i$.

4.3.2 Bad Message Density

The number of requests received by an agent exceeds a threshold value.

4.4 Cooperative Behaviours

In AMAS theory, an agent can exhibit two types of behaviour, nominal and cooperative. An agent's nominal behaviours in a reliable MAS are explained in Section 3.

As aforementioned, the cooperative behaviours are categorized into tuning, reorganization, and evolution.

One would presume an autonomous and cooperative agent is able to modify its replicas by adjusting its internal parameters by adopting a tuning behaviour. However, the tuning behaviour is not considered in this work since the consistency between replicas is one of the main issues in reliable systems. If the internal parameters of agent replicas are modified by triggering a tuning behaviour to overcome any NCSs, consistency between replicas cannot be continuously maintained. Since the tuning behaviour is not adopted in this approach, a cooperative agent adopts a reorganization behaviour in which it tries to change the way it interacts with others. The reorganization behaviours of reliable agents are implemented using the *Reorganization* plan.

The last category behaviour that may be adopted by a reliable agent is the evolution. In evolution behaviour, a replica can either be created or removed (e.g., if ineffective, it must leave the system). In these two last levels, the propagation of a problem to other agents is indeed possible if an agent is not able to execute its nominal behaviour.

The evolution behaviours of reliable agents are implemented using the *Evolution* plan (illustrated in Figure 3), which corresponds to the creation and/or removal of reliable agent's replicas. There is an underlying assumption that no replica can be created or removed simultaneously.

NCS is suppressed by executing the aforementioned plans as described in the following subsections. By executing two plans, an agent's criticality value will decrease, as these NCS increase the agent's criticality. The evolution behaviour of reliable agents will be explained first.

4.5 Suppression of "Unable to Create Replica" NCS

When an environment agent does not provide a resource to an $Agent_i$ and the complain variable $freq_a$ exceeds a predefined threshold $thresh_a$, the $Agent_i$ sends the feedback message *Need a new replica* to its neighbours. This message specifically requests an increase in the number of replicas (line 6 in Algorithm 1).

When an agent receives feedback from one or more neighbours (or from its environment), it may retro-propagate a feedback to its own neighbours. When the agent receives the *Need a new replica* message, the *Evolution* plan in Figure 3 is initialized. If the agent is a neighbour, it then stores the number of the *Need a new replica* messages (no_{fa_i}) in a data structure (line 1 in Algorithm 2 in Figure 3). If no_{fa_i} exceeds a certain value $thresh_c$ (line 2 in Algorithm 2), it tries to cooperate with the $Agent_i$.

If its criticality decreases over a certain number of periods (i.e. $freq_{dec}$ is larger than $thresh_{dec}$), it then kills one of its replicas (line 3 and 4 of the *Evolution* plan). After removing the replica, the other replicas in the group also delete this agent from their membership lists and it informs the environment agent (line 5 of Algorithm 2 in Figure 3). Upon receiving a new replica creation request, the environ-

Algorithm 2 Evolution

1. **if** receiving a message containing "need a new replica" $no_{fai} \leftarrow no_{fai} + 1$
2. **if** $no_{fai} > thresh_c$
3. **if** $freq_{dec} > thresh_{dec}$ and the number of replicas>1
4. delete one of the replicas and
5. send ("inform","available resource:IP",Environment Agent)
6. **else** send("request","need a new replica", Neighbour Agents) **endif**
7. **endif**
8. **endif**

Figure 3. Algorithm for the evolution behaviour

ment agent will then create a new replica in the host that has been released by the neighbour's replica. In this case, the number of replicas of $Agent_i$ increases by 1, thus the $Agent_i$ stops complaining and sets the value $freq_a$ to zero (line 9 of Algorithm 1 in Figure 2). If there is no decrease in the neighbour's criticality value of $Agent_i$ (i.e., it is still complaining) during a certain number of periods, it then forwards the *Need a new replica* message to its neighbours (line 6 in Algorithm 2).

4.6 Suppression of "Bad Message Density" NCS

When the number of requests received by $Agent_i$ exceeds a threshold value $thresh_m$, the $Agent_i$ sends the querying agent a message informing that the message density is inadequate.

When the number of messages received by the querying agent is high, the primary action of the *Reorganization* plan is activated by identifying the provision as the sender agent (i.e. $Agent_i$). Whenever $Agent_n$ has received this message, the value of no_{fm} is increased by 1 (line 2 in Algorithm 3 in Figure 4). If the value of no_{fm} exceeds a threshold $thresh_{fm}$, $Agent_n$ searches for a new agent in order to send its queries. Therefore, it asks the Directory Facilitator (DF) to send $Agent_j$ so that it can provide the same service as $Agent_i$ for its queries (line 3 in Algorithm 3 in Figure 4). In order to prevent interaction with $Agent_i$, $Agent_i$ is removed from the knowledge base when DF sends the identifier of $Agent_j$ (line 4 and 5 in Algorithm 3).

Next, when a new agent's identifier has been received from the Directory Facilitator (DF), the new agent's identifier is included to the knowledge base of $Agent_n$ in order to contact with $Agent_j$. Afterwards, it will be possible for it to send its queries to that agent (line 6 in Algorithm 3 in Figure 4). In this case, both the

agent's activity and criticality value decrease.

Algorithm 3 Reorganization

```
1. if receiving a message containing "the number of messages is high"
2.     no_fm ← no_fm +1
3.     if no_fm >thresh_fm send ("request","an Available Agent ", Directory Facilitator)
4.         if Directory Facilitator sends the Agent Identifier of an Available Agent
5.             Remove the Sender Agent that complains and
6.             send requests to the Available Agent
7.         endif
8.     endif
9. endif
```

Figure 4. Algorithm for the reorganization behavior

5 EXPERIMENTS

In order to evaluate the self-organizing replication based on AMAS theory, a library system was designed that included two specific agents – library assistant agents and user agents. The latter were designed to query library assistant agents. Each library assistant agent manages a different library and stores the library knowledge (i.e. bibliographical information) using the library ontology. Instances of this ontology hold the properties of all periodicals and books; for example, the title, the author(s), the ISBN number, and keywords related to documents.

In the case study, each user agent interacted with a user when it received a book and/or periodical request and then forwarded the request directly to all of the library assistant agents. Next, the library assistant agent executed a single plan to match the request to the document's ontology instance(s) and responded with the sources of the bibliographical information contained in a message. When the user agent received the response from the library assistant agents, it then selected a library where the source was located and provided the result to the user. In this case study, the user agents were dependent upon the library assistant agents since the library assistant agent was a critical and reliable agent for the library system's operation.

The library system was implemented by using Semantic Web Enabled Multiagent System Development Framework (SEAGENT) [42] and Java Version 1.5.0. The tests were run on a computer with Intel Core i7 CPU and 64 GB of RAM.

5.1 Costs of Reliability

In a reliable multiagent system, there must be multiple replicas of an agent. According to the preferred replication approach, those replicas may run concurrently, possibly in different environments. The cost of reliability of a multiagent system using a replication approach is the sum of the cost of replica creation/deletion, replica usage, and overheads incurred by the coordination of its replicas. Moreover, a constant number of resources in a system are reserved to provide redundancy for the performance of a certain task. However, usage of a constant number of resources in a system can be expensive. Varying the replication degree in a multiagent system can decrease the cost caused by replication of critical agents of the system. Adaptive replication techniques enable a multiagent system to change its replication degree in accordance with its environment.

In order to evaluate the costs of applying a self-organizing replication, a test environment was implemented, which included library assistant agent leaders ranging from 10 to 60, plus their replicas, and a user agent. In the first case, the library assistant agent leaders applied a static replication technique in the system. Next, the programmer manually deployed replicas of the leader agents in the system. When the static replication technique was applied, the number of replicas increased, doubling the number of the leaders.

In the second case, the library assistant agent leaders applied the self-organizing replication in the system. The programmer deployed the library assistant agent leaders in the system whereby they were automatically and dynamically replicated in accordance with their criticalities at runtime. The highest number of possible replicas in the system was set to two times that of the deployed leaders. The time of the test's sampling period was set to 400. In this test, the threshold value $thresh_a$ was set to 2, the threshold value $thresh_c$ was set to 1, and the threshold value $thresh_{dec}$ was set to 2.

In the third case, in order to compare the systems' costs, whether in static or self-organizing replication techniques, the systems' response times were compared to the control (i.e. response time of a system without using a replication approach). The user agents sent their requests to the library assistant agents without applying any replication technique. In this test, it was observed if the effects of self organizing replication influenced the overall performance of the system.

In the first and second case, semi-active replication was employed in the library systems. In order to measure the cost of the self-organizing replication approach, the user agent sent queries to the leaders. The number of requests sent to the leaders changed the agent's criticality accordingly. However, in all cases, the total number of requests sent to the organization were equal in every step of the tests. Response times, for queries were measured in all cases. The response time is the time it takes a querying agent to receive the reply after sending its request to a leader agent.

5.1.1 Test Results

The results of tests are illustrated as graphs in Figure 5. When the test environment included library assistant agent leaders ranging from 10 to 30, the systems' response times were very close to each other since the number of agents was not large in the systems and the computer performed all operation very fast. As illustrated by the graphs, the slopes of the average response times of the systems applying the static, self-organizing replication techniques, and no replication increased with the number agents, as seen from Figure 5.

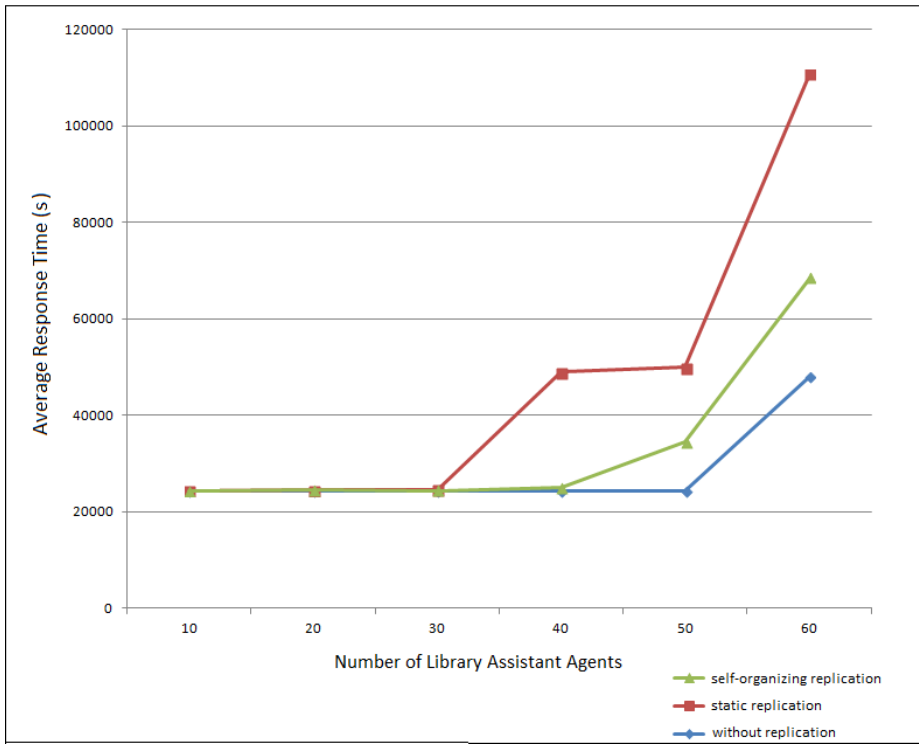

Figure 5. The effect of the self-organizing replication approach

The increase in response times was anticipated due to the fact that the leader of the group multicasts all incoming requests to the replicas; thus, the number of requests sent to the system increased. All replicas processed these requests. Finally, the number of messages exchanged also increased with the number of agents due to leader's multicasting of requests and heartbeat messages. Moreover, in SEAGENT, the communication module uses the RMI based communication infrastructure and all functionalities of internal architecture are based on threads. Therefore, when all agents were created in a single machine, then agents' threads were initialized and

the amount of time contributed by computation was slightly increased because of a computational load of the computer.

5.1.2 Discussion

According to the results, the self organizing replication approach is very promising and outperforms the static replication. As indicated in Figure 5, the system's response time, when the static replication is applied, is longer than the system's response time when the self-organizing replication is applied. Further, the system's response times, when the self-organizing replication is applied, are very close to the system's response time when no replication is applied. This result was expected, since the number of replicas in the system effects the system's response times, as mentioned in the previous sections.

When the self organizing replication technique is applied to the system, the number of replicas in each group may change with respect to the criticalities of the leader agents. The highest number of replicas in the system applying self organizing replication can equal the total number of replicas in the system applying static replication, for the same number of leaders. Sometimes, the agents may not receive any requests for a period of time or the number of requests they have received decreases. Thus, the degree of agents' activities will decrease as will the agents' criticalities during these periods. If a decrease in criticality is present for a certain period of time, the leader agent decreases the number of its replicas by removing useless replicas. In this case, the system's response time decreases as the number of replicas in the system decreases. If its criticality increases, it increases the number of its replicas in order to decrease its criticality. As the number of replicas in the system increases, the response time of the system also increases because the leader agent multicasts the received requests to its replicas and the replicas process the requests simultaneously.

However, when applying static replication, the programmer himself/herself decides on the number of replicas before executing the application. During execution, the number of replicas are fixed to a certain value in the organization ($2 \times$ number of leaders). In the system applying self organizing replication, the highest number of possible replicas can total the number of replicas in the system applying static replication. Although, the number of requests sent to the systems might be equal, the number of total replicas can change in accordance to the criticalities of the agents in self organizing replication. Therefore, the system's average response time when static replication is applied is longer than the system's average response time when self organizing replication is applied.

The most important advantage of the self organizing replication is the lack of centralized control. Indeed, the presence of a centralized control actually increases the system's response times since the centralized control mechanism needs to have global information of the system. Having the system's global information increases the number of messages sent and received; therefore, the response time also increases in the system applying a centralized self-adaptive reliable approach compared to the

system applying a static approach as explained in [35]. The agents in the system applying the self-organizing replication use the local information and behave in a cooperative manner in their system. Thus, the reliable self-organizing system takes advantage of applying the AMAS theory.

High reliability and availability are of the utmost importance to air traffic control (ATC) systems. The Advanced Automation System (AAS) [38] was a distributed real-time system that integrated all the services of the US air traffic control network. Critical services were replicated using either the active or passive approach, according to the application semantics and the hardware configuration. ATC systems are complex in the sense of complex systems. The control and knowledge are distributed in these systems and the flight data will be more complicated than the processed data at present to be processed by these systems. Therefore, adoption of new technologies and procedures must be taken into consideration when ATC systems are built. The self-organizing approach presented in this paper can be used in such systems in order to alleviate the effects of software failures and provide a great support to achieve high reliability and availability in ATC systems.

5.2 Evaluation of Suppression of "Bad Message Density" NCS

In order to observe whether a *Bad Message Density* NCS was suppressed or not, a user agent sent 80 queries to one of the library assistant agents with low processing capacity in each sampling period. Since the threshold $thresh_m$ was set to 25, the library assistant agent sent *The number of messages high* message to the user agent. The user agent receiving this message increased no_{fm} by 1. When the user agent detected that no_{fm} was larger than $thresh_{fm}$ (it was set to 1), it asked the DF agent to obtain a new library assistant agent's identifier that it could get the same service. The DF agent sent another library assistant agent's identifier to the user agent. The user agent removed the old library assistant agent's identifier from its knowledge base and added the new library assistant agent's identifier. As a result, the user sent its queries to the new agent. After receiving *The number of messages high* message, all operations were executed in a time frame ranging from 1 921–2 682 ms.

5.3 Evaluation of Robustness of the Self-Organizing Approach

A test bed consisting of five library assistant agent leaders and five user agents, and a failure simulator were designed and implemented in order to evaluate the robustness of the self-organizing replication approach. The failures ranging from 85 to 111 were injected into the system by a failure simulator which simply sent "kill" messages to the agents within a certain time frame. The agents receiving the "kill" messages stopped executing their threads. The majority of the kill messages were sent to the library assistant agents since they were the critical agents for the system's operation. A smaller number of kill messages were sent to the user agents. In this test, the number of available resources was set to 100; therefore, the maximum number of replicas in the whole multiagent system could be 100.

As shown in Figure 6, the number of failures injected to the multiagent system gradually increased, and the rate of success was observed and recorded. The rate of success is defined as the percentage of the replica groups that could accomplish their tasks in MAS within a certain time frame. The data in Figure 6 revealed the rate of success declined while the number of failures in multiagent system rose. This association was expected due to the fact that only 100 replicas existed in the multiagent system in order to tolerate failures.

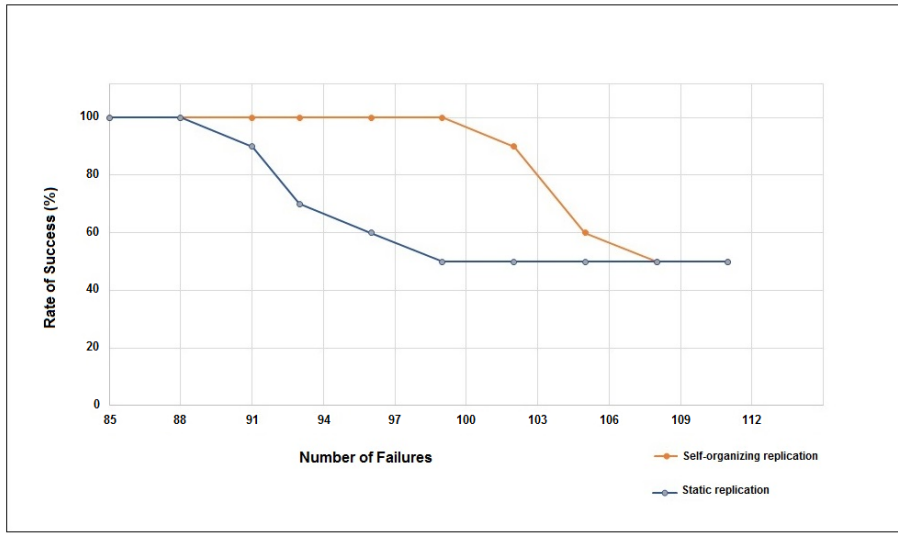

Figure 6. Robustness of the self-organizing replication approach

In the static replication approach, the replication degrees of the groups were fixed to certain numbers; therefore, the number of failures and agents' activities never changed the numbers of replicas in the groups. Since the majority of the failures influenced the library assistant agents, most of the library assistant agent groups failed to complete their tasks during these experiments; however, there existed the user agent groups in the multiagent system at the end of the experiments.

When the system employing self-organizing replication started, the replication manager determined the numbers of replicas of each library assistant agent leader and each user agent leader with respect to their criticalities. As the number of kill messages received by the replica groups linearly increased, each leader took into consideration the number of failures in order to calculate the change in the agent's criticality and it required a resource to create a new replica if the agent's criticality increased. Therefore, when the number of failures increased in the system, the self-organizing approach showed a stronger performance as compared to the static replication approach. From these experiments, it was observed that a replication approach required f+1 replica library assistant agents in order to survive up to

f library assistant agent crashes in the replica group. Moreover, the number of replicas in a multi-agent system should be at least equal to the number of library assistant agent leaders which were critical for the system's operation.

6 CONCLUSION

In this paper, a self-organizing approach based on AMAS theory was proposed for improving reliability in MAS. Based on observations, when the self organizing replication based on the AMAS theory was applied to the groups, the cost of reliability decreased. Therefore, this new approach to adaptive replication highly outperformed static replication.

In conclusion, the results indicated that the efficient replication was sustainable using this approach. Reliable systems can benefit from the self-organizing replication that utilizes resources more efficiently, and can also achieve greater flexibility. Some other future research opportunities can also be pursued to evaluate the performance of the various algorithms based on self-organization mechanisms inspired by nature. In the future, adaptive immune system concepts can be utilized to design and implement self-organizing reliable multiagent systems. Moreover, stigmergy and genetic algorithms can be used as the mechanisms to enhance reliability in multiagent systems. Especially, adaptive stigmergic mechanisms are good candidates for providing self-organization to reliable multiagent systems. Future work concerns a deeper analysis of those mechanisms, costruction of the reliable systems in different agent platforms, or implementation of a system that improves reliability in the cloud.

REFERENCES

[1] Brueckner, S. A.—Van Dyke Parunak, H.: Self-Organising MANET Management. In: Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., Zambonelli, F. (Eds.): Engineering Self-Organising Systems (ESOA 2003). Springer, Berlin, Heidelberg, Lecture Notes in Artificial Intelligence, Vol. 2977, 2004, pp. 20–35, doi: 10.1007/978-3-540-24701-2_2.

[2] Montresor, A.—Meling, H.—Babaoglu, O.: Messor: Load-Balancing Through a Swarm of Autonomous Agents. In: Moro, G., Koubarakis, M. (Eds.): Agents and Peer-to-Peer Computing (AP2PC 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2530, 2003, pp. 125–137, doi: 10.1007/3-540-45074-2_12.

[3] Ko, S. Y.—Gupta, I.—Jo, Y.: Novel Mathematics-Inspired Algorithms for Self-Adaptive Peer-to-Peer Computing. Proceedings of the First International Conference on Self-Adaptive and Self-Organising Systems (SASO 2007), 2007, pp. 3–12, doi: 10.1109/SASO.2007.40.

[4] Li, Y.—Chen, F.-H.—Sun, X.—Zhou, M.-H.—Jiao, W.-P.—Cao, D.-G.—Mei, H.: Self-Adaptive Resource Management for Large-Scale Shared Clusters. Jour-

nal of Computer Science and Technology, Vol. 25, 2010, No. 5, pp. 945–957, doi: 10.1007/s11390-010-9379-0.

[5] Wrzesinska, G.—Maassen, J.—Bal, H. E.: Self-Adaptive Applications on the Grid. Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '07), 2007, pp. 121–129, doi: 10.1145/1229428.1229449.

[6] Foukia, N.: IDReAM Intrusion Detection and Response Executed with Agent Mobility. Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '05), Utrecht, The Netherlands, IEEE Press, New York, 2005, pp. 264–270, doi: 10.1145/1082473.1082513.

[7] Forrest, S.—Hofmeyr, S. A.—Somayaji, A.—Longstaff, T. A.: A Sense of Self for Unix Processes. Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, IEEE, 1996, pp. 120–128, doi: 10.1109/SECPRI.1996.502675.

[8] Holland, O.—Melhuish, C.: Stigmergy, Self-Organization, and Sorting in Collective Robotics. Artificial Life, Vol. 5, 1999, No. 2, pp. 173–202, doi: 10.1162/106454699568737.

[9] Buyurgan, N.—Meyyappan, L.—Saygin, C.—Dagli, C. H.: Real-Time Routing Selection for Automated Guided Vehicles in a Flexible Manufacturing System. International Journal of Manufacturing Technology Management, Vol. 18, 2007, No. 2, pp. 169–181, doi: 10.1108/17410380710722881.

[10] Clair, G.—Kaddoum, E.—Gleizes, M.-P.—Picard, G.: Self-Regulation in Self-Organising Multiagent Systems for Adaptive and Intelligent Manufacturing Control. Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '08), IEEE, 2008, pp. 107–116, doi: 10.1109/SASO.2008.19.

[11] Valckenaers, P.—Van Brussel, H.—Kollingbaum, M.—Bochmann, O.: Multi-Agent Coordination and Control Using Stigmergy Applied in Manufacturing Control. In: Luck, M., Mařík, V., Štěpánková, O., Trappl, R. (Eds.): Multi-Agent Systems and Applications (ACAI 2001). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2086, 2001, pp. 317–334, doi: 10.1007/3-540-47745-4_15.

[12] Van Brussel, H.—Wyns, J.—Valckenaers, P.—Bongaerts, L.—Peeters, P.: Reference Architecture for Holonic Manufacturing Systems: PROSA. Computers in Industry, Vol. 37, 1998, No. 3, pp. 255–274, doi: 10.1016/S0166-3615(98)00102-X.

[13] Capera, D.—George, J.-P.—Gleizes, M.-P.—Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-Organizing Cooperative Agents. Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises (WET ICE 2003), IEEE, 2003, pp. 383–388, doi: 10.1109/ENABL.2003.1231441.

[14] Ottens, K.—Gleizes, M.-P.—Glize, P.: A Multi-Agent System for Building Dynamic Ontologies. Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07), ACM, 2007, Art. No. 227, doi: 10.1145/1329125.1329399.

[15] Combettes, S.—Sontheimer, T.—Rougemaille, S.—Glize, P.: Weight Optimization of Aircraft Harnesses. In: Demazeau, Y., Müller, J., Rodríguez, J., Pérez, J. (Eds.): Advances on Practical Applications of Agents and Multi-Agent Systems. Springer, Berlin, Heidelberg, Advances in Intelligent and Soft Computing, Vol. 155, 2012, pp. 229–232, doi: 10.1007/978-3-642-28786-2_26.

[16] Bernon, C.—Capera, D.—Mano, J.-P.: Engineering Self-Modeling Systems Application to Biology. In: Artikis, A., Picard, G., Vercouter, L. (Eds.): Engineering Societies in the Agents World IX (ESAW 2008). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5485, 2009, pp. 248–263, doi: 10.1007/978-3-642-02562-4_14.

[17] Lacouture, J.—Rodriguez, I.—Arcangeli, J.-P.—Chassot, C.—Desprats, T.—Drira, K.—Garijo, F.—Noel, V.—Sibilla, M.—Tessier, C.: Mission-Aware Adaptive Communication for Collaborative Mobile Entities. Handbook of Research on Mobility and Computing Evolving Technologies and Ubiquitous Impacts, 2011, pp. 1056–1076, doi: 10.4018/978-1-60960-042-6.ch064.

[18] Videau, S.—Bernon, C.—Glize, P.—Uribelarrea, J.-L.: Controlling Bioprocesses Using Cooperative Self-Organizing Agents. In: Demazeau, Y., Pěchoucěk, M., Corchado, J. M., Pérez, J. B. (Eds.): Advances on Practical Applications of Agents and Multiagent Systems. Springer, Berlin, Heidelberg, Advances in Intelligent and Soft Computing, Vol. 88, 2011, pp. 141–150, doi: 10.1007/978-3-642-19875-5_19.

[19] Guivarch, V.—Camps, V.—Péninou, A.: AMADEUS: An Adaptive Multi-Agent System to Learn a User's Recurring Actions in Ambient Systems. ADCAIJ Advances in Distributed Computing and Artificial Intelligence Journal, Vol. 1, 2013, No. 3, pp. 1–10.

[20] Kaddoum, E.—Georgé, J.-P.: Collective Self-Tuning for Complex Product Design. Proceedings of 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012), IEEE, 2012, pp. 193–198, doi: 10.1109/SASO.2012.14.

[21] Brax, N.—Andono, E.—Gleizes, M.-P.: A Self-Adaptive Multi-Agent System for Abnormal Behavior Detection in Maritime Surveillance. In: Jezic, G., Kusek, M., Nguyen, N. T., Howlett, R. J., Jain, L. C. (Eds.): Agent and Multi-Agent Systems. Technologies and Applications (KES-AMSTA 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7327, 2012, pp. 174–185, doi: 10.1007/978-3-642-30947-2_21.

[22] Mano, J.-P.—Georgé, J.-P.—Gleizes, M.-P.: Adaptive Multi-Agent System for Multi-Sensor Maritime Surveillance. In: Demazeau, Y., Dignum, F., Corchado, J. M., Pérez, J. B. (Eds.): Advances in Practical Applications of Agents and Multiagent Systems (PAAMS 2010). Springer, Berlin, Heidelberg, Advances in Intelligent and Soft Computing, Vol. 70, 2010, pp. 285–290, doi: 10.1007/978-3-642-12384-9_34.

[23] Gürcan, Ö.: An Emergent Model for Mimicking Human Neuronal Pathways in Silico. Proceedings of the 12th European Conference on Artificial Life (ECAL 2013), MIT Press, 2013, pp. 1172–1173, doi: 10.7551/978-0-262-31709-2-ch180.

[24] Gürcan, Ö.—Bernon, C.—Türker, K. S.—Mano, J.-P.—Glize, P.—Dikenelli, O.: Simulating Human Single Motor Units Using Self-Organizing Agents. Proceedings of the 2012 IEEE Sixth International Conference on Self-

Adaptive and Self-Organizing Systems (SASO 2012), IEEE, 2012, pp. 11–20, doi: 10.1109/SASO.2012.18.

[25] Gürcan, Ö.—Türker, K. S.—Mano, J.-P.—Bernon, C.—Dikenelli, O.—Glize, P.: Mimicking Human Neuronal Pathways in Silico: An Emergent Model on the Efective Connectivity. Journal of Computational Neuroscience, Vol. 36, 2014, No. 2, pp. 235–257, doi: 10.1007/s10827-013-0467-3.

[26] Guessoum, Z.—Briot, J.–P.: From Active Objects to Autonomous Agents. IEEE Concurrency, Vol. 7, 1999, No. 3, pp. 68–76, doi: 10.1109/4434.788781.

[27] Guessoum, Z.—Briot, J. P.—Sens, P.—Marin, O.: Toward Fault-Tolerant Multi-Agent Systems. European Workshop on Modeling an Autonomous Agent in a Multi-Agent World (MAAMAW 2001), Annecy, France, 2001.

[28] Marin, O.—Bertier, M.—Sens, P.: DARX – A Framework for the Fault-Tolerant Support of Agent Software. 14th International Symposium on Software Reliability Engineering (ISSRE 2003), 2003, pp. 406–416, doi: 10.1109/ISSRE.2003.1251062.

[29] Faci, N.—Guessoum, Z.—Marin, O.: DimaX: A Fault Tolerant Multi-Agent Platform. Proceedings of the Fifth International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (ICSE '06, SELMAS '06), ACM, Shangai, China, pp. 13–20, 2006.

[30] Guessoum, Z.—Faci, N.—Briot, J.-P.: Adaptive Replication of Large-Scale Multi-Agent Systems – Towards a Fault-Tolerant Multi-Agent Platform. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (Eds.): Software Engineering for Multi-Agent Systems IV (SELMAS 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3914, 2005, pp. 238–253, doi: 10.1007/11738817_15.

[31] Guessoum, Z.—Briot, J.-P.—Marin, O.—Hamel, A.—Sens, P.: Dynamic and Adaptive Replication for Large-Scale Reliable Multi-Agent Systems. In: Garcia, A., Lucena, C., Zambonelli, F., Omicini, A., Castro, J. (Eds.): Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2603, 2003, pp. 182–198, doi: 10.1007/3-540-35828-5_12.

[32] Guessoum, Z.—Ziane, M.—Faci, N.: Monitoring and Organizational-Level Adaptation of Multi-Agent Systems. Proceedings of the Third International Joint Conference on Autonomous Agents (AAMAS '04), Vol. 2, 2004, pp. 514–522.

[33] Bora, S.—Dikenelli, O.: Implementing a Multi Agent Organization That Changes Its Fault Tolerance Policy at Run-Time. In: Dikenelli, O., Gleizes, M. P., Ricci, A. (Eds.): Engineering Societies in the Agents World VI (ESAW 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3963, 2006, pp. 153–167, doi: 10.1007/11759683_10.

[34] Bora, S.—Dikenelli, O.: Applying Feedback Control in Adaptive Replication in Fault Tolerant Multi-Agent Organizations. Proceedings of the Fifth International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (ICSE '06, SELMAS '06), ACM, Shangai, China, 2006, pp. 5–12, doi: 10.1145/1138063.1138066.

[35] Bora, S.—Dikenelli, O.: A Centralized Self-Adaptive Fault Tolerance Approach Based on Feedback Control for Multi-Agent Systems. Turkish Journal of Electrical

Engineering and Computer Sciences, Vol. 24, 2016, pp. 4707–4723, doi: 10.3906/elk-1405-58.

[36] Fedoruk, A.—Deters, R.: Improving Fault-Tolerance by Replicating Agents. Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 2002, pp. 737–744, doi: 10.1145/544862.544917.

[37] Bora, S.—Dikenelli, O.: Replication Based on Role Concept for Multi-Agent Systems. In: Aldewereld, H., Dignum, V., Picard, G. (Eds.): Engineering Societies in the Agents World X (ESAW '09). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5881, 2009, pp. 165–180, doi: 10.1007/978-3-642-10203-5_15.

[38] Cristian, F.—Dancey, B.—Dehn, J.: Fault-Tolerance in the Advanced Automation System. Proceedings of the 4th Workshop on ACM SIGOPS European Workshop (EW 4), ACM, 1990, pp. 6–17, doi: 10.1145/504136.504156.

[39] Elnozahy, E. N.—Zwaenepoel, W.: Replicated Distributed Processes in Manetho. Fault-Tolerant Computing. Digest of Papers, Twenty-Second International Symposium on FTCS-22, IEEE, 1992, pp. 18–27.

[40] Stollberg, M.—Rhomberg, F.: Survey on Goal-Driven Architectures. Technical Report DERI-TR-2006-06-04, DERI, Austria, 2006.

[41] Bora, S.: Implementing Fault-Tolerant Services in Goal-Oriented Multiagent Systems. Advances in Electrical and Computer Engineering, Vol. 14, 2014, No. 3, pp. 113–122, doi: 10.4316/AECE.2014.03015.

[42] Dikenelli, O.—Erdur, R. C.—Gumus, O. et al.: SEAGENT: A Platform for Developing Semantic Web Based Multi Agent Systems. Fourth International Joint Conference on Autonomous Agents (AAMAS '05), 2005, pp. 1271–1272, doi: 10.1145/1082473.1082728.

Sebnem Bora received her Ph.D. degree from Ege University, Turkey in 2006. She is currently Assistant Professor in the Computer Engineering Department at Ege University. Her research interests include dependable computing, self-adaptive systems, and agent-based modeling and simulation.