Plan-based Replication for Fault-tolerant Multi-Agent Systems

Alessandro de Luna Almeida, Samir Aknine, Jean-Pierre Briot, Jacques Malenfant

***Abstract*— The growing importance of multi-agent applications and the need for a higher quality of service in these systems justify the increasing interest in fault-tolerant multi-agent systems. In this article, we propose an original method for providing dependability in multi-agent systems through replication. Our method is different from other works because our research focuses on building an automatic, adaptive and predictive replication policy where critical agents are replicated to avoid failures. This policy is determined by taking into account the criticality of the plans of the agents, which contain the collective and individual behaviors of the agents in the application. The set of replication strategies applied at a given moment to an agent is then fine-tuned gradually by the replication system so as to reflect the dynamicity of the multi-agent system.**

***Index Terms*—Adaptation, Fault tolerance, Planning, Replication**

I. Introduction

THE notion of agent (and multi-agent systems) is getting increased attention as a very promising approach for designing and building future cooperative distributed applications (e.g., crisis management systems [1], air traffic control, industrial plant automation, e-commerce, communication network management…). Reduced to the simplest terms, a multi-agent system (MAS) is a distributed system consisting of a set of agents where: each agent has incomplete information or capabilities for solving the problem; there is no global system control; data are decentralized; and computation is asynchronous [2]. Being distributed systems, MASs are susceptible to the same faults that any distributed system is susceptible to, such as software bugs, system crashes, shortage of resources, slow downs or failures in the communication links [3].

When a fault occurs in an MAS, interactions between the agents may cause the fault to spread throughout the system in unpredictable ways. This is extremely undesirable in critical applications, where the occurrence of a fault may cause the loss of lives, delays in the manufacturing process of products, or suboptimal utilization of a network.

In order to prevent this problem, many fault tolerance approaches have been proposed, most of which are based on the concept of redundancy: replication of the critical components. But in most cases, replication is decided and applied statically, before the application starts.

However, recent applications, especially those designed as multi-agent systems, can be very dynamic because of the process of reallocation of tasks, flexible organizations, replanification, changes in the roles of the agents, etc. Thus it is very difficult to decide at design time which software components (which agents) are most critical, because this may vary greatly. Replicating every agent is not a feasible approach since not only the available resources are often limited, but also the overhead imposed by the replication could degrade performance significantly.

Consequently, it is necessary to replicate in an automatic and dynamic way. This involves the study of mechanisms to determine when to replicate the agents, which agents are to be replicated, the quantity of replicas to be made and where to deploy those replicas.

In this paper, we will introduce our approach to building reliable multi-agent systems. It is based on the concept of criticality, a value (evolving in time) associated to each agent in order to reflect the effects of its failure on the overall system. This value is calculated using the plans of the agent, i.e., the actions that the agent has planned to execute in the near future.

A plan-based fault-tolerant mechanism acts as a promising preventive method since it takes into account the prediction of the future behavior of the agents and their influence over the other agents of the society. Note that our project already studied other types of information to compute criticality, e.g., based on the notion of role [4].

The remainder of this paper is organized as follows. Section 2 describes the general architecture of the middleware developed for fault-tolerant MAS and the general architecture for replication control. Section 3 explains how the plans of the agents can be used as an approach to the problem of reliability in multi-agent systems. Section 4 provides an overview of the state of the art. Finally, in section 5 we present our conclusions and perspectives for future work.

Manuscript received November 29, 2005. This work was supported in part by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), Brazil.

A. L. Almeida is with the LIP6, Université de Paris 6, 75015, FR (phone: +33-144278794; fax: +33-144277000; e-mail: Alessandro.Luna-Almeida@lip6.fr).

S. Aknine, is with the LIP6, Université de Paris 6, 75015, FR (e-mail: Samir.Aknine@lip6.fr).

J. P. Briot, is with the LIP6, Université de Paris 6, 75015, FR (e-mail: Jean-Pierre.Briot@lip6.fr).

J. Malenfant, is with the LIP6, Université de Paris 6, 75015, FR (e-mail: Jacques.Malenfant@lip6.fr).

II. General architecture

A. The DARX Framework

In our project, the first step was to design and build a framework, named DARX (as for Dynamic Agent Replication eXtension) to support dynamic replication [5]. DARX relies on the notion of replication group (RG). Every agent of the application is associated to an RG, which DARX handles in a way that renders replication transparent to the application at runtime. Each RG has exactly one ruler, which communicates with the other agents. Other RG members, referred to as subjects, are kept consistent with their ruler according to the replication strategies. Several different strategies, ranging from passive to active, may be applied within a replication group.

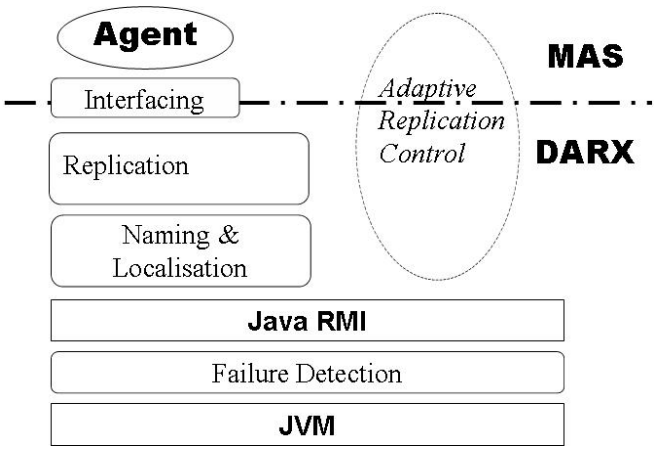

Fig. 1. DARX framework design.

As shown in Fig. 1, DARX offers several services. Failure detection enables to suspect host and process failures. Naming and localisation provides a means to supply agents and their replicas with unique identifiers throughout the system, and to retrieve their location whenever the application requires it. The replication service is used by the adaptive replication control module to provide a suitable replication scheme for every agent.

DARX is coded in Java 1.4 and uses its RMI feature as a means to simplify the coding of network issues. It can be easily integrated to any agent platform by means of an interfacing component. Current implementation provides the integration to DIMA [6] and Madkit [7] multi-agent platforms.

B. Adaptive Replication Control Architecture

The adaptive replication control module, shown in Fig. 2, was inspired by the architecture proposed by [4]. We will associate a monitoring agent to each agent of the system and a host monitor to each machine.

The monitoring agent receives the local plans of the monitored agent and is responsible for the calculation and update of its criticality. As we will see later in section 4, the computation of the criticality of an agent may rely on the criticality of other agents (because of possible dependence between their tasks). Thus their respective monitoring agents need to communicate information.

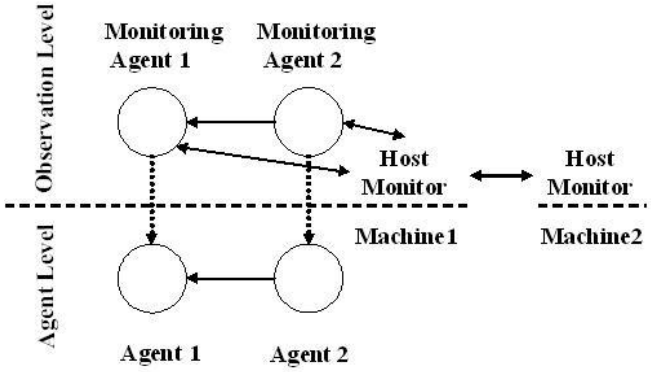

Fig. 2. Architecture for replication control.

Each host monitor contains a piece of the global information of the application, such as:

- The sum of the criticalities of the agents deployed in its machine (criticalities are obtained from the monitoring agents, as shown by the arrows in Figure 2);
- The total number of replicas in its machine and the number of replicas still available;
- The reliability of its machine.

They exchange messages with their local information in order to keep their vision about other hosts up to date (total number of replicas in the system, sum of the criticalities of all the agents, …) and, consequently, to make it viable the mechanism of replication chosen.

C. What Information to Use to Compute Criticality

In order to decide the criticality of an agent, we may use various kinds of information as inputs. In the project, we have already studied the following ones:

- System-level information: communication load, processing time [8].
- Semantic-level information: the role taken by an agent in an organization (e.g., role of broker, manager…) [4].

In this paper, we propose using another type of semantic-level information, the plans of the agent and also dependencies between their tasks.

III. Our plan-based criticality assessment method

In our model, we consider that each agent of the system knows which sequence of actions (plan) must be executed in order to accomplish its current goal. Since unexpected events may occur in dynamic environments, agents usually interleave planning and execution. Consequently, their plans are established just for the short term. We assume that at each given instant of time the agent is executing at most one action.

An action *A* is defined by an n-tuple (I, D, J, R, C, P), where:

- I is the identifier of the action;
- D is its expected duration, an approximate normalized

value, independent of the machine;

• J is the set of agents which will jointly perform the action (J may be unitary);

• R is the set of required resources;

• C is the absolute criticality of the action, a fixed and predetermined value;

• P is the set of antecedent actions, all of which must be performed before *A*. The action *A* is a *child* of each action in P.

Using the same approach established by [9], we represent the plan of an agent as a directed acyclic AND/OR graph where each node represents an action. The nodes are connected by AND or OR edges. A node *n* which is connected to *k* nodes (n_1, n_2, ..., n_k) by means of AND edges represents an action A_n after which *all* the actions A_{n1}, A_{n2}, …, A_{nk} will be executed. However, if a node *n* is connected to *k* nodes (n_1, n_2, ..., n_k) by means of OR edges, it suffices that *at least one* of the actions A_{n1}, A_{n2}, …, A_{nk} be executed after the execution of the action A_n.

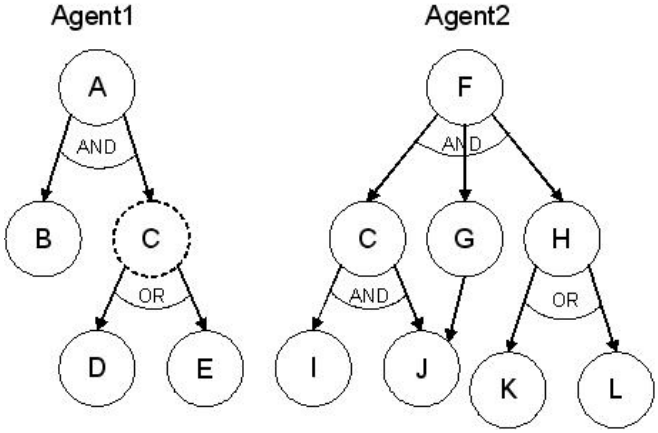

Fig. 3. Example of plans of two interacting agents.

In the example of Fig. 3, after performing the action *A*, $Agent_1$ needs to have both *B* and *C* executed in order to accomplish its plan. However, after *C*, only one of *D* or *E* needs to be performed so that $Agent_1$ accomplishes its plan.

Definition 3.1: An *external* action is an action belonging to the plan of an agent which will be executed by others. For example, consider the action *C* belonging to the plan of $Agent_1$ in Fig. 3. Since this action is performed by $Agent_2$, it is an external action in the current plan of $Agent_1$.

Definition 3.2: A *terminal* action is an action after which no other known action will be performed. In Fig. 3, *B*, *D*, *E*, *I*, *J*, *K*, and *L* are terminal actions.

A. Action Criticality

So as to calculate the criticality of an action, we distinguish its absolute criticality from its relative criticality. The *absolute criticality (AC)* of an action is defined without taking into account the current plans of the agents. It is given a priori by the system designer and can be determined in function of a number of factors:

• Number of agents capable of performing the action: an action that can be done by many agents can be considered not too critical, since it is probably easier to reschedule it, if it ever fails, than if only a few agents were capable of performing it.

• Duration of the action: under some circumstances, actions which are expected to take too long can be considered more critical than short ones.

• Resources required for the execution of the action: the size of the set of required resources can also be used to determine the absolute criticality. A large set may lead to a more critical action than a small one.

• Semantic information: the system designer can use semantic information to determine the criticality of the action, since, depending on the field of application, some actions are more important than others.

The *relative criticality (RC)* of an action belonging to the plan of an agent is proportional to the criticality of the agent when it is executing the action or waiting that some other agent executes it. As a consequence, the relative criticality of an action may be different for each agent whose plan it belongs to. If the action is external, its relative criticality depends only on the relative criticality of the actions that the agent will execute afterwards.

If the agent is executing the action (possibly jointly with other agents), the relative criticality reflects the importance of the action in the multi-agent system. In this case, it depends on the absolute criticality of the action and on the usefulness of its results to all the agents which depend on it to perform their tasks. In other words, in order to determine the relative criticality of an action executed by an agent, we must estimate the impact of its failure to the multi-agent system as a whole.

The relative criticality is calculated as follows:

• For an external action, it is equal to the *local relative criticality (LRC)*. The LRC is obtained using the AND-aggregation function if the action is connected to its children by means of AND edges or the OR-aggregation function if it is connected by OR edges. The parameters of these two functions are the relative criticalities of the children of the action. We use as an AND-aggregation function the sum of its parameters and as OR-aggregation function, the mean of its parameters. If the action has only one child, its LRC is equal to the relative criticality of its child. If the action is terminal (i.e. it has no child), its local relative criticality is equal to zero.

• For a non-external action *a*, its relative criticality is equal to its absolute criticality plus the sum of the local relative criticalities of *a* in each plan to which it belongs.

Table I shows the relative criticalities of each action in the example of Fig. 3 if the corresponding absolute criticalities are considered.

TABLE I
CALCULATION OF CRITICALITY

Action	Absolute Criticality	Relative Criticality
A	4	15
B	8	8
C (Agent1)	4	3
C (Agent2)	4	13
D	5	5
E	1	1
F	6	30
G	3	7
H	2	4
I	2	2
J	4	4
K	3	3
L	1	1

In order to obtain those values for the relative criticalities, the method previously described is used. For example, the action *B* belonging to the plan of $Agent_1$ is a non-external action. Then its relative criticality is calculated by adding its absolute criticality with its local relative criticality. Since it is a terminal action, its local relative criticality is equal to zero.

$$RC(B) = AC(B) + LRC(B, Agent1) = 8 + 0 = 8$$

However, we calculate the relative criticality of the action *C* in the plan of $Agent_1$ differently because it is an external action (it will be executed by $Agent_2$). In this case, the relative criticality is simply equal to the value of the local relative criticality. In order to calculate the LRC of *C* in $Agent_1$'s plan, we use the mean aggregation function with the relative criticalities of the children of action *C* (namely *D* and *E*) as parameters.

$$RC(C, Agent1) = LRC(C, Agent1) =$$
$$Mean(RC(D), RC(E)) = Mean(5, 1) = 3$$

The problem with this approach is that it does not consider the time when the actions will actually start to be executed. In fact, using the strategy just described above, one terminal action which is located far from the root in the agents' graph (and possibly having a very late start time) has an equal impact on the final relative criticality of the root as another action with the same absolute criticality and nearer to the root.

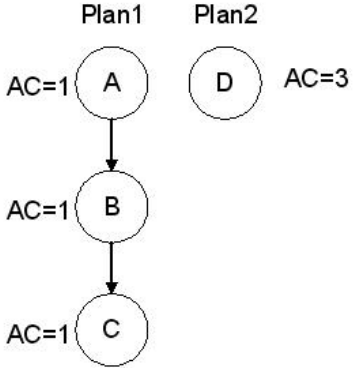

Fig. 4. Impact of time in the criticality of actions.

Nevertheless, in dynamic environments, actions with an estimated late start time will be executed less possibly than actions with early start time. In Fig. 4, action *A* in $Plan_1$ and action *D* in $Plan_2$ would have a relative criticality of 3. However, due to the fact that it is not certain that actions *B* and *C* are needed to be executed by the agent, action *D* is more critical in $Plan_2$, than the action *A* in $Plan_1$.

Consequently, we also propose another approach to calculate the relative criticalities, where we multiply the relative criticality of the actions by a factor which varies along time, taking into account the expected time that the action will start to be executed. Let *t* be the estimated starting time of the action and RC_{old} its relative criticality calculated using the previous approach. Then we calculate the relative criticalities in the new approach (RC_{new}) using the following exponentially decreasing function:

$$RC_{new} = RC_{old} / b^t, \text{ where } b \geq 1$$

We compute the estimated starting time of the actions using a topological sorting in the graph (top-down) considering the elapsed times of the antecedents and siblings' actions [10].

In the example of Fig. 4, if we consider that the duration of all the actions is equal to three units of time, and the amortizing base *b* is equal to *e*, the following relative criticalities would be obtained:

TABLE II
CALCULATION OF CRITICALITY CONSIDERING TIME

Action	Relative Criticality
A	$RC(A) = (AC(A) + LRC(A, Agent))/e^{ta} = (AC(A) + RC(B))/e^{ta} = (1 + e^{-3} + e^{-9})/e^0 = 1 + e^{-3} + e^{-9}$
B	$RC(B) = (AC(B) + LRC(B, Agent))/e^{tb} = (AC(B) + RC(C))/e^{tb} = (1+ e^{-6})/e^3 = e^{-3} + e^{-9}$
C	$RC(C) = AC(C)/e^{tc} = 1/e^6 = e^{-6}$
D	$RC(D) = AC(D)/e^{td} = 3/e^0 = 3$

Using this new mechanism for calculating the relative criticalities, the action *A* in $Plan_1$ is less critical than the action *D* in $Plan_2$, as desired.

B. Agent Criticality

The criticality of an agent can be calculated based on the criticalities of the actions which belong to its plan. An agent who executes important actions should be considered critical.

In a given time *t*, the criticality of the agent will be given by the relative criticality of the current root of its plans' graph.

Since multi-agent systems are often dynamic and non-determinist, it is not possible to know in advance the complete plan of the agent. Actually, during the execution of multi-agent plans, one or more agents might determine that the context has changed so much that the agents' partial plan should be modified. For example, this can be due to: lack of resources, dynamicity of the agent society (agents can enter or leave the society), impossibility of other agents to accomplish its engagements, etc.

Consequently, the initial criticality of the agents in the instant $t = 0$ is quite precise, but it needs to be updated along

time. The question is when and how to update those criticalities. We propose two main types of strategies to revise the criticality: *time-driven strategies* and *event-driven strategies*.

Time-driven strategies are based on local clocks associated to each agent. Whenever the clock alarms, the criticality of the corresponding agent is re-evaluated. The interval of time between two consecutive alarms can be *fixed* or *variable*. Using an initial approach, at each fixed interval Δt, the clock will sound the alarm and the criticality will be updated. The value of Δt could be variable so as to reflect the dynamicity of the system. If this is the case, the length of the interval is initially set to a predefined value. It is reduced if a substantial modification in the criticality has been noticed in the last interval of time or, inversely, it would be increased if almost no change has been observed in the criticality.

Event-driven strategies are based on critical events that might change the criticality in a substantial way. Whenever one of these events is detected, the criticality is updated. There are two main types of events: those which depend on the application (completion of an action, changes in the plan of the agent, …) and those related to failures (failure of an agent or a machine).

C. Agent Replication Mechanism

In [4], an agent replication mechanism has been proposed to decide which agents to replicate and how many copies to make. In this previous work, an agent $Agent_i$ is replicated according to:

- c_i: his criticality;
- C: the sum of criticality of all agents in the system;
- *min*: the minimum number of replicas that an agent must have (introduced by the designer);
- *max*: the quantity of replicas available to the system.

The number of replicas n_i of the agent $Agent_i$ can be determined as follows:

$$n_i = rounded(min + (c_i \times max / C))$$

In other words, it is directly proportional to the number of available resources and inversely proportional to the sum of criticality of all agents in the system. It is important to notice that the value *max* is equal to the number of replicas available after allocating the *min* replicas to each agent of the system. If *QR* is the quantity of replicas available in the whole system in a given instant of time and *QA* is the quantity of agents in the system,

$$max = QR - (QA \times min)$$

At each interval of time Δt, for each agent, the value of n_i is calculated and used to update its number of replicas. If the agent possesses more replicas than it is allowed, it is obligated to free the corresponding exceeding quantity. Conversely, if the agent holds fewer replicas than it should, it chooses randomly n replicas, where n is the difference between the current value of n_i and its previous value.

One problem with this technique of calculating the number of replicas that should be given to each agent is that it does not take into consideration the future failure probability of the replica. In fact, it is better to have only one replica which will have in the future an almost zero probability of failure (since it will be deployed in a very robust machine) than having many replicas which are not reliable. Additionally, it does not address the problem of where deploy efficiently the replica.

Hence, we will propose another mechanism of replica allocation, which considers the probability of crash in the machines. In this new mechanism, we define the *value* of the replica r_k (denoted by v_k), as the probability that its machine will not crash. A value of one will be attributed to a completely reliable resource, whereas an unreliable one shall have a near zero value. The probability of failure of a given set of replicas $R = \{r_1, r_2, \dots, r_n\}$, is given by:

$$P\,(Failure\,(R) = 1) = (1\text{-}v_1) \times (1\text{-}v_2) \times \dots \times (1\text{-}v_n)$$

Let S be the sum of the values of all the replicas in the system. Then, an agent $Agent_i$ is allowed to be replicated using a total value of replicas (t_i) proportional to the percentage of its criticality (c_i) with respect to the sum of agents' criticalities (C), as given by the equation:

$$t_i = c_i \times S / C$$

The system of replication will then allocate to the agent the set of replicas $R = \{r_1, r_2, \dots, r_n\}$, such that $v_1 + v_2 + \dots + v_n \leq t_i$ and its probability of failure is minimal among all the possible sets of replicas.

One can apply the same possible strategies used as the agent criticality update policy (time-driven or event-driven) to decide when to re-calculate the values of t_i. For instance, one can use a variable window of time Δt for each agent $Agent_i$. If the quantity of replicas (whose total value does not exceed t_i) that the agent $Agent_i$ can acquire does not change significantly, the window of time Δt can be increased, otherwise it is decremented. Another possibility is to recalculate the value of t_i whenever the value of c_i is updated.

IV. Related Work

Several approaches have addressed the multi-faceted problem of fault tolerance. In fact, many toolkits include replication facilities to build reliable applications. However, most of them are not quite suitable for implementing large-scale, adaptive replication mechanisms.

Hägg [11] proposes an approach to the problem in which sentinel agents monitor inter-agent communication, build models of other agents and take corrective actions. Since the sentinels analyze the entire communication going on in the system to detect state inconsistencies, it would be far too expensive in terms of computation and communication to take total control of possible fault situations and global consistency. Additionally, sentinels are themselves points of failures.

Decker et al [12] describe different levels of adaptation, but concentrates only on execution adaptation where agent cloning is used in load balancing. Fault tolerance aspects are not addressed. Furthermore, they do not propose mechanisms such as merging of two agents or self-extinction of

underutilized agents to control agent proliferation.

Kumar et al [13] propose a fault tolerant architecture of brokers and AgentScape middleware offers a replication service to achieve fault tolerance [14]. However, in both cases, agent failure is not completely dealt with, since only some agents (brokers) or part of them can be replicated.

Fedoruk and Deters [3] also use replication to improve fault tolerance. Their work implements the passive strategy (hot-standby) of replication in a transparent way using proxies. All messages going to and from a replicate group are funnelled through the replicate group message proxy. Kraus et al [15] define the problem of fault tolerance as a deployment problem and propose a probabilistic approach to deploy the agents in a multi-agent application. The main problem of these two works is that replication is applied statically before the application starts. This is not desirable in the case of dynamic and adaptive multi-agent applications because the criticality of agents may evolve dynamically during the course of computation.

There are some software infrastructures for adaptive fault tolerance [16]–[18] where existing strategies can be dynamically changed. Nevertheless, such a change must have been devised by the application developer before runtime or the modifications must be specified and applied in a non-automatic way during the execution of the system.

V. Conclusion

In recent years, research on multi-agent systems has addressed the problem of agent reliability since they must often run without any interruption. To make these systems reliable, we proposed an original predictive method to evaluate dynamically the criticality of agents. Our approach takes profit of the specificities of multi-agent applications and analyses the agents' plans to determine their importance to the system. The agent criticality is then used to replicate agents in order to maximize their dependability based on available resources.

To validate the proposed approach, we are using the combination of the DARX framework and the DIMA multi-agent platform in order to develop our plan-based replication strategy. In this integration, an agent is implemented as a DIMA agent and uses DARX in order to acquire replication capabilities.

The implementation phase is very advanced (we have already implemented the algorithms to determine the agent's criticality and we are finishing the new agent replication mechanism). We are going to start very soon the experimentation phase using two different real-world applications (the personal meeting assistants and the patrolling agents [19]) so that we can compare our new approach to the previously developed ones.

Acknowledgment

The authors would like to thank the members of the fault-tolerant multi-agent systems project for the fruitful discussions.

References

[1] H. Boukachour, C. Duvallet, A. Cardon, "Multiagent systems to prevent technological risks," in *Proc. of International Conference on Artificial and Computational Intelligence for Decision Control and Automation in Engineering and Industrial Application (ACIDCA'2000)*, Monastir, Tunisie, March 2000.

[2] K. Sycara, "Multiagent systems," *AAAI AI Magazine*, vol. 19, no. 2, pp. 79-92, 1998.

[3] A. Fedoruk, R. Deters, "Improving fault-tolerance by replicating agents," In *Proc. AAMAS-02*, Bologna, 2002, pp. 737-744.

[4] Z. Guessoum, J.-P. Briot, O. Marin, A. Hamel, P. Sens, "Dynamic and adaptive replication for large-scale reliable multi-agent systems," In *Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, LNCS 2603, pp. 182-198, April 2003.

[5] O. Marin, P. Sens, J.-P. Briot, Z. Guessoum, "Towards adaptive fault-tolerance for distributed multi-agent systems," In *Proc. of ERSADS'2001*, Bertinoro, Italy, May 2001.

[6] Z. Guessoum, J.-P. Briot, "From active objects to autonomous agents,". *Special Series of Actors and Agents, IEEE Concurrency*, vol. 7, no. 3, pp. 68-76, 1999.

[7] MadKit Web Site. Available: http://www.madkit.org/

[8] Z. Guessoum, J.-P. Briot, N. Faci. "Towards fault-tolerant massively multiagent systems," In *Toru Ishida, Les Gasser and Hideyuki Nakashima editors, Massively Multi-Agent Systems*, Lecture Notes in Computer Science, Springer Verlag, to be published.

[9] B. Horling et al., "The TAEMS White Paper," January 1999.

[10] Hillier and Lieberman, *Introduction to Operations Research*. Third Edition. Holden-Day Inc, pp. 246-259.

[11] S. Hägg, "A sentinel approach to fault handling in multi-agent systems," In *Proc. of the Second Australian Workshop on Distributed AI*, Cairns, Australia, August 27, 1996.

[12] K. S. Decker, K. Sycara, "Intelligent adaptive information agents," *Journal of Intelligent Information Systems*, vol. 9, pp. 239 - 260, 1997.

[13] S. Kumar, P.R. Cohen, H.J Levesque, "The adaptive agent architecture: achieving fault-tolerance using persistent broker teams," In *Proc. Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, Boston, MA, USA, July 7-12, 2000.

[14] F.M.T. Brazier, M. van Steen, N.J.E. Wijngaards, "Distributed shared agent representations," *Multi-Agent-Systems and Applications II, Lecture Notes in Computer Science*, vol. 2322, pp. 213-220.

[15] S. Kraus, V.S. Subrahmanian, N. Cihan, "Probabilistically survivable MASs," In *Proc. of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Aug. 2003, pp. 789-795.

[16] Z. Kalbarczyk, S. Bagchi, K. Whisnant, R.K. Iyer,. "Chameleon: a software infrastructure for adaptive fault tolerance," *IEEE Transactions on Parallel and Distributed Systems*, June 1999, pp. 560-579.

[17] M. Cuckuern et al, "AQuA: an adaptive architecture that provides dependable distributed objects," In *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98)*, West Lafayette, Indiana, October 20-23, 1998, pp. 245-253.

[18] F. Favarim, F. Siqueira, J. S. Fraga, "Adaptive fault-tolerant CORBA components," In *Middleware Workshops 2003*, pp. 144-148.

[19] A. L. Almeida et al, "Recent advances on multi-agent patrolling," In *Proc. SBIA 2004*, pp. 474-483.