Towards Adaptive Fault Tolerance For Distributed Multi-Agent Systems

Olivier Marin [1,2], Pierre Sens[2], Jean-Pierre Briot[2], Zahia Guessoum[2]

[1]Laboratoire d'Informatique du Havre
University of Le Havre
25 rue Philippe Lebon
BP540 76058 Le Havre cedex

[2]Laboratoire d'Informatique de Paris 6
University Paris 6 - CNRS
4 place Jussieu
75252 Paris Cedex 05, France

emails:
Olivier.Marin@univ-lehavre.fr,
Pierre.Sens@lip6.fr,
Jean-Pierre.Briot@lip6.fr,
Zahia.Guessoum@lip6.fr

***Abstract*— This paper studies how to bring flexibility to fault-tolerant systems. Firstly, multi-agent systems are identified as a very valuable basis for reaching this goal, and reliability is also shown to be a rare and attractive feature for such systems. We then propose a framework for building applications that provide adaptive fault tolerance, and put forward the promising results obtained when testing the implementation of this framework. We conclude with drawing some perspectives of evolution of our work.**

Keywords: decentralized systems, multi-agent programming framework, fault tolerance, replication

I. Introduction

A priori, the main motivation for multi-agent systems lies in the distributed nature of information, resources and action. It seems also intuitive that one of the fundamental issues of distributed computer systems is the possibility of host or network failures. However, it is to be noticed that most of the current distributed multi-agent platforms and applications do not yet address, in a systematic way, this possibility of failures [NS00] [MCM99]. The main explanation appears to be that most multi-agent systems and applications are still developed on a small scale:

- they run on a single computer or on a few highly coupled (farm of) computers.
- they run for short-timed experiments.

In the distributed systems community some abstractions (group communication, replication, atomic multicast, ...) have been proposed to ensure some dependability of critical systems. But the design and application of such dependability protocols have mostly been static, in the sense that they do not change once the application starts. Fortuitously, another important issue for multi-agent systems resides in dynamic applications, where the relative significance of the different entities involved may change during the course of computation. An example of application domain is the field of crisis management systems [BDC00] where software is developed in order to assist various teams in the process of coordinating their knowledge and actions. Possibility of failures is high and the criticality of each element, should it be an information server or an agent assistant, evolves during the management of the crisis. A concrete example can be that of a toxic gas spreading in an urban area, the crisis being handled with the help of a multi-agent application where each agent assists a public service team sent on the spot. The importance of the paramedical teams is then foremost, and the assisting agents require a high degree of dependability; but it can appear finally that the gas is harmless. If panic has stricken the population nonetheless, it is at this moment the law and order teams which are to be assisted the most reliably. One naive solution would be to apply dependability protocols, for example replication schemes, to every element. But this is not feasible in practice because these protocols are costly; thus one needs to apply them optimally when and where they are most needed.

In this paper, we propose an architecture for fault-tolerant computing. As opposed to main conventional distributed programming architectures, our architecture offers dynamic properties: software elements can be replicated and unreplicated on the spot and it is possible to change the current replication strategies on the fly. We have developed a solution to interconnect this architecture with a multi-agent platform, namely DIMA [GB99]. We consequently conducted some simple experiments to evaluate our architecture.

The paper is organized as follows. Section 2 briefly presents replication issues in multi-agent systems, as well as a quick insight on replication in distributed systems. Section 3 describes the goals we intended to achieve through developing a solution that would allow the elaboration of applications liable to provide adaptive fault tolerance. Section 4 presents the general design of our proposal for a framework dedicated to reliable multi-agent systems. Section 5 reports on the performances of the software we implemented on the basis of the solution we proposed. Finally, perspectives are drawn in section 6.

II. Related Work

A. Fault tolerance in multi-agent systems

Within the family of reactive multi-agent systems, some systems offer high redundancy. A good example is a system based on the metaphor of ant nests [DCF95], where a common task is fulfilled by a multitude of very basic agents which present exactly the same behaviour and the same characteristics at the beginning ; that is, they share the same code. Unfortunately:

- it is very hard and time-consuming to design applications in terms of such reactive multi-agent systems; basically, nobody has yet found the right methodology,
- such a simple redundancy scheme cannot be applied onto more cognitive multi-agent systems as this would increase the advent of inconsistencies between copies of a single agent; indeed, the more advanced and complex agent definitions involve intrinsic non-determinism.

Some solutions [DSW97] [SN98] offer dynamic cloning of specific agents in multi-agent systems. The motivation is different, though: to improve the availability of an agent in case of congestion. Such work appears to be restricted to agents having functional tasks only, and no changing state. Thus it doesn't represent an appropriate support for fault tolerance.

Other solutions [BCS99] [GH00] enable some reliability through persistence on stable storage. This methodology, however, does not solve situations which involve the definite failure of a machine hosting an agent. Besides, recovery delays become hazardous and computations may not be fully restored.

B. Replication in distributed computing

The replication of data and/or computation is the only efficient way to achieve fault tolerance in distributed systems. A replicated software component is defined as a software component that possesses a representation on two or more hosts [GS97]. There are two main types of replication protocols:

1) the **active** one in which all replicas process every input message concurrently,
2) and the **passive** one in which all input messages get processed by a single replica which transmits its current state periodically to the other replicas in order to maintain consistency.

Active replication strategies lead to a high overhead. If the degree of replication is n, which means there are n replicas involved, there will be n treatments to produce one result. Passive replication economizes processor utilization by activating redundant replicas in case of failures only. That is: if the active replica is found to be faulty, a new replica is elected among the set of passive ones and the execution is restarted from the last saved state. This technique requires less CPU resources than the active approach but it involves a checkpoint management which remains expensive in processing time and space.

The active replication provides a fast recovery delay. This kind of technique is dedicated to applications with real-time constraints which require short recovery delays. The passive replication scheme has a low overhead under failure free execution but does not provide short recovery delays.

The choice of the most suitable strategy is directly dependent of the environment context, especially the failure rate, and the application requirements in terms of recovery delay and overhead. Active approaches should be chosen either if the failure rate becomes too high or if the application design specifies hard time constraints. In all other cases, passive approaches are preferable.

Many toolkits include replication facilities to build reliable applications [RBS96]. However, most products are not flexible enough to implement an adaptive replication mechanism.

MetaXa [GOL98] implements active and passive replication in a flexible way using Java. The authors extended Java with a reflective meta-level architecture. A very interesting feature is that the provided replication scheme is transparent to the supported application. However, MetaXa relies on a modified Java Virtual Machine.

GARF [GGM96] realizes fault-tolerant Smalltalk machines using active replication. Similar to MetaXa, GARF uses a meta level but does not provide different replication strategies.

III. Objectives and Approach

As mentioned earlier, the intended achievement of this work is twofold:

1) provide efficient fault-tolerance to multi-agent systems through selective agent replication,
2) take advantage of the specificities of multi-agent platforms to develop a suitable architecture for performing dynamic fault-tolerance within applications.

Fundamentally, multi-agent applications rely on the collaboration amongst agents. It follows that the failure of one of the involved agents can bring the whole computation to a dead end. Replicating specific agents which are identified as crucial to the application may allow to bypass easily this problem.

However, one must keep in mind that replication may often be very costly in processing as well as in communications. Moreover, a software element may loose in criticality at some point of the application's progress. Therefore, it is important to be able to go back on the previous choices and replicate other elements. The intrinsic properties of multi-agent systems, in particular the fact that they are very flexible and dynamic, point them out as a strong basis for elaborating a fault-tolerant mechanism likely to allow on-the-fly changes.

From the start, it was decided to try and keep the solution as independent as possible from the overlying multi-agent platform, so as to be still valid in case of drastic evolutions of the platform. Furthermore, this constitutes a first step towards enabling the developed architecture to be reusable by any multi-agent system, and eventually offering interoperability between such systems.

For portability and compatibility issues, it was chosen that the architecture would be Java-based. Indeed, the Java language and more specifically the JVM provide – relative – hardware independence, an invaluable feature for distributed systems. Moreover, a great number of the existing multi-agent platforms are implemented in Java. In addition to all this, the remote method invocation (RMI) facility offers many useful high-level abstractions for the elaboration of distributed solutions.

Finally and most important of all, in order to simplify the tackling of the problem, the following hypothesis was made: the assumed system model would be synchronous at first, and the obtained solution ought to be conceived in such a way that it could be extended later for asynchronous models. This implies that an upper bound on communication delays is assumed for failure detection; upgrading prospects are presented in VI.

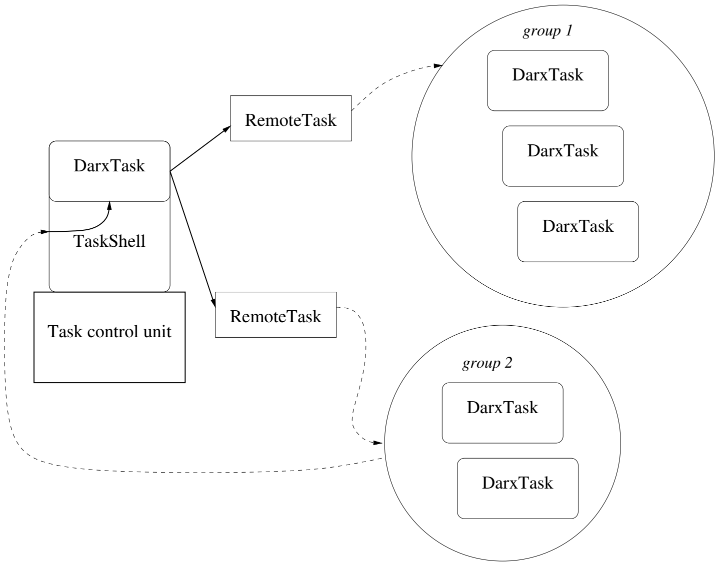

Fig. 1. DARX application architecture

IV. The DARX Framework

In this section, we present our solution, DARX, and roughly depict its features.

A. Design and Architecture

DARX is a framework to design reliable distributed applications. Each task can be replicated an unlimited number of times and with different replication strategies (see the two following sections). DARX includes group membership management to dynamically add or remove replicas. It also provides atomic and ordered multicast inside every replication group.

A replication group is an opaque entity underlying every application task. The number of replicas and the internal strategy of a specific task are totally hidden to the other application tasks. Each replication group has exactly one leader which communicates with the other tasks. The leader also acts as a fixed sequencer, providing totally-ordered multicast and failure detection within its replication group.

DARX provides a global naming service, whereas RMI does not include this functionality. To each application task corresponds a generic name which is independent of the current location of the replication group elements. The naming service is called upon when the ongoing location of the leader in charge of a generic application task is requested. In case of failure of a leader, it is also the naming service which is responsible for electing a new one among the set of remaining replicas; thus ensuring at all times that every replication group comprises exactly one leader.

In order to benefit from the fault tolerance abilities offered by DARX, each agent gets to inherit the functionalities of a `DarxTask` object (Figure 1), enabling the underlying system to handle the agent execution and communication. It thus becomes possible for DARX to act as an intermediary for the agent, committed to deciding:

- when an agent should really be started, stopped, suspended and resumed,
- and exactly when a message reception should take effect.

Each task is itself wrapped into a `TaskShell`, which acts as a replication group manager (see IV-B) and is responsible for delivering received messages to all the members of the replication group, thus preserving the transparency for the supported application. Input messages are intercepted by the `TaskShell`, enabling message caching. Hence all messages get to be processed in the same order within a replication group, and messages duplicated by mistake can be discarded. When replicating an agent, its replication group is suspended and the corresponding `DarxTask` is copied to a newly created `TaskShell` on the requested host before resuming the group execution.

A task can communicate with a remote task, disregarding whether it is a single agent or a replication group, by using a local proxy implemented by the `RemoteTask` interface. Each `RemoteTask` references a distinct remote entity considered as the leader of its replication group.

B. Replication and Failure management

DARX provides both passive and active replication strategies. The main originality is that the strategy is not fixed. Each agent can change, at runtime, its replication protocol and tune internal parameters such as the number of replicas or the period between back-ups in case of passive replication. As mentioned earlier, the replication features of each agent are totally transparent to the others.

A replication manager is associated to each agent. It provides four functionalities:

- **group information maintenance:** It keeps track of all the replicas included in the group, and of the current strategy in use.
- **activity suspension/resumption:** It is responsible for suspending or restarting the group. Suspension is necessary when the internal parameters of the group, such as the number of members or the strategy, are changed. Theses functions depend on the current strategy. In the active replication scheme, group suspension implies sending a suspend order to each replica. In the passive one, only the leader has to be suspended.
- **message diffusion:** It provides the means to propagate communications within the replication group. This function also varies following the current strategy. For example, in the active replication scheme, every message will be broadcasted to the replicas; whereas in the passive one, the message will only be computed by the leader of the replication group.
- **replication strategy switching:** Finally, the replication manager carries out the change of the current replication policy, and the tuning of its parameters. When switching from an active to a passive scheme, the operation solely consists in informing the replicas of the policy change; whereas in the opposite case, the whole group is suspended and all the replicas are updated before resuming the execution.

This replication manager represents a fast way of handling failure recovery. When the active replica of a replication group fails, any other replica within the group already possesses the right information in order to be elected as the new active replica.

V. PERFORMANCE

This section presents a performance evaluation of the basic DARX components. Measures were obtained using JDK 1.1.6 on a set of Ultra-SparcII 333 MHz linked by a Fast Ethernet (100 Mb/s).

A. Migration

Firstly, the cost of adding a new replica at runtime is evaluated. In this protocol, a new DARX task is created on a remote host and the leader sends its local data to the new replica. This mechanism is very close to a task migration.

Figure 2 shows the time required to “migrate” a server as a function of its data size. A relatively low-cost migration is observed. For a 1 megabytes server, the time to add a new copy is less than 0.6 seconds.

The performance of our server migration mechanism is also compared with the Voyager framework, which itself provides a migration facility to move agents across the network. In this particular test the server is moved between two Pentium III/550MHz PCs running linux with JDK 1.1.8. As shown in Figure 3, DARX is generally more than twice faster than Voyager. The anomalies in this graph come from the fact that the performances

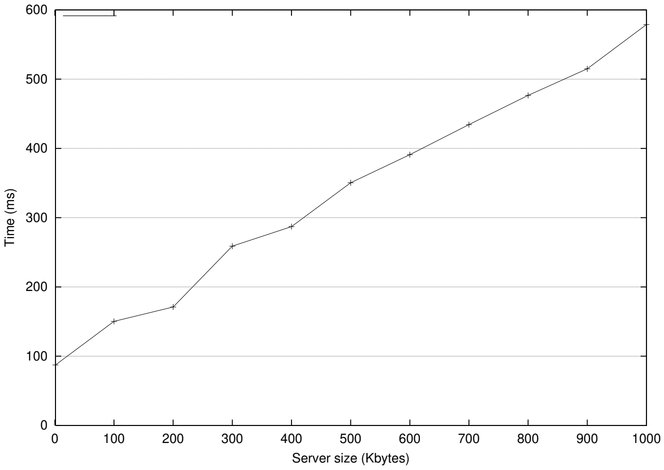

Fig. 2. Server migration cost

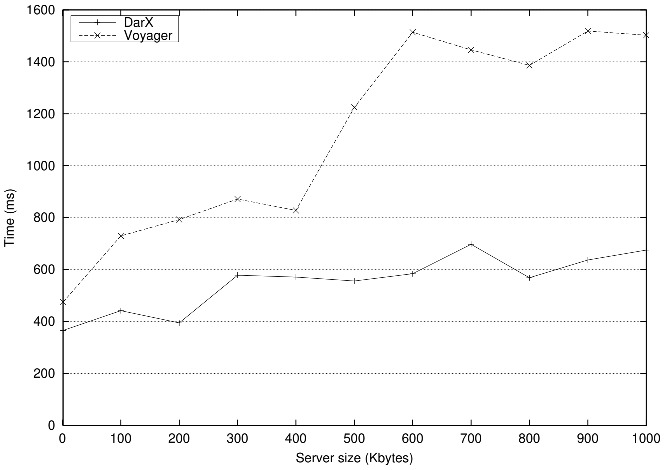

Fig. 3. Comparison of Server migration costs

were measured in a non-dedicated environment subject to chaotic variations.

B. Active replication

The cost of sending a synchronous message to a replication group using the active replication strategy is then evaluated. The time displayed is that measured between the emission of a message whose size is given in abscissa and the reception of a basic reply. Each message is sent to a set of replicas. Figure 4 presents three configurations with different replication degrees. In the RD-1 configuration, the task is local and not replicated. In the RD-2 configuration, the task is replicated on a remote host. In the RD-3 configuration, there are three replicas; the leader being on the sending host and the two other replicas residing on two distinct remote hosts.

C. Passive replication

To estimate the cost induced by passive replication, the time to update remote replicas is measured. The updating of a local replica was set aside as the obtained response times were too small to be significant. Figure 5 illustrates the measured performances when updating a replication group at different replication degrees.

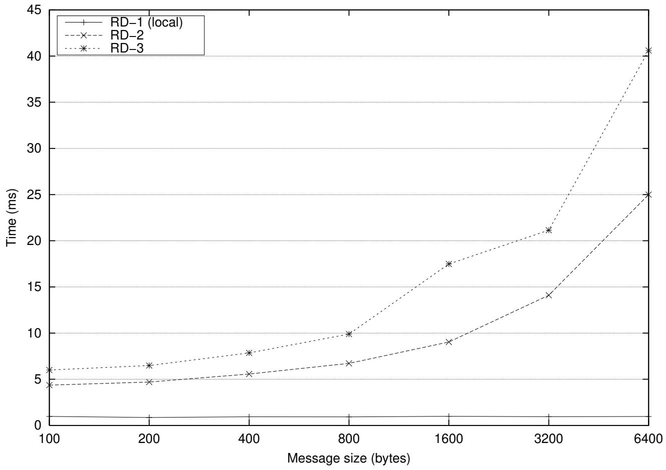

Fig. 4. Communication cost as a function of the replication degree

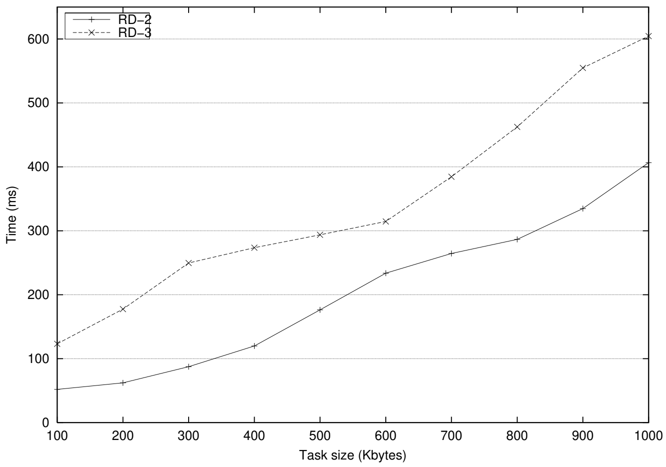

Fig. 5. Update cost as a function of the replication degree

D. Replication policy switching

Finally, we evaluate the times required to switch replication strategies:

- in the RD-2 configuration, it takes 4.5ms to proceed from the active strategy to the passive one;
- whereas in the RD-3 configuration, 4.56ms are required for the same operation.

VI. Conclusion

We have presented in this paper a framework for fault-tolerant multi-agent systems. Our architecture does not only provide the means to build reliable distributed applications, but also brings a certain amount of flexibility to such software. Indeed it gives one the possibility to decide which parts of the computation are more critical than the others, and hence should be made to bypass failures through replication. It moreover allows to control **dynamically** the way the application safeguards its components, enabling the reliability of the computation to be fine-tuned on the fly.

The proposed architecture is a reality: it has been implemented using Java and its RMI facility and presently interfaces with the DIMA multi-agent platform. Tests have already been conducted with the current versions of DARX and DIMA; as shown in the above section, the results are promising.

There are still some issues left unsolved as of today, however. For instance, the global naming system mentioned in section IV-A constitutes DARX's main weak spot as it is centralized and based on a synchronous system model. We are presently studying the creation of a distributed global naming service that would fully benefit from the characteristics of multi-agent systems, as well as integrating asynchronous failure detection [HN99].

Another field to work on is the way the replication schemes are chosen. Presently, it is the sole application designer's responsibility to decide in advance, for every component:

- which one is to be replicated,
- how many replicas must be made,
- where those replicas should be located,
- which strategy should be used,
- and when a given replication scheme ought to be modified.

Within the DARX project, research on the artificial intelligence side is currently under way concerning agent-monitored decisions [LBGS00]. We are looking into decentralized solutions for observing the application behaviour so as to provide an optimized fault tolerance for each agent. The low-level informations collected by DARX, once passed on to the multi-agent system and matched with the agent semantics and needs, could be used to automatically reach intelligently fine-tuned decisions regarding the right replication schemes to choose.

Other system-specific research issues include pinpointing the common needs and constraints of multi-agent platforms in terms of mobility and communication, so as to provide interfacing with other multi-agent systems, and furthermore offer a simple way of interoperating them.

In order to validate the work achieved up until now, several applications are being developed. Those include a distributed agenda, as well as a basic crisis management system destined to test viability and utility of our architecture in terms of such software.

REFERENCES

[BCS99] P. Bellavista, A. Corradi and C. Stefanelli, "A Secure and Open Mobile Agent Programming Environment" In *Proc. Fourth International Symposium on Autonomous Decentralized Systems (ISADS '99)*, Tokyo, Japan, March 21-23, 1999, pages 238-245, IEEE Computer Society Press.

[BDC00] H. Boukachour, C. Duvallet and A. Cardon, "Multiagent systems to prevent technological risks" In *Proceedings of IEA/AIE'2000*, Springer Verlag 2000.

[DCF95] Drogoul A., Corbara B. and Fresneau D, "MANTA : New experimental results on the emergence of (artificial) ant societies" In *Artificial Societies: the computer simulation of social life*, Nigel Gilbert & R. Conte (Eds), UCL Press, London, 1995

[DSW97] K. Decker, K Sycara and M. Williamson, "Cloning for Intelligent Adaptive Information Agents." In *Multi-Agent Systems: Methodologies and Applications*, Lecture Notes in Artificial Intelligence 1286, Zhang, C. and Lukose, D., (eds.), Springer, 1997, pages 63–75.

[GB99] Z. Guessoum and J.-P. Briot, "From active objects to autonomous agents" In *Special Series on Actors and Agents*, edited by Dennis Kafura and Jean-Pierre Briot, IEEE Concurrency, 7(3):68-76, July-September 1999.

[GH00] IKV++, "Grasshopper - A Platform for Mobile Software Agents" http://213.160.69.23/grasshopper-website/links.html

[GOL98] M. Golm, "MetaXa and the Future of Reflection" In *OOPSLA Workshop on Reflective Programming in C++ and Java* , October 18, 1998, Vancouver, British Columbia.

[GGM96] R. Guerraoui, B. Garbinato and K. Mazouni, "Lessons from Designing and Implementing GARF" In *Objects Oriented Parallel and Distributed Computation*, Lecture Notes in Computer Science 791, page 238–256, Springer Verlag 1996

[GS97] R. Guerraoui and A. Schiper "Software-based replication for fault tolerance" In *IEEE Computer*, 30(4):68-74, April 1997.

[HN99] S. Haddad and F. Nguilla, "Combining Different Failure Detectors for Solving a Large-Scale Consensus Problem" In *14th ISCA-CATA* Cancun, Mexico, April 1999.

[LBGS00] G. Lacôte, J.-P. Briot, Z. Guessoum and P. Sens, "Towards Fault-Tolerant Agents" In *ECOOP'2000 Workshop on Distributed Objects Programming Paradigms*, Cannes, juin 2000.

[MCM99] D. Martin, A. Cheyer and D. Moran "The Open Agent Architecture: A Framework for Building Distributed Software Systems" In *Applied Artificial Intelligence*, 13(1-2):91-128, January-March 1999.

[NS00] Niranjan Suri et al.,, "An Overview of the NOMADS Mobile Agent System" In *Proceedings of ECOOP'2000*, Nice, France, 2000.

[RBS96] R. van Renesse, K. Birman, and S. Maffeis, "Horus: A flexible group communication system", In *CACM*, 39(4):76–83, April 1996.

[SN98] W. Shen and D. H. Norrie, "A Hybrid Agent-Oriented Infrastructure for Modeling Manufacturing Enterprises." In *Proceedings of KAW'98*, Banff, Canada, 1998 (Agent-5:1–19).