GENERIC FAULT-TOLERANT LAYER SUPPORTING PUBLISH/SUBSCRIBE MESSAGING IN MOBILE AGENT SYSTEMS

Milovan Tosic, Arkady Zaslavsky
School of Computer Science and Software Engineering, Monash University, 900 Dandenong Road Caulfield East, Victoria 3145, Australia

Keywords: Reliability, Fault-tolerance, Agents, Multi-agent Systems, Publish/Subscribe Messaging, Web-services

Abstract: With the introduction of clustered messaging brokers and the fault-tolerant Mobile Connector, we can guarantee the exactly-once consumption of messages by agents. The context-aware messaging allowed us to decrease the messaging overhead which has to be present in any fault-tolerant solution. This paper proposes a complete fault-tolerant layer for multi-agent systems (EFTL) that does not restrict agent autonomy and mobility in any way. An application can choose if it wants EFTL support and that decision is based on support costs. A persistent publish/subscribe messaging model allows the creation of an external platform-independent fault-tolerant layer. In order to support the multi-agent platforms of different vendors, a large part of the application logic is moved from those platforms to an application server. We present the EFTL system architecture, the algorithm of exactly-once message consumption and the system's performance analysis.

1 INTRODUCTION

The use of web-services in many domains of distributed computing has proven its effectiveness. However, the research community has not paid a lot of attention to the application of web-services in domain-independent fault-tolerant support systems. Our External Fault-Tolerant Layer (EFTL) introduces a new dimension in this research area – moving the components of the fault-tolerant system out of the multi-agent platforms using standard tools, web-services and messaging brokers. Moreover, an application or an agent can choose if it wants EFTL support and that decision is based on support costs.

The most important factors which can affect reliability of multi-agent systems are related to the reliability levels of their components. Multi-agent systems are comprised of different entities where the most important ones are the agents and agent hosts. In regard to the basic systems theory, the performance of the whole system and its outputs depends on the actions and performance of its entities. That means that the performance of the complete multi-agent system depends on the performance of its agents and hosts. On the other hand, not all the entities within a system have the same level of importance, so the failure of some entities might not cause the failure of the whole system. That is why fault-tolerant approaches can ignore some failures in order to lower the cost of a fault-tolerant solution.

Another category that can cause a failure of the system is communication. Knowing that the achievement of a goal usually depends on cooperation between the agents, we can conclude that any fault of a communication subsystem can produce the difference between the real and expected outcomes of a system. Agent migration between the hosts can be viewed as a special type of communication because many agent platforms use the same mechanisms for message and agent transfer. If an agent is lost during transmission from one host to another, then it is not an agent failure but a migration failure.

The persistent publish/subscribe messaging model allows the creation of an external platform-independent fault-tolerant support system. The most important part of any distributed fault-tolerant support system is its messaging subsystem. With the introduction of clustered messaging brokers and the fault-tolerant Mobile Connector, we can guarantee the exactly-once consumption of messages by agents. The Mobile Connector is a lightweight platform-independent component which does not restrict agent autonomy and mobility.

This paper is organized as follows: firstly, we shall present related work from the area of multi-agent system reliability. Then, we shall explain the reliability model which has been used in our research and describe the architecture of the External Fault-Tolerant Layer (EFTL) with focus on the Mobile Connector component. After that, we shall present a few scenarios in EFTL functioning and explain what needs to be done to develop an application that will be supported by EFTL. The last sections of this paper will present performance analysis of EFTL, the conclusions and motivations for future work.

2 RELATED WORK

A group of authors proposed checkpointing as a good procedure which saves agent states to a persistent storage medium at certain time intervals. Later, if an agent fails, its state can be reconstructed from the latest checkpoint (Dalmeijer et al, 1998). This approach depends on the reliability of the host because we have the so-called blocking problem when the host fails. The agents which have been saved at a particular host can be recovered only after the recovery of that host (Mohindra et al, 2000). The second approach that tries to ensure an agent's reliability is replication. In this approach, there are groups of agents which exist as replicas of one agent, and can be chosen to act as the main agent in case of its failure. The number of agents is increased and they have to cooperate so the complexity of the system is also increased. In order to preserve the same view to the environment from all the members of the replica group, (Fedoruk, Deters, 2002) have proposed the concept of a group proxy, which is an agent acting as proxy through which all the interactions between the group and the environment have to pass. When the proxy agent approach is broadened with the primary agent concept, in (Taesoon et al, 2002) and (Zhigang, Binxing, 2000), then the primary agent is the only one which does all the computations until its failure. Then all the slaves vote in another primary agent from their group. Therefore, any slave agent can become a primary.

In order to watch the execution of an agent from an external entity, (Eustace et al, 1994), (Patel, Garg, 2004) and (Lyu, Wong, 2004) have proposed the usage of supervisor and executor agents. The supervisor agents watch the execution of the problem-solving agents and detect all the conditions which can lead to, or are, the failures, and react upon detected conditions. Hosts can also be used as the components of a fault-tolerant system (Dake, 2002). Basic services which are provided by the hosts can be extended by certain services which help the agents achieve a desirable level of reliability. Depending on the implementation of the fault-tolerant system, it cannot cope with all kinds of failures. That is why some systems do not even try to recover from certain types of failures. In order to determine the feasibility of the recovery, (Grantner et al, 1997) proposed the usage of fuzzy logic.

Moving on to the recovery of an agent host, if the state of the host has not been saved to a persistent storage medium, we can simply restart the host. Then, if a host is very important for the functioning of the whole agent platform, we can replicate it (Bellifemine et al, 2003). If our agents used the transaction-based approach which relied on the services provided by the host and not by an underlying application server or a database, then the host is the one which has to undo all the uncommitted actions after its restart (Patel, Garg, 2004).

In order to deliver a message to an agent, we have to track the agent's location to determine where to forward the message. The authors have proposed different solutions, such as the registration of the agent locations at some central entity (Moreau, 2002) or the usage of the forwarding pointers principle (Zhou et al, 2003). Then, when we know the exact location of the agent, we have to deliver the message. Two main delivery principles have been specified in (Cao et al, 2002). In the "push" principle, we have to interfere with an agent's autonomy and to constrain its mobility until we deliver the messages to it. In the "pull" principle, the agent is the one which decides when it wants to receive messages, and which messages it wants to receive. (Cao et al, 2004) have proposed the mailbox as a separate entity that is also mobile and moves to be at the same host as its agent or somewhere close to that host.

The benefits of the publish/subscribe messaging model in mobile computing have been presented in (Padovitz et al, 2003). Their approach specifically concentrates on context-aware messaging, where an agent can subscribe to receive only the messages which satisfy its subscription filter. This solution leads us to a highly effective notification mechanism for the mobile agents.

Another communication problem, the inaccessibility in the case of, for example, network fragmentation can be solved using the doubler agents, presented in (Pechoucek et al, 2003).

Multicasting is the delivery of the same message to multiple receivers, and is often described by the “all or none” principle. Researchers usually used the two-phase commit protocol to solve this problem, as in (Macedo, Silva, 2002).

3 RELIABILITY MODEL

The reliability of multi-agent systems has to be measured differently from the reliability of other conventional distributed systems. Since almost all multi-agent systems share characteristics such as network fragmentation, component autonomy and mobility, then standard factors of reliability, like system availability, cannot be applied to them. Therefore, we have to find another reliability model able to describe the events which can cause multi-agent system failures and allow us to evaluate our research achievements.

As described in (Luy, Wong, 2004), reliability in multi-agent systems can be evaluated by measuring the reliability of each individual agent on a more general level. From the viewpoint of the whole system, each agent can either successfully complete its tasks or fail to do so. Therefore, the reliability of the whole system depends on the percentage of agents which managed to achieve their goals.

The same authors proposed that the agent tasks should be defined as scheduled round-trips in a network of agent hosts.

In order to evaluate reliability, we can assume that the agents and agent hosts are prone to different types of failures. The agents can die unexpectedly or become non-responsive. A host can die and cause the failures of all the agents which resided on it at the moment of its death. Only the agent which managed to arrive at the final host and which has a state consistent with the states of all the other successful agents can be considered a successful finisher.

4 DESIGN OF EFTL

EFTL (External Fault-Tolerant Layer) is an application-independent fault-tolerant layer that provides multi-agent systems with extra reliability features. The system diagram is presented in Figure 1. In order to support multi-agent platforms from different vendors, a large part of application logic is moved from those platforms to the application, web and messaging servers. The only platform-dependent components are the Reliable Agent Layer and the Platform Listener. They support only the basic operations which EFTL has to perform on an agent or agent platform. Those operations include the control of an agent’s life cycle and listening to the platform-wide events which are important from a reliability perspective.

The Platform Listener is not deployed at any agent host prior to EFTL execution time. The usage of a web-server allowed us to decouple the agent platform and the Platform Listener. It is installed by the Reliable Agent Layer only when EFTL decides that the listener functionality is needed. The Reliable Agent Layer downloads a Platform Listener class from a web-server and deploys it at an agent host.

The costs of EFTL support can be expressed in monetary terms or system resources that have to be used for the functioning of EFTL. An application or an agent can decide whether those costs are acceptable in line with the additional reliability that EFTL provides.

Our fault-tolerant solution employs a persistent publish/subscribe messaging model. It was the premise that allowed us to develop an almost completely external and platform-independent system. With the introduction of clustered messaging brokers and the fault-tolerant Mobile Connector, we can guarantee the exactly-once consumption of messages by the agents.

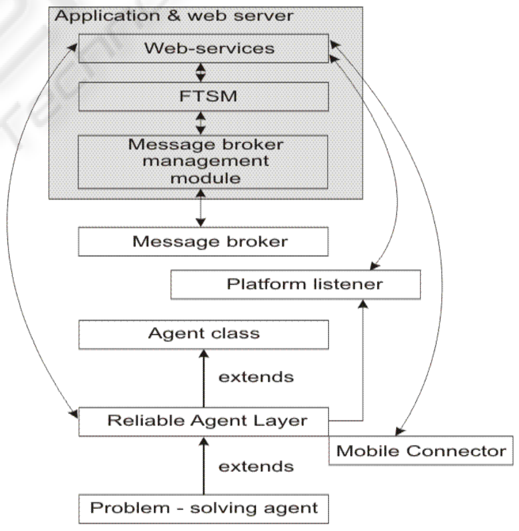

Figure 1: EFTL Architecture

4.1 The Mobile Connector

After an agent registers with EFTL, it obtains the credentials needed to make subscriptions or to publish a message to a message topic. The Mobile Connector is a facility that allows agents to communicate independently to the changes in their life cycles. It defines message selectors which can be used to allow context-aware messaging within a multi-agent platform.

The Mobile Connector is used to subscribe and/or publish to a message topic. If a message-receiving acknowledgement does not reach the message broker, due to link problems, then the message is resent. The agent would receive another copy of the same message. To enforce the exactly-once property, every message published in EFTL is uniquely numbered. This allows the Mobile Connector to discard messages which have already been consumed. This process can be represented by the following pseudo-code:

```
while(subscribed to a topic)
begin
    wait for next message;
    receive message;
    read unique MsgID;
    if(MsgID <= ConsumedMsgID)
    begin
        discard the message;
    end
    else
    begin
        consume the message;
        ConsumedMsgID = MsgID;
    end
end-while;
```

The actions of the Mobile Connector depend upon the changes in agent lifecycle and that is why the Reliable Agent Layer informs its Mobile Connector about each relevant change. Then, the Mobile Connector is able to perform all the operations that precede or follow events which can cause temporary disconnections from a message broker. At the moment of its creation, the Mobile Connector registers itself with the message broker, and creates both a publisher and a durable subscriber. Once it establishes the connection to the broker, this component makes sure that the link is reliable and if it detects a disconnection, it reconnects to the broker.

To solve the problem of message loss during periods of disconnection from the message broker, the Mobile Connector only disconnects from the message broker prior to the next migration step, but its primary subscription stays valid at the broker so that all the missed messages are forwarded to the agent as soon as it reconnects to the broker. This way, the Mobile Connector does not affect an agent's autonomy but guarantees message delivery.

4.2 The EFTL Web-services

Reliable agents use web-services for cost evaluation, contract signing, initial registrations with EFTL, deregistration from EFTL and redundant ways of communication if the standard publish/subscribe model fails. In order to register an agent and provide it with fault-tolerant support, web-services need information about the agent platform and that particular agent.

Web-services act as a gateway from the Reliable Agent Layer to the other EFTL components - the Fault-Tolerant System Manager and the Messaging Broker Management Module.

A multi-agent system can be comprised of reliable agents, which are supported by EFTL, and non-reliable agents. However, cooperation in the system is not constrained by the category to which the agents belong. EFTL controls only the agents with the Reliable Agent Layer, but listens to communication between all the agents within a platform. It can recover only reliable agents. The recovery of an agent might require the use of information about communication of that agent with all the other agents, irrespective of whether those agents were reliable or not.

4.3 Negotiable EFTL Support

Based on application domain, system goals and specific performance requests, developers can identify the agents which are critical to their mission. According to that information, the agents which have to be supported by EFTL and therefore provided with fault-tolerant support, need to have valid contracts with EFTL. A multi-agent system can be classified in one of three groups in regard to its demand for reliability: it can be either a high, medium or low demand system. If a system has a high demand for reliability, it probably has a mission critical application which wants EFTL support by all means and at any cost. Therefore, it will sign a contract with EFTL without any negotiation. If it has a medium demand for reliability, it needs to negotiate the costs with EFTL before it makes a decision about its support. If a system has a low demand for reliability, it will not use EFTL support.

The negotiation of costs by a system which has medium demand for reliability can be done by each agent separately or by an external application.

If an agent negotiates the costs with EFTL, it has to read an activation profile which defines the costs the agent is ready to accept. Those costs are sent to the Contractor web-service which compares them to the real costs dependant on the platform type and agent size. If the accepted costs are lower or equal to the real costs, the agent signs a contract for EFTL support.

If an external application negotiates the costs with EFTL, it has to forward information about the agent platform and agents which are going to be used, to the Contractor web-service. In order to facilitate the negotiation process, the application can use the EFTL Negotiator class which is distributed as part of the EFTL system. The Contractor web-service returns the real costs so that the application can choose whether or not it wants EFTL support. If it decides to sign a contract with EFTL, that contract holds information about all the agents that are going to be supported. Therefore, the application has to forward the contract details to every agent in order to allow them to register with EFTL.

4.4 Recovery Procedures

The checkpointing procedures used in EFTL do not affect agent mobility. When an agent decides to move quickly to another host, its Reliable Agent Layer might not have time to save a local checkpoint. If the agent fails, EFTL will have to try to find an earlier checkpoint in order to recover that agent. This kind of checkpointing is developed in order to preserve agent autonomy and not restrict mobility.

When the Platform Listener detects an event which might impair the functioning of the overall system, it notifies FTSM (Fault-Tolerant System Manager). Following the detection of the agent's death, FTSM, which listens to the topic that the Platform Listener published the notification to, decides to recover the agent. It sends the recovery command to another reliable agent closest to the place where the recovery is going to take place. That reliable agent performs the whole recovery procedure. The dead agent is recovered from its last checkpoint if its host is alive and functioning. If the host is not alive, in order to prevent the problem of blocking while we wait for the host to recover, EFTL, using the web service which has access to the hosts registry, finds out which other hosts provide the same services as the failed host. Then, the agent is sent to the host which is most similar to the failed one. When the agent is recovered, EFTL resends all the messages which have been produced by other agents from the moment of its checkpoint save until the moment of its death. This way the state of the agent is driven to the point in which it is consistent with the states of all the other agents present in the system, regardless of whether they are reliable or not.

When EFTL detects that an agent's life cycle has not changed during a longer period of time, it may decide to ping the agent using the publish/subscribe messaging subsystem. If the agent does not respond to the pings, EFTL concludes that it is non-responsive, removes it from the multi-agent system and recovers it from the latest checkpoint.

EFTL checks whether the recovery process is finished in a timely manner. An agent taking responsibility for the recovery of another agent might fail during that recovery process. Then EFTL tries to find another agent capable of doing that work. EFTL ceases the recovery if it cannot be finished within a predefined period of time. Moreover, EFTL will not be able to recover an agent which did not save any of its checkpoints.

EFTL uses the web service to access hosts registry in case of resource unavailability. When an agent is blocked due to inaccessible resources, EFTL sends it to a host which has exactly the same or similar resources.

In the case where network partitioning is detected, EFTL watches the agent actions and prevents them from running into situations which can cause their failures. When an agent (e.g. Agent A) is unable to move to a destination host, EFTL searches the system to find another agent (e.g. Agent B) of the same type which can move to the destination host. If such an agent is found, EFTL clones Agent B and sends it to the destination host. Then, using the publish/subscribe system and Java reflection mechanism, EFTL updates Agent B's state with the state of Agent A which is removed from the system.

5 APPLICATION DEVELOPMENT WITH EFTL

The external part of EFTL has to be deployed to the application and web servers. It is distributed in the form of Java archive files, so the process of deployment is simple. Those files have to be copied to the deployment folders of the application and web servers. The next step is initialization of the messaging subsystem. If the messaging subsystem is a part of the application server, it is usually started at the same time as the server instance. If the messaging subsystem is a separate application, it has to be started and configured for proper use by EFTL. Configuration of each messaging system depends on what levels of reliability, fail-over and scalability are required within the messaging subsystem.

The Reliable Agent Layer class has to be visible to a problem-solving agent class. The developer does not have to implement any new methods related to the special fault-tolerant features. If the

Reliable Agent Layer is implemented as a separate class, the public methods of the basic Agent class are defined as *final*. Then the problem-solving agent cannot implement those methods in its code. It can only implement the methods of the Reliable Agent Layer class which have similar names. For example, if the Agent's method name was *beforeMove*, the Reliable Agent Layer's name would be *beforeReliableMove*. If the agent platform's license allows changing of its source code, the Reliable Agent Layer functionality can be embedded in the basic Agent class. Then, the developers could use the same method names as in the Agent class.

Special property files have to be present at the host in which the reliable agents are being initialized. These configurable files are read by the Reliable Agent Layer. They provide the layer with the information on how to connect to the rest of EFTL. The developer or the system administrator has to edit the connection values in these files after the application, web and messaging servers have been configured.

6 PERFORMANCE ANALYSIS

Performance analysis of EFTL had to include two distinct categories which are applicable to any fault-tolerant system: its reliability level and messaging overhead. Context-awareness of the EFTL messaging subsystem was designed with one objective in mind - to reduce messaging overhead between components of the fault-tolerant system.

Tests were conducted in the JADE environment, using a fixed number of hosts distributed on different computers. Types of failures simulated in these tests were host and agent deaths. The host death rate was constant - one host failure per ten seconds. Every failed host was restarted after five seconds. The agent death rate was variable, and was a parameter of the simulation process. The choices of which hosts or agents should be killed for test purposes were random.

All the tests included a number of mobile agents which had to complete their round-trips across the network of hosts. Their itineraries were dynamically determined at the time of start-up. They stayed five seconds at each of the hosts. The percentage of agents which succeeded in visiting all the hosts and returning to the place of their origin determined the level of system reliability.

Our first experiment included a fixed number of mobile agents and a variable number of agent faults. The results of this experiment are shown in Figure 2.

It can be concluded that EFTL greatly improves system reliability, even in the cases of high failure rates. If we compare the reliability of a multi-agent system with and without EFTL support, we can see that EFTL is capable of delivering a high reliability level to a system which would completely fail without its support.

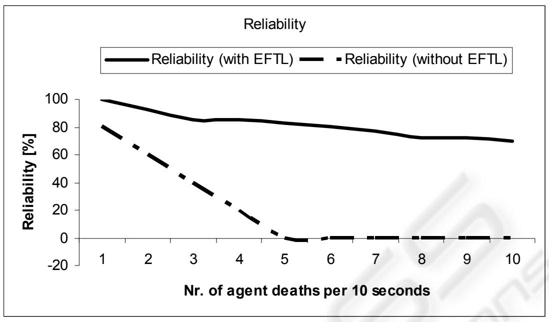

Figure 2: System reliability with and without EFTL support

The following experiments calculated the messaging overhead which was generated by EFTL. In the first experiment calculating this overhead, we used the formula:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{m} M_{ij}$$

O – messaging overhead
n – number of reliable agents
m – number of published EFTL messages
M_{ij} – size of the message *j*, published or received by the agent *i*
(equal to 0 if the agent *i* did not publish or receive message *j*)

The experiments showed that there was no dependency between the agent failure rate and the messaging overhead. However, the overhead was related to the number of mobile agents presented in the test-bed system.

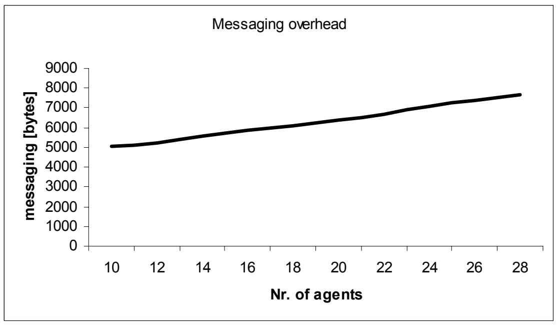

Figure 3: Messaging overhead [size of published messages]

As can be seen in Figure 3, messaging overhead slightly increases with the number of reliable agents in the system. However, this messaging overhead is so small that it can be compared to the overhead of the migration of one agent between two hosts.

In addition, Figure 4 shows that the number of messages published in the EFTL internal messaging subsystem grows with the number of reliable agents present in a multi-agent system. The publishing and delivery of these messages is rapid because they are small in content size. This does not generate any notable operation slowdown on an agent level.

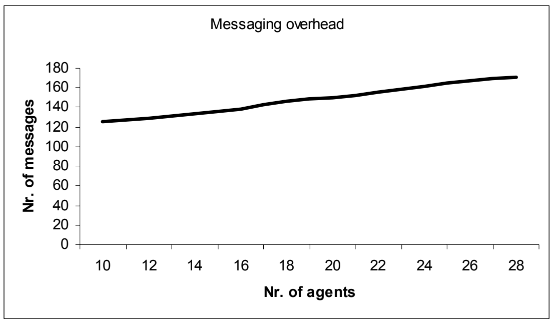

Figure 4: Messaging overhead [number of published messages]

7 CONCLUSION AND FUTURE WORK

EFTL introduced a substantial amount of platform-independence in the multi-agent fault-tolerant approaches. The idea to move as many components of the fault-tolerant system out of the agent platforms, using standard tools, allowed us to preserve agent autonomy and mobility. In addition to this, we proposed the use of the persistent publish/subscribe messaging model which employs context-aware message selection at the message brokers. This allowed us to decrease the messaging overhead which has to be present in any fault-tolerant solution. Our modification of the model, with the usage of the Mobile Connector, guarantees the exactly-once consumption of messages. Since all the components of EFTL can inherit fault-tolerance and scalability of the application, web and messaging servers, we can claim that our approach offers an extra level of reliability and high availability. Moreover, EFTL introduced negotiable fault-tolerant support based on the costs. An application or an agent can choose if it wants an extra level of reliability accompanied by costs that can be expressed in monetary terms or the additional usage of system resources.

Our future work will be focused on the development of different platform-independent checkpointing procedures with the use of Java reflection, and on the more adaptive mechanisms of EFTL usage and deployment in regard to its overheads.

REFERENCES

Bellifemine, F.; Caire, G.; Trucco, T.; Rimassa, G., 2003. *JADE administrator's guide*, TILAB S.p.A., Italy

Cao, J.; Feng, X.; Lu, J.; Chan, H.; Das, S.K., 2002. Reliable message delivery for mobile agents: push or pull, *Parallel and Distributed Systems, 2002. Proceedings Ninth International Conference on*, 314 – 320.

Cao, J.; Zhang, L.; Yang, J.; Das, S.K., 2004. A reliable mobile agent communication protocol, *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, 468 – 475.

Dake, W.; Leguizamo, C.P.; Mori, K., 2002. Mobile agent fault tolerance in autonomous decentralized database systems, *Autonomous Decentralized System, 2002. The 2nd International Workshop on*, 192 – 199.

Dalmeijer, M.; Rietjens, E.; Hammer, D.; Aerts, A.; Soede, M., 1998. A reliable mobile agents architecture, *Object-Oriented Real-Time Distributed Computing, 1998. (ISORC 98) Proceedings. 1998 First International Symposium on*, 64 – 72.

Eustace, D.; Aylett, R.S.; Gray, J.O., 1994. Combining predictive and reactive control strategies in multi-agent systems, *Control, 1994. Control '94. Volume 2., International Conference on*, 989 – 994.

Fedoruk, A.; Deters, R., 2002. Improving fault-tolerance by replicating agents, *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, ACM Press New York, NY, USA, ISBN:1-58113-480-0, 737 – 744.

Grantner, J.L.; Fodor, G.; Driankov, D., 1997. Using fuzzy logic for bounded recovery of autonomous agents, *Fuzzy Information Processing Society, 1997. NAFIPS '97. 1997 Annual Meeting of the North American*, 317 – 322.

Lyu, R. M.; Wong, Y. T., 2004. A progressive fault tolerant mechanism in mobile agent systems, Retrieved April 25, 2004, from http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/ SCI2003.pdf

Macedo, A.; Silva, F., 2002. Coordination of mobile processes with mobile groups, *Dependable Systems and Networks, 2002. Proceedings. International Conference on*, 177 – 186.

Mohindra, A.; Purakayastha, A.; Thati, P., 2000. Exploiting non-determinism for reliability of mobile agent systems, *Dependable Systems and Networks,* *DSN 2000. Proceedings International Conference on*, 144 – 153.

Moreau, L., 2002. A fault-tolerant directory service for mobile agents based on forwarding pointers, *Proceedings of the 2002 ACM symposium on Applied computing*, ACM Press New York, NY, USA, ISBN:1-58113-445-2, 93 – 100.

Padovitz, A.; Zaslavsky, A.; Loke, S. W., 2003. Awareness and Agility for Autonomic Distributed Systems: Platform-Independent Publish-Subscribe Event-Based Communication for Mobile Agents, *the 1st International Workshop on Autonomic Computing Systems, DEXA 2003*, Prague, Czech Republic

Patel, R. B.; Garg, K., 2004. Fault-tolerant mobile agents computing on open networks, Retrieved April 18, 2004, from http://www.caip.rutgers.edu /~parashar/AAW-HiPC2003/patel-aaw-hipc-03.pdf

Pechoucek, M.; Dobisek, M.; Lazansky, J.; Marik, V., 2003. Inaccessibility in multi-agent systems, *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, 182 – 188.

Taesoon, P.; Ilsoo, B.; Hyunjoo, K.; Yeom, H.Y., 2002. The performance of checkpointing and replication schemes for fault tolerant mobile agent systems, *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*, 256 – 261.

Zhigang, W.; Binxing, F., 2000. Research on extensibility and reliability of agents in Web-based Computing Resource Publishing, *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, Volume: 1, 432 – 435.

Zhou, J.; Jia, Z.; Chen, D., 2003. Designing reliable communication protocols for mobile agents, *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, 484 – 487.