Toward Fault-Tolerant Multi-agent Systems

Zahia Guessoum, Jean-Pierre Briot, Pierre Sens, and Olivier Marin
LIP6, University of Paris 6
OASIS team (Objects and Agents for Simulation and Information Systems)
and SRC team (Cooperative and Distributed Systems)
4 place Jussieu, 75252 cedex 5
{Zahia.Guessoum, Jean-Pierre.Briot, Pierre.Sens, Olivier.Marin}@lip6.fr
http://www-poleia.lip6.fr/OASIS/
http://www-src.lip6.fr/

Abstract. Most existing multi-agent architectures and multi-agent methodologies deal with domains which have static organization structures, static agent behaviors and small number of agents. This paper introduces an adaptive multi-agent model that is well adapted to domains which have dynamic organization structures, adaptive agents and a big number of agents. It first presents an example of application (Fault tolerant multi-agent systems) to underline the need of such model. It describes then the main components of this new model and their activities.

Key Words. Multi-Agent, Organization, Adaptation, Interdependence, Fault-Tolerance

1. Introduction

To make concrete the various research disciplines in the multi-agent area, several distributed agent architectures [M ller, 1998] have been proposed, they address different key features an agent should have. Most of these architectures describe the functional components of an agent and how these components work together. They are founded on very interesting proposals such as those presented in [Castelfranchi, 1995]; [Gasser, 1990]; [Durfee et al., 1987].

Recent works additionally address some key features of organization such as the concepts of groups, roles, etc. [Ferber and Gutknecht, 1998]. For instance, Gaia is a methodology for agent-oriented analysis and design [M. Wooldridge et al., 1999].

The proposed architectures and methodologies are well adapted to domains which have these main characteristics:

- static organization structures,
- static behaviors of agents and static services they provide,
- small number of agents,

- etc.

However, the case of dynamic and adaptive organizations has not enough been studied. In most existing systems, the adaptation of the organization structures relies on the adaptation of the agent behavior and it is not well known how local behavior rules lead to the emergence of a global behavior of the complex system.

Another issue in building multi-agent systems is how to observe, understand and control coordination in a dynamic organization of agents. These systems have several characteristics which make them more difficult to monitor and control than conventional single agent systems. The first problem is the element of distribution: Each agent is an autonomous and pro-active entity, and coordination is strictly limited to what can be done through message passing (see [Gasser9, 1992]). These factors make difficult the monitoring of multi-agent systems. J. Pitrat underlines that an intelligent system must have the ability to observe its own behavior [Pitrat, 1990]. In multi-agent systems, each agent has a local goal, so it can have a satisfaction function. But in most existing agent architectures, there is no mean to represent the global satisfaction function of the multi-agent system.

This paper deals with the problem of building multi-agent systems for domains which have these main characteristics:

- dynamic organization structures and complex web of interconnection,
- large number of different and flexible agents,
- etc.

The aim of our proposal is to introduce a dynamic and adaptive multi-agent model. We focus our work on fault-tolerant multi-agent systems. The organization structure, that we use to build fault-tolerant multi-agent systems, defines the interdependence among a population of agents [Castelfranchi, 1998].

This paper first describes an example: Fault tolerant multi-agent systems. Then it describes the proposed multi-agent model which has three main components: domain agents, monitor agents and an interdependence net. Finally, we discuss the advantages of our model to design complex multi-agent systems.

2. Fault Tolerant Multi-agent Systems

The main strength of the agent paradigm consists in the collective resolution of low-capable but interacting agents. As a distributed system, however, multi-agent systems are exposed to high rates of failure of their hardware and/or software components.

The failure of a component often evolves the failure of the whole system. A symptomatic example would be a crisis management system, where various and decentralized teams would cooperate.

Meanwhile, we noticed that most multi-agent architectures and multi-agent platforms do not address the issue of fault tolerance. We think that one major reason is that for the majority multi-agent systems and applications are still developed at a small scale:

- they run on a single computer or a group of highly coupled of computers,
- they run for small temporal experiments.

Moreover, most exiting works in distributed systems are not suitable for dynamic systems [Marin et al., 2001]. Few recent works (see for example [Golm,1998]) underlined the need of an adaptive fault-tolerant mechanism for complex system, but no solution has been proposed yet.

The following subsections present the replication in distributed computing, the framework DARX that we use to replicate agents and its implementation.

2.1. 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 [Guerraoui et al., 1997]. There are two main types of replication protocols:

- the **active** one in which all replicas process concurrently all input messages,
- and the **passive** one in which only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency.

Active replication strategies lead to a high overhead. If the degree of replication is **n**, the **n** replicas are activated simultaneously to produce one result.

Passive replication economizes processor utilization by activating redundant replicas only in case of failures. 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 needs a checkpoint management which remains expensive in processing time and space.

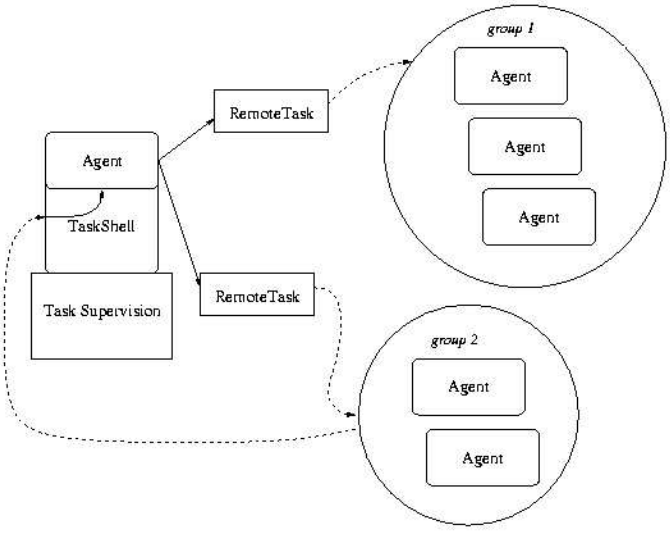
The active replication provides a fast recovery delay. This kind of technique is dedicated to applications with real-time constraints which require short recovery delays. The passive replication scheme has a low overhead under failure free execution but does not provide short recovery delays. The choice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

Many toolkits [Renesse et al., 1996] include replication facilities to build reliable applications. However, most products are not quite flexible to implement an adaptive replication mechanism.

The following section presents the framework DARX. The latter provides efficient fault-tolerance to multi-agent systems through selective agent replication. 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.

2.2. The DARX Framework

DARX is a framework to design reliable distributed applications. Each task can be replicated an unlimited number of times and with different replication strategies. DARX includes group membership management to dynamically add or remove replicas. It also provides atomic and ordered multi-cast for the replication groups' internal communication.



INCORPORER

Fig. 1 DARX application architecture

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 checks the liveness of each in turn its replicas and is responsible for reliable broadcasting. In case of failure of a leader, a new one is automatically elected among the set of remaining ones.

DARX provides a global naming, whereas RMI does not include this functionality. Each replicated task has a global name which is independent of the current location of its replicas.

To make multi-agent systems fault-tolerant, each agent gets to inherit the functionalities of a *DarxTask* object (Fig. 1), enabling the underlying system to handle the agent's 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 agent is itself wrapped into a *TaskShell*, which acts as a replication group manager 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.

A task can communicate with a remote task, unregarding 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.

2.3. Discussion

We consider a multi-agent system where:

- agents run on several distributed computers,
- they run for long and important experiments,
- the resources (number of machines, ...) are limited,
- etc.

DARX provides the needed mechanisms to replicate agents. It provides also passive and active strategies. The active replication strategy evolves several problems in the case of multi-agent systems. An agent behavior is not deterministic. His replicas cannot therefore always behave identical. In this paper, we consider only passive strategy.

The problem is therefore to find the good answers to the following kinds of questions:

- Which agents must be replicated?
- How many replicas of these agents must be made?
- etc.

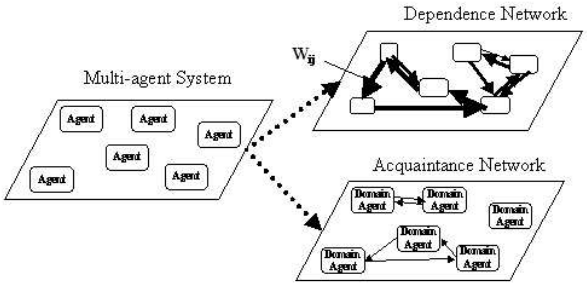


Fig.2. Examples of multi-agent system components

Distributed systems can be easily replicated before run time. The number and criticality of components are often static. However, multi-agent systems are more complex. They usually have several dynamic structures and criticality of agents relies on these structures. So, the agents criticality cannot be determined before runtime.

C. Castelfranchi defines the interdependence and power structure, the communication structure, etc. (see Fig. 2). The replication mechanism can be therefore based on the interdependence and power structure. The problem is how to define and how to update this structure?

In the following section, we introduce a new model for dynamic and adaptive multi-agent organizations. This model provides a high level of dynamicity and it can be used to build fault-tolerant multi-agent systems.

We aim to show that some multi-agent characteristics can be modeled and implemented as applications of the existing multi-agent architectures.

3. Adaptive Multi-agent Model

In most existing agent and multi-agent architectures, a multi-agent system is a set of interacting agents. The organization structures (such as the interdependence net) are therefore not explicitly represented. These representations are well suitable for static

organization structures. However, they cannot be used to represent dynamic and adaptive structures. We propose to consider the organization as the most important property of a multi-agent system. We adopt L. Gasser's definition: an organization is a particular set of settled and unsettled questions about beliefs and actions through which agents view other agents. So, the organization relies on: mutual commitments, global commitments, and mutual beliefs [Gasser, 1990].

The following subsections describe the components and activities of our adaptive multi-agent model.

3.1. Multi-agent System Components

We consider a multi-agent system where agents run on a set of computers. Each computer has one or several agents.

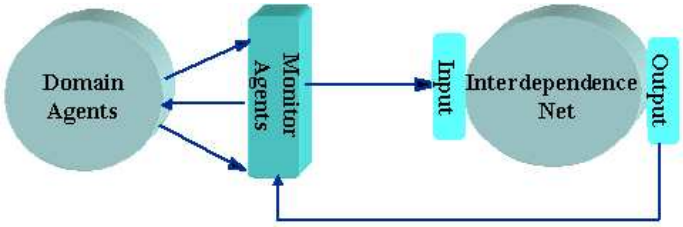


Fig. 3. Adaptive multi-agent system components

We propose to reify the adaptive part of the organization (see Fig. 3). A multi-agent system is therefore composed of [Guessoum and Cardon, 1999]:

- A settled part (domain agents) which is defined by the set of domain agents related in a classical acquaintances net. This net defines the agent explicit communication with specialized message passing (for example, KQML or ACL messages), and domain knowledge.
- An unsettled part (adaptive part) which is defined by a set of nodes related in an interdependence net. It reifies the set of trends of the system. As it was underlined in C. Castelfranchi, this part represents the glue of groups, it links the agent with joint goal and the common solution, it links members with each other [Castelfranchi, 1998].
- A set of agents (monitor agents) that continuously observe the settled part, build a state of the system and deliver this information to the unsettled part. They can also control the settled part (add new agents, add new replicas, remove replicas, ...).

3.2. Adaptive Multi-agent System Activity

An adaptive multi-agent system activity includes three sub-activities:

- the activity of domain agents,
- the monitoring which is the activity of the monitor agents,
- the adaptation which is the activity of the interdependence net.

3.2.1. Domain activity

The domain agents are related and distributed according to the application domain characteristics. They continuously act according to their environment changes and the interdependence structure [Castelfranchi, 1998]. The end of this activity evolves the end of the whole system.

Domain agents run on a set of distributed computer. Each computer integrates a DARX server. The latter schedules the activity of the various agents running on the associated computer and observes them.

We associate a *TaskShell* (see section 2.2) to each domain agent. This *TaskShell* acts as a replication group manager. It defines a set of events such as the received and sent messages. These events are then analyzed by the associated monitor agent.

3.2.2. Monitoring

The monitoring activity relies on events of domain agents and their environment. For example, for fault-tolerant multi-agent systems, we can associate a monitor agent to each machine and we can then observe:

- the number of sent and received messages,
- the quantity of information,
- the progression of agent problem solving process,
- the historic of the agent activity,
- etc.

We can also consider some data related to the computer (historic of failures) and the network. A subset of this data is given by DARX. For instance, DARX enables to build a set of events for each agent.

Each monitor agent defines therefore an array $D_{i,\ i=1,n}$ for each agent $Agent_i$ of the associated machine. For example, $D_i[1]$ may represent the number of messages

sent by the $Agent_i$ and n is the number of domain agents. This array is updated at each time interval Δt.

The built data represents the attributes that characterize the multi-agent system.

Another role of the monitor agents is the interpretation of the interdependence net provided by the adaptive part. Each $W_{i,j}$ describes the dependence between $Agent_i$ and $Agent_j$. $W_{i,j}$ is different from $W_{j,i}$, the interdependence function is not symmetric. In the example of distributed agenda, the dependence function between $Agent_i$ and $Agent_j$ can be proportional to the number of messages exchanged by the two agents.

The interdependence net is therefore analyzed to determine the most important agents in order to replicate them. To do this, the monitor agents calculate the weights of the domain agents. The weight w_i of each agent $Agent_i$ is calculated as follows:

$$w_i = \Sigma_{j=1,n} W_{i,j}$$

An agent can be then replicated according to his weight, the maximum weight (W_{max}) and the maximum and minimum number of replicas (R_m and r_m). The number of replicas nb_i of $Agent_i$ can be determined as follows:

$$nb_i = r_m + w_i / W_{max} \times (R_m - r_m)$$

So, the organization of agents which realize monitoring gets in touch two kinds of information: the direction of the behavior of the agents and the direction of variation of the interdependence net. Any modification of the domain agents is perceived by the monitor agents and propagated to the interdependence net which is adapted accordingly. Moreover, any modification of the interdependence net is interpreted and the used to act (replicate in the considered example) on domain agents.

In several application domains, the interdependence data can be used by domain agents to reason on other agents (see [Castelfranchi, 1998]).

3.2.3. Adaptation

The modifications of the interdependence net are controlled by the organizational knowledge. The latter aims to provide the system with the following properties:

- adaptation which allows the system to deal with new constraints and new data of the external world,
- self-organization to ensure the stability of the system dynamic,
- generalization, based on the examples that the system can solve.

To represent this knowledge, we choose a Kohonen's self-organization map [Kohonen, 1987] which provides the above properties (adaptation, generalization,...) and some other required multi-agent properties such as the building facility and distribution.

The input array of this map represents the data built by the monitor agents in the last interval Δt. The interdependence net is thus represented by the output array of Kohonen's map where each neuron represents a node. It is defined by the internal connections $W_{i,j}$.

The $W_{i,j}$ are updated by the learning algorithm proposed by T. Kohonen:

1. Initialize:

   the weights $W_{i,j}$,

   the learning coefficient ($\eta = 1$),

   the neighborhood V_i.

2. Evaluate correlation,

3. Activate the net,

4. Determine the neuron i_0 with maximal output,

5. Update weights

   $$W_{i,j} = W_{i,j} + \eta (C_{i0} - W_{i,j})$$

   for each V_i,

6. Update the learning coefficient

   $$\eta = \eta - 0.001 \Delta t$$

7. Go to 2

The application designer can choose several methods to initialize the weights $W_{i,j}$. He can:

- choose a default value (0 for example),
- define the values,
- etc.

4. Conclusion

This paper deals with the problem of how to build multi-agent systems for domains which have these main characteristics:

- dynamic organization structures and complex web of interconnection,
- large number of different and flexible agents,
- etc.

We proposed a new adaptive multi-agent model. The main idea of this work is to build multi-agent systems that can be autonomous. So, they can self-configure in order to avoid failures or to resolve an unknown problem.

This model has been implemented with the platform DIMA [Guessoum and Briot, 1999] and it was used to build fault tolerant multi-agent systems.

In order to validate the presented adaptive multi-agent model and it application to fault-tolerant multi-agent system, small applications are currently being developed. Those include:

- a distributed agenda,
- a basic crisis management system,
- etc.

We are also using this model to simulate economic models and to study the coevolution of new organizational forms [Lewin et al., 1999].

These examples are destined to test our model's and architecture's viability and utility. They aim also to complete the model and adjust the parameters. For example, several kinds of events can be observed and then used to calculate dependence between agents. The problem therefore is how to choose relevant events and then how to define the suitable correlation function.

5. References

[Avouris and Gasser, 1992] N. A. Avouris, and L. Gasser, editors (1992). Distributed Artificial Intelligence, chapter An Overview of DAI, pages 1--25. Kluwer Academic Publisher, Boston.

[Castelfranchi, 1998] C. Castelfranchi. Modelling social action for AI agents. Artificial Intelligence, pp. 157--182.

[Castelfranchi, 1995] C. Castelfranchi. A point missed in multi-agent, DAI and HCI. In Wooldridge, M. J. and Jennings, N. R., editors, Intelligent Agents - Theories, Architectures, and Languages, number 890 in LNAI, pages 49--62. Springer Verlag.

[Decker et al., 1997] 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.

[Durfee et al., 1987] E. H. Durfee, V. Lesser, and D. D. Corkill (1987). Coherent cooperation among communicating problem solvers. IEEE Transactions on Computers, 36(11):1275--1291.

[Ferber and Gutknecht, 1998] J. Ferber, J. and O. Gutknecht. Alaadin: a meta-model for the analysis and design of organizations in multi-agent systems. In Demazeau, Y., editor, ICMAS'98, pages 128--135, Paris.

[Gasser, 1990] L. Gasser. Conceptual modeling in distributed artificial intelligence. Journal of the Japanese Society of Artificial Intelligence, 5(4).

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

[Guerraoui et al., 1996] 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

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

[Guessoum and Cardon, 1999] Z. Guessoum, and A. Cardon. Self-adjustable autonomy in multi-agent systems. In AAAI 1999 Spring Symposium Series, Agents with adjustable autonomy.

[Guessoum and Briot, 1999] 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.

[Kohonen, 1987] T. Kohonen. Self-Organization and Associative Memory. Springer-Verlag.

[Lewin et al., 199] A. Y. Lewin, C. P. Long, and T. N. Carroll. ''The Coevolution of New Organizational Forms'', In Organization Science, 10(5):535--549, April 1996.

[Marin et al., 2001] O. Marin, P. Sens, J.-P. Briot and Z. Guessoum; ''Towards Adaptive Fault-Tolerance for Distributed Multi-Agent Systems'' To appear in Proc. of ERSADS'2001, Bertinoro, Italy, May 2001.

[M ller, 1998] J. M ller . Agent architectures. In ATAL'98, page 1:8.

[Pitrat, 1990] J. Pitrat. An intelligent system must and can observe it own behavior.

In Cognitiva 90.

[Renesse et al., 1996] R. Van Renesse, K. Birman, and S. Maffeis. ''Horus: A flexible group communication system'',In CACM, 39(4):76--83, April 1996.

[Shen and Norrie, 1998] 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).

[M. Wooldridge et al., 1999] M. Wooldridge, N. Jennings and D. Kinny (1999). The methodology gaia for agent-oriented analysis and design. AI, 10(2):1--27.

PROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPR
OGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROG
COMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCO
MPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMP
PROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPR
OGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROGCOMPPROG
COMPPROGCOMPPROGCOMP