Proceedings of the 6th WSEAS Int. Conf. on EVOLUTIONARY COMPUTING, Lisbon, Portugal, June 16-18, 2005 (pp135-140)

Modeling Fault-Tolerant and Secure Mobile Agent Execution

K.Mohammadi and H. Hamidi
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 in building a mobile agent system and many fault-tolerant schemes have been proposed so far. Security is a major problem of mobile agent systems, especially when money transactions are concerned . Security for the partners involved is handled by encryption methods based on a public key authentication mechanism and by secret key encryption of the communication. To achieve fault tolerance for the agent system, especially for the agent transfer to a new host, we use Distributed Transaction Processing.
We derive the 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 we propose a solution for effective agent deployment using dynamic agent domains.

Key-Words*:* Fault Tolerant, Mobile agent, Security, Network Management, checkpointing.

1 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]. Mobile agents are becoming a major trend of distributed systems and applications in the coming years. It can bring benefits such as reduced network load and overcoming of network latency [9]. Nevertheless, security is one of the blocking factors of the development of these systems. The main unsolved security problem lies on the possible existence of malicious hosts that can manipulate the execution and data of agents [10]. Most distributed applications we see today are deploying the *client/server paradigm.* There are certain problems with the client/server paradigm, such as the requirement of a high network bandwidth, and continuous user-computer interactivity.
In view of the deficiencies of the client/server paradigm, the *mobile code paradigm* has been developed as an alternative approach for distributed application design. In the client/server paradigm, programs cannot move across different machines and must run on the machines they reside on. The mobile code paradigm, on the other hand, allows programs to be transferred among and executed on different computers. By allowing code to move between hosts, programs can interact on the same computer instead of over the network. Therefore, communication cost can be reduced. Besides, *mobile agent* [11] programs can be designed to work on behalf of users autonomously. This autonomy allows users to delegate their tasks to the mobile agents, and not to stay continuously in front of the computer terminal. The promises of the mobile code paradigm bring about active research in its realization. Most researchers, however, agree that security concerns are a hurdle [12]. In this paper, we investigate these concerns. . First, in Section 2, In Section 2 ,presents the model for fault-tolerant mobile agent . In Section 3,security issues of the mobile agent are discussed. In Section 4, discusses Security Modeling and Evaluation for the Mobile Agent. In Section 5, Simulation Results and Influence of the size of the Agent are discussed.

2 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.

2.1 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).

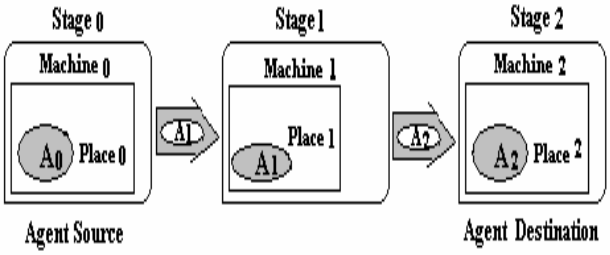

Figure1:Model of mobile agent execution with 3 stages.

2.2 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 [2,3].

2. 3 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 [19]. 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

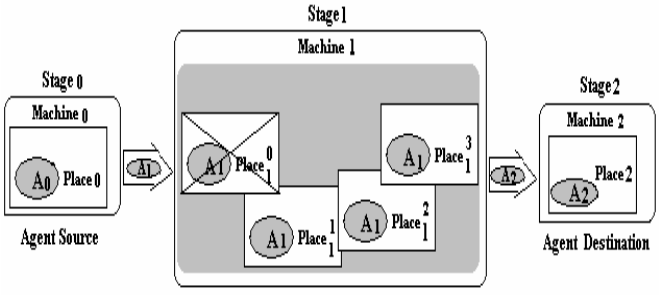

Figure2: The redundant places mask the place failure.

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_1 fails while executing a_1 (fig. 2). While P_1 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].

3. Security Issues of the Mobile Agent

Any distributed system is subject to security threats, so is a mobile agent system. Issues such as encryption, authorization, authentication, non-repudiation should be addressed in a mobile agent system. In addition, a secure mobile agent system must protect the hosts as well as the agents from being tampered by malicious parties.

First, hosts must be protected because they continuously receive agents and execute them. They may not be sure where an agent comes from, and are at the risk of being damaged by malicious code or agents (Trojan horse attack). This problem can be effectively solved by strong authentication of the code sources, verification of code integrity, and limiting the access rights of incoming agents to local resources of hosts. This is mostly realized by the Java security model [11. The main security challenge of mobile agent systems lies on the protection of agents. When an agent executes on a remote host, the host is likely to have access to all the data and code carried by the agent. If by chance a host is malicious and abuses the code or data of an agent, the privacy and secrecy of the agent and its owner would be at risk.

Seven types of attack by malicious hosts [10] can be identified:(1)Spying out and manipulation code; (2) Spying out and manipulation of data; (3) Spying out and manipulation of control flow ; (4) Incorrect execution of code; (5) Masquerading of the host;(6) Spying out and manipulation of interaction with other agents; and (7) Returning wrong results of system calls to agents.

There are a number of solutions proposed to protect agents against malicious hosts [9], which can be divided into three streams:

- Establishing a closed network: limiting the set of hosts among which agents travel, such that agents travel only to hosts that are trusted.
- Agent tampering detection :using specially designed state-appraisal functions to detect whether agent states have been changed maliciously during its travel.
- Agent tampering prevention: hiding from hosts the data possessed by agents and the functions to be computed by agents, by messing up code and data of agents, or using cryptographic techniques.

None of the proposed solutions solve the problem completely. They either limit the capabilities of mobile agents, or are not restrictive enough. A better solution is being sought, and there is no general methodology suggested to protect agents. In the mean time, developers of mobile agent systems have to develop their own methodologies according to their own needs . Apart from attacks by malicious hosts, it is also possible that an agent attacks another agent. However,

this problem, when compared with the problem of malicious hosts, is less important, because the actions of a (malicious) agent to another agent can be effectively monitored and controlled by the host on which the agent runs, if the host is not malicious.

4 Security Modeling and Evaluation for the Mobile Agent

There is no well-know model for mobile agent security. One of the few attempts so far is given by [12]. Software reliability modeling is a successful attempt to give quantitative measures of software systems. In the broadest sense , security is one of the aspects of reliability . A system is likely to be more reliable if it is more secure. One of the pioneering efforts to integrate security and reliability is [24]. In this paper, these similarities between security and reliability were observed.

Security	Reliability
Vulnerabilities	Faults
Breach	Failure
Fail upon attack effort spent	Fail upon usage time elapsed

Table1: Analogy between Reliability and Security

Thus, we have *security function, effort to next breach distribution,* and *security hazard rate* like the *reliability function, time to next failure distribution,* and *reliability hazard rate* respectively as in reliability theory. One of the works to fit system security into a mathematical model is [14], which presents an experiment to model the attacker behavior. The results show that during the "standard attack phase", assuming breaches are independent and stochastically identical, the period of working time of a single attacker between successive breaches is found to be exponentially distributed.

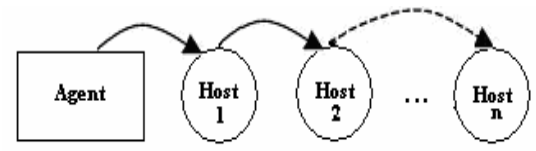

Figure3: A Mobile Agent Travelling on a Network

Now, let us consider a mobile agent travelling through *n* hosts on the network, as illustrated in Figure3 . Each host , and the agent itself , is modeled as an abstract machine as in [12]. We consider only the standard attack phase described in [14] by malicious hosts. On arrival at a malicious host, the mobile agent is subject to an attack effort from the host. Because the host is modeled as a machine, it is reasonable to estimate the attack effort by the number of instructions for the attack to carry out, which would be linearly increasing with time. On arrival at a non-malicious host, the effort would be constant zero. Let the agent arrive at host i at time T_i, for $i = 1, 2, ..., n$.

Then the effort of host i at total time / would be described by the *time-to-effort function:*

$$E_i(t) = k_i(t-T_i)$$, where k is a constant

We may call the constant k_i the *coefficient of malice.* The larger the k_i, the more malicious host i is (k_i=0 if host i is non-malicious). Furthermore, let the agent stay on host i for an amount of time t_i, then there would be breach to the agent if and only if the following breach condition holds:

$E_i(t_i+T_i)$ > effort to next breach by host i
i.e., $k_i t_i$ > effort to next breach by host i

As seen from [24,25], it is reasonable to assume exponential distribution of the effort to next breach, so we have the *probability of breach at host i,*

P(breach at host i) = P(breach at time t_i+T_i)
= P(breach at effort $k_i t_i$)
$= 1 - exp(-vk_i t_i)$, v is a constant
$= 1 - exp(-\lambda_i t_i)$, $\lambda_i = vk_i$

We may call v the *coefficient of vulnerability* of the agent. The higher the v, the higher is the probability of breach to the agent. Therefore, the *agent security E* would be the probability of no breach at all hosts, i.e.,

$$E = \prod_{i=1}^{n} e^{-\lambda_i t_i} = e^{-\sum_{i=1}^{n} \lambda_i t_i}$$

Suppose that we can estimate the coefficients of malice k_i's for hosts based on trust records of hosts, and also estimate the coefficient of vulnerability v of the agent based on testing and experiments, then we can calculate the desired time limits T_i's to achieve a certain level of security E. Conversely, if users specify some task must be carried out on a particular host for a fixed period of time, we can calculate the agent security E for the users based on the coefficients of malice and vulnerability estimates.

5 Evaluation Results and Influence of the Size of the Agent

We evaluate transactional agents in terms of access time compared with client- server model. The computation of mobile agents is composed 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.

Access time from time when the application program starts to time when the application program ends, is measured for Agents and client-server model. Figure 4 shows the access time for number of object servers, the non- fault tolerant and secure mobile agents shows that

mobile agents classes are not loaded when an agent A arrives at an object server. Here, the agent can be performed after Aglets classes are loaded. On the other hand, the fault tolerant and secure mobile agents means that an agent manipulates objects in each object server where mobile agents classes are already loaded, i.e. the agent comes to the object server after other agents have visited on the object server. As shown in Figure4, the client-server model is faster than the transactional agent. However, the transactional agent is faster than the client-server model if object servers are frequently manipulated, i.e. fault tolerant and secure mobile agents classes are a priori loaded.

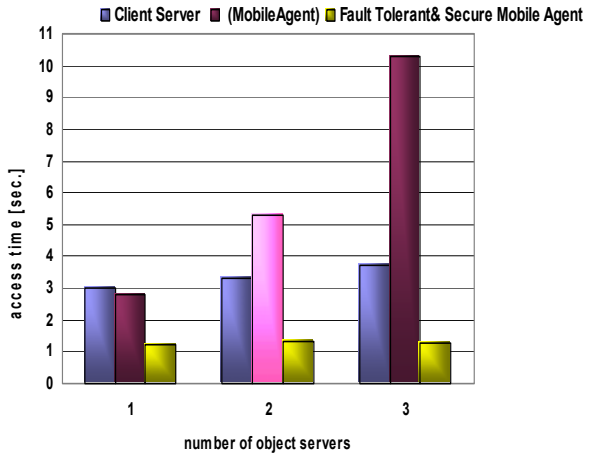

Figure4: Access Time for Number of Object Servers

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 5 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.

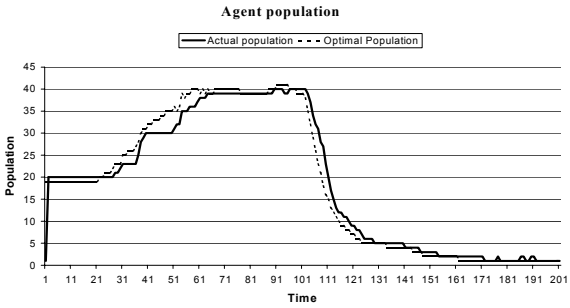

Figure 5: The size of the agent population under changing network conditions

To measure the performance of fault tolerant mobile agent system 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 f air 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. 6 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.

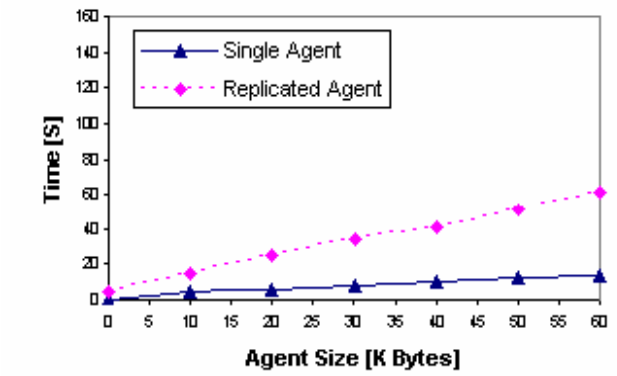

Figure6: Costs of single and replicated agent execution increasing agent size .

6 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 Fault – Tolerant Mobile Agent System 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!'1 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 15-24, June 1999.

[8] M.Shiraishi, T.Enokido and M.Takizawa" Fault – Tolerant Mobile Agents in Distributed Objects Systems"proc,of the Ninth IEEE Workshop on Future Trends of Distributed Computer Systems(FTDCS,03), pp. 11-20 ,2003.

[9] H.W.Chan,K.M.Wong , R.Lyu " Design ,Implementation ,and Experimentation on Mobile Agent Security for Electronic Commerce Application," 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 sy

[11] F.Hohl."A Model of Attacks of Malicious Hosts Against Mobile Agents".proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Object systems:Secure Internet Mobile Computations ,p.105-120.INRIA,France ,1998.

[12] Fritz Hohl. "A Model of Attacks of Malicious Hosts Against Mobile Agents". In *Fourth Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations*, http://cuiwww.unige.ch/~ecoopws/ws98/papers/hohl.ps, 1998.

[13] Sarah Brocklehurst, Bev Littlewood, Tomas Olovsson and Erland Jonsson. "On Measurement of Operational Security". In Proceedings of the Ninth Conference on Computer Assurance (COMPASS'94): Safety, Reliability, Fault Tolerance and Real Time, Security, p.257-266.1994.

[14] Erland Jonsson. "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior". In *IEEE Transactions on Software Engineering, Vol. 23, No. 4*. IEEE, April 1997.

stem (SRDS ' 98),pp. 43-50, oct. 1998.