Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 4707 – 4723

doi:10.3906/elk-1405-58

# A centralized self-adaptive fault tolerance approach based on feedback control for multiagent systems

**Şebnem BORA*, Oğuz DİKENELLİ**
Department of Computer Engineering, Faculty of Engineering, Ege University, İzmir, Turkey

**Received:** 08.05.2014 • **Accepted/Published Online:** 29.08.2015 • **Final Version:** 06.12.2016

**Abstract:** Our research introduces a self-adaptive fault tolerance approach for multiagent systems that enables the system to avoid crash failures. It is a replication-based approach that exploits a feedback control loop and a proportional (P) controller within a replication infrastructure. Thus, we are able to both observe the agents' behaviors to estimate criticalities and determine the number of replicas in replica groups with respect to the agents' criticalities and the number of available resources. Thus, agents that are to be replicated and the numbers of replicas in replica groups are automatically and adaptively identified in dynamic environments. We implement this approach to demonstrate performance gained in a set of experiments undertaken in different operating conditions.

**Key words:** Software agents, self-adaptive, feedback control, fault tolerance

## 1. Introduction

Fault tolerance is a property that allows a computing system to continuing functioning, albeit proportionally to the severity of the failure, in the likely event of a system failure due to faults or failures in the system. Moreover, the primary objective of fault tolerance is to design a dependable system in order to avoid failures, even when faults are present.

According to Laprie, dependability is defined "as a property of a computing system that allows reliance to be justifiably placed on the service it delivers" [1]. Dependability is characterized by a number of attributes, such as reliability, availability, maintainability, and security. A strong relationship exists between these attributes and autonomic computing [2], or self-* properties [3]; subsequently, dependability and fault tolerance are concerned with the self-healing facet of autonomic computing.

Autonomic computing or self-* systems are organized in a way that allows automatic adaptations to respond to environmental changes. Although Huebscher and McCann asserted that researchers use the terms autonomic computing, self-healing, self-optimizing, self-configuring, and self-adaptive interchangeably [4], the self-adaptive software domain is more limited, whereas autonomic computing has emerged in a broader context. For the purposes of this paper, we simply use the term self-adaptive systems, which include its many variants.

A self-adaptive fault-tolerant system exhibits the ability to change its fault tolerance approach and/or its level of redundancy in accordance with its user's preferences and environment without human intervention. Therefore, self-adaptive fault tolerance has both its own mechanisms and approaches to sustain fault tolerance.

Our research is a replication-based fault tolerance approach that develops a self-adaptive fault-tolerant

*Correspondence: sebnem.bora@ege.edu.tr

multiagent system (MAS). Agents that are to be replicated and the numbers of replicas in replica groups are automatically and adaptively identified by using our self-adaptive replication mechanism. When applying a replication-based fault tolerance policy to a MAS, one of the challenges is identifying which agent needs to be, and to what degree it must be, replicated in dynamic and large-scale environments. Therefore, agents in a fault-tolerant MAS are observed by the self-adaptive replication mechanism in order to estimate their criticalities. Since there are a limited number of resources in the system, they are adaptively shared between agents with respect to their activities and criticalities in the system.

Our approach exploits a feedback control loop within a self-adaptive replication mechanism to provide fault tolerance to the MAS. In addition, control theory methodology and analysis are examined to explain the replication; however, applying a theoretical perspective to software is more complicated since system modifications are required when building software. Our aim is to describe how to apply the feedback control methodology to MASs in order to enhance their fault tolerance.

Further, we provide data that illustrate that the control theory methodology is able to enhance fault tolerance in MASs. The remainder of this paper is structured as follows: Section 2 presents related work, Section 3 introduces the method used in our self-adaptive fault tolerance approach, Section 4 presents the implementation details of the self-adaptive replication mechanism, Section 5 describes how to employ self-adaptive fault tolerance, Section 6 examines a case study, and, finally, Section 7 presents the conclusion.

## 2. Related work

Just a small number of multiagent platforms offer fault tolerance, thus providing useful solutions to fault tolerance problems in MASs.

Fedoruk and Deters' [5] replication-based fault tolerance model was implemented via proxies; however, they did not provide any mechanism that could mask the failure from users in the case of the failure of the proxy. In addition, it ignored the idea of changing replication strategies and degrees of replica groups at run-time. Thus, replication could only be realized by a programmer before an application was initiated.

To improve fault tolerance in a MAS, Guessoum et al. designed an adaptive multiagent architecture association with a DIMA [6] platform and DarX middleware [7]. DarX agents can be either unreplicated or replicated and the replication strategy can be changed at run time. DarX can be integrated into any MAS to improve fault tolerance, but it is not that flexible for it is impossible to make an agent fault-tolerant after it has already been initialized.

DimaX, an integration of DarX and DIMA, is a fault-tolerant multiagent development platform [8] comprising 3 different levels: 1) the system (DARX middleware); 2) the application (agents); and 3) monitoring. DARX supplements the required mechanisms for distributing and replicating agents, while DimaX fulfills an observation service operating at both middleware and application levels. DIMA supplements a set of libraries in order to construct multiagent applications at the application level. At the monitoring level, the control of replication is monitored by using the observation service.

Our system design is unique because it both dynamically and automatically identifies the agents to replicate and the number of replicas of replica groups by employing a proportional (P) controller and organizing as a feedback control system. Our self-adaptive replication mechanism monitors agents in the environment to evaluate the criticalities of agents by taking into account communication and interaction with other agents and the number of failures in replica groups. Since the available resources are limited in the environment, it shares available resources between agents with respect to their activities and criticalities in the environment.

To our knowledge, our self-adaptive replication mechanism is the first one that uses feedback control for managing adaptive replication, not only in MASs but in general.

## 3. Method

In this section, we introduce our proposed abstract architecture, which provides self-adaptive replication-based fault tolerance to MASs. An illustration of this architecture is shown in Figure 1.

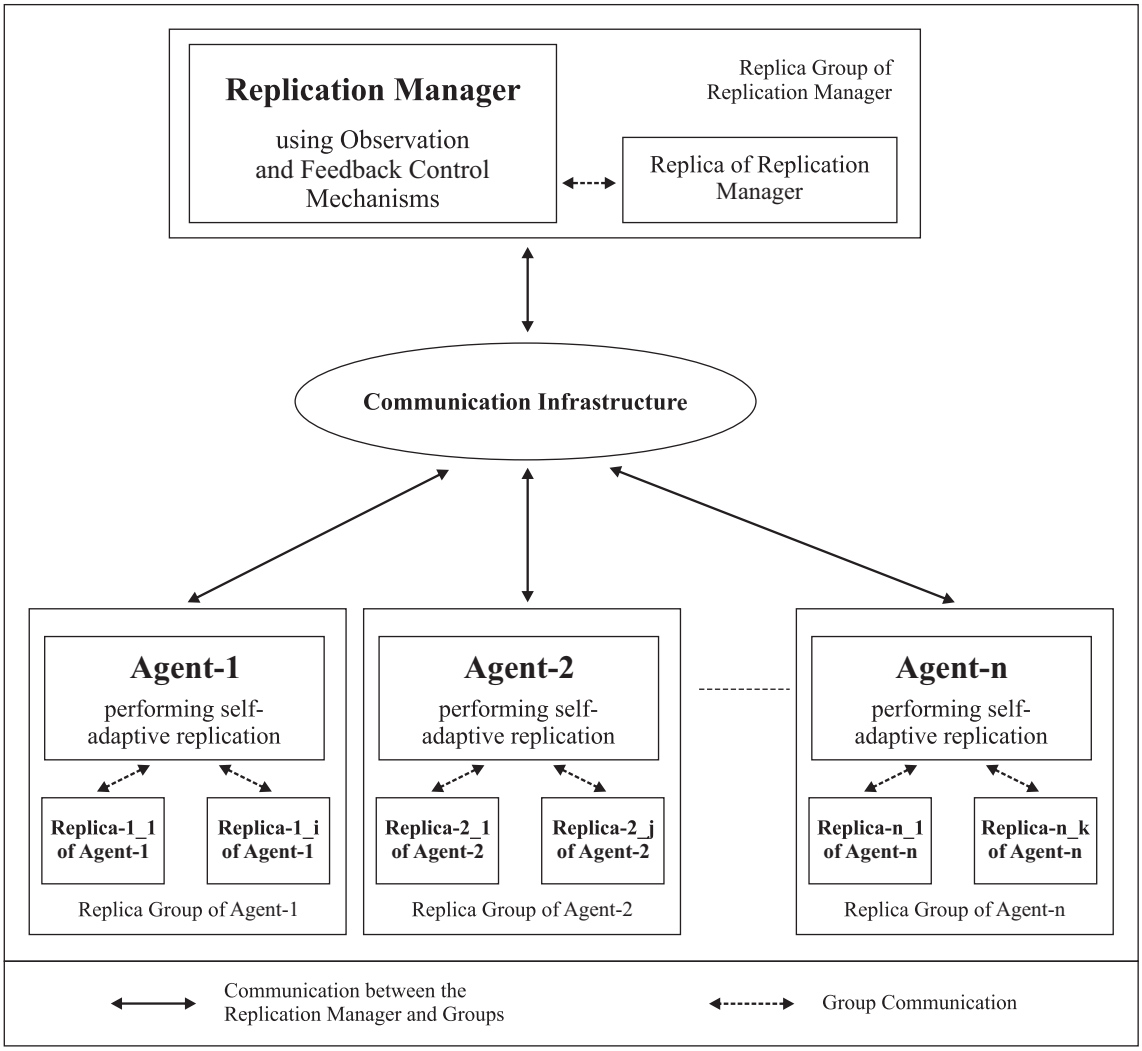


**Figure 1.** Abstract architecture for self-adaptive fault tolerance approach.

Our replication-based fault-tolerant MAS consists of agents (Agent-1 to Agent-n) as leaders and their replicas. When agents and their replicas are run on computers, they are subject to fail since computers experience hardware and/or network problems. When a failure occurs on a replica, other replicas prevent the system's failure and ensure the availability of the system. In this paper, we assume that the considered system tolerates only crash failures [9].

In order to effectively manage self-adaptive fault tolerance in the MAS, we associate an "adaptive replication manager" as illustrated in Figure 1. We prefer that kind of hierarchy in our system. An advantage is

that decision making is more flexible since the adaptive replication manager makes decisions without experiencing any interference from other agents. In other words, the hierarchal design actually decreases the agent's communication load since it does not need to know the global state of the system to decide on replicating itself.

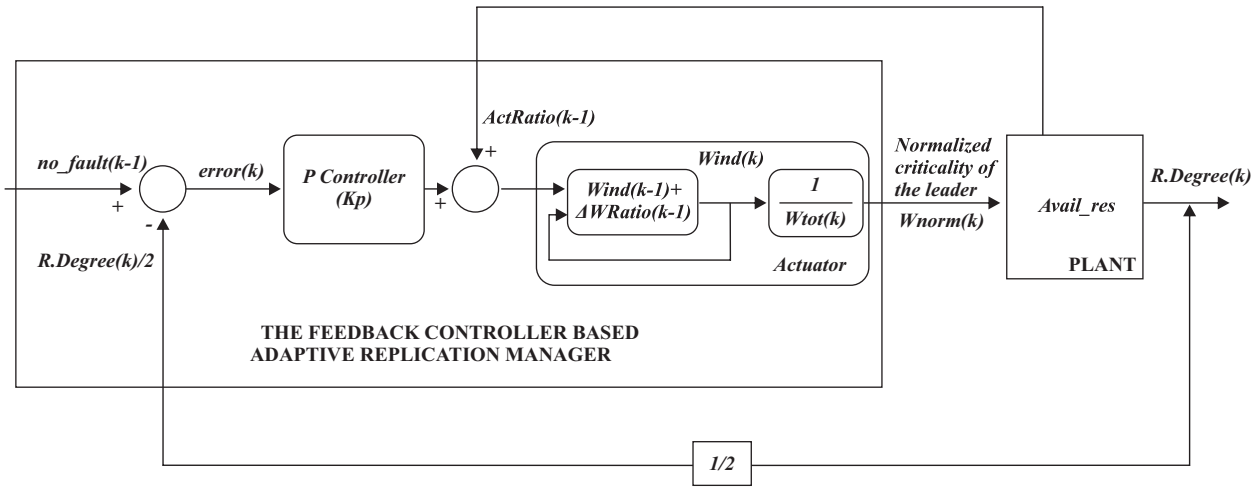


**Figure 2.** The architecture for feedback controller-based self-adaptive replication.

The adaptive replication manager adaptively calculates the criticality of each leader agent. The criticalities of leader agents are quantities input in calculations in order to distribute the limited resources to replica groups. The adaptive replication manager uses both the observation and feedback control mechanisms to achieve this goal.

The observation mechanism gathers the data from the environment. Next, the collected data are handled in the feedback control mechanism, where each agent's criticality is calculated. This value signifies a quantity that emphasizes the importance of an agent to the system. The criticality of each leader agent is embedded within the content of a message transmitted to a leader, which applies the self-adaptive fault tolerance policy.

The goal of a leader is to be fault-tolerant against crash failures. In order to achieve this goal, it performs adaptive replication and uses a replication service. The leader forwards the message received from the adaptive replication manager to its replication service.

The replication service applies the replication approaches to the groups. In our system, we implement both semiactive and passive replication approaches [9–11]. The system can use either the semiactive replication approach or the passive replication approach.

The replication service creates new replicas or removes replicas with respect to the content of the message received from the adaptive replication manager. The replication service uses the membership service, the failure detector, and the group communication service to perform removal and replication operations. In order to provide fault tolerance to multiagent systems, we implement these services. The implementation details of these services are given in [12].

Our self-adaptive replication mechanism uses a feedback control loop for managing adaptive replication. Therefore, we focus on presenting this mechanism in this paper. The next section describes in detail how the feedback controller-based self-adaptive replication mechanism is implemented.

## 4. Feedback controller-based self-adaptive replication

In order to monitor the replication of agents in a multiagent system, we derived a self-adaptive replication mechanism that exploits a feedback control loop. A system using feedback control is referred to as a feedback control system. A typical feedback control system comprises a controller, a plant, and sensors. Moreover, it defines the controlled variable, the quantity of the output that is measured and controlled. The difference between the current value of the controlled variable and the system input (set point) is the error. The manipulated variable is the quantity that is varied by the controller so as to affect the value of the controlled variable. The system operates as follows:

1. The system periodically ascertains and compares the controlled variable to the input to determine if any errors have occurred.
2. If an error is found, the controller computes the necessary control with a control function to remedy any errors.
3. The actuator changes the manipulated variable's value to control the system.

In order to employ feedback control in replication, we pinpoint the input, the controlled variable, the control function, and the manipulated variable. The criticality of a replica group leader is characterized as the manipulated variable in our approach. The controlled variable (the system's output) is the replication degree of a replica group.

The input is characterized by the number of faults that affect the members of the group and may cause members of the replica groups to crash as well. In our design, we presume that the number of faults is directly related to the number of replicas that crash in a group. The system compares the number of replicas that have currently crashed in a group (input) with the replication degree of a replica group (controlled variable). In order to survive up to f agent crashes in the replica group, an active replication approach requires 2f+1 replica agents in a group because of solving consensus. In other words, if the number of crashed replicas exceeds one-half of a replication degree, then the error will be a positive value and will increase the next value of the replication degree, since the controller uses the value of the error to influence the replication degree.

One assumption that we hold is that the controller's type in the feedback control mechanism is P control for self-adaptive replication. It is a type of proportional-integral-derivative (PID) controller, which are generic feedback control mechanisms used in industrial control systems. The PID controller algorithm involves 3 separate parameters: the proportional, the integral, and the derivative values. The proportional value determines the reaction to the current error.

In MASs, the environmental context is extremely dynamic. In adaptive fault tolerance policy, the replication mechanism of a system relies on one part of this environment. Further, not only does it depend on the current input, but also on the system's previous inputs, such as communication and interaction between agents, and failure rates. It is often difficult to create a precise model because the environment is very active. Nevertheless, P control does not require a precise analytical model since it is used in the self-adaptive replication mechanism. Essentially, it adapts to the system's output in accordance with environmental changes.

The plant in this approach consists of replica groups and their leaders. If a replica group's leader fails, then the "Leader Election Algorithm" elects a new leader; thus, replica groups are fault-tolerant against their agents' failures [12]. The actuator uses the controller's output and the agent's activity information, which is gathered by an observation mechanism, to compute the agent's criticality. After normalizing, the criticality value is sent to the leader in the message's content. When the leader receives this message, it executes the "Adaptive

Fault Tolerance" task to replicate agents or kill replicas. In the next section, the observation mechanism of the feedback controller-based self-adaptive replication architecture is explained in detail.

### 4.1. The observation mechanism

In order to determine the criticality of agents in self-adaptive fault tolerance policies, it is essential to monitor the system. Critical information is accessed by an observation mechanism that is either system-level information, such as communication load and processing time, or from an application level, such as the importance of messages [13–15].

The agent's criticality indicates just how reliant and important agents are for specific agents; consequently, in terms of fault tolerance, the agent's criticality is of the utmost importance. If a critical agent fails, agents that rely on that particular agent will experience difficulty achieving their individual goals.

If warranted, there are many different types of dynamic metrics to estimate and update the agents' criticalities in the system. The dependency of other agents on a specific agent is an important metric that illustrates the agent's criticality. The dependency on an agent is based on the number and performatives of the messages received. Some performatives of messages may have more importance than others, such as request messages and accept messages in agent systems. Guessoum et al. distinguished 6 unique classes of performatives [16]. In our study, if messages are to be considered as an agent's criticality, their performatives should contain requests and other variations (request-whenever, query-if, query-ref).

In order to monitor the environment, the adaptive replication manager queries the leader agents related to their communication loads and the number of replicas that have currently crashed in their replica groups. Thus, all data from system and application levels are ascertained by querying agents in a specific time period. The data collection period is programmed when the system is initialized.

We created a data structure to store individual agent data. The data consist of: 1) the agent's name; 2) the agent's address; 3) the total number of requests received by the agent; 4) the total size of messages received by the agents; 5) the number of replicas that have crashed (we call it "the number of failures in the replica group"); and 6) the agent's normalized criticality information reported from the previous period. The information in the data structure is automatically updated when the agent receives a new request or a new failure report.

### 4.2. The feedback control mechanism using the proportional controller

In this section, we detail how the data are processed to obtain the criticality information of each agent. In order to process the collected data and determine replication degrees of groups, a feedback control mechanism is the service used by the adaptive replication manager (as illustrated in Figure 2).

In this architecture, when the system is initialized, a sampling period T is established for the multiagent system. This period is defined over a time window ((k–1)T, kT), where k is the sampling instant. The criticality of each agent is calculated by using the following formulas:

$$error(k) = no\_fault(k-1) - \frac{R.Degree(k)}{2} \tag{1}$$

*no_fault(k − 1)*: The number of replicas that have crashed in the *(k− 1)*th sampling window. The data related to the number of failures in the replica group are based on the failure reports sent by the failure detectors at every sampling period.

*R.Degree(k)*: The replication degree of the group in the $k$th sampling window;

*error(k)*: The difference between *no_ fault (k − 1)* and *R.Degree(k)/2* is the "*error*" of the feedback control mechanism. If the *error(k)* is a negative value or is equal to 0, then the controller has been disabled, and we should only consider the degree of the agent's activities (the value of the communication load of the agent) in the calculation. Even if there is no change in an agent's activity, the self-adaptive replication mechanism will replicate new replicas to keep the replication degree constant since the criticality of the agent has not changed.

If the value of *error(k)* is greater than 0, then the number of failures in a replica group is greater than the set point. In this case, the contribution of the number of failures to the criticality must be increased. As a result, the P controller is initialized and the proportional response of the P controller is adjusted by multiplying the "*error*" by a constant *Kp*, called the proportional gain. The proportional gain of the controller is restricted by the following:

$$0 < Kp < 4 * \frac{W_{tot}(k-1)}{Avail\_res} \tag{2}$$

*Avail_res*: The available resources that encompass the maximum number of simultaneous replicas.

$W_{tot}(k-1)$: Total criticality of all agents in the MAS for the previous sampling period.

Based on the above inequality, *Kp* is determined by the parameters defined by the inequality. We derive this value from the stability analysis of the feedback control mechanism. The proof for Eq. (2) will be presented at the end of this section.

The output of the controller is given by the following:

$$Fault(k) = error(k) * Kp \tag{3}$$

where *Fault(k)* represents the number of failures affecting the criticality of an agent in the $k$th sampling window.

The change in the criticality of an agent, $\Delta W\,Ratio$, is given by the following formula:

$$\Delta WRatio(k-1) = Fault(k) + ActRatio(k-1) \tag{4}$$

*ActRatio(k − 1):* The degree of an agent's activity at the $(k-1)$th sampling instant and stored in the data structure and determined for the $(k-1)$th sampling period. In calculating *ActRatio(k − 1)*, only messages received from other agents in the system are considered by the communication loads of fault-tolerant agents, not messages exchanged between replicas in groups. By doing so, the dependency of other agents on a specific agent is a metric that we use.

The formula calculated by the actuator of the feedback control mechanism is given below:

$$W_{ind}(k) = W_{ind}(k-1) + \Delta WRatio(k-1) \tag{5}$$

$W_{ind}(k)$: New criticality value calculated for the next period. $W_{ind}(k-1)$ is the criticality value of the agent stored in the data structure and determined in the $(k-1)$th sampling period.

Before the $(k+1)$th sampling period begins, the data structure is updated with *ActRatio(k)* and $W_{ind}(k)$. In the next step, we determine the normalized criticality as follows:

$$W_{norm}(k) = \frac{W_{ind}(k)}{W_{tot}(k)} \tag{6}$$

$W_{norm}(k)$: Normalized criticality of the agent in the $k$th sampling period. It expresses the importance of the agent among other agents in a MAS. $W_{norm}(k)$ values are directed to the leaders and then used to execute adaptive replication in replica groups.

**Proof.** In order to prove the inequality in Eq. (2), the z transformation of the self-adaptive replication infrastructure should be used in a discrete time domain [17]. The z transformation of the feedback control mechanism is illustrated in Figure 3. In this figure, the z transformation of the actuator's output is given as follows:

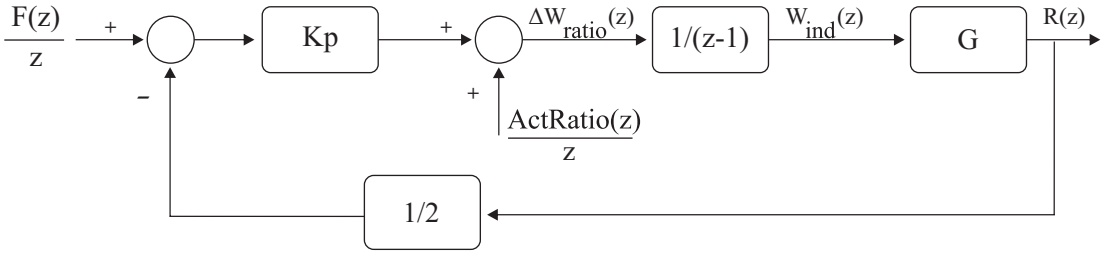


**Figure 3.** The z transform of the feedback control mechanism.

$$W_{ind}(z) = \frac{\Delta WRatio(z)}{z-1} \tag{7}$$

$$\Delta WRatio(z) = \left[Kp * \left(\frac{F(z)}{z} - \frac{R(z)}{2}\right) + \frac{ActRatio(z)}{z}\right] \tag{8}$$

The output, R(z), of the feedback controller-based self-adaptive replication infrastructure in the z domain is given as follows:

$$\left[Kp * \left(\frac{F(z)}{z} - \frac{R(z)}{2}\right) + \frac{ActRatio(z)}{z}\right] * \frac{(G)}{(z-1)} = R(z) \tag{9}$$

Then R(z) is derived from Eq. (9):

$$R(z) = \frac{(2 * F(z) * Kp * G)}{(2 * (z-1) + Kp * G) * z} + \frac{(2 * ActRatio(z) * G)}{(2 * (z-1) + Kp * G) * z} \tag{10}$$

$F(z)$: The z transform of *no_ fault (k)*.

$R(z)$: The z transform of *R.Degree(k)*.

$Kp$: The controller's proportional gain.

$G$: The gain of the actuator, which equals $Avail\_res/\Omega W$. Here, $\Omega W$ is the sum of the criticalities of the agents applying adaptive fault tolerance in the system.

*ActRatio(z)*: The z transform of *ActRatio(k)*.

$R(z)/F(z)$ is the transfer function of the feedback controller-based self-adaptive replication infrastructure and is given in Eq. (11). It can be derived from Eq. (10).

$$\frac{R(z)}{F(z)} = \frac{2 * Kp * G}{(2 * (z-1) + Kp * G) * z} \tag{11}$$

Our feedback controller-based self-adaptive replication infrastructure is stable if and only if all the poles of $R(z)/F(z)$ are in the unit circle of the z-plane. In Eq. (12), the pole of the transfer function is derived by setting the denominator of the transfer function to zero.

$$(2 * (z - 1) + Kp * G) * z = 0 \tag{12}$$

For Eq. (12), we obtain 2 poles for z.

$$z = 0 \tag{13}$$

$$z = 1 - \frac{Kp * G}{2} \tag{14}$$

If a system is stable, the following condition must be satisfied by the system:

$$|z| < 1 \tag{15}$$

For z = 0, the system is stable. We need to check whether the other pole is in the unit circle of z-plane.

$$-1 < \left(1 - \frac{Kp * G}{2}\right) < 1 \tag{16}$$

$$0 < Kp < \frac{4}{G} \tag{17}$$

If $Kp$ is satisfied in eq. (17), the closed-loop system is stable. It seems that the inequalities in Eqs. (17) and (2) are the same. The next section describes how to apply self-adaptive replication to replicate agents into groups.

## 5. Applying the self-adaptive replication approach

The first task of applying self-adaptive replication is to determine the replication degree of the group by using the *Wnorm* value transmitted by the adaptive replication manager. The replication degree of each leader is calculated usıng the following formula: $R.Degree = rounded(Wnorm * Avail\_res)$ (18)

where *R.Degree* is equivalent to the number of replicas found in the next sampling period. Let us consider a set of hosts (computers) *H={h1, h2, h3,. . . ,hnh}* where the processes can reside, and let *nsj* be the number of resources available in the host *hj*. *Avail_res* is given as follows:

$$Avail\_res = \sum_{j=1}^{nh} ns_j \tag{18}$$

If we assume that the local machines that host replicas have limited resources, then the replication degree is determined locally as follows:

$$R.Degree = rounded(Wnorm * \sum_{j=1}^{nh} ns_j) \tag{19}$$

$$R.Degree = \sum_{j=1}^{nh} rd_j \tag{20}$$

where *rdj* is the number of replicas of a leader that must reside in machine *hj* and *rdj* is given as follows:

$$rd_j = rounded(Wnorm * ns_j) \tag{21}$$

Then the number of replicas to be adapted is determined locally as follows:

$$rc_j = rd_j - nr_j \tag{22}$$

*nrj*: The number of replicas in the host *hj*.

*rcj*: The number of replicas needed to be replicated in the host *hj* if $rc_j$ is 1) a positive integer or number of replicas that must be killed, or 2) a negative integer in the host *hj*.

If *rcj* is positive, then the "Create New Replicas" task starts to execute its code. In the "Create New Replicas" task, as many messages containing copy requests are prepared as the number of *rcj* and sent to the leader itself to initiate the "Cloning a Replica" task. After the leader receives copy request messages, the "Cloning a Replica" task is executed as given in [18].

If *rcj* is negative, then the "Decrease the Replication Degree" task starts to execute its code to decrease the replication degree of the group in the host *hj*. In the "Decrease the Replication Degree" task, as many request messages are prepared as *rc* and sent to the leader itself to initiate the "Decreasing Replication Degree" task. After the messages are received by the leader, the "Decreasing Replication Degree" task starts to execute as given in [18].

In this paper, another resource assumption made is that the resource requirement for replicating all agents is the same. However, different agents require different amounts of resources, and this reality adds some constraints when deploying agents such as the memory space available on a computer, the memory space necessary for each agent, and the maximal interaction cost among the agents. Determining where agent replicas must be deployed is an important issue called the resource allocation problem, and it was discussed in [19].

In order to give an idea about our feedback controller-based self-adaptive replication mechanism, we will next present a case study example. In our case study, we examine a MAS that consists of a limited number of agents disseminated throughout a network. The agents are only able to communicate by transmitting messages to one another.

## 6. Case study

The fault tolerance approach featured in this study was implemented within the internal architectural framework of the Semantic Web Enabled Multiagent Development System (SEAGENT) [20]. Moreover, the framework is goal-oriented and designed for the FIPA (Foundations for Physical Agents)-based MAS architecture.

In order to verify our methodological approach, first we designed an agent system that included both library assistant agents and user agents that were specially designed to query library assistant agents (as illustrated in Figure 4). Each library assistant agent managed a different library and stored library knowledge using a reference ontology. Instances of this ontology contained specific elements of data that included the title of the book, the ISBN, authors' names, and key words of the books. In our case study, each user agent directly sent a book request to all library assistant agents. Next, the library assistant agent has only one plan (a partially ordered sequence of primitive tasks) that matches the request to the book ontology instance(s) and returns the matched books' descriptions within a message. When the user agent receives responses from the library assistant agents, it selects a book based on the responses and provides the results to the user. In this case study, the user agents depend on the library assistant agents: therefore, each library assistant agent is

a single point of failure. Due to the fact that the library assistant agent is a critical agent for the system's operation, it is initialized as a fault-tolerant agent. Although our agent plan is basic, the general characteristic of the case study is extremely reliable in terms of fault tolerance.

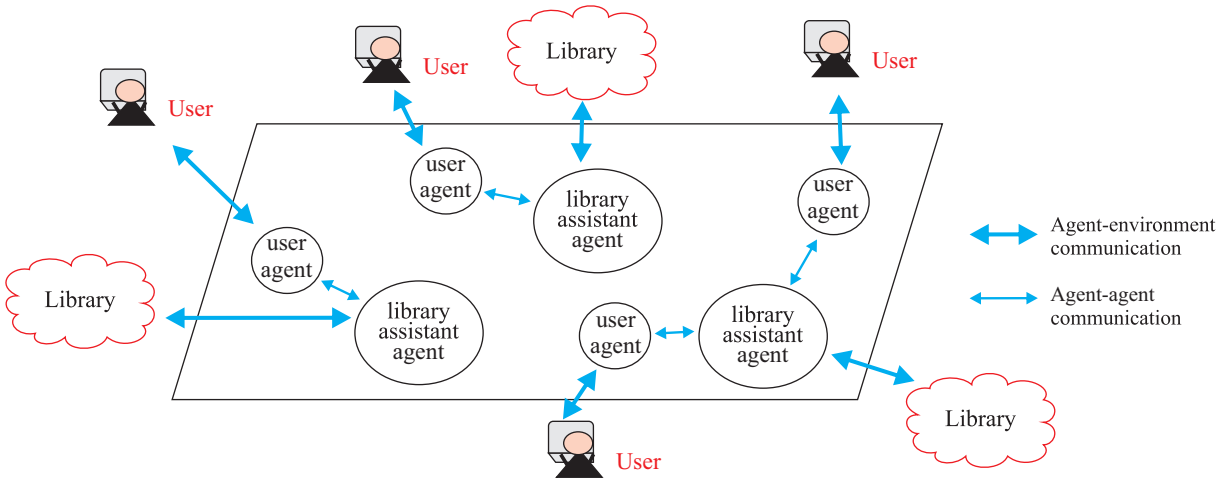


**Figure 4.** An agent-based library system.

The agent system is implemented in the SEAGENT platform and Java version 1.5.0. The tests are performed on 2 computers with Intel P4 running at 1.5 GHz and 2 GB of RAM.

The evaluation consists of 4 tests. In the first test ("effect of monitoring and feedback control"), we evaluate the cost resulting from including observation and feedback control mechanisms in the system as the number of library assistant agent leaders increases in the MAS. In the second test ("effect of sampling period"), we evaluate how response times vary with different sampling periods in the system. In the third test ("effectiveness of the feedback control mechanism"), we evaluate the effectiveness of the feedback controller-based self-adaptive replication architecture as the number of failures sent to the MAS increases up to 130. In the fourth test ("effect of value of $Kp$"), we employ the feedback control mechanism for different values of $Kp$ according to Eq. (2) to observe the effect of the value of $Kp$ on the system's operation. In the next sections, the tests and their results will be given in detail.

### 6.1. Effect of monitoring and feedback control

In order to evaluate the cost of the feedback controller-based self-adaptive replication mechanism, we decided to implement a test bed containing 10 to 70 library assistant agent leaders. In addition, the test bed contained their replicas and a user agent. In our initial case, the library assistant agent leaders executed a static fault tolerance plan (which means that no monitoring and feedback control are applied). In the second case, the library assistant agent leaders utilized the adaptive fault tolerance policy simply by employing the observation and feedback control mechanisms in order to periodically monitor the test bed. The sampling period of time for the test was fixed at 400 s.

The system allows the leader applying static fault tolerance policy to automatically restart the failed replicas. The system allows the leader applying the adaptive fault tolerance policy to restart the failed replicas only after an update is received from the feedback controller-based adaptive replication manager. In both cases, the library assistant agent leaders are distributed to the first and second computers and all replicas are deployed or automatically replicated in the second computer. The numbers of replicas are equal and they are replicated

in both cases. In the first case, due to the static fault tolerance policy, we deploy replicas to the second computer manually. In the second case, we only deploy the library assistant agent leaders to the computers, and eventually they replicate themselves to the second computer.

In order to determine the cost (efficiency) of monitoring and feedback control, the querying agent sends 100 queries (there is no waiting time between queries) to the leaders specifically to record the response times. This measurement is then determined by recording the time allotted by a user agent to receive a reply from a leader agent. In this test, the effects of monitoring and feedback control on the overall system performance were observed and noted.

The results of Test1 are illustrated in Figure 5. The graph indicates that as the number of library assistant agent leaders increase, the response times of the systems applying the static and self-adaptive fault tolerance policies also increase. The increase in response times of systems is not surprising since the number of messages received and sent between the agents has been multicast as heartbeat messages. Figure 5 shows the increase in response times when systems apply self-adaptive fault tolerance and static fault tolerance policies.

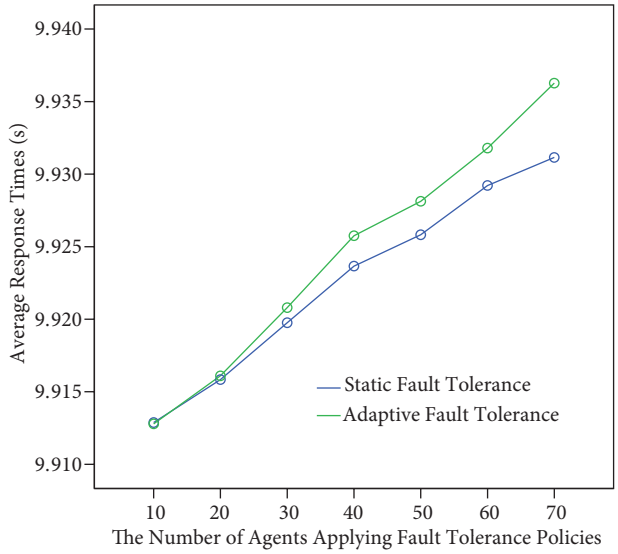


**Figure 5.** The effect of the feedback control mechanisms.

In the extent of determining the effect of the observation and feedback control mechanisms in the library system, it is crucial to measure the differences between the response times of the library systems. As seen in Figure 5, the cost resulting from including monitoring and feedback control mechanisms increases as the number of agents increase. This result was expected because as the number of library assistant agent leaders applying the self-adaptive fault tolerance policy increases, the observation and feedback control mechanisms of the adaptive replication manager have to continuously monitor more agents, thus determining the criticality values for themselves and then relaying the values to the agents. When a message containing the criticality value is received by the agent, it executes a self-adaptive replication mechanism to calculate a new replication degree. According to the new value, the agent will either clone itself or remove its replicas. Thus, every single operation increases the response time.

At first glance, the response time of the system applying adaptive fault tolerance is longer than the response time of the system without monitoring and feedback mechanisms. However, the difference between response times when the self-adaptive fault tolerance policy is applied is not relatively large when compared

to monitoring cost response times (9,935,000 ms vs. 5000 ms from Figure 5). Therefore, the cost of including feedback control in a system for self-adaptive replication is not high as the number of leader agents increases in the system. If the setting for the appropriate sampling period is increased, the response time of the system applying the self-adaptive fault tolerance policy may improve compared to the system applying the static fault tolerance policy. The effect of the sampling period on the response of the system's performance is evaluated in the next section.

### 6.2. Effect of sampling period

The sampling period is just one of many important elements that can affect the response time of the observation and feedback control mechanisms. A test bed represented by library assistant agent leaders, their replicas, and an agent that queries the library assistant agents was implemented by us. The library assistant agent leaders administer a self-adaptive fault tolerance policy. The tests involved 2 cases. In the first case, there were 30 library assistant agent leaders and the sampling period of the system was gradually increased from 1 s to 200 s. In the second case, there were 60 agents in the system, the querying agent sent 20 queries to the leaders, and the response time of the system was measured.

The data in Figure 6 reveal that the response times of the systems executing self-adaptive fault tolerance policies declined while the sampling period of the system rose. This was expected due to the fact that as the sampling period decreased, the number of samples per unit of time increased and vice versa. When the period of sampling decreases, the adaptive replication manager has to frequently monitor agents, determine the criticality values for them, and send these values to the agents, thus creating a rise in the response time of the system [21]. Moreover, for the same sampling periods, as the number of agents in the system climbs, the response times of the system applying self-adaptive fault tolerance policies also rises, as seen from the previous tests.

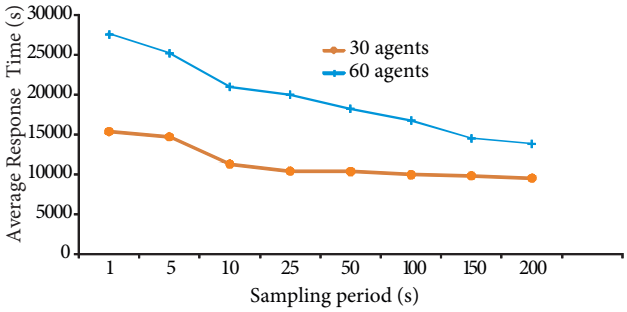


**Figure 6.** The effect of the sampling period.

### 6.3. Effectiveness of the feedback control mechanism

To evaluate the effectiveness of the feedback control mechanism, a test bed consisting of 5 library assistant agent leaders and 5 user agents and a failure simulator was carried out by us. Failures in the number range from 100 to 130 are injected into the system by the failure simulator. It injects an ascending number of failures during each period (it injects 100 failures during the first period, then 102 failures during the second period, and so on).

In order to model the presence of failures, the failure simulator sends "kill" messages to the agents within a certain time frame. The majority of the kill messages are received by the most critical agents, while a smaller number of kill messages are sent to agents that are less critical. Of these kill messages, typically 80% are sent to agents with the highest activity within the current period and 15% are sent to agents with a moderate activity.

The remainder of the kill messages are transmitted to less active agents within a certain time frame. In this particular model, there is an assumption that the system has been tested under stress conditions. The agents getting the kill messages will eventually cease executing their threads.

As mentioned before, the values of the agents' activities are important to calculate the criticality of the agent. To create the effect of communication load on the MAS, user agents uniformly send a number of requests to the library assistant agents. In this test, the library assistant agents are the critical agents. User agents are the agents with the smallest activity and therefore are not critical agents. In this test, *Avail_res* is given as 100, which also defines the maximum number of replicas in the MAS to be 100. The library assistant agent leaders and user agents (both organized as 5 different groups) share available resources present in their activities and criticalities in the environment. For example, when the system started, the replication manager determined that each library assistant agent leader had 17 replicas and each user agent had 3 replicas with respect to their criticalities. Then the failure simulator injected 100 failures to the system by sending kill messages. Of the 80% of kill messages sent to the library assistant agents all were evenly distributed. Of 5% of kill messages sent to the user agents all were evenly distributed.

In the initial case, the feedback control mechanism is deactivated, thus canceling the closed loop of the feedback control mechanism. In doing so, we only consider the communication load of an agent. Thus, according to our observations of the behavior of the control mechanism, it is similar to our previous research enquiries and has been thoroughly explained in the works of Guessoum [6–9,13,16]. In the second case, the feedback control mechanism is employed, in which $K_p$ is set to one-half of the maximum bound given by Eq. (2). In this particular test, we reported a noticeable difference when comparing the feedback control mechanism in the two separate cases. As illustrated in Figure 7, we increased the number of failures injected to the MAS and observed the rate of success. We recorded the rate of success as the percentage of the replica groups that could complete their tasks in the MAS within a certain time frame.

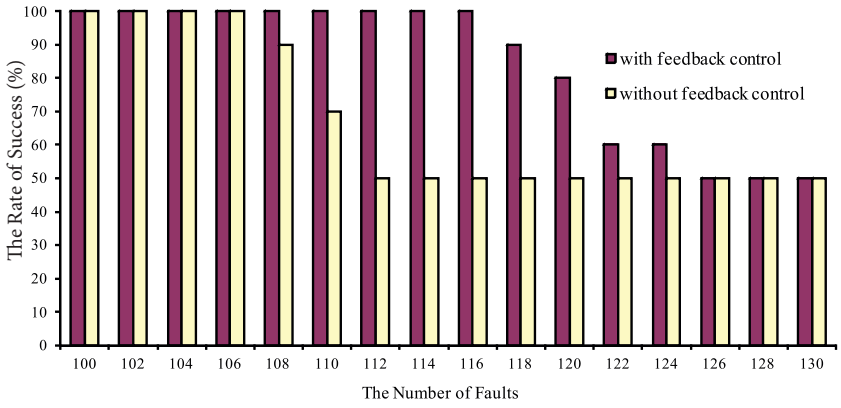


**Figure 7.** The effectiveness of the feedback control mechanism.

Furthermore, as the number of failures exceeds 100, over a period of time the success rate declines because only 100 replicas can exist in the MAS. As the number of failures sent to the group linearly rises, the feedback control mechanism takes into consideration the number of failures in order to determine the agents' criticality and adapts to the number of replicas. By deactivating the feedback control loop, the number of failures was not considered in the determination of the agents' criticality. Therefore, when the number of failures increased in the system, the feedback control mechanism exhibited a stronger performance as compared to the other control mechanisms.

We performed a t-test (paired samples) to evaluate the differences in means between the two mechanisms' results. The t-test indicated that the difference between the mechanism with feedback control and the mechanism without feedback control was significant (P = 0.001). Although the mean ±gstandard deviation of the success rate of the mechanism without feedback control equaled 56.6667, the mean ±gstandard. deviation of the success rate of the mechanism with feedback control equaled 87.7778. This finding confirms that the integration of control theory to adaptive replication enhances fault tolerance.

### 6.4. The effect of the value of Kp

In order to conduct this test (the results are illustrated in Figure 8), we used the test bed and the same scenario in the third test. We utilized the feedback control mechanism for different values of *Kp* according to Eq. (2). We applied a specific set of values of *Kp* such as *(0, 1/4*(4* $W_{tot}$ (k-1) / Avail_res))*, *(1/2*(4* $W_{tot}$ (k-1) / Avail_res))*,*(4* $W_{tot}$ (k-1) / Avail_res)*, and *(3/2*(4* $W_{tot}$ (k-1) / Avail_res))* for each run. As illustrated in Figure 8, we increased the number of failures injected to the MAS and observed the rate of success. One general change that we observed was the effect of the *Kp* value over the feedback control mechanism; as different values were substituted for *Kp*, the rate of success gradually declined as the number of faults sent to the MAS increased. We anticipated this result due to the fact that there were only 100 available resources shared between replica groups. As a result, the rate of success decreased as the number of the failures increased to more than 100.

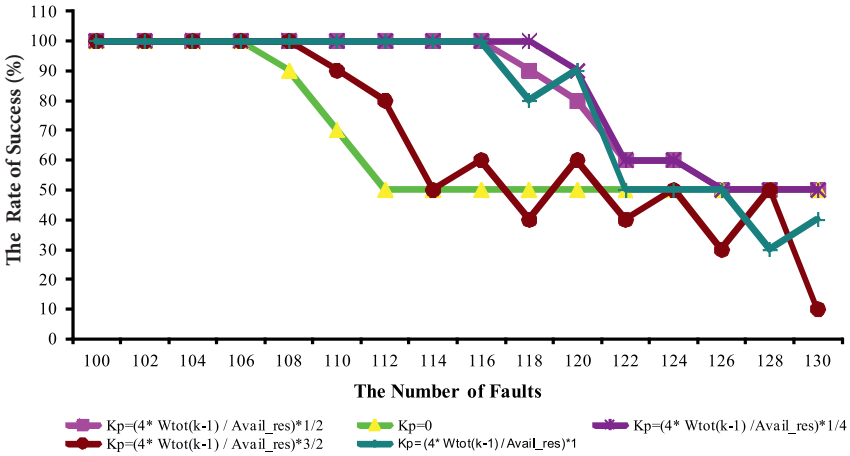


**Figure 8.** The effect of the value of *Kp* on the success rate.

By setting the *Kp* to zero, the number of failures was not considered to determine the agents' criticalities. Moreover, that value of *Kp* was not assigned a value between the bounds according to the inequality given in Eq. (2). Therefore, when the failure rate increased in the organization, the feedback control mechanism for specific values of *Kp* (selected between the bounds of inequality given in Eq. (2)) exhibited a better performance compared to the mechanism with $Kp = 0$.

Moreover, our test concludes that the values of *Kp* should be assigned between the bounds (e.g., *(1/4*(4* $W_{tot}$ (k-1) / Avail_res))* and *(1/2*(4* $W_{tot}$ (k-1) / Avail_res))*) defined by the inequality given in Eq. (2). When *Kp* is not assigned within the bounds such as *(4* $W_{tot}$ (k-1) / Avail_res)* and *(3/2*(4* $W_{tot}$ (k-1) / Avail_res))*, the performance significantly declines and unstable behaviors are readily observed. Those values of *Kp* increased the effect of failures in the calculation of the agent's criticalities and provided the failed groups with unnecessary

resources, increasing their replication degrees. Therefore, the rate of success for those values of $Kp$ declines faster compared to the rate of success for the values of $Kp$ assigned according to Eq. (2).

## 7. Conclusion

This paper examined a new approach developed for self-adaptive replication based on classical control theory. This innovative approach was dynamically used to examine the criticality of specific agents. The quantities that defined the agents' criticalities were sent via messages to the agents in order to stimulate replication services. According to the contents of the messages, one of two events occurred: either an agent began to clone itself or it removed the replicas.

Although the data indicated that the adaptive fault tolerance policy was a useful technique in terms of fault tolerance, the analysis revealed when the self-adaptive fault tolerance policy was applied, the groups' monitoring cost only slightly increased due to the observation and feedback control mechanisms.

Furthermore, the self-adaptive replication mechanism in our creative approach provided a higher performance when compared to other replication mechanisms, since they took into account only the agents' interaction and communication load in the calculation of the agents' criticality. In summary, our research suggests that effective replication can be achieved when an effective adaptive replication mechanism is systematically and accurately applied. In the future, our goal is to implement and utilize self-organizing and emergent functionality in the adaptive replication mechanism.

## References

[1] Laprie JC. Dependable computing and fault tolerance: concepts and terminology. In: 15th IEEE International Symposium on Fault Tolerant Computing; 1985; Ann Arbor, MI, USA. pp. 2-11.

[2] Kephart JO, Chess DM. The vision of autonomic computing. Computer 2003; 36: 41-50.

[3] Babaoglu O, Jelasity M, Montresor A, Fetzer C, Leonardi S, Moorsel SA, Steen M. Self-Star Properties in Complex Information Systems: Conceptual and Practical Foundations. Secaucus, NJ, USA: Springer-Verlag, 2005.

[4] Huebscher MC, McCann JA. A survey of autonomic computing: degrees, models, and applications. ACM Comput Surv 2008; 40: 1-28.

[5] Fedoruk A, Deters R. Improving fault-tolerance by replicating agents. In: 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems; July 15–19 2002; Bologna, Italy. New York, NY, USA: ACM. pp. 737-744.

[6] Guessoum Z, Briot JP. From active objects to autonomous agents. IEEE Concurr 1999; 7: 68-76.

[7] Marin O, Sens P, Briot JP, Guessoum Z. Towards adaptive fault-tolerance for distributed multi-agent systems. In: 3rd European Research Seminar on Advanced Distributed Systems; 2001; Annecy, France. pp. 195-201.

[8] Faci N, Guessoum Z, Marin O. DimaX: A fault tolerant multi-agent platform. In: Fifth International Workshop on Software Engineering for Large-Scale Multi-Agent Systems; 22–23 May 2006; Shanghai, China. New York, NY, USA: ACM. pp. 13-20.

[9] Powell D. Delta-4: A Generic Architecture for Dependable Distributed Computing. Berlin, Germany: Springer, 1991.

[10] Cristian F, Dancey B, Dehn J. Fault-tolerance in the advanced automation system. In: 20th International Conference on Fault-Tolerant Computing; 26–28 June 1990; Newcastle upon Tyne, UK. New York, NY, USA: IEEE. pp. 6-17.

[11] Elmootazbellah N, Zwaenepoel W. Replicated distributed processes in Manetho. In: Twenty-Second International Symposium on Fault Tolerant Computing; 8–10 July 1992; Boston, MA, USA. New York, NY, USA: IEEE. pp. 18-27.

[12] Bora S. Implementing fault-tolerant services in goal-oriented multi-agent systems. Adv Electr Comput En 2014; 14: 113-122.

[13] Guessoum Z, Ziane M, Faci N. Monitoring and organizational-level adaptation of multi-agent systems. In: 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems; 19–23 July 2004. New York, NY, USA: ACM. pp. 514-522.

[14] Bora S, Dikenelli O. Applying feedback control in adaptive replication in fault tolerant multi-agent organizations. In: Fifth International Workshop on Software Engineering for Large-Scale Multi-Agent Systems; 22–23 May 2006; Shanghai, China. New York, NY, USA: ACM. pp. 5-11.

[15] Bora S, Dikenelli O. Experience with feedback control mechanisms in self-replicating multi-agent systems. In: Burkhard D, Lindemann G, Verbrugge R, Varga Z, editors. Multi-Agent Systems and Applications V, 5th International Central and Eastern European Conference on Multi-Agent Systems. Berlin, Germany: Springer Verlag, 2007. pp. 133-142.

[16] Guessoum Z, Faci N, Briot J. Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform. ACM SIGSOFT 2005; 30: 1-6.

[17] Phillips CL, Harbor RD. Feedback Control Systems. Englewood Cliffs, NJ, USA: Prentice Hall, 1999.

[18] Bora S, Dikenelli O. Implementing a multi agent organization that changes its fault tolerance policy at run-time. In: Dikenelli O, Gleizes MP, Ricci A, editors. Engineering Societies in the Agents World VI, 6th International Workshop. Berlin, Germany: Springer Verlag, 2006. pp. 153-167.

[19] Almeida A, Aknine S, Briot J. Dynamic resource allocation heuristics for providing fault tolerance in multi-agent systems. In: 2008 ACM Symposium on Applied Computing; 16–20 March 2008; Fortaleza, Brazil. New York, NY, USA: ACM. pp. 66-70.

[20] Dikenelli O, Erdur RC, Gumus O, Ekinci EE, Gurcan O, Kardas G, Seylan I, Tiryaki AM. SEAGENT: A platform for developing semantic web based multi agent systems. In: Fourth International Joint Conference on Autonomous Agents and Multiagent Systems; 25–29 July 2005; Utrecht, the Netherlands. New York, NY, USA: ACM. pp. 1271-1272.

[21] Bora S, Dikenelli O. On the choice of sampling rates in a fault-tolerant multi-agent system. In: International Symposium on Innovations in Intelligent Systems and Applications Conference; 2–4 July 2012; Trabzon, Turkey. New York, NY, USA: IEEE. pp. 1-5.