Evaluation of Fault Tolerant Mobile Agent Execution

H. Hamidi and K.Mohammadi
Department of Electrical Engineering
Iran University of Science & Technology
Iran-Tehran

Abstract: **The reliable execution of a mobile agent is a very important design issue to build a mobile agent system and many fault-tolerant schemes have been proposed. Hence, in this paper, we present evaluation of the performance of the fault-tolerant schemes for the mobile agent environment. Our evaluation focuses on the checkpointing schemes and deals with the cooperating agents.**
We derive the FANTOMAS (Fault-Tolerant approach for Mobile Agents) design which offers a user transparent fault tolerance that can be activated on request, according to the needs of the task, also discuss how transactional agent with types of commitment constraints can commit.Furthermore this paper proposes a solution for effective agent deployment using dynamic agent domains.

Keywords**: Checkpointing, FANTOMAS, Fault Tolerant, Mobile Agent, Network Management, Transactional Agent**.

I. Introduction

A mobile agent is a software program which migrats from a site to another site to perform tasks assigned by a user. For the mobile agent system to support the agents in various application areas, the issues regarding the reliable agent execution, as well as the compatibility between two different agent systems or the secure agent migration, have been considered. Some of the proposed schemes are either replicating the agents [1,2] or checkpointing the agents [3,4]. For a single agent environment without considering inter-agent communication, the performance of the replication scheme and the checkpointing scheme is compared in [5] and [6].

In the area of mobile agents, only few work can be found relating to fault tolerance. Most of them refer to special agent systems or cover only some special aspects relating to mobile agents, e. g. the communication subsystem. Nevertheless, most people working with mobile agents consider fault tolerance to be an important issue [7,8].

Johansen et al. [9] detect and recover from faulty migrations inside the TACOMA [10] agent system. When an agent migrates, a rear-guard agent is created that stays on the origin node. It monitors the migrated agent on the destination node. This very simple concept does not tolerate network partitioning. In the scope of the TACOMA project, Minsky et al. [11] propose an approach based on an itinerary concept, i. e. a plan, which nodes the mobile agent has to visit and what it has to do there. They assume that the itinerary is known at start time and the order of visited nodes is fixed. Fault tolerance is obtained by performing every itinerary step on multiple nodes concurrently and sending the results (resp. the Mobile Agent) to all nodes of the next step. The majority of the received inputs from the last step becomes the input of the new task for this step and so on. Thus, a certain number of faults can be tolerated in each step.

This work is partially supported by The Iran Communication Research Center.

Disadvantages of this fault tolerance concept are the very inflexible description of the itinerary and the simple model of Mobile Agents. For example, communication between different mobile agents is not included in this concept, so it is not suited for distributed or parallel applications.

Strasser and Rothermel present a more flexible itinerary approach for fault tolerance within the Mole system [12]. Independent items in this enhanced itinerary can be reordered. Each itinerary stage comprises the action that has be to done on one node. When the mobile agent enters a new stage by migrating to the next node of the itinerary, called a worker node, it is also replicated onto a number of additional nodes, called observers. If the worker becomes unavailable (due to a node or network fault), the observer with the highest priority is selected as the new worker by a special selection protocol. A voting protocol ensures the abortion of wrong multiple workers in case of a network fault. The voting protocol is integrated into a 2-phase commit protocol (2PC) that encloses every stage execution. These protocols cause a significant communication overhead. Just as in the work of Minsky et al., nothing is said about the interaction between different mobile agents that are executed concurrently.

Vogler et al. [13] introduce a concept for reliable migrations of mobile agents based on distributed transactions. The migration protocol is derived from known transaction protocols like the already mentioned 2PC. Besides the migrations, no other fault tolerance aspects of mobile agents are treated.

II. The Mobile Agent Model

A. Mobile agents

An agent is a program which can be autonomously performed on multiple object servers . A home computer of an agent is one where the agent is initiated . A current server of an agent is one where the agent exists . An agent issues methods to an object server to manipulate objects in the current object server . Every object server is assumed to support a platform to perform agents .

First, an agent A is autonomously initiated on an object server . Suppose an agent A lands at an object server D_i to manipulate an account object through a method increment . Here, suppose another agent B is resetting the account object . Since a method reset conflicts with a method increment, the agent A cannot be started . A pair of agents A_1 and A_2 manipulate a same object through conflicting methods. After landing at an object server D_j , the agent A is allowed to be performed on the object server D_j if there is no agent on an object server D_j which conflicts with an agent A.

An agent A can be replicated in multiple replicas A_1 , … A_m ($m \geq 2$) . Each replica A_i is autonomously performed . By replicating an agent, parallel processing and fault-tolerant computings of the agent can be realized . Even if some replica is faulty, other replicas of the agent can be performed . In addition, replicas are in parallel performed on object servers .

An agent autonomously finds a destination object server to which the agent moves . If the destination object server is

faulty, the agent finds another destination object server. An agent is not faulty by escaping faulty servers .

B. Fault Model

Several types of faults can occur in agent environments. Here, we first describe a general fault model, and focus on those types, which are for one important in agent environments due to high occurrence probability, and for one have been addressed in related work only insufficiently.

- Node failures: The complete failure of a compute node implies the failure of all agent places and agents located on it. Node failures can be temporary or permanent.
- Failures of components of the agent system: Failures of agent places, or components of agent places become faulty, e. g. faulty communication units or incomplete agent directory. These faults can result in agent failures, or in reduced or wrong functionality of agents.
- Failures of mobile agents: Mobile agents can become faulty due to faulty computation, or other faults (e. g. node or network failures).
- Network failures: Failures of the entire communication network or of single links can lead to isolation of single nodes, or to network partitions.
- Falsification or loss of messages: These are usually caused by failures in the network or in the communication units of the agent systems, or the underlying operating systems. Also, faulty transmission of agents during migration belongs to this type.

Especially in the intended scenario of parallel applications, node failures and their consequences are important. Such consequences are loss of agents, and loss of node specific resources. In general, each agent has to fulfill a specific task to contribute to the parallel application, and thus, agent failures must be treated. In contrast, in applications where a large number of agents are sent out to search and process information in a network, the loss of one or several mobile agents might be acceptable.

C. Concepts for a Fault Tolerance Approach

As shown in the previous sections, despite the generally agreed-upon necessity, existing agent systems contain only limited provisions for fault tolerance, if at all. Especially treatment of node or agent failures with support for communicating agents is covered only insufficiently. However, such support is essential for distributed and/or parallel applications. This section discusses possibilities and approaches to augment an agent system to achieve fault tolerance, with focus on these fault and application types. First, the goals are described, then the fault model for an agent system is explained, and the faults examined with respect to their occurrence probability and treatment in existing systems. From this, a set of faults is determined, for which further treatment is still needed. After that, an overview over fault tolerance approaches in known environments is given, and examined, if and how they are suited for mobile agents. From these investigations, the FANTOMAS concept is developed.

III. transactional Agents

A. Commitment conditions

- A transactional agent A manipulates objects in multiple object servers by moving around the object servers . A scope Scp (A) of an agent A means a set of object servers which the agent A possibly to manipulate.

- The atomic, majority, and at-least-one commitment conditions are shown in forms of $\binom{n}{r}$, $\binom{n}{(n+1)/2}$, and $\binom{n}{1}$ commitment conditions, respectively . generalized consensus conditions with preference are discussed in a paper [14] . A commitment condition com(A) is specified for each agent A by an application . There are still discussions on when the commitment condition com(A) of an agent A can be checked while the agent A is moving around object servers . Let H (A) be a history of an agent A, i.e. set of object servers which an agent A has so far manipulated($H(A) \subseteq Scp(A)$) .The commitment condition Com(A) can hold only if at least one object server is successfully manipulated, i.e. $|H(A)|$ =1 in the at-least-one commitment condition .

If an agent A leaves an object server D_i , an agent named surrogate of A is left on D_i [Figure1] . The surrogate agent A_i still holds objects in the object server D_i which are manipulated by the agent A on behalf of the agent A.

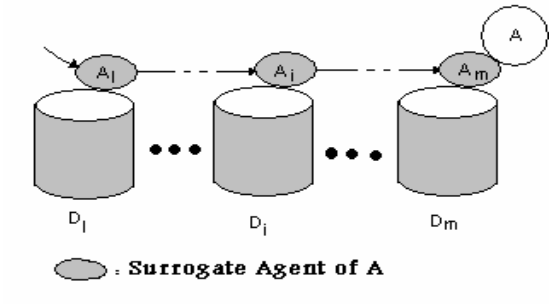

Figure 1. Surrogate agents.

B. Commitment

There are two types of agents, ordered agents and unordered agents . Every pair of ordered agents manipulate objects in a well-defined way . Each ordered agent A is assigned a precedent identifier pid(A). An agent A_1 precedes another agent A_2($A_1 \rightarrow A_2$) if pid(A_1)< pid (A_2). For example, timestamp [15] can be used as identifier of an agent . That is, the identifier pid(A) of an agent A is time ts(A) when the agent A is initiated at the home server . An agent A_1 precedes another agent A_2 only if ts(A_1)<ts(A_2). An agent A_1 is concurrent with another agent A_2 ($A_1 \| A_2$) if neither A_1 precedes A_2 nor A_2 precedes A_1. Here , the agents A_1 and A_2 can be performed on an object server in any order . If a pair of the agents A_s and A_t conflict on object servers D_i and D_j the agents A_s and A_t are required to be performed in the precedence order at the object servers D_i and D_j . There never occurs deadlock . Like locking protocols , an unordered agent can obtain an object if no conflicting agent obtains the object .

C. The FANTOMAS Concept

From these considerations, we choose independent checkpointing with receiver based logging as base for our fault tolerance approach for mobile agents. Adhering to the agent paradigm, and exploiting the already available facilities of the mobile agent resp. the agent environment, an agent is used as the stable storage for the checkpointed state and the message log. For each mobile agent (called user agent in the following), for that fault tolerance is enabled, a logger agent is created. A user agent and its logger agent form an agent pair (figure 2). The logger agent does not participate actively in the

application's computation, and thus needs only a small fraction of the available CPU capacity. It follows the user agent at a certain, non-zero, distance on its migration path through the system. They must never reside on the same node, so that not a single fault destroys both of them. User and logger agent monitor each other, and if a fault is detected by one of them, it can rebuild the other one from its local information.

The creation of the agent pair is readily derived from the already existing migration facilities. To create a logger agent, the user agent serializes its state in the same way as for a migration, and sends it to a remote agent place. There, a new agent is created from this data. Different from migration, the new agent does not start the application module that was sent with the state information, and the user agent continues normal execution. Further, the communication unit of the agent is exchanged against a version that first forwards each incoming message to the logger agent before delivering it locally.

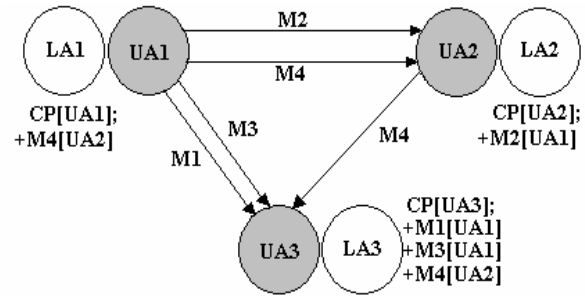

Figure 2. Example for an application with three user and logger agents with checkpoints (CP) and messages (M) [16].

IV. Checkpointing and Logging Schemes

A. Checkpointing Schemes

The checkpointing schemes considered for the analysis are as follows:

- **Loosely Coordinated Checkpointing Coordination(LCCP) Scheme:** The checkpoints of an agent are sequentially numbered. On the receipt of a message from another agent, a forced checkpoint is taken before processing the message, if the sender's checkpoint number is larger than the receiver's checkpoint number.
- **Communication Mode Based Checkpointing Coordination(CMCP) Scheme:** An agent is either in the "sending" mode or in the "receiving" mode, regarding the communication status. A forced checkpoint is taken before the agent changes its communication status from a "sending" mode to a "receiving" mode.
- **Lazy(LAZY) Scheme:** This scheme is an extension of the LCCP scheme. The rule of the LCCP scheme is applied if the sender's checkpoint sequence number is a multiple of *Z,* where *Z* can be any integer value.
- **Timer Based Checkpointing Coordination(TBCP) Scheme:** The rule of the LCCP scheme is applied. However, once an induced checkpoint is taken, the timer for the next basic checkpoint is reset so that frequent checkpointing can be avoided.

B. Logging Schemes

The following three logging schemes are considered for the evaluation.

- **Pessimistic Message Logging (PML) Scheme:** On the receipt of a message from another agent, the message is logged into the stable storage before it is processed.
- **Dependency Based Message Logging (DML) Scheme:** A message received from another agent is first copied into the volatile log space. The volatile logs are flushed into the stable storage before the agent sends out any message to another agent.
- **Optimistic Message Logging (OML) Scheme:** A message received from another agent is first copied into the volatile log space.

V. The functioning of an agent

The lifecycle of an agent consists of three stages: Normal operational phase, when agents roam their domains performing the regular tasks; trading phase, where domain corrections are initiated and cloning/merging phase, where heavily loaded agents can multiply, or two under loaded agents can merge.

First we have to define the idea of logical topology. Logical network topology is a virtual set of connections stored in the hosts. If the physical topology of the managed network is rare in connections the use of logical topology can facilitate the correct functioning of the algorithm. The logical topology should follow the physical topology as setting up an arbitrary set of connections will increase the migration time.

As in most mobile agent applications the greatest reason again using them is security. Some people still look at mobile agents as a form of viruses and mobile agent platforms as security holes allowing foreign programs run on the system. Concerning the general threats of mobile agents is out of scope of this paper. Instead we would like to outline the security issues concerning network management. The main difference between general mobile agents and network management agents that the latter ones cannot be closed in a separate running environment because network management agents must have the privilege to modify the configuration of the host to perform its tasks. Misusing these privileges can severely harm the nodes.

If the agent domains are not separated into distinct partitions we call the domains coherent. By keeping the coherence the migrating times can be much smaller comparing to the case when the agent domains can be partitioned into remote parts. On the other hand keeping domain coherence places limit to the trading process as the agent must know its "cutting" nodes that cannot be traded without splitting its area into distinct pieces.

VI. Simulation Results and Evaluation

A simulator was designed to evaluate the algorithm. The system was tested in several simulated network conditions and numerous parameters were introduced to control the behavior of the agents.

We also investigated the dynamic functioning of the algorithm. Comparing to the previous case the parameter configuration has a larger effect on the behavior of the system. The most vital parameter was the frequency of the trading process and the predefined critical workload values.

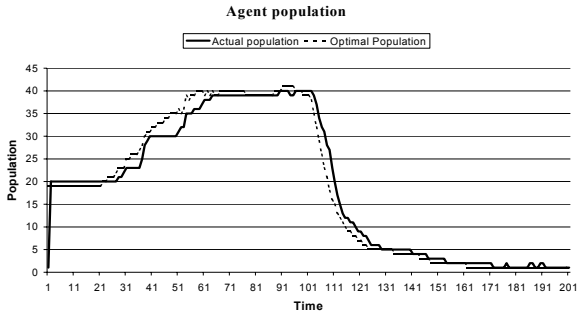

Figure 3: The size of the agent population under changing network conditions.

Figure 3 shows the number of agents on the network. In a dynamic network situation. The optimal agent population is calculated by dividing the workload on the whole network with the optimal workload of the agent.

Simulation results show that choosing correct agent parameters the workload of the agents is within a ten percent environment of the predefined visiting frequency on a stable network. In a simulated network overload the population dynamically grows to meet the increased requirements and smoothly returns back to normal when the congestion is over.

We evaluate transactional agents in terms of access time compared with client-server model . In the evaluation, three object servers D_1 , D_2 and D_3 are realized in Oracle which are in sun workstation (SPARC 900MHz x 2) and a pair of Windows PCs(Pentium3 1.0GHz x 2 and pentium3 500MHz), respectively. The object servers are interconnected with 100base LAN . JDBC classes are already loaded in each object server .
An application program A manipulates table objects by issuing select and update to some number of object servers at the highest isolation level, i.e. select for update in Oracle . The program is implemented in Java for Aglets and a client-server model . In the client-server model . the application program A is realized in Java on a client computer . In the transactional agent model , the application A is realized in Aglets .

The computation of Aglets is composed of moving, class loading, manipulation of objects, creation of clone, and commitment steps. In the client-server model, there are computation steps of program initialization , class loading to client, manipulation of objects, and two-phase commitment .

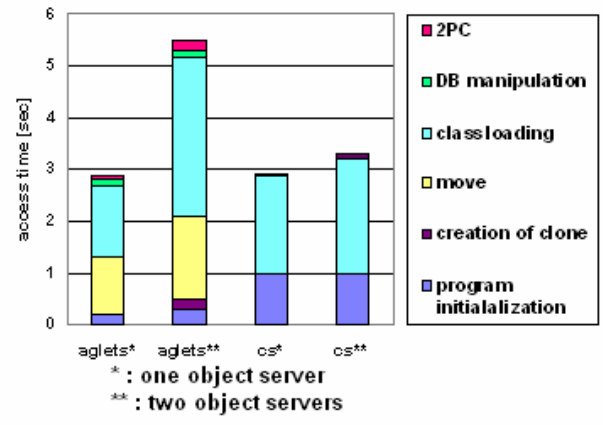

Figure4. Access time.

Figure 4 shows how long it takes to perform each step for two cases, one for manipulating one object server and another for manipulating two object servers, in client-server (cs) and transactional agent (TA) models. In the transactional agent model, Aglets classes are loaded to each object server before an agent is performed**.** A mobile agent system consisting of *Ns* sites and *Na* agents has been simulated. The working time on each site follows an exponential distribution with a mean *Tw*. The next site for the visit is selected randomly out of *Ns* sites and the migration time of *T*m is assumed. While working on a site, an agent communicates with another agent by sending and receiving a message. The message-sending rate of each agent follows a Poisson process with a rate λ c . For each message-sending event, the receiver is selected randomly and the communication delay *of Tc* is assumed.

An agent takes a basic checkpoint before the migration into a volatile storage. The fixed time of *Tcp* is assumed to save an induced checkpoint into a stable storage and to log the messages into the stable storage, the fixed time of *Tl* is also assumed. A system site fails with an interval following an exponential distribution with a mean *Tf*.

First, to measure the influence of system parameters on the checkpoint and the recovery cost, five test cases are simulated and for each case, the *checkpointing time,* the *logging time,* the *number of rollback agents,* the *lost time* and the *total overhead* are obtained.

The system environment with the following parameter values is simulated: N_s*=40,* N_a*=lO,* T_w*=3000* , λ *c=1/400* , T_f*=320000,* T_{cp}*=100,* T_l *=10* and *Tc=10.*
Case 1: The value of λ c changes to 1/800.
Case 2: The value *of* T_w changes to 1500.
Case 3: The value of T_{cp} changes to 50.
Case 4: The value of N_a changes to 5.
The Cases 1-4 differs only one system parameter value compared to the Case "Basic".

From the results, it is observed that the CMCP scheme is very sensitive to the changes of the communication rate and among the logging schemes the OML scheme shows the most stable performance as the changes of the system parameters regardless of the rollback propagation.

Figure 5 analyze the influence of the communication rate in more detail, where T_f*=32000,* T_w*=3000* and T_{cp}*=100.* The CMCP scheme and the DML scheme show the most drastic changes in the checkpoint cost and the lost time as the communication rate varies. However, the CMCP scheme shows very slow increases in the number of rollback agents for the communication rate increase.

Figure 5-a shows experimental results of the four test cases, in which the values of T_f*=320000,* T_w*=3000* , λ c=1/400 and T_{cp}*=100* are used for Figure 5-a; for Figure 5-b, the value of λ c changes to 1/6400 ; the value of T_{cp} also changes to 10 in Figure 5-c .

As shown in the figure 5, the LAZY schemes work well for the environment with high checkpointing overhead and low failure rate, while the CMCP scheme is good for the cases with low checkpointing overhead and high failure rate. In most of the cases, logging schemes can be a good choice except the cases where both the checkpointing overhead and the failure rate are low.

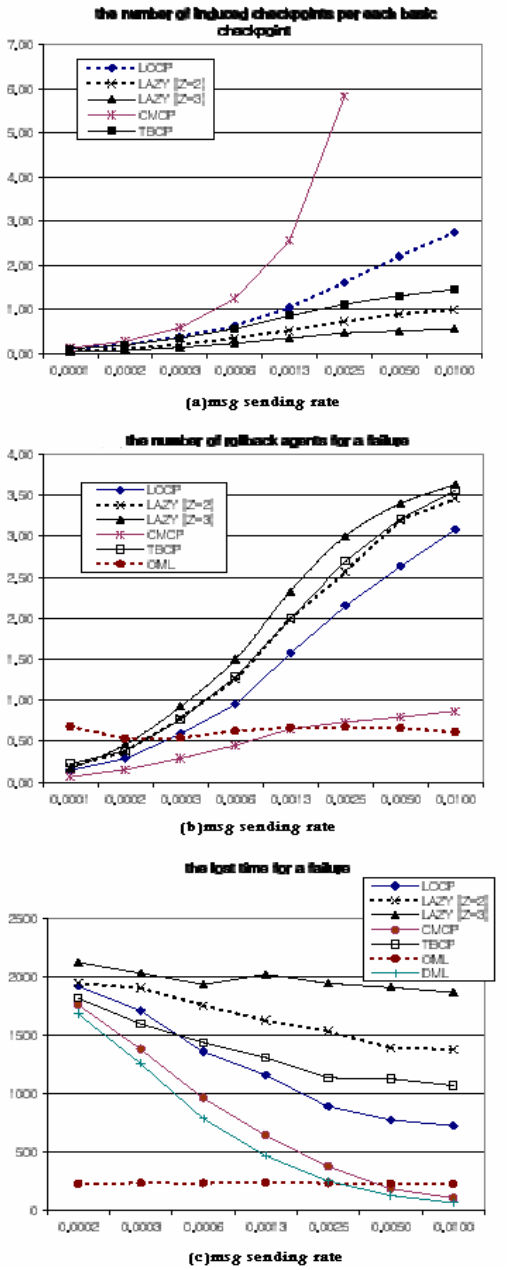

Figure 5. The Influence of Communication Rate.

VII. Conclusion

This paper discussed a mobile agent model for processing transactions which manipulate object servers . An agent first moves to an object server and then manipulates objects. We showed the evaluation of the mobile agent-based transaction systems for applications . If Aglets classes are a priori loaded, the transactional agents can manipulate object servers faster than the client-server model.

Also, we have presented the experimental evaluation of the performance of the fault-tolerant schemes for the mobile agent environment. General possibilities for achieving fault tolerance in such cases were regarded, and their respective advantages and disadvantages for mobile agent environments, and the intended parallel and distributed application scenarios shown. This leads to an approach based on warm standby and receiver side message logging.

In the paper dynamically changing agent domains were used to provide flexible, adaptive and robust operation.

References

[1] H.Hamidi and K.Mohammadi, "Modeling and Evaluation of Fault Tolerant Mobile Agents in Distributed Systems ," *Proc. Of the 2th IEEE Conf . on Wireless & Optical Communications Networks (WOCN2005),pp.91-95,* March 2005.

[2] S. Pleisch and A. Schiper, "Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agree Problems," *Proc. of the 19th IEEE Symp. on Reliable Distributed Systems,* pp. 11-20,2000.

[3] S. Pleisch and A. Schiper, "FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Ap-proach," *Proc. 2001 Int'l Conf on Dependable Systems and networks,pp.215-224,luI.2001*

[4] M. Strasser and K. Rothermel, "System Mechanism for Partial Rollback of Mobile Agent Execution," *Proc. 20th Int'l Conf on Distributed Computing Systems,* 2000.

[5] T. Park, I. Byun, H. Kim and H.Y. Yeom, "The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systems," *Proc. 21th IEEE Symp. on Reliable Distributed Systems,* 2002.

[6] L. Silva, V. Batista and 1.G. Silva, "Fault-Tolerant Execution of Mobile Agents," *Proc.In!''1 Conf on Dependable Systems and Networks,* 2000.

[7] M. Izatt, P. Chan, and T. Brecht. Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. In Proc. ACM 1999 Conference on Java Grande, pages 1524, June 1999.

[8] D. Wong, N. Paciorek, and D. Moore. Java-based mobile agents. Communications of the ACM, 42(3):92-102, March 1999.

[9] D. Johansen, R. van Renesse, and F. B. Schneider. Operating System Support for Mobile Agents. Proceedings of the 5th. IEEE Workshop on Hot Topics in Operating Systems. , USA, pages 42-45, 1995.

[10] D. Johansen, R. van Renesse, and F. B. Schneider. An Introduction to the TACOMA Distributed System, Version 1.0. Report, Institute of Mathematical and Physical Science, University of Tromsf3, Norway, 1995.

[11] Y. Minsky, R. van Renesse, F. B. Schneider, and S. D. Stoller. Cryptographic Support for Fault-Tolerant Distributed Computing. In Proc. 7th ACM SIGOPS European Workshop, pages 109-114. ACM Press, September 1996.

[12] J. Baumann, F. Hohl, and K. Rothermel. Mole - Concepts of a Mobile Agent System. Technical Report TR-1997-15, Fakultat Informatik, Germany, 1997.

[13] H.Vogler , T . Kunkelmann – and M . – L . Moschgath . An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions. In Proc 2002 International Conference on Parallel and Distributed Systems (ICPADS ' 2002) . IEEE Computer Society , December 1997 .

[14] I. Shimojo, T. Tachikawa, and M. Takizawa . "M-ary commitment protocol with partially ordered domain". proc.of the 8th Int'l Conf . on Database and Expert Systems Applications (DEXA'97), pages 397-408,1997 .

[15]P.A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency control and recovery in database systems . In Addison Wesley, 1987 .

[16] S. Petri and C. Grewe. A Fault-Tolerant Approach for Mobile Agents. In Dependable Computing - EDCC-3, Third European Dependable Computing Conference, Fast Abstracts. Czech Technical University in Prague, September 1999.