A Distributed Heterogeneous Database System based on Mobile Agents

A. Di Stefano, L. Lo Bello, C. Santoro

Universita' di Catania - Facolta' di Ingegneria
Istituto di Informatica e Telecomunicazioni
Viale A.Doria, 6 - 95125 Catania (ITALY)
tel: +39 95 339449, fax: +39 95 338280
email: {ad, llobello, csanto}@iit.unict.it

Abstract

This paper investigates the suitability of the mobile agents approach to the problem of integrating a collection of local DBMS into a single heterogeneous large-scale distributed DBMS. The paper proposes a model of distributed transactions as a set of mobile agents and presents the relevant execution semantics. In addition, the mechanisms which are needed to guarantee the ACID properties in the considered environment are discussed.

1 Introduction

The aim of this paper is to study a transactional model based on mobile agents for the development of a heterogeneous, large-scale distributed database. The distributed environment we will refer to is made up of a set of sites connected by a network, each of which has a local DBMS. The well-known advantages of the mobile agent paradigm (autonomy, local interactions, fault-tolerance improvement [1, 2, 8, 5, 6, 15]), as compared with the traditional client/server model used in the actual implementations of distributed DBMSs, led us to consider the possibility of constructing an interface to integrate the collection of local DBMSs into a single distributed DBMS in which the transactions are mobile agents. More specifically, we will show how it is possible to model a distributed transaction as a set of mobile agents which execute interactions with remote databases by visiting the relative sites. In dealing with this topic we will use an object–oriented approach, deriving all the entities our model comprises from a library of object classes which will implement the necessary basic functions. We will then analyse some of the mechanisms used to guarantee the ACID properties in a distributed environment, assessing their functions in relation to the specific properties of mobile agents [1, 4, 5]. The paper is structured as follows. Section 2 illustrates the distributed mobile agent transaction model proposed; Section 3 presents a model of the transactional environment, illustrating the entities involved in a distributed transaction and their behaviour; Section 4 analyses the impact on a mobile agent environment of some of the

existing protocols to guarantee the ACID properties, and Section 5 gives our conclusions.

2 Model of a Distributed Mobile Agent Transaction

A distributed mobile agent transaction is defined here as a portion of code which has to be executed respecting ACID properties [11, 3] and can migrate from one site to another, performing the appropriate computations and/or interactions at each site. Wishing to make reference to an object–oriented environment, we defined the transaction by encapsulating it in an object which has the functions of a mobile agent and, at the same time, features a precise execution and interaction semantics to guarantee ACID properties. In the model proposed, a distributed mobile agent transaction, which we will call a transactional agent, starts execution on one site in the network, called the *home-site*, and can then migrate to other sites, called *visited-sites*, to which the objects with which the transaction has to interact belong. As we shall discuss below, a transactional agent has to possess adequate lock mechanisms and implement appropriate protocols for the control of concurrency and commitment, in order to ensure the ACID properties.

To fulfil the potential of a transactional scheme, an agent has to be able to create other transactional agents, thus allowing the transaction to be split up into smaller computation portions called *subtransactions* [11] which can operate in parallel in such a way as to increase the execution throughput of the transaction. Therefore, a distributed transaction T can be split into a set of subtransactions $S_1,\ldots,S_n$, and so is indicated as $T = (S_1,\ldots,S_n)$, where the subtransaction is defined as a portion of a distributed transaction which contains a sequence of commands or operations closely related to interaction with one or more objects belonging to the same site in a network. Our transactional agent model is based on this concept of subtransaction, which is incorporated in a mobile agent; the latter, created on the site to which the global transaction belongs (home-site), migrates to its own site and performs its operations there, interacting locally with the objects taking part in the transaction. In executing a distributed transaction according to this model, it may not be possible to activate all the various subtransactions it comprises simultaneously: according to how the transaction is designed, some may be linked to others by a cause and effect relationship. This means that, to be executed, some of them may need the results of the computation performed by another subtransaction. In executing a distributed transaction, therefore, some subtransactions (a subset of $S_1,\ldots,S_n$) will be launched when the transaction itself begins, while the others will be activated by the subtransactions on whose results they depend. The result of this is the creation of a tree of subtransactions (Figure 1) which we will call the *Mobile Transaction Family.* The transaction T which originates the first-level subtransactions is called a *top-level subtransaction.*

It should be pointed out that there is not always a one-to-one correspondence between a subtransaction and the relative mobile agent; if the transaction comprises several subtransactions which have to be executed sequentially, they can all be included in the same mobile agent. The mobile agent will migrate to the

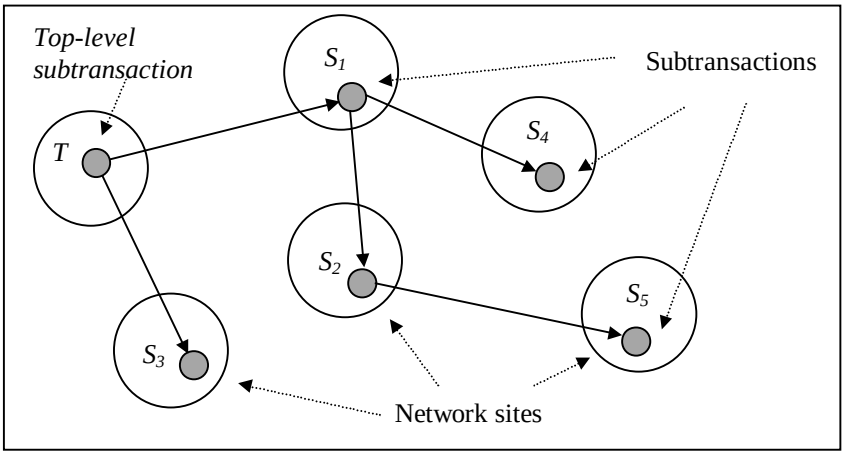

Figure 1: Mobile Transaction Family

site each subtransaction belongs to, execute the relative subtransaction, and then migrate towards to the next subtransaction and so on, until the list of subtransactions ends.

3 Distributed Mobile Agent Transactional Environment

The distributed environment we will refer to comprises a set of heterogeneous sites, connected in a network, each of which hosts and local Database by means of a DBMS. To devise a model for a Distributed DBMS based on mobile agents, we introduce the following entities at each site in the network (Figure 2):

- an *interface layer*, called *MADI - Mobile Agent Database Interface*, which integrates the collection of heterogeneous local Databases into a single distributed Database, presenting it as such to the user;
- an *execution engine* for the distributed mobile agent transactions, called *MAEF - Mobile Agent Execution Framework*.

The *Mobile Agent Database Interface* is made up of a set of modules and a library of classes that the transactional agent can use, the aim of which is to regulate access to the local DBMS by the distributed mobile agent transactions. The functions of the *MADI* can be summarised as follows:

- mapping the subtransactions of the distributed transaction onto the transactions of the local DBMS;
- presenting the transactional agents with a common interface for access to the site's Database, irrespective of the site and the DBMS used;
- implementing protocols to provide the distributed mobile agent transactions with concurrency control and atomicity;
- implementing mechanisms for the recovery of distributed transactions.

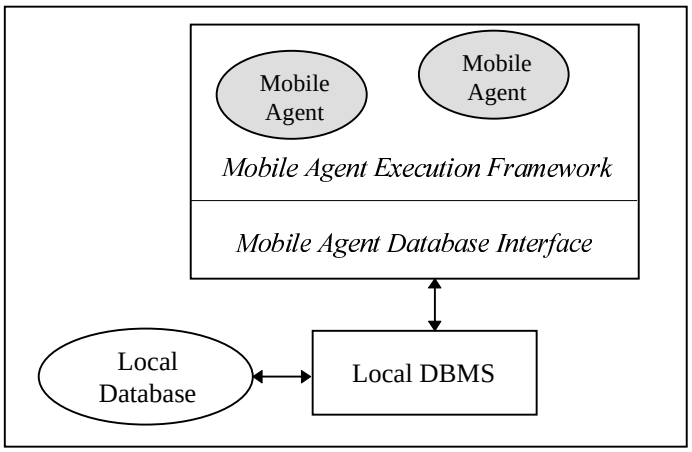

Figure 2: Modular Composition of the Environment

These functions have to be implemented taking into account that a distributed transaction is not based on client/server semantics but on mobile agents and therefore possesses all the latter's characteristics [1, 2, 5, 4]. As we will see in Section 4, these features can affect the protocols for concurrency control, commitment, fault-detection and fault-recovery, thus requiring accurate analysis and re-elaboration as compared with the client/server model.

The *Mobile Agent Execution Framework* is the environment which regulates the execution of each mobile agent making up the distributed transaction. Its functions include:

- management of the agent migration algorithm (context saving, dispatching, context restoring, execution) [15];
- access to the services supplied by the *MADI*;
- management, together with the *MADI*, of the database location services;
- management, together with the *MADI*, of the fault-recovery protocols.

In this context, in order to comply with the subtransaction model illustrated in Section 2, we have modelled a distributed mobile agent transaction as comprising (Figure 3) an object of the `MTransaction` type, which implements the top-level subtransaction, and a set of `MSubTransaction` objects, each of which incorporates one of the subtransactions making up the distributed transaction. The `MTransaction` and `MSubTransaction` classes are therefore the basic classes of objects which in our model represent the actors in a generic distributed mobile agent transaction and which, by the inheritance mechanism, can be used by the programmer to create his own distributed transactions.

3.1 Execution of a mobile agent distributed transaction

In our environment, a transaction is created by instancing an `MTransaction` object at the home-site. The latter is a static agent that can be considered as comprising two parts:

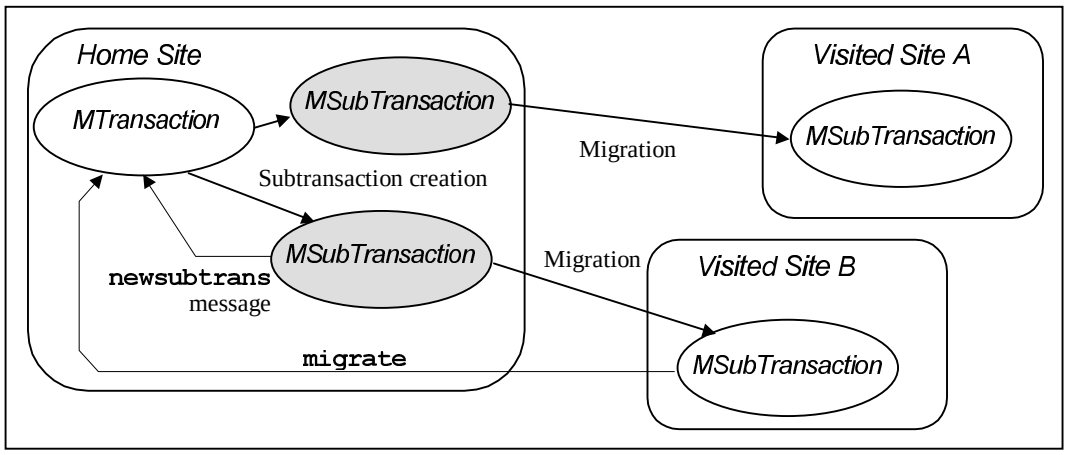

Figure 3: Objects involved in a distributed transaction

- a *transaction-dependent* part, composed of the method `MTransaction::run()`, which the programmer has to re-define and in which the code of the top-level subtransaction of the transaction has to be implemented. This implementation is also responsible for activating, by using the method `MTransaction::createSubTransaction()`, the first-level subtransactions of the Mobile Transaction Family.
- a *mobile transaction management* part, present in the method `MTransaction::coordinate()`, which is independent of the specific transaction implemented and deals with managing, controlling and co-ordinating the execution of the distributed transaction. These functions, as we will see later on, include storing information about the location and status of each subtransaction, and co-ordination of the termination of the transaction (which occurs by means of a *two–phase commitment protocol* [11, 3, 9]).

As soon as an agent of the `MTransaction` class is activated, the *MAEF* generates two threads: on the first the `run()` method of the agent instanced is executed, and on the second the `coordinate()` method is activated, the code of which - already contained in the basic `MTransaction` class - implements the mobile transaction management functions. The `run()` method generally terminates execution after generating the first-level subtransactions of the Mobile Transaction Family, while the `coordinate()` method continues execution until all the subtransactions have completed their tasks.

Each subtransaction has to be encapsulated in an agent belonging to the `MSubTransaction` class, re-defining its `run()` method, in which it is necessary to insert the real operating code of the subtransaction. After it is created and before it is executed, an `MSubTransaction` agent has to notify the relative `MTransaction` agent of its presence by sending a `newsubtrans` message (Figure 3). This message is necessary as the `MTransaction` agent always has to be aware of the complete composition of the Mobile Transaction Family so that it can reach all the subtransactions during execution of the commitment phase. It should be pointed out that although the `newsubtrans` message is useless for first-level subtransactions (since the `MTransaction` agent itself creates them),

it is fundamental for the subtransactions of the subsequent levels which the `MTransaction` agent does not see directly.

The programmer is unaware of the notification of the creation of a subtransaction as it is done automatically by the *MAEF* before activating the `MSubTransaction::run()` method. To this end, each `MSubTransaction` agent has to possess an *object reference* or a *proxy* to the relative `MTransaction` agent, so that the latter can be reached by management messages; this object reference is passed by the `MTransaction` agent to the first-level subtransactions while they are being created and then propagated by them following the creation of subtransactions belonging to the subsequent levels.

Once the creation operations are concluded, the `run()` method of the subtransaction created is activated. The first instructions of the code of this method generally refer (1) to the *localization of the Database* with which interaction is to occur, using the services provided by *MADI* and *MAEF*, and (2) to *migration* to the relative site, by calling the `MSubTransaction::dispatch()` method. When a subtransaction reaches its destination site after a migration, a `migrate` message sent by the *MAEF* to the relative `MTransaction` agent (Figure 3) informs the latter of the subtransaction's new location. This message is needed as each `MSubTransaction` agent will then have to be contacted individually during execution of the final commitment protocol.

When a subtransaction has concluded its interaction with the site's DBMS, it can: (1) conclude its execution, (2) migrate to another site and continue execution there, or (3) generate new subtransactions. In the first case, a `done` message is sent to the `MTransaction` agent and the subtransaction waits for the commitment protocol to be activated. In the second case, the `MSubTransaction` agent can locate the new site and reach it, again by invoking the `dispatch()` method. This situation is not, however, the same as a simple migration. As the global transaction has not been concluded, in fact, it is necessary to leave an object referring to the transaction in the site the subtransaction is about to leave, so that it can then take part in the two-phase commitment protocol. Also, in view of the fact that an aborted transaction will have to be re-executed, it may be convenient to clone the `MSubTransaction` agent and leave a copy at the original site: if the transaction needs to be re-executed it will not be necessary to re-create and transfer the `MSubTransaction` agents and the execution can be co-ordinated by simple message passing between the agents already present at the various sites. For these reasons, when an `MSubTransaction` agent invokes the `dispatch()` method in order to continue executing on another site, the *MAEF* clones the agent, makes the original wait for commitment and transfers the copy to the destination site, where it will continue execution[1]. From the point of view of the `MTransaction` agent, this operation corresponds to the creation of a new subtransaction and so it has to be notified of the event by sending a `newsubtrans` message followed by a `migrate` message.

In the third case, a subtransaction can conclude its execution by creating further subtransactions using the `MSubTransaction::createSubTransaction()` method. In this case, the code of the new `MSubTransaction` agent classes to be instanced is often not present on the site on which the new subtransactions

[1]This operation is performed automatically by the *MAEF* only when the `dispatch()` method is invoked for the second time, as during the first migration the agent abandons the home-site where it was created and so cloning is not necessary as it has not executed any operations at this site.

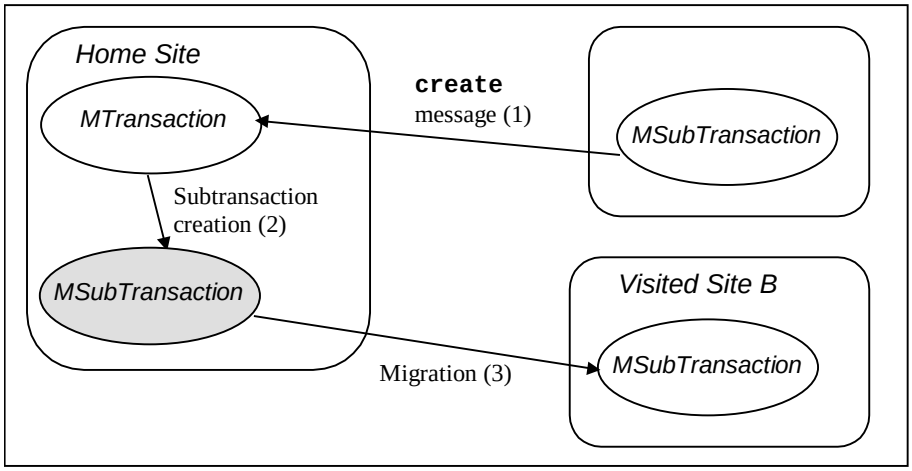

Figure 4: Creation of a subtransaction belonging to a level other than the first

are being created. The programmer, in fact, generally places all the classes relating to a transaction's subtransactions in the home-site, so they can then be transferred to the sites on which they are to be instanced. The creation of subtransactions belonging to a level other than the first (Figure 4) therefore occurs by sending a `create` message to the `MTransaction` agent, which actually creates the subtransaction required. Then, once the latter is activated, it can migrate to the relative site using the mechanism already described for first-level subtransactions.

A distributed mobile agent transaction of the type described concludes when all the subtransactions it comprises have finished executing, i.e. when all the `MSubTransaction` agents have sent a `done` message to the `MTransaction` agent. In this case the `MTransaction` agent co-ordinates a two–phase commitment protocol [11, 3, 9], the participants of which are the `MSubTransaction` agents of the family. It should be pointed out that whereas execution of the distributed transaction follows a hierarchical model, as it is structured as a tree (Figure 1), the two-phase commitment is based on a *flat* model, since it is the `MTransaction` agent that directly contacts all the subtransactions the transaction comprises.

4 ACID Properties & Fault-tolerance

As illustrated in [1, 15, 5, 8], one of the main features of mobile agents is their *autonomy* which, from the point of view of execution semantics, is equivalent to an *asynchronous* operating mode. After activating an agent, a user does not have to wait until terminates its execution but can "forget" it: the agent itself will notify the user when it terminates. Mobile agents are therefore particularly suitable for distributed environments in which certain sites are susceptible to faults and cannot always count on a continuous, reliable network connection [8, 5, 15]. In this kind of environment, if a site becomes unreachable during the execution of a distributed computation (due to a fault on the site itself or an interruption in a network link), a retry mechanism is needed, using a time-out on expiry of which the computation is considered to have failed. In the

client/server model, the choice of the duration of the time-out is quite critical, as the interaction is strictly *synchronous*. On the other hand, a mobile agent can wait even a long time and then it can try again later, until the fault is repaired.

Whereas this increase in execution time causes no problems for generic mobile agent computations, it is a highly significant parameter in the case of transactions. The presence of *blocking* management protocols like two-phase locking [11, 3, 9, 13] to control concurrency may degrades performance quite quickly if, for example an `MSubTransaction` agent has acquired a lock on a resource and repeatedly tries to contact a remote site (with a view to migrating there) which is temporarily unreachable. In this kind of situation, the solution found in the client/server model is to abort the transaction, considering it to have failed; in our environment, on the contrary, we have entrusted the `MSubTransaction` agent with a suitable *rollback, retry & restart* protocol comprising the following phases:

- *(rollback)* the `MSubTransaction` agent orders a temporary rollback of the transaction, sending a `temp-rollback` message to the `MTransaction` agent which consequently releases any resources blocked by the various subtransactions. Even though a rollback occurs, the transaction is not yet considered to have failed;
- *(retry)* the `MSubTransaction` agent sends `echo` messages to the site in question until it receives an answer;
- *(restart)* once it has ascertained that the site is reachable, the `MSubTransaction` agent sends the `MTransaction` agent a `restart` message which causes the distributed transaction to be restarted. As said previously (Section 3.1), some subtransactions may already reside on that site and so the execution can be co-ordinated by a message passing mechanism.

The protocol described, execution of which is completely transparent to the user activating the transaction, increases the probability of successful conclusion of a transaction in the event of temporary faults on a site or network connection, thus avoiding a degradation in performance (the blocked resources are released). The only cost is an increase in the execution time of the distributed transaction, when the latter is affected by the fault.

The same applies to the commitment protocol. As is known [11, 3, 10, 7], in all commitment protocols there is a particularly critical phase for faults or network partitioning. In the two-phase commitment protocol [11, 3], for instance, the occurrence of a permanent or long-term network partitioning during the second phase of the protocol may cause one or more sites to remain isolated; as their subtransactions cannot be reached by the concluding *commit/rollback* message, they have to activate a recovery protocol which will allow them to trace the decision made by the co-ordinator (in our case the `MTransaction` agent). The recovery protocols proposed in literature [11, 10, 12, 14] solve the problem partially but none of them provides a solution for the case in which there is a single isolated subtransaction. A situation of this kind is quite likely to occur in environments with a low degree of network connection reliability, the same which, as said above, are suitable for mobile agents. In our model, as the commitment problem cannot be solved completely, we propose a solution whereby

the transaction can at least be terminated. This solution involves the following steps:

- the programmer who has created the `MTransaction` agent for the transaction has to provide it with a parameter, called `defaultDecision`, which is propagated to all the `MSubTransaction` agents of the transaction;
- when a site remains isolated, its `MSubTransaction` agent, which cannot receive notification of the result of the transaction from the `MTransaction` agent, takes the autonomous decision indicated by the `defaultDecision` parameter, on expiry of a certain time-out;
- if the `MTransaction` agent cannot reach a subtransaction during the second phase of the two-phase commitment, it continues its action, notifying the result to all the other subtransactions. If, however, the result is different from the value of the `defaultDecision` parameter, the transaction is concluded by notifying the user of a possible loss of consistency, including information about the site/s that cannot be reached.

This protocol, which has to be activated when a failure recovery procedure [11, 10, 12, 14] has not been able to reach the decision taken by the co-ordinator, ensures termination of the transaction and detects any problems of loss of consistency that may arise.

Of course, the discussion so far has only dealt with some of the problems mobile agents present in a transactional environment, but it is a good starting point for tackling the problems that arise due to the inherent features of a mobile agent environment and the different execution semantics for a distributed mobile agent transaction as compared with the client/server model.

5 Conclusions

In this paper we have studied a transactional environment based on mobile agents for the management of a heterogeneous, large-scale distributed Database. We have considered a distributed environment formed by a set of network sites each of which has a local DBMS. On each of these sites an interface layer is constructed to integrate the heterogeneous local DBMSs into a single distributed DBMS in which the transactions are mobile agents. To this end, a distributed mobile agent transactional model has been developed, identifying the objects involved and their features; these have been modelled, following an object–oriented approach, highlighting the basic classes of objects which have been given the features of mobile agents. Finally, after illustrating the execution semantics of our distributed mobile agent transaction model, we have analysed some protocols to guarantee the ACID properties, in the light of the features of mobile agents and the distributed environments which are suitable for them. In this sense, the paper is a contribution to research which aims at developing the mobile agent programming paradigm, laying the bases for the introduction of mobile agents in transactional environments.

References

[1] D. B. Lange D. T. Chang. Mobile agents: A new paradigm for distributed computing on the WWW. *OOPSLA '96 Workshop*, 1996.

[2] G. Harrison et al. Mobile agents: are they a good idea ? Technical report, IBM T. J. Watson Research Center, 1995.

[3] K. Kinderberg G. Coulouris, J. Dollimore. *Distributed Systems: Concepts and Design.* Addison-Wesley, 1995.

[4] F. Somers L. Hurst, P. Cunningham. Mobile agents - smart messages. In *First International Workshop on Mobile Agents*, 1997.

[5] Horizon Systems Laboratory. Mobile agents computing - A White Paper. Technical report, Mitsubishi Electric ITA, 1996.

[6] M. Schwehm M. Strasser. A performance model for mobile agent systems. In *Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, 1997.

[7] S. J. Mullender. *Distributed Systems.* Addison-Wesley, 1993.

[8] K. Heilmann D. Kihanya A. Light P. Musembwa. Intelligent agents: A Technology and Business Application Analysis. Technical report, University of Nancy, 1996.

[9] T. Johnson R. Chow. *Distributed Operating Systems & Algorithms.* Addison-Wesley, 1996.

[10] A. Schiper R. Guerraoui, M. Larrea. Non blocking atomic commitment with an unreliable failure detector. In *IEEE Symposium on Reliable Distributed Systems*, 1995.

[11] G. Pelagatti S. Ceri. *Distributed Database Systems.* McGraw Hill, 1985.

[12] Dale Skeen. A quorum-based commit protocol. Technical report, Computer Science Department - Cornell University, 1982.

[13] A. S. Tanenbaum. *Modern Operating Systems.* McGraw Hill, 1991.

[14] Peter Triantafillou. Independent recovery in large-scale distributed systems. *IEEE Transaction on Software Engineering - Vol. 22, No. 11*, 1996.

[15] Jim White. Mobile agents white paper. Technical report, General Magic corp., 1995.