On Fault Tolerance in Law-Governed Multi-Agent Systems

Maíra A. de C. Gatti, Carlos J. P. de Lucena
Laboratório de Engenharia de Software – LES
Pontifícia Universidade Católica do Rio de Janeiro, PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro – Brasil
+55 -21- 2540-6915, ext. 103
{mgatti, lucena} @inf.puc-rio.br

Jean-Pierre Briot
Laboratoire d'informatique de Paris 6 - LIP6
Universit´e Pierre et Marie Curie
8 rue du Capitaine Scott, 75015 Paris, France
Jean-Pierre.Briot@lip6.fr

ABSTRACT

There has been much research about frameworks and tools to build multi-agent systems in different domains in recent years. These systems have particular features such as autonomy, distribution, sociability, cooperation and others implemented in another software entity, known as an agent. In order to achieve some previously defined goals, the agents interact between themselves to complete their tasks. One issue that arises from this kind of software is how can we ensure their dependability, considering the reliability of critical applications and the availability of those agents that play important roles with their responsibilities; i.e., how to dynamically and automatically identify the most critical agents and increase their availability and reliability? To this end, over the past few years there has been work on this problem proposing different approaches, each one solving a restricted problem involving dependability and leaving the global problem to be solved afterwards. This paper describes a solution to increase the availability of such systems through a technique of fault tolerance known as agent replication, and to increase its reliability through a mechanism of agent interaction regulation called law enforcement mechanism. The main contribution of this work is to improve the capability of calculating how critical an agent is to the system through its interactions with other agents and to provide a framework that uses this information to ensure availability and reliability.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems, Languagens and structures; C.4. [**Performance of Systems**]: Reliability, availability, and serviceability;

General Terms

Reliability

Keywords

Open systems, dependability, criticality, law-enforcement.

1. INTRODUCTION

There are many definitions in the literature for agents and, consequently, multi-agent systems. And despite their differences, all of them basically characterize a multi-agent system (MAS) as a computational environment in which individual software agents interact with each other, in a cooperative manner, or in a competitive manner, and sometimes autonomously pursuing their individual goals. During this process, they access the environment's resources and services and occasionally produce results for the entities that initiated these software agents [1]. As the agents interact in a concurrent, asynchronous and decentralized manner, this kind of system can be categorized as a complex system [2].

The absence of centralized coordination data makes it hard to determine the current state of the system and/or to predict the effects of actions. Moreover, all of the possible situations that may arise in the execution context led us to be uncertain about predicting the behavior of agents. However, in critical applications such as business environments or government agencies (hospitals, police, justice, etc.), the behavior of the global system must be taken into account and structural characteristics of the domain have to be incorporated [10].

A particular issue that arises from this kind of software is: how we can ensure their dependability (which is the ability of a computer system to deliver service that can justifiably be trusted [3]) considering the reliability of critical applications and availability of these agents, which play important roles with their responsibilities? To this end, there already has been some work addressing this problem ([3][4][5][6], for instance, for availability and [7][8] for reliability) which have been proposed in the last few years using different approaches; each one solved a restricted problem involving dependability while leaving the global problem to be resolved afterwards.

This paper describes a solution to increase the availability of such systems through a technique of fault tolerance known as agent replication, and to increase their reliability through a mechanism of agent interaction regulation called law enforcement mechanism. The main contribution of this work is to improve the capability of calculating how critical an agent is to the system through its interactions with other agents and to provide a framework that uses this information to ensure availability and reliability.

The subsequent sections are organized as follows: Section 2 introduces the main concepts related to dependability. We focus on the strategies of fault tolerance for multi-agent systems, and on the law enforcement approach for increasing the reliability of these systems.

Section 3 states a scenario for the problem description. And Section 4 details the proposed solution for the problem as an integrated architecture. This architecture is the integration of both approaches presented in Section 2. And finally, Section 5 concludes this paper.

2. DEPENDABILITY IN MULTI-AGENT SYSTEMS

The concepts and techniques of dependability are well established due to major concerns regarding ubiquitous computing systems that control critical structures such as railroads, airplanes, and nuclear plants [11].

The main notion of dependability is consolidated within three concepts: the attributes of, the threats to and the means by which it is attained [12][13].

The attributes of system dependability consist of: (i) availability, the deliverance of correct service at a given time; (ii) reliability, the continuous deliverance of correct service for a period of time; (iii) safety, the absence of catastrophic consequences to users and the environment; (iv) confidentiality, the absence of unauthorized disclosure of information; (v) integrity, the absence of improper system state alterations; (vi) maintainability, the ability to undergo repairs and modifications.

Different emphasis may be put on each attribute depending on the application intended for the system. Several other dependability attributes may be defined and may be either combinations or specializations of the above. In this paper we will address availability and reliability attributes as a way of achieving or increasing multi-agent systems dependability.

The threats are the second concept mentioned previously and consist of failures, errors and faults. The ways in which a system can fail are its failure modes, characterized by the severity and the symptoms of a failure. And a fault is active when it produces an error; otherwise, it is dormant.

Finally, the means to attain a system's dependability, according to [12][13], were regrouped in four techniques: fault prevention, fault removal, fault tolerance and fault forecasting. The focus of the work presented here is fault tolerance, i.e., how to deliver correct service in the presence of active faults. It is generally implemented by error detection and subsequent system recovery, and possibly by error containment. Recovery transforms a system state that contains one or more errors (and possibly faults) into a state that can be activated again without detected errors and faults.

2.1 Fault tolerance

There are four essential characteristics of a multi-agent system: a MAS is composed of autonomous software agents, a MAS has no single point of control, a MAS interacts with a dynamic environment, and the agents within a MAS are social (agents communicate and interact with each other and may form relationships).

All these situations contribute to a failure state. A failure occurs when the system produces results that do not meet the specified requirements. A fault is defined to be a defect within a component of a MAS that may lead to a failure. There are several faults that may occur. For instance, we can have program bugs, which are errors in programming that are not detected by system testing. We can also have unforeseen states, i.e., the programming does not handle a particular state and testing team did not test for this state. We can have processor faults, which can be a system crash (permanent/fail-silent) or a shortage of system resources. There would be communication faults which can occur due to slow downs, failures or other problems with the communication links; And, finally, unwanted emerging behavior, i.e., system behavior which is not predicted. Emerging behavior may be profitable or detrimental. When a fault occurs in a MAS, interactions between agents may cause the fault to spread throughout the system in unpredictable ways. The mechanism proposed in this paper aims to solve the processor and communication faults through the agent replication technique, and the unwanted emerging behavior through the law enforcement mechanism.

Several approaches (for instance [4][14][15]) address the multi-faced problem of fault tolerance in multi-agent systems. Some of them handle the problems of communication, interaction and coordination of agents with the other agents of the system. Others address the difficulties of making reliable mobile agents, which are more exposed to security problems. Some of them are based on replication mechanisms [9], and as mentioned before they have solved many problems of ubiquitous systems. However, the main limit of current replication techniques for multi-agent systems is that most of them are not quite suitable for implementing adaptive replication mechanisms, which is a problem as the criticality of agents may evolve dynamically during the course of computation and it not possible to predict how critical the agent is previously.

Therefore, there is a framework called DimaX [6] that allows dynamic replication and dynamic adaptation of the replication policy (e.g., passive to active, changing the number of replicas). It was designed to easily integrate various agent architectures, and the mechanisms that ensure dependability are kept as transparent as possible to the application. Basically, DimaX is the integration between a multi-agent system called Dima and the dynamic replication architecture for agents called DarX.

Among the several approaches to fault tolerance in MASs, basically we can group them in: agent-centric approaches, which build fault tolerance into the agents; and system-centric approaches, which move the monitoring and fault recovering into a separate software entity [5]. Agent replication uses aspects of both agent-centric and system-centric approaches.

Agent replication is the act of creating one or more replicas of one or more agents, and the number of each agent replica is the replication degree; everything depends on how critical the agent is while executing its tasks. Then there are two cases that might be distinguished: 1) the agent's criticality is static and 2) the agent's criticality is dynamic. In the first case, multi-agent systems have often static organization structures, static behaviors of agents, and a small number of agents. Critical agents, therefore, can be identified by the designer and can be replicated by the programmer before run time.

In the second case, the agent criticality cannot be determined before run time due to the fact that the multi-agent systems may have dynamic organization structures, dynamic behaviors of agents and a large number of agents. Then it is important to determine these structures dynamically in order to evaluate agent criticality. [16] proposed a way of determining it through role

analysis. It could be done by some prior input from the designer of the application who specifies the roles' weights, or there would be an observation module for each server that collects the data through the agent execution and their interactions. In the second approach, global information is built and then used to obtain roles and degree of activity to compute the agent criticality.

Another way of dynamically determining these structures to evaluate agent criticality is to represent the emergent organizational structure of a multi-agent system by a graph [6]. The hypothesis is that the criticality of an agent relies on the interdependences of other agents on this agent. First, the interdependence graph is initialized by the designer, and then it is dynamically adapted by the system itself. Some algorithms to dynamically adapt and describe it are proposed in [6].

We will present here an enhancement of these approaches and it will be further described in Section 3. Basically, we improved the agent criticality calculation through dynamic elements present during interactions with other agents. These elements will be described in the next section while the law enforcement approaches, especially the one that was chosen, are detailed.

2.2 Law-Governed Interaction

Open multi-agent systems, as we have already seen, are built of distributed software agents that are independently implemented, i.e., the development takes place without a centralized control. Thus, we want to ensure the reliability of these systems in a way that all the interactions between agents will occur according to the specification and that these agents will obey the specified scenario. For this, these applications must be built upon a law-governed architecture.

In this kind of architecture, enforcement that is responsible for the interception of messages and the interpreting of previously described laws is implemented. The core of a law-governed approach is the mechanism used by the mediator to monitor the conversations between agents. Among the models and frameworks that were developed to support this mechanism (for instance, [7][8][17][18]), XMLaw [7] was chosen for three main reasons. First, because it implements a law enforcement approach as an object-oriented framework, which brings the benefits of reuse and flexibility. Second, it allows normative behavior that is more expressive than the others through the connection between norms and clocks. And finally, it permits the execution of Java code through the concept of actions.

Thus, in this section, we explain the description language and the XMLaw framework [7]. Basically, interactions should be analyzed and subsequently described using the concepts proposed in the model during the design phase. After that, the concepts will be mapped to a declarative language based on XML. It is also important to point out that agent developers from different open MASs must agree upon interaction procedure. In fact, each open MAS should have a clear documentation about the interactions' rules. By doing that, there is no need of agent developers' interaction.

Interaction's definitions are interpreted by a software framework that monitors component interaction and enforces the behavior specified by the language. Once interaction is specified and enforced, despite the autonomy of the agents, the system's global behavior is better controlled and predicted. Interaction specification of a system is also called the laws of a system. This is because besides the idea of specification itself, interactions are monitored and enforced. Then, they act as laws in the sense that they describe what can be done (permissions), what cannot be done (prohibitions) and what must be done (obligations).

Among the model elements, the outer concept is the LawOrganization. This element represents the interaction laws (or normative dimension) of a multi-agent organization. A LawOrganization is composed of scenes, clocks, norms and actions. Scenes are interaction contexts that can happen in an organization. They allow modularizing interaction breaking the interaction of the whole system into smaller parts. Clocks introduce global times, which are shared by all scenes.

Norms capture notions of permissions, obligations and prohibitions regarding agents' interaction behavior (as mentioned before). Actions can be viewed as a consequence of any interaction condition; for example, if an agent acquires an obligation, then action "A" should be executed.

Scenes define an interaction protocol (from a global point of view), a set of norms and clocks that are only valid in the context of the scene. Furthermore, scenes also identify which agents are allowed to start or participate in the scene.

Events are the basis of the communication among law elements; that is, law elements dynamically relate with other elements through event notifications. Basically, we can understand the dynamic of the elements as a chain of causes and consequences, where an event can activate a law element; this law element could generate other events and so on.

The framework provides compliance with both the model of interactions proposed previously and the XMLaw declarative language. It has a set of modules that supports three types of users: (i) "Law developer" represents the developer responsible for specifying the laws. He must understand the application under construction, know the law concepts, and then, specify the laws for the application; (ii) "Agent developer" represents the developer responsible for building the agents of a multi-agent system. He knows about the existence of the laws and should design the agents in compliance with them; (iii) "Software infrastructure developer" deals with law enforcement software support.

Most of the framework is implemented as mediator agent modules. The mediator agent monitors all interactions and makes sure that interactions are compliant with the specifications. The mediator performs a number of activities. First, the mediator waits to receive messages. Once a message has arrived, it checks if the message belongs to the mediator protocol. If it does, the mediator proceeds with the protocol execution. Otherwise, if the message belongs to some agent conversation, the mediator starts the process of enforcing, and if it is compliant with the laws, the message is redirected to the addressee agent. This sequence of activities is repeated while the mediator agent is running.

The communication among the modules is mainly based on event notifications. This approach leads to a low coupling level among modules and also leads to more flexible system designs.

The proposal here is not to detail the framework, so further details can be found in [19]. The next sections will address both DimaX and XMLaw and how their integration works.

3. PROBLEM DESCRIPTION

In this section we are going to describe a scenario where two agents exchange messages in order to achieve their goals. During the interaction, they are regulated by rules that do not allow them to send some types of messages (that we can call performatives) and some other normative elements. The idea of illustrating this scenario is to find out how to answer two main questions: how and which elements (norms, clocks, etc.) of the XMLaw could improve the agent criticality analysis that is done by DimaX? And how can it be best accomplished, considering coupling, modularity and reuse?

First, imagine a scenario where there are two agents mentioned: the customer and the seller of an institution. Suppose that an open multi-agent system exists where the agents that want to buy a product may enter or leave at any time, and that there are sellers in this institution that want to sell the product for the highest price that they can achieve. Then, we have a negotiation scene where each agent wants to succeed and there is a protocol in this scene that represents all the messages that can be exchanged and all the rules that rule this scene and the participants.

At any time, any agent can enter into the scene and initiate the protocol. If we specify this scene in XMLaw, we have to specify the protocol as a state machine, where each transition of the protocol is activated by a message sent by an agent and it can activate the other elements of XMLaw, as clocks and norms.

Basically, the negotiation proceeds as follows: a customer initiates a negotiation by sending a proposal for a book to a seller. He informs the maximum price that he will pay for the book. The seller can accept with a proposal or can refuse it. If he accepts, he can send proposals with lesser or equal price informed by the customer. When the customer receives the proposal, he has 2 minutes to decide if he will accept it or not. After 2 minutes, if the customer hasn't answered the seller, he can sell the product to another customer. Otherwise the seller is not allowed to sell it to anybody else. If the customer refuses it, the seller can re-propose another price. If the customer accepts it, the seller informs the bank where the payment must be made. Then the customer has the obligation of paying for the product and of informing the number of the voucher to the seller. The scene ends then when the customer informs that he paid it with the proof of payment (Figure 1).

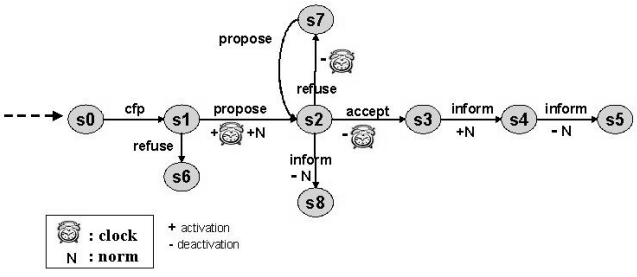

Figure 1 - Protocol State Machine Representation

If we consider that when an event (such as clock activation/deactivation, norm activation/deactivation, etc.) occurs during the scene execution, the agent criticality can increase or decrease, since the agent becomes more or less important; thus, each element should be analyzed in order to calculate it in the best way. Moreover, other elements and events that might not be handled by XMLaw should be analyzed in order to evaluate how they could influence the agent criticality analysis. For instance, when an agent starts playing a role its criticality may increase or decrease.

In the context of the negotiation scene, when the customer must answer the seller if he will accept his proposal or refuse it since the clock activation event will be fired, his criticality should increase, since the seller cannot sell the product while the customer doesn't answer him. Thus, the customer is very important to the seller at this time and should not crash, for example. Then, when the clock deactivation is fired, the customer criticality should decrease. Another situation would be of the payment for the product. Since the customer has the obligation of paying for the product when he accepts the price, his criticality should also increase. Those variations are shown in Figure 2.

We can see the protocol execution on the left side of the picture. Next to it is a draft of the main criticality variation. This main result is based on the criticality variation that occurs as a result of each event, as previously mentioned. The clock's picture represents the clock activation/deactivation event and the letter **N** represents the norm activation/deactivation event during the protocol execution, according to the plus or minus sign that comes before the picture or letter.

For instance, in an analogous manner, if we analyze the seller criticality during the scene execution, his criticality should increase when the customer proposes a price for the product because he has the obligation to answer him.

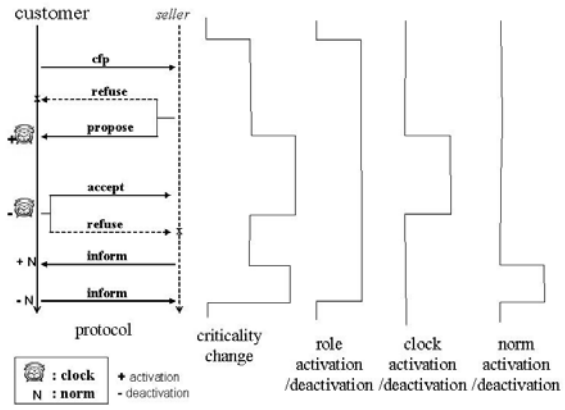

Figure 2 - Criticality variation for customer role

Within conclusion, all events that may be fired during a protocol execution can increase or decrease the agent criticality according to the type of the event and to its semantic. In the next section we will explain how we extended both XMLaw and DimaX to include this analysis at the design time and at the run-time.

4. PROPOSED SOLUTION: THE INTEGRATION

In this section we will describe the proposed solution first from the XMLaw point of view, second from the DimaX point of view, and finally from the integration point of view.

We analyzed XMLaw and studied which elements should be inserted, which events should be sensed by the new elements and so on. It was not a trivial effort considering that XMLaw is not an extensible framework for adding new elements or generating new events. Section 4.1 describes these elements and events.

Then we analyzed how to integrate it with DimaX and how to extend the criticality analysis done by DimaX. Section 4.2 describes how we extended the agent criticality calculation and Section 4.3 describes the integration itself.

4.1 XMLaw Extensions

We have extended XMLaw with two new elements: Role and Criticality Analysis. With the new Role element, when an agent requests to enter an organization, it has to inform the role it wants to play; and when a scene is executed, the agent, if accepted, will have to play its role. An organization has one or more roles to be played by agents and an agent can play different roles in different organizations.

Each organization's role has an identification and a list of norms associated with its Rights. Some norms can be activated and/or deactivated according to agents' rights. Rights describe the permissions regarding the resources and services available in the environment and about the behavior of the agents. It can activate or deactivate some norms related to that role.

The Criticality Analysis element has two elements: Increases and Decreases. The Increases element contains the list of events that contribute to increasing agent criticality. And the Decreases element contains the list of events that contribute to decreasing agent criticality. The Increase and Decrease elements have these attributes: the event identification from the event that was fired, the event type from the event that was fired, the value which is a weight for the increase or decrease contribution of that event and the agent role identification, which has all the references to the agent whose criticality will be updated in runtime. The weight is a number between 0 to 1.

```
<CriticalityAnalysis>
    <Increases>
        <Increase event-id="customer" event-type="role_activation" value="0.3"
        role-id="customer" />
        <Increase event-id="seller" event-type="role_activation" value="0.7" role-
        id="seller" />
        <Increase event-id="time-to-decide" event-type="clock_activation"
        value="0.5" role-id="customer" />
        <Increase event-id="customer-payment-voucher" event-
        type="norm_activation" value="0.5" role-id="customer" />
    </Increases>
    <Decreases>
        <Decrease event-id="customer" event-type="role_deactivation"
        value="0.3" role-id="customer" />
        <Decrease event-id="seller" event-type="role_deactivation" value="0.7"
        role-id="seller" />
        <Decrease event-id="seller-permission-to-cancel" event-
        type="norm_activation" value="0.5" role-id="customer" />
    </Decreases>
</CriticalityAnalysis>
```

Figure 3 - Example of Criticality Analysis Specification

For instance, Figure 3 shows the XMLaw specification for the Criticality Analysis. When an agent starts playing the customer role, its criticality has to be recalculated and updated by a weight of 0.3. The same happens when an agent starts playing the seller role, its criticality has to be updated by a weight of 0.7. Those actions are executed when the role activation event is fired.

4.2 DimaX Extensions

In our work we have proposed the same reasoning as done in [12] for updating the agents' criticality. Each value of increasing or decreasing agent's criticality is stored on a table T, which defines the weights of its event. So, for example, there would be three different tables in our negotiation scene problem: Tr, which defines the weights of role activation or deactivation; Tc, which defines the weights of clocks activation or deactivation; and Tn, which defines the weights of norms activation or deactivation.

Then the criticality of the agent Agent i is computed as follows:

$$wi(t) = (a1 * \text{aggregation}(Tr[rij]\ j=1,nr) + a2 * \text{aggregation}(Tc[cij]\ j=1,nc) + a3 * \text{aggregation}(Tn[nij]\ j=1,nn) + a4 * awi)) / \sum_{i=1}^{4} a_i$$

Where a1, a2, a3 and a4 are the weights given to the four kinds of parameters (roles, clocks, norms and degree of activity), which are introduced by the designer by XMLaw specification. And awi is the degree of activity of the agent i.

And the number of replicas nbi of Agent i, which is used to update the number of replicas of the domain agent, can be determined as follows:

$$nbi(t) = \text{rounded}(rm + wi(t) * Rm/W)$$

Where:

– wi: its criticality,

– W: the sum of the domain agents' criticality,

– rm: the minimum number of replicas which is introduced by the designer,

– Rm: the available resources that define the maximum number of possible simultaneous replicas.

4.3 DimaX and XMLaw Integration

In order to complete the full integration, there are two tasks that we are looking forward to accomplish. First, we will evaluate the new architecture with a larger case study. We are working on this task at the moment.

After that, we will conduct a comparison between this case study running on this new architecture with the same case study running separately in DimaX and XMLaw.

The idea is to benchmark the two attributes of dependability that we are tackling: availability and reliability. Thus, we will have a deep analysis of how much the solution proposed improves dependability of multi-agent systems.

5. CONCLUSIONS AND FUTURE WORK

As we have already seen, the Open Multi-Agents System dependability can be achieved by fault tolerance. Among other existing fault tolerance techniques, there is a specific one that has been used in the recent years for achieving dependability in multi-agent systems. It is the Agent Replication technique. It has been used in several approaches and we used it in our work from the point of view of [6][16], since it is an effective way to implement fault tolerance for distributed systems.

Furthermore, we used XMLaw because we expect that it also increases the reliability of Open Multi-Agents System through a law enforcement approach for regulating agents' interactions through a higher control.

This work presents an extension of the XMLaw conceptual model described in Section 3 as a way of improving its dependability. We propose to use new elements that help specify the attributes concerning the agent criticality during its interaction with other agents.

Moreover, considering that XMLaw framework is an event-based framework, other elements from the law's specification that are perceived by events can improve the criticality analysis done by DimaX, which is used for calculating the agent number of replicas as clock activations, norms activations, etc.

We extended XMLaw with two new elements that were introduced in the conceptual model: Role and Criticality Analysis. The first one (Role) was necessary because, until now, XMLaw does not have this element and it would be very difficult to associate an event activation to the agent without its reference. By doing that, we realized the need of associating specific norms to an agent when it starts playing a role or stops playing it. Then we created the concept of Rights, which describe the permissions on the resources and services available in the environment and about the behavior of the agents. It can activate or deactivate some norms related to that role.

The second element, Criticality Analysis, was introduced in order to monitor the events that should improve the criticality analysis done by DimaX. The events are divided into two groups: the ones that increase the agent's criticality and the ones that decrease it. By doing that, any event considered important by the designer of the application while specifying its law can be taken into account.

Second, it was necessary to extend DimaX in order to provide another algorithm for calculating the agent's criticality. Considering the reasoning done by the Role Analysis described in [16], it was easy to extend it; instead of receiving one table with the weights, receiving tables related to XMLaw events with the weights.

Two issues arose during the XMLaw instantiation. First, we perceived that XMLaw could be improved in order to make it more extensible. And second, the specification done by the designer of the events, which increase or decrease agent criticality, could be more appropriate if it wasn't so sensitive. In fact, we believe that it should be based on safety cases.

Thus, we are going to study how dependability cases [20] can help the "Law developer" of critical systems, since it defines a bottom-up approach for specifying critical events.

6. REFERENCES

[1] http://agtivity.com/agdef.htm, accessed in Oct/2005.

[2] Jennings, Nicholas R., An Agent-Based Approach for building Complex Software Systems, Communications of the ACM, 44(4), 35-41, April 2001.

[3] Peng Xu, Ralph Deters. "Using Event-Streams for Fault-Management in MAS," iat, pp. 433-436, IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04), 2004.

[4] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In AAMAS2002, Boulogna, Italy, 2002.

[5] Fedoruk, A. and Deters, R. 2003. Using dynamic proxy agent replicate groups to improve fault-tolerance in multi-agent systems. In Proc. of the Sec. int. Joint Conf. AAMAS '03. ACM Press, New York, NY, 990-991.

[6] Guessoum, Z., Faci, N., Briot, J.-P., Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform. Proc. of ICSE'05, 4th Int. Workshop on Soft. Eng. for Large-Scale Multi-Agent Systems, ACM Software Engineering Notes, 30(4) : 1-6, July 2005.

[7] R. Paes, G. R. Carvalho, C.J.P. Lucena, P. S. C. Alencar, H.O. Almeida; and V. T. Silva. Specifying Laws in Open Multi-Agent Systems. In: Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM), AAMAS2005, 2005.

[8] Murata, T. and Minsky, N. "On Monitoring and Steering in Large-Scale Multi-Agent Systems", Proceedings of ICSE 2003, 2nd Intn'l Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003).

[9] Guerraoui, R. and Schiper, A. Software-based replication for fault tolerance. IEEE Computer Journal, 30(4):68--74, 1997.

[10] Vpazquez-Salceda, J., Dignun, V., Dignun, F., Organizing Multiagent Systems, Autonomous Agents and Multi-Agent Systems, 11, 307-360, 2005.

[11] Lussier, B. et al. 3rd IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, Manchester (GB), 7-9 September 2004, 7p.

[12] Laprie, J. C., Arlat, J., Blanquart,J. P., Costes, A., Crouzert, Y., Deswarte, Y., Fabre, J. C., Guillermain, H., Kaâniche, M., Kanoun, K. Mazet, C., Powel, D., Rabéjac, C. and Thévenod, P. Dependability Handbook (2nd edition) Cépaduès – Éditions, 1996. (ISBN 2-85428-341-4) (in French).

[13] Avizienis, A., Laprie, J.-C., Randell, B. Dependability and its t-hreats - A taxonomy. IFIP Congress Topical Sessions 2004: 91-120.

[14] Decker, K., Sycara,K. and Williamson, M. Cloning for intelligent adaptive infor-mation agents. In ATAL'97, LNAI, pages 63–75. Springer Verlag, 1997.

[15] Hagg, S. A sentinel approach to fault handling in multi-agent systems. In C. Zhang and D. Lukose, editors, Multi-Agent Systems, Methodologies and Applications, number 1286 in LNCS, pages 190–195. Springer Verlag, 1997.

[16] Guessoum, Z., Briot, J.-P., Faci,N. Towards Fault-Tolerant Massively Multiagent Systems, Massively Multiagent Systems n.3446, LNAI, Springer Lecture Note Series, Verlag, 2005, pg. 55-69.

[17] Minsky, N.H., Ungureanu, V., Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACM Trans. Softw.Eng.Methodol. 9 (3) (2000) 273-305.

[18] Esteva, M., Eletronic institutions:from specification to developement, Ph.D. thesis, Institut d'Investigació en Intel.ligència Artificial, Catalonia - Spain (October 2003).

[19] Paes, R., Alencar, P., Lucena, C. Governing Agent Interaction in Open Multi-Agent Systems. Monografias de Ciência da Computação nº 30/05, Departamento de Informática, PUC-Rio, Brazil, 2005.

[20] Weinstock, C.B., Goodenough, J.B., Hudak, J.J., Dependability Cases, Technical Note, CMU/SEI-2004-TN-016, 2004.