A Fault Tolerance Mechanism for MAgNET: A Mobile Agent based E-commerce System

Prithviraj(Raj) Dasgupta
Department of Computer Science
University of Nebraska, Omaha, NE 68182.
E-mail: pdasgupta@mail.unomaha.edu

Abstract

Mobile agents are an attractive paradigm for implementing communication intensive applications such as distributed databases, network management and more recently e-commerce. Like all other software systems, reliability is a major concern in mobile agent systems. The problem is complicated for mobile agent systems as the remote sites on which the mobile agent executes cannot be controlled by the application programmer. In this paper, we use MAgNET, a mobile agent enabled e-commerce system to study and propose solutions for the reliability of mobile agent systems. In MAgNET, human buyers create and deploy mobile shopping agents that visit different seller sites on the Internet to comparison shop for a particular item. MAgNET employs passive replication of the mobile shopping agents to protect against faults on the remote seller sites. Process faults are addressed by cloning the shopping agent on each seller site prior to execution. To address network and node faults, we have used the bounded failure rate hypothesis along with passive replication of the shopping agent at each seller site.

1 Introduction

With the rapid growth of the Internet over the last few years, the amount of information available online has increased manifold. More and more people are using the Web for activities including business transactions, personal communication, information retrieval and even for entertainment. These activities involve large amounts of data transaction and impose a huge communication requirement on the Internet. Most of the current online applications employ the client-server model for network communication. Mobile agents provide a suitable paradigm for improving the performance of Internet applications. Instead of transfering data betweeen a client and a server, mobile agents carry the application code with them from the client to the server. Since the size of the code is lesser than the amount of data interchanged between the client and the server in online applications, mobile agent based systems provide considerable improvement in performance over client-server based computing.

Reliability is an important issue for Internet applications because applicatiions that crash while running might expose confidential data, lead to user dissatisfaction and even leave the entire system in an unusable state. In this paper we investigate the reliability issues in mobile agent based systems.

Mobile agents have been used in a variety of applications such as distributed database access and management, network management and e-commerce. We have selected the last application for our study because large amounts of information are exchanged between online buyers and sellers and the open nature of the Internet provides an opportunity to study reliability concerns for e-commerce. Fault tolerance guarantees the uninterrupted operation of a distributed software system despite network or node failures. In this work, we use fault tolerance techniques to protect an e-commerce application against network and node faults.

2 Related Work

Fault tolerance in mobile agents has been relatively less investigated than other aspects of mobile agent systems. Schneider [4, 7] proposes active replication with majority voting to obtain a consensus on the computation performed by a set of replicated nodes. A stage in the system accepts an agent sent from the previous stage only if the majority of the replicated agents from the previous stage bear a message called a forward signed by the previous stage. Active replication is unsuitable for e-commerce as the remote seller sites that need to be

replicated are not under the control of the owner of the mobile shopping agent.

Strasser and Rothermel [9] have used passive replication with the primary replica in each stage determined by a bully algorithm. Multiple agents resulting from network partitioning are resolved by majority voting after rejoining. De-Assis Silva [6, 8] has employed a nested transaction model with checkpoints stored within a distributed context database (DCD) to implement fault tolerance. The model uses passive replication and the primary replica at each stage is determined by using a pre-assigned ordering between the replicated nodes. Both of these models rely upon replication of remote nodes to provide fault tolerance. As mentioned earlier, in e-commerce, the owner of the mobile shopping agent does not have any control over the remote seller sites. Thus, it is not possible to have replicated copies of the same seller site. Therefore, in our model we have used passive replication of the mobile agents within a single seller site to provide fault tolerance.

We have used the MAgNET system that we have developed to study the reliability concerns for mobile agent based systems operating over the Internet. MAgNET is a Java Aglets [5] based mobile agent system that enables buyers to compare the attributes of an item available at different online sellers and select the offer that best matches the buyers' needs. The security architecture of MAgNET [3] provides the basic reliability of MAgNET. However, security protocols are not sufficient to prevent system faults arising from network partitioning or when the host on which the mobile agent is executing goes down. Thus, fault tolerance is an important aspect in MAgNET.

3 MAgNET: Mobile Agents for Networked Trading

MAgNET [2] is a mobile agent system for comparison shopping over the Internet. MAgNET has been implemented using Java, and the Aglets Software Development Kit (ASDK) [5] as the underlying mobile agent platform. The ASDK provides an object-oriented programming interface for writing Java-based mobile agents, called aglets.

The participants in MAgNET are human buyers, mobile shopping aglets created and deployed by the buyers, and Internet sellers represented by their online sites. Buyers are made aware of the identity of sellers through advertisements and possibly through mobile sales agents sent by sellers [1]. Each buyer site maintains a list of potential sellers for products. As shown in Figure 1, the buyer and seller sites in MAgNET run aglet servers that enable them to host aglets. The buyer site contains a stationary controller aglet that accepts purchase requests from human buyers and creates the shopping aglets. Every seller site contains a stationary service aglet that interfaces between visiting shopping aglets and the inventory on the back-end of the seller's site. The shopping list for a buyer and the inventory on a seller site are stored as XML annotated files. The shopping aglets in MAgNET employ a stop-and-shop scheme. A shopping cycle in MAgNET consists of the following steps:

1. The human buyer submits a shopping list to the buyer site through a GUI.

2. The controller aglet on the buyer site creates one mobile shopping aglet for each item on the shopping list.

3. The controller aglet provides the list of seller sites that each shopping aglet has to visit and dispatches the shopping aglet on its itinerary.

4. On arriving at a seller site, a shopping aglet searches through the seller's inventory and determines the best offer. If the present offer is better than any offer previously seen on its itinerary, the shopping aglet reserves the item by spawning a stationary surrogate aglet. Surrogate aglets spawned on previous sites are aborted by sending a cancellation message. The data collected from the seller are stored within the shopping aglet.

5. After visiting all the seller sites on its itinerary, the shopping aglet returns to its origin and reports the best deal that it has found. The buyer either confirms the deal and proceeds with the monetary transaction, or aborts the query and terminates the shopping aglet.

The correct operation of MAgNET relies on the fault-free operation of the aglet-servers on each of the seller sites, the reliability of the computers used by the sellers, and the consistency of the links connecting the different sellers that are on the shopping aglet's itinerary. If any of these develop faults, the shopping aglet encounters problems and is either destroyed or rendered unusable. Therefore, a fault tolerance protocol that guarantees accurate operation of the shopping aglet is an important feature that ensures the reliability of MAgNET. The Aglets SDK provides rudimentary fault tolerance through

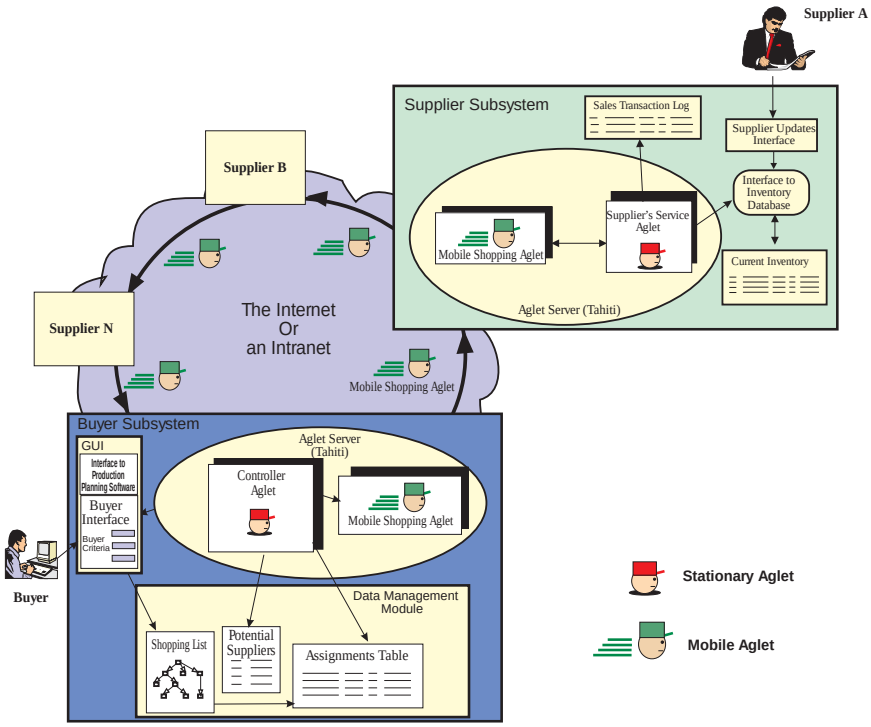

Figure 1: The architecture of the MAgNET system.

the persistence feature of aglets, that allows them to be stored temporarily in the secondary storage of the computer on which they are executing. However, temporary storage within the hard disk of a computer is not sufficient to address all the software faults in MAgNET. Therefore, we have developed additional strategies that enable a shopping aglet in MAgNET to recuperate from network and node faults.

4 Reliability Issues in MAgNET

In distributed systems that use replication, the system designer determines the number of replicas at each node, maintains the replicas and also decides the backup and recovery strategies. To make MAgNET fault tolerant, we employ replication of mobile shopping agents at the seller sites that the mobile agent visits. However, in MAgNET, replication is complicated because the remote seller sites at which the mobile shopping agent execute are beyond the control of the buyer originating the shopping aglet. Moreover, each seller site is likely to have its own limitation on system resources available to shopping aglets, and therefore, it is difficult for the buyer to determine and enforce the number of copies of the mobile shopping aglet that can be run simultaneously at every seller site. Running multiple copies of shopping aglets at every seller site also aggravates security and performance issues in MAgNET. Therefore, active replication of the shopping aglets cannot be used to provide fault tolerance in MAgNET.

We have identified two types of software faults in MAgNET:

- **Process Fault.** The aglet-server process on a seller site on which the mobile shopping agent is executing crashes, but the computer on the seller site continues running.

- **Network and Node Fault.** The seller site that was executing the shopping aglet goes down or the shopping aglet is lost in transit due to a link fault.

Process faults can be rectified by maintaining a replica of the shopping aglet within the seller's computer at a location where the replica is not affected by the aglet-server process crash. Therefore, a lightweight protocol that replicates the shopping aglet after it arrives on a seller site can solve process faults. On the other hand, if network and node

faults destroy the entire shopping aglet, then the shopping aglet has either to be created afresh, or its latest saved version has to be retrieved and executed. This requires the protocol for solving network and node faults to be more robust than the protocol for solving process faults. Therefore, we have developed two protocols, a lightweight protocol for recovery from process crashes and a more complicated protocol for recovery from network and node failures.

5 Surviving Aglet-server Process Faults

The protocol for protecting against aglet-server process failures uses the persistence feature of aglets along with passive replication on the seller site on which the shopping aglet is currently executing.

5.1 Fault Tolerance Features in Aglets SDK

The Aglets SDK provides the persistence feature of aglets, that allows an aglet executing within an aglet-server to be deactivated and temporarily stored in secondary storage. The *PersistencyListener* class, that can be added to an aglet, listens for *PersistencyEvents* for deactivating the aglet and saving it in temporary storage or, for reactivating the aglet from its dormant state in secondary storage. An aglet that is stored in temporary storage using the *deactivate(duration)* method is reactivated after the time given by the parameter *duration* expires. This makes the aglet resilient to aglet-server faults that might occur during the period it was deactivated. If the aglet-server is rebooting or unavailable when the deactivated aglet is re-activated, the aglet persists in secondary storage and is re-activated when the aglet-server process is restored.

A shopping aglet in MAgNET can be protected against aglet-server faults using the persistence feature provided by the ASDK. However, since the shopping aglet contains the main thread of execution, it is not feasible to deactivate the shopping aglet itself and affect the performance of the system. Therefore, a clone of the shopping aglet serves as a backup that can used in case of aglet-server fault on the seller site.

5.2 Protocol for Recovery from Process Crashes

The aglet-server running on a seller site in MAgNET represents a single point of failure. If the aglet-server crashes, then all of the aglets executing within it are lost. To prevent the loss of a shopping aglet that is executing on a seller's site, the shopping aglet is cloned upon arrival on the seller site. The original shopping aglet is designated as the "primary" and performs all the computation required by the shopping aglet. The clone serves as the backup. It is stored in the secondary storage and is synchronized with the primary at certain intervals. The protocol for preventing against process faults works as follows:

1. As shown in Figure 2, the controller aglet on the buyer site creates a shopping aglet and sends it on its itinerary.

2. On arriving at a seller site, the shopping aglet makes a clone of itself. The original shopping aglet is marked as the primary while the clone is marked as the backup.

3. The backup shopping aglet is stored in secondary storage while the primary shopping aglet executes its code. The seller site allows only the primary shopping aglet to spawn temporary surrogate aglets on its site.

4. The primary shopping aglet is provided with checkpoints at certain intervals during its execution. At each checkpoint, the backup is retrieved from secondary storage, synchronized with the primary and stored back.

5. If the aglet-server on the seller site crashes while the primary shopping aglet was executing, the backup shopping aglet is retrieved when the aglet-server comes up again. Then, the backup is designated as the primary on the seller site. It creates a copy of itself that acts as the backup. The new primary then continues executing from the last stored checkpoint.

The above protocol protects MAgNET against process faults. However, it does not guarantee that the mobile shopping aglet will be resilient to network and node faults. If the computer at the seller site goes down, both the primary and backup shopping aglets are lost. Also, if the network connection over which the shopping aglet is transported goes down while the aglet is in transit, the shopping aglet is lost. To prevent either kinds of faults,

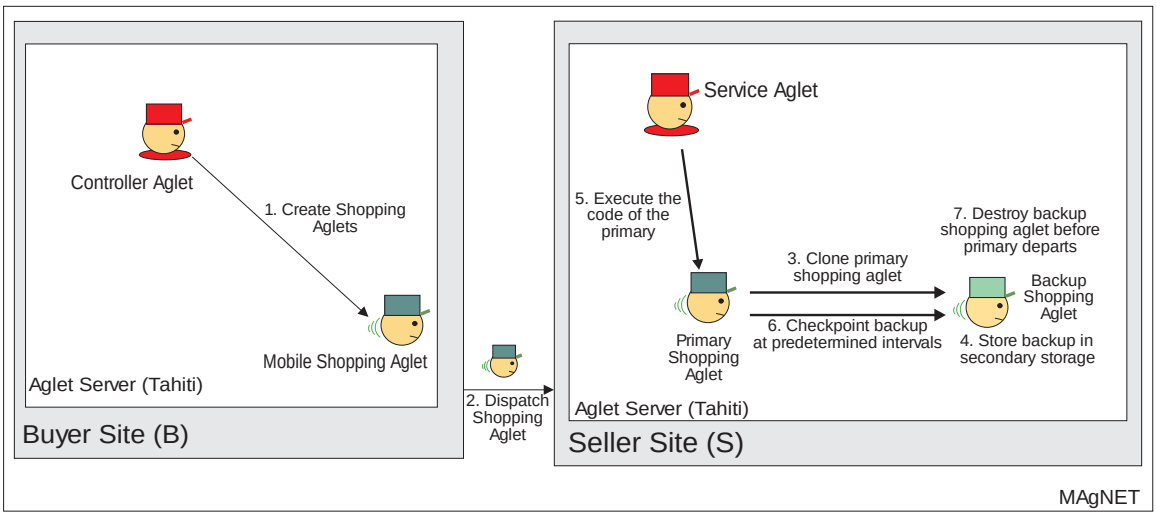

Figure 2: Protocol for recovery from process faults.

a fault tolerant protocol should be designed that saves the endeavor of creating the shopping aglet on the buyer site.

6 Surviving Network and Node Faults

A naive approach to protect against network and node faults could be to use a timeout for receiving heartbeat messages from the shopping aglet at the buyer site. However, this increases the number of messages transferred and affects the performance of MAgNET. Here we have used the bounded failure rate hypothesis along with passive replication at different seller sites to make our system reliable.

A **Bounded Failure Rate** [4] guarantees that a mobile agent will encounter at most f faults while traversing h hosts, where $0 < f \leq h$. As shown in Figure 3, after agent $a1$ has completed execution on site $S0$, three copies $A2, A3$ and $A4$ are sufficient to guarantee that at least one of the replicas survives all network and node faults before completing the entire itinerary, according to the bounded failure rate hypothesis. As the state of the agent changes after the computation on every node, therefore the agent needs to be replicated before each hop. However, since the number of remaining hops reduce as the agent progresses through its itinerary, the number of replicas determined by the bounded failure rate also reduces. In MAgNET, if a mobile shopping aglet has n sellers remaining to visit during its itinerary, $n+1$ copies of the shopping aglet are made and sent to $n+1$ seller sites. The bounded failure rate hypothesis then guarantees that at least one of the replicas of the shopping aglet survives all network and node faults.

6.1 Protocol for Recovery from Network and Node Faults

The protocol for recovery from network and node faults works as follows:

1. When a shopping aglet completes execution at seller S after which there are n seller sites to visit, seller S makes n copies of the shopping aglet and sends them to each of the remaining n seller sites. One of the n seller sites is the seller that is next on the shopping aglet's itinerary. This seller is designated as seller $S+1$. The original shopping aglet remains at seller S. Seller S starts a timer $aglet-ack-timer_S$ for receiving the responses from the n sellers.

2. When a seller receives a copy of the shopping aglet it returns an *aglet-recd-ack* to seller S.

There can be three scenarios emerging from this situation:

3(a) Seller S receives the *aglet-recd-ack* from all of the n sellers. This means that there have been no faults. Seller S sends a *exec-end* message to seller $S-1$ that terminates $exec-timer_{S-1}$. Seller S then starts $exec-timer_S$ and sends an *exec-begin* message to seller $S+1$. Seller $S+1$ becomes the leader [9] of stage $S+1$ once it

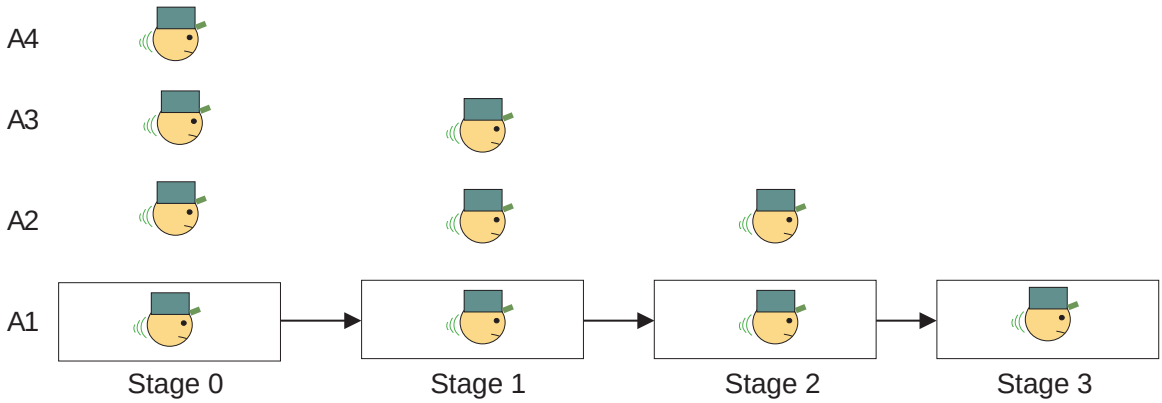

Figure 3: Scenario illustrating a mobile agent that has to perform three hops. By the bounded failure rate hypothesis, $n+1$ copies of the mobile agent should be maintained at a stage after which there are n hops outstanding.

receives the *exec-begin* message. It then sends a *leader-elected* message to the remaining $n-1$ sellers of that stage.

3(b) Seller S receives between one and $n-1$ *aglet-recd-ack*s. Seller S then reattempts to send the aglet to the $n-m$ seller sites that did not respond, assuming that their failure is transient. If the reattempts also fail before the timer $aglet-ack-timer_S$ expires, one of the sellers that did not respond is removed from the itinerary of the shopping aglet. Seller S then reattempts the protocol for $n-1$ sellers instead of n. It also sends a message to the buyer site informing the removal of the unresponsive seller from the shopping aglet's itinerary.

3(c) Seller S receives no *aglet-recd-ack*s. This implies that seller S has developed a fault, either the computer of seller S has gone down or a link connecting seller S to the Internet has failed. Seller S then does not send a *exec-end* message to seller $S-1$. As a result, $exec-timer_{S-1}$ expires. The computation of the aglet is then rolled back till the completion of execution on seller $S-1$. The protocol is then retried by sending $n+1$ copies of the shopping aglet for the sellers in stage S.

The above protocol can result in cascading faults, if the sellers in successive stages fail in succession. In such a case, the shopping aglet is transferred to a relay point for execution. A relay point is a trusted host that is guaranteed to be resilient to faults.

The protocol for recovery from network and node faults allows uniterrupted execution of the shopping aglet at the different seller sites on its itinerary. While travelling along its itinerary the shopping aglet has to exchange messages with the buyer site to convey about temporary reservations it makes at different sellers. To ensure the consistency of the stationary controller aglet on the buyer site, a backup copy of the controller aglet is maintained inside secondary storage. If the aglet-server on the buyer site crashes, the backup takes the place of the controller aglet when the aglet-server is restored. If the buyer site goes down or a link to the buyer site develops a permanent failure, replication of the entire buyer site on another processor can guarantee consistency. A seller site in MAgNET services shopping aglets from several buyers. In contrast a buyer site manages only the shopping aglets that it creates itself. The reliability of a seller site in MAgNET is thus more important than that of a buyer site. Therefore, our protocol ensures that the seller sites operate consistently while keeping the software on the buyer site simple and lightweight.

7 Conclusion

The fault tolerance protocols for MAgNET presented in this paper protects a shopping aglet from faults on the seller sites that it visits and from faults in the links connecting those seller sites. One of the objectives of the security protocol in MAgNET [3] is to reduce multiple copies and clones made by the shopping aglet. This is counter-intuitive to the fault tolerance protocol, which relies on cloning and replicating the shopping aglet. Thus, the issues of security and fault tolerance are somewhat orthogonal for mobile agent based systems. However, the fault tolerance protocol can be integrated with the

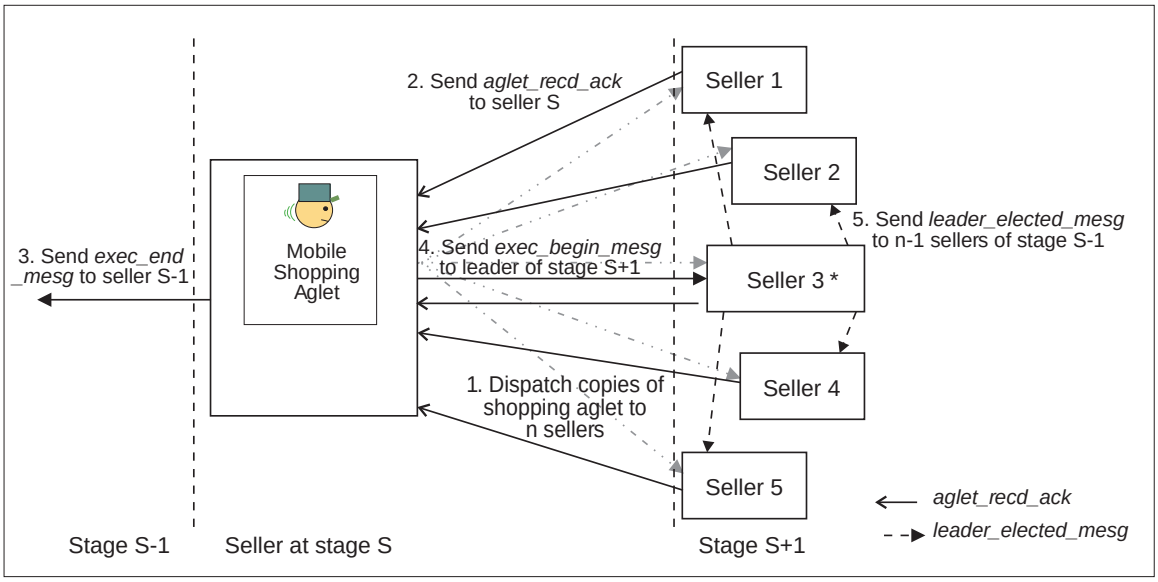

Figure 4: Protocol for recovery from network and node failures.

security protocol to perform the authentication on a seller site before granting permission for a shopping aglet to execute.

The protocol presented here assumes that the operations of storing and retrieving aglet from secondary storage, leader election and stage termination are atomic. However, in the real world, an application can fail after it has partially executed these instructions. We are currently investigating enhancements to the protocols presented in this paper that will make mobile agents resilient against more complicated faults.

References

[1] P. Dasgupta, N. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "A supplier driven marketplace using mobile agents," *Proceedings of the First International Conference on Telecommunications and E-commerce,* pp. 42-50, Nashville, TN, November 1998.

[2] P. Dasgupta, N. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "MAgNET: Mobile agents for networked electronic trading," *IEEE Tranasactions on Knowledge and Data Engineering, Special Issue on Web Applications,* vol. 11, no. 4, pp. 509-525, July-August 1999.

[3] P. Dasgupta, L. E. Moser, and P. M. Melliar-smith, "The security architecture for MAgNET: A mobile agent e-commerce system", *Proceedings of the Third International Conference on Telecommunications and E-commerce,* pp. 289-298, Dallas, TX, November 2000.

[4] D. Johansen, K. Marzullo, F. Schneider, K. Jacobsen and D. Zagorodnov, "NAP: Practical fault-tolerance for itinerant computations," *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX, pp. 180-189, May-June 1999.

[5] D. Lange and M. Oshima, "Programming and deploying Java mobile agents with aglets", Addison Wesley, MA, 1998.

[6] F. De Assis Silva and R. Popescu-Zeletin, "Mobile agent-based transactions in open environments," *IEICE Transactions on Communications*, vol. E83-B, no. 5, pp. 973-987, May 2000.

[7] F. Schneider, "Towards fault-tolerant and secure agentry," *Proceedings of the 11th International Workshop on Distributed Algorithms,* pp. 1-14, Saarbrucken, Germany, September 1997.

[8] L. M. Silva, V. Batista and J. G. Silva, "Fault-tolerant execution of mobile agents," *Proceedings of the International Conference on Dependable Systems and Networks*, New York, NY, pp. 135-143, June 2000.

[9] M. Strasser and K. Rothermel, "Reliability concepts for mobile agents", *International Journal of Cooperative Information Systems,* vol. 7, no. 4, pp. 355-382, December 1998.