A New Approach to Fault Tolerant Mobile Agent Execution in Distributed Systems

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

***Abstract*: Mobile agents are no longer a theoretical issue since different architectures for their realization have been proposed . With the increasing market of electronic commerce it becomes an interesting aspect to use autonomous mobile agents for electronic business transactions. Being involved in money transactions, supplementary security features for mobile agent systems have to he ensured. .In this paper, we present FATOMAS , a java – based fault – tolerant mobile agent system based on an algorithm presented in an earlier paper . In contrary to the standard "place – dependent" architectural approach , FATOMAS uses the novel „ agent – dependent „ approach introduced in the paper . In this approach , the protocol that provides fault tolerance travels with the agent . This has the important advantage to allow fault – tolerant mobile agent execution with out the need to modify the underlying mobile agent plat form .**

We derive the FATOMAS (Fault-Tolerant Mobile Agent System) 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*: FATOMAS, Fault Tolerant, Mobile agent, Network Management, checkpointing , Redundant .**

I. Introduction

A mobile agent is a software program which migrates 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].Failures in a mobile agent system may lead to a partial or complete loss of the agent[1] .To achieve fault – tolerance , the agent owner (i.e., the person or application creating and configuring the agent) can try to detect the failure of its agent, and upon such an event launch a new agent. However , this requires the ability to correctly detect the crash of the agent ,i.e., to distinguish between a failed agent and an agent that is delayed by slow processors or slow communication links. Unfortunately this can not be achieved in systems such as the Internet. An agent owner who tries to detect the failure of his agent thus cannot prevent the case where he mistakenly thinks that the agent has crashed. In this case , launching a new agent leads to multiple executions of the agent, i.e., to the violation of the desired exactly-once property of agent execution. Even though this may be acceptable for certain applications (e.g., applications whose operations do not have side-effects ,i.e., are idempotent),others clearly forbid it. Fortunately ,multi-ple agent executions can be prevented in environments with unreliable failure detection by replicating agents with an adequate protocol. Indeed, if failure detection is unreliable, replication by itself dose not ensure the exactly-once execution property .for example, exactly- once execution is not ensured by the protocol of [9], which assumes a perfect failure–detection mechanism. Some systems have tried to address the exactly –once issue in the context of unreliable failure detection , and have proposed complex solution based on transactions and leader election [2,10]. Other approaches address the exactly- once execution problem by detecting duplicate agents at the end of the agent execution , and undoing at that moment the effects of multiple executions [11 ,12] . However , undoing the effects of duplicate agents at the end of the execution is not simple , and often limits dramatically the overall system throughput . In contrast to these different approaches , we have presented in an earlier paper [13] an approach that ensures the exactly- once execution property using a very simple principle:

The mobile agent execution is modeled as a sequence of agreement problems . In the current paper , we present FATOMAS , a java- based Fault – Tolerant Mobile Agent System , that implements this approach .

In order to characterize the architecture of FATOMAS , we start by introducing two approaches called place dependent and agent – dependent . Place–dependent is the standard approach that integrates fault tolerance into the mobile agent platform (the plat form , that provides the support for mobile agents). Agent dependent is the new approach introduced by FATOMAS . In this approach , the protocol that provides fault tolerance travels with the agent .

This has the important advantage to allow fault – tolerant agent execution without having to modify the underlying mobile agent platform . Currently, FATOMAS supports object space's Voyager mobile agent platform [14] . However our design enables to easily port FATOMAS to other mobile agent platforms .

II. Model

We assume an asynchronous distributed system, i.e., there are no bounds on transmission delays of messages or on relative process speeds. An example of an asynchronous system is the

Internet. Processes communicate via message passing over a fully connected network.

A. Mobile agent Model

A mobile agent executes on a sequence of machines, where a places $p^i (0 \le i \le n)$ provides the logical execution environment for the agent. Each place runs a set of services, which together compose the state of the place. For simplicity, we say that the agent" accesses the state of the place, " although access occurs through a service running on the place. Executing the agent at a place p_i is called a stage S_i of the agent execution. We call the places where the first and last stages of an agent execute (i.e., $p_0 andp_n$) the agent source and destination, respectively. The sequence of places between the agent source and destination respectively. The sequence of places between the agent source and destination(i.e., $p_0, p_1, \cdots p_n$) is called the itinerary of a mobile agent. Whereas a static itinerary is entirely defined at the agent source and does not change during the agent execution, a dynamic itinerary is subject to modifications by the agent itself.

Logically , a mobile agent executes in a sequence of stage actions (Fig. 1).Each stage actions a_i consists of potentially multiple operations $op_0, op_1, \cdots$ Agent ($0 \le i \le n$) at the corresponding stage S_i represents the agent a that has executed the stage action on places p_j j <i) and is about to execute on place p_i .The execution of a_i on place p_i results in a new internal state of the agent as well as potentially a new state of the place (if the operations of an agent have side effects, i.e., are non idempotent).We denote the resulting agent a_{i+1}. Place p_i forwards to p_{i+1} (for i < n).

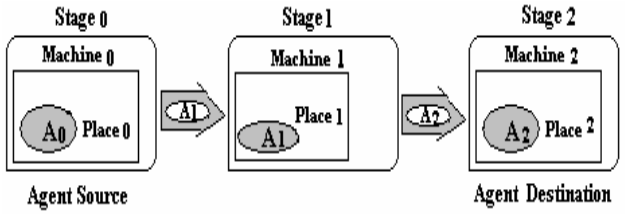

Fig 1.Model of mobile agent execution with 3 stages.

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. Model Failures

Machines, places, or agents can fail and recover later. A component that has failed but not yet recovered is called down; otherwise, it is up. If it is eventually permanently up, it is called good [15]. In this paper, we focus on crash failures (i.e., processes prematurely halt). Benign and malicious failures (i.e., Byzantine failures) are not discussed. A failing place causes the failure of all agent running on it. Similarly, a failing machine causes all places and agents on this machine to fail as well. We do not consider deterministic, repetitive programming errors (i.e., programming errors that occur on all agent

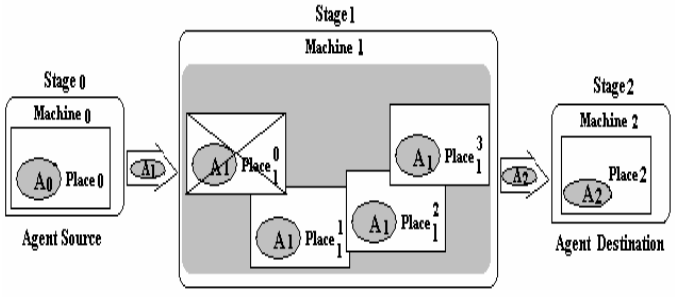

Fig 2.The redundant places mask the place failure.

replicas or places) in the code or the place as relevant failures in this Context. Finally a link failure causes the loss of the messages or agents currently in transmission on this link and may lead to network partitioning. We assume that link failures (and network partitions) are not permanent. The failure of a component (i.e., agent, place, machine, or communication link) can lead to blocking in the mobile agent execution. Assume, for instance that place p_2 fails while executing a_2 (fig. 1). While p_2 is down, the execution of the mobile agent cannot proceed, i.e., it is blocked. Blocking occurs if a single failure prevents the execution from proceeding . In contrast, and

execution is non blocking if it can proceed despite a single failure ,the blocked mobile agent execution can only continue when the failed component recovers .this requires that recovery mechanism be in place, which allows the failed component to be recovered. If no recovery mechanism exists, then the agents state and, potentially, even its code may be lost. In the following, we assume that such a recovery mechanism exists (e.g., based on logging [13]. Replication prevents blocking. Instead of sending the agent to one place at the next stage, agent replicas are sent to a set M_i of places $p_i^0, p_i^1, \ldots$ (Fig. 2). We denote by a_i^j the agent replica of a_i executing on place p_i^j ,but will omit the superscripted index if the meaning is clear from the context. Although a place may crash (i.e., stage1 in Fig. 2), the agent execution does not block. Indeed, p_2^1 can take over the execution of a1 and thus prevent blocking. Note that the execution at stages S_0 and S2 is not replicated as the agent is under the control of the user. Moreover, the agent is only configured at the agent source and presents the results to the agent owner at the agent destination. Hence, replication is not needed at these stages.

Despite agent replication, network partitions can still prevent the progress of the agent. Indeed, if the network is partitioned such that all places currently executing the agent at stage S_i are in one partition and the places of stage S_{i+1} are in another partition, the agent cannot proceed with its execution . Generally (especially in the Internet), multiple routing paths are possible for a message to arrive at its destination. Therefore, a link failure may not always lead to network partitioning. In the following, we assume that a single link failure merely partitions one place from the rest of the network .Clearly ,this is a simplification , but it allows us to define blocking concisely. Indeed , in the approach presented in this article, progress in the agent execution is possible in a network partition that contains a majority of places .If no such partition exists , the execution is temporally interrupted until a majority partition is established again ,Moreover , catastrophic failures may still cause the loss of the entire agent. A failure of all places in M1 (Fig. 2), for instance ,is such a catastrophic. Failure (assuming no recovery mechanism is in place). As no copy of a1 is available any more , the agent a1 is lost and ,obviously ,the agent execution can no longer proceed .In other words ,replication does not solve all problems .The definition of non blocking merely addresses single failures per stage as they cover most of the failures that occur in a realistic environment.

In the next section ,we classify the places in Mi into iso-places and hetero –places according to their properties [16].

III. Concept for a Fault Tolerance Approach for Mobile Agents

For each user agent(UA) ,an additional logger agent(LA) is created ,if and when fault tolerance is needed for the UA(see fig.3) .When the UA migrates , the LA follows it in a certain "distance" (e.g. on a neighbour node) .To able to tolerate node failures ,the LA must never be on the same node as the UA.

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 Fault Tolerant Approach for Mobile Agent concept is developed.

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 3). 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.

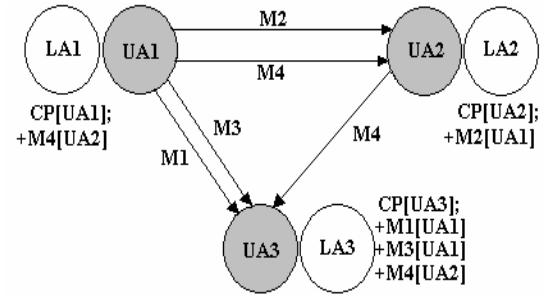

Figure 3. Example for an application with three user and logger agents with checkpoints (CP) and messages (M) [17].

IV. 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.

V. Simulation Results and Influence of the size of the Agent

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 pre-defined critical workload values.

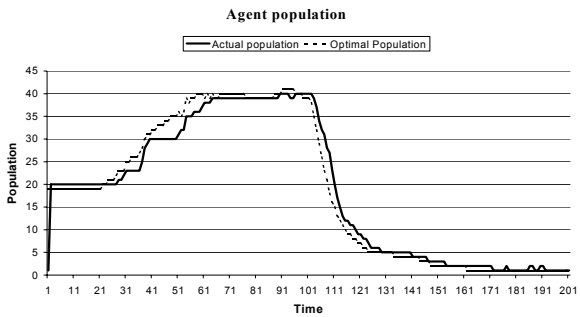

Figure 4: The size of the agent population under changing network conditions

Figure 4 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.

To measure the performance of FATOMAS our test consists of sequentially sending a number of agents that increment the value of the counter at each stage of the execution. Each agent starts at the agent source and returns to the agent source, which allows us to measure its round–trip time.
Between two agents , the places are not restarted. Consequently, the first agent needs considerably longer for its execution, as all classes need to be loaded into the cache of the virtual machines. Consecutive agents benefit from already cached classes and thus execute much faster . We do not consider the first agent execution in our measurement results. For a fair comparison, we used the same approach for the single agent case(no replication). Moreover , we assume that the Java class files are locally available on each place. Clearly, this is a simplification, as the class files do not need to be transported with the agent. Remote class loading adds additional costs because the classes have to be transported with the agent and then loaded into the virtual machine.
However, once the classes are loaded in the class loader, other agents can take advantage of them and do not need to load these classes again.

The size of the agent has a considerable impact on the performance of the fault-tolerant mobile agent execution .To measure this impact, the agent carries a Byte array of variable length used to increase the size of the agent. As the results in fig. 5 show , the execution time of the agent . increases linearly with increasing size of the agent . Compared to the single agent, the slope of the curve for the replicated agent is steeper.

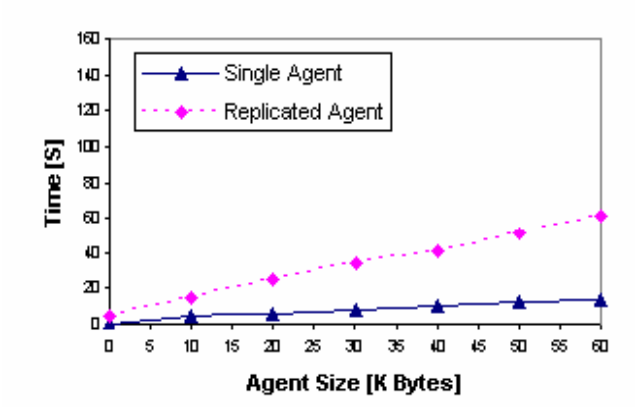

Figure5: Costs of single and replicated agent execution increasing agent size .

VI. Conclusion

In this paper ,we have identified two important properties for fault-tolerant mobile agent execution: non-blocking and exactly-once. Non-blocking ensures that the agent execution proceeds despite a single failure of either agent ,place ,or machine .Blocking is prevented by the use of replication.
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.

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. The performance measurement of FATOMAS show the overhead introduced by the replication mechanisms with respect to a non-replicated agent .Not surprisingly ,They also show that this overhead increases with the number of stages and the size of the agent.

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 Approach," *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 In!'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 Systemss ," *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!'l Conf on Dependable Systems and lIIenvorks,* 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] N. Budhirja, k. Marzullo, F.B. Schneider, and s. Toueg "The primary-Backup Approach," Distributed systems, s. Mullender,ed., second ed., pp. 199-216, Reading, Mass.: Addison-wesley , 1993.

[10] X.Defago,A. schiper,and N. sergent, "semi-passive Replication,"proc. 17th IEEE symp. Reliable Distributed system (SRDS ' 98),pp. 43-50, oct. 1998.

[11] MJ. Fischer,N.A. Lynch and M.S. paterson, "Impossibility of Distributed consensus with one Faulty process,"Proc.second ACM SIGACT-SIGMOD symp. Principles of Database system,pp. 17, Mar.1983.

[12] T.D. chandra and s. Toueg, "unreliable Failure Detectors for Reliable Distributed system ," J .ACM, VOL.43,NO.2,PP.225-267,MAR. 1996.

[13] D.chess, C.G. Harrison, and A. kershenbaum, "Mobile Agents:Are They a Good Idea?"Mobile Agents and security, G. Vigna,ed., pp. 25-47,springer verlag, 1998.

[14] D. Chess, B. Grosof, C. Harrison, d. Levine, C. parris, and G.Tsudik, "Itinerant Agents for Mobile computing," IEEE personal comm..Systems,vol. 2, no. 5,pp.34-49. oct.1995.

[15] M.K. Aguilera, w. chen, and s. Toueg, "Failure Detection and consensus in the crash-Recovery Model," Distributed computing,vol. 13,no. 2,pp. 99-125,2000.

[16] S. Pleisch and A. Schiper, " Fault-Tolerant Mobile Agent Execution," *IEEE TRANSACTIONS ON COMPUTERS, VOL. 52 ,NO .2 ,Feb 2003.*

[17] 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.