MAgNET: Mobile Agents for Networked Electronic Trading

Prithviraj Dasgupta, Student Member, IEEE, Nitya Narasimhan, Student Member, IEEE, Louise E. Moser, Member, IEEE, and P.M. Melliar-Smith, Member, IEEE

Abstract—Electronic commerce technology offers the opportunity to integrate and optimize the global production and distribution supply chain. The computers of the various corporations, located throughout the world, will communicate with each other to determine the availability of components, to place and confirm orders, and to negotiate delivery timescales. In this paper, we describe MAgNET, a system for networked electronic trading that is based on the Java mobile agent technology, called aglets. Aglets are dispatched by the buyer to the various suppliers, where they negotiate orders and deliveries, returning to the buyer with their best deals for approval. MAgNET handles the deep supply chain, where a supplier may need to contact further suppliers of subcomponents in order to respond to an enquiry. Experimental results demonstrate the feasibility of using the Java aglet technology for electronic commerce.

Index Terms—Electronic commerce, Java, mobile agents, procurement, supply chain.

1 INTRODUCTION

The Internet and the World Wide Web have made electronic commerce possible. Web sites that display products and take orders are becoming common for many types of businesses. Such Web sites are, however, still designed for interactive use by humans and do not yet fully support automated electronic commerce among computers.

The global production and distribution supply chain is complex, involving procurements of component parts from many suppliers. Rapid response to changes in demand and customer preference, and the ability to exploit new technologies, are becoming critical. Although price is undoubtedly a factor in procurements, availability in the timescale required is often more important, as are appropriately phased deliveries to support just-in-time operation. Timely interactions require close coordination of the planning and logistics systems of the buyers and suppliers. The aim of the new technology must be to support close interaction between the procurement, production planning and logistics systems of business enterprises in the entire supply chain. Those who achieve this daunting objective will dominate commerce in the next century.

The complex and tight interactions required between the software systems of companies around the world will be greatly aided by the Internet and by private intranets. To reduce the delays that hamper tight interaction between software systems, it is appropriate to investigate the use of mobile software agent technology. This technology allows an agent in the form of program code, data, and execution state to be packaged into a message and sent across the network to a remote computer. As Fig. 1 shows, the agent can conduct multiple interactions with software resident on the remote computer to achieve the intricate negotiation or planning that is required. The initial delay of sending the agent across a wide-area network will be longer than the delay for a typical interaction message, because the software agent is larger, but that initial delay is incurred only once. For activities that require hundreds or thousands of interactions, the savings in communication time can be substantial.

Although researchers proposed mobile software agents some years ago, until recently mobile agents were wishful thinking. Now we have Sun’s Java language, which allows programs to be run on different platforms without modification or recompilation. Java applet technology allows programs to be downloaded across the network and executed on a remote host, and IBM’s Java agent technology [1, 14], [15] allows executing programs to move from one...
host to another within the network. The software infrastructure for mobile agents is available, perhaps not yet for development into a commercial product but certainly for experimentation.

Section 2 of this article discusses the use of mobile agent technology in electronic commerce, and Section 3 describes briefly the mobile agent technology of IBM’s Aglet Software Development Kit. Section 4 presents our prototype system, called MAgNET, that uses mobile agents for procurement of component parts in the global production and distribution supply chain based on the pull strategy of marketing, and Section 5 reports on our experimental performance results. Section 6 discusses related research, and Section 7 considers opportunities for further research.

2 THE USE OF MOBILE AGENTS IN ELECTRONIC COMMERCE

The aspect of electronic commerce on which we focus is the procurement of the many component parts needed to manufacture a complex product. Those parts are typically purchased from a single supplier, but several alternative suppliers may exist. Alternative choices of major components can lead to choices of different subcomponents.

Our objective is to find a set of suppliers able to provide the component parts required for manufacture of the product on a delivery schedule that is compatible with production plans. Price is, undoubtedly, a consideration, but availability and delivery schedule are often more important. The time to complete the procurement decisions and commit the production plan is also critical, because it determines a company’s ability to respond to sales enquiries from its customers.

A buyer, possibly a human or possibly production planning software, assembles a list of required components with a desired delivery schedule and a list of preferred suppliers for each component. If there are alternatives and dependencies in the list, the structure may be quite complex. The buyer dispatches one or more mobile agents, each with its list of required components and potential suppliers. The mobile agents then visit the suppliers to obtain quotations and delivery schedules for the required components. If a quotation is attractive, the mobile agent may reserve the components before moving to the next site in its itinerary. Some suppliers may be able to fill a large order only partially, and the mobile agent may need to seek the remainder of the order from another supplier.

In due course, the mobile agents return to the buyer with the quotations that they have obtained and the reservations that they have made. The buyer determines the best combination of offers and sends messages or agents to the suppliers to commit or cancel the reservations. The rapid decisions made possible by mobile agents ensure that the reservations are held for only a short period of time, thus reducing the reservation fees that must be paid. Multiple mobile agents can compete with each other to produce the best offer for a component part, or a single agent can handle all of the related components by visiting all of the potential suppliers of those components.

If all of the components are in stock and available for offer, this scheme works quite well. In practice, however, with an objective of maintaining minimum inventory, the suppliers may need to plan the manufacture of components to meet the requirements, and also to approach their suppliers of subcomponents and raw materials. Those suppliers may also need to approach their suppliers and, thus, a single initial request may ripple through the supply chain. With the electronic speed of mobile agents and automated planning software, it is possible to respond promptly and accurately to the initial request, with information obtained from the entire supply chain.

3 JAVA MOBILE AGENT TECHNOLOGY

The Aglets Software Development Kit (ASDK), developed by IBM, is a Java-based framework for implementing mobile agents [1], [14], [15]. ASDK provides an object-oriented programming interface, mechanisms for moving code, data and state information from one machine to another, a platform-independent development and runtime environment, and security mechanisms.

The aglet runtime environment is provided by a graphically enhanced aglet server, called Tahiti, that provides basic functionality to create, clone, dispatch, and dispose of aglets. Aglets use an event-driven model to listen for aglet mobility events. When an event occurs, a method of the corresponding eventlistener object is invoked to perform housekeeping chores either before or after the aglet movement, as shown in Fig. 2.

The basic functionality and runtime properties of aglets are defined by the Java Aglet, AgletProxy, and

![Fig. 2. A mobile aglet runs on server A and is dispatched to server B. Before the aglet leaves server A, its onDispatching method is invoked. Similarly, the aglet’s onArrival method is invoked before the aglet starts to run on server B.](image-url)
AgletContext classes. The abstract class Aglet defines the fundamental methods that control the mobility and lifecycle of an aglet. The Aglet class also provides access to the inherent attributes of an aglet, such as creation time, owner, codebase and trust level, as well as dynamic attributes, such as arrival time at a site and address of the current context.

The AgletProxy class provides a handle that is used to access the aglet. The AgletProxy object provides location transparency by forwarding requests to remote hosts and returning results to the local host. It also acts as a shield that protects the methods of the aglet object from direct access by other objects. All communication with an aglet occurs through its aglet proxy.

As Fig. 3 shows, when a buyer’s stationary agent creates a mobile agent, a reference to the mobile agent’s proxy is returned automatically to the stationary agent. The buyer’s stationary agent passes a reference to its own proxy to the mobile agent, so that the mobile agent can communicate with it.

The AgletContext class provides the runtime execution environment for aglets within the Tahiti server. When an aglet is dispatched to a remote site, it is detached from the current AgletContext object, serialized into a message bytestream, sent across the network, and reconstructed in a new AgletContext, which provides the execution environment at the remote site.

As Fig. 4 shows, when a buyer’s mobile agent arrives at a supplier’s site, it interrogates the aglet context at that site to obtain a reference to the proxy of the supplier’s stationary agent. The mobile agent can then invoke methods on the supplier’s stationary agent.

Besides the Aglet, AgletProxy, and AgletContext classes, the aglet technology also provides additional classes for synchronous and asynchronous messaging between aglets.

Security is a prime concern for mobile agent technology, and aglets provide a security model in the form of an AgletSecurityManager, as a subclass of the Java SecurityManager.

4 THE ARCHITECTURE OF THE MAGNET SYSTEM

The MAgNET system is based on the pull model of marketing in which buyers approach suppliers of products with their requirements. The MAgNET system involves a buyer, a mobile agent for the buyer, and a supplier. The buyer maintains a list of potential suppliers along with their lists of products. A buyer who is interested in acquiring a product creates a mobile agent, provides it with an itinerary of supplier sites, specifies criteria for the acquisition of the product, and dispatches the mobile agent to the potential suppliers. The mobile agent visits each supplier site, searches the product catalogs according to the buyer’s criteria, and returns to the buyer site with the best deal it finds. The buyer either confirms the deal and proceeds with the monetary transaction, or aborts the query and disposes of the mobile agent.

As Fig. 5 shows, the architecture of the MAgNET system consists of the subsystem at the buyer site, the buyer’s mobile agent, and the subsystem at the supplier site. Each site may be both a buyer and a supplier, as Fig. 6 shows.

The buyer’s subsystem consists of the buyer’s stationary agent, a graphical user interface (GUI), a data structure containing unsatisfied purchase requests from which the instructions to the buyer’s mobile agents are generated, and a data structure of pending purchase reservations and confirmations. The buyer’s stationary agent creates and manages the buyer’s mobile agents, receives messages from them, and updates the data structures in response to messages. It also interacts with the human user (buyer) through the GUI and with the
buyer’s production planning software through an Open Buying on the Internet (OBI) interface [17]. If it receives an attractive quotation, the buyer’s mobile agent makes a reservation for the product at the supplier site. Before it moves to the next site, the buyer’s mobile agent leaves behind a temporary surrogate agent that waits to receive instructions from the buyer’s stationary agent either to confirm the purchase or to cancel the reservation.

The supplier’s subsystem consists of the supplier’s stationary agent which interacts with the buyer’s mobile agents, an inventory contained in a product catalog, a transaction log which records the interactions between the supplier’s stationary agent and the buyer’s mobile agent, and an interface to production planning software that can be interrogated to obtain quotations for products that are not in the current inventory. The planning software is not part of the MAgNET system, but interfaces to the supplier’s own buyer subsystem to dispatch mobile agents to obtain quotations for subcomponents. These requests may ripple through many levels of buyers and suppliers, so that the entire supply chain may be activated to meet the original buyer’s requirements.

At the buyer’s site, the products to be acquired are described by an XML [13], [26] annotated data structure that represents a directed acyclic graph (DAG). The DAG contains the components that must be acquired to make the product, with the end product at the root of the DAG. In the example shown in Fig. 7, the component to be acquired is a computer, and the root of the DAG is labeled with the product computer. Subcomponents of the computer, such as a keyboard, display, and disk drive, form the second level of the DAG. The node labeled disk drive may again be the root of a subDAG containing the head, motor, and controller nodes. At each level of the DAG, the buyer opts either to purchase the completed product or to purchase and assemble the constituent subcomponents.

Each node of the DAG contains the name, maximum price, required quantity, and required delivery date of the product to be acquired, and a list consisting of the potential supplier URL, supplier identifier, and product identifier for that supplier. The information in the DAG about potential suppliers is assembled from product descriptions presented to the buyer by mobile agents dispatched by the supplier [8] and from other sources. To accomplish this, the suppliers need to know the identities of the buyers a priori. The buyers’ identities can be obtained beforehand from a centralized server or database that stores a list of buyers, wholesale retailers, warehouses and other potential sales outlets. The supplier then dispatches its mobile agents to those potential buyers with lists of its products. Where such a product matches a potential need of the buyer, and the supplier satisfies business partner requirements, the buyer site enters the supplier and its products into the DAG.

4.1 The Buyer’s Site

The buyer’s site provides the portal through which the human buyer can access the MAgNET system, logging on to the system using a secure password. As Fig. 7 and Fig. 8 show, the buyer or the planning software initiates a procurement by forming a DAG, with the intention of procuring the components represented by the nodes of the DAG. The buyer’s stationary agent traverses the DAG in a depth-first manner using the method traverseDAG, which extracts the buyer’s requirements (price, quantity, delivery deadline, and priority) from the DAG and encodes them into an assignment table from which the instructions to the buyer’s mobile agents are generated. Once the DAG has been traversed, the buyer’s mobile agent creates the mobile agents and initializes them. It then dispatches each mobile agent to the first URL on its itinerary list of suppliers.

It is possible that components represented by different nodes of the DAG can be procured from the same set of
suppliers, resulting in multiple mobile agents visiting the same suppliers and possibly competing to acquire the same components. This redundancy is eliminated by identifying components that can be obtained from the same set of suppliers and by generating a single mobile agent to handle all such components.

The buyer’s stationary agent handles two types of messages from the agents. The temporary surrogate agent, left behind by the buyer’s mobile agent at a supplier site, sends a message reporting a reservation to the buyer’s stationary agent. If the supplier’s stationary agent has asked for more time to produce quotations and the buyer’s mobile agent has already left the site, the temporary surrogate agent sends a message reporting the quotation to the buyer’s stationary agent.

When the buyer’s mobile agent returns to the buyer’s site, it gives the stationary agent at that site the list of reservations that it has made. The buyer’s stationary agent compares this list with the lists it has received from the surrogate agents. It then presents the best deal to the human buyer for approval, or to the planning software, so that it can construct a quotation. If the buyer confirms the deal, the buyer’s stationary agent sends a message to the surrogate agent instructing it to cancel the reservation and pay any reservation fee. Some reservations may have to await the return of other mobile agents before those reservations can be confirmed or canceled.

To increase the probability of procuring the required components, the buyer’s request can be satisfied by multiple suppliers, each of which satisfy the order partially, instead of reserving the entire quantity desired by the buyer from a single supplier. The buyer’s stationary agent coalesces the responses of the multiple suppliers before it confirms the purchase. For a buyer who is tightly constrained by time, the reservation scheme supports a one-shot confirmation in which an immediate purchase confirmation is made at the first supplier site that satisfies the buyer’s requirements.

The pseudocode for the buyer’s stationary agent is shown in Fig. 8.

4.2 The Buyer’s Mobile Agent

When the buyer’s stationary agent creates a mobile agent, it passes to the mobile agent the records from the assignment table that contain details of the products for which the buyer’s mobile agent is to search. These records contain the list of suppliers to be visited, the products to be obtained, the price, quantity and delivery deadline for each product and a priority that determines which of these three
Fig. 6. Each site is both a buyer and a supplier. When the buyer's mobile agent requests items that are not in the current inventory from the supplier, the supplier's stationary agent communicates with a planner, which decides how to obtain the requested items. The planner may invoke the buyer's stationary agent at its own site to dispatch mobile agents to other suppliers to obtain subcomponents.

Fig. 7. The structure of the buyer's subsystem, consisting of the buyer's stationary agent, a graphical user interface (GUI), a data structure containing unsatisfied purchase orders submitted by the users, and a data structure of pending purchase requirements, reservations, and confirmations.
attributes has highest priority. The buyer's mobile agent constructs, for each product, an object named bestYet. This object stores the current best offer that a supplier has made, and is initialized to the buyer's minimally acceptable deal. If the mobile agent encounters a better quotation satisfying the buyer's search criteria, the previous best offer is overwritten.

The buyer's mobile agent follows a stop-and-shop scheme, visiting in turn each supplier site listed in its itinerary. On arrival at a new location, the mobile agent checks its location and determines which components to request at that location. The mobile agent then initiates a query dialog with the supplier's stationary agent, using its obtainQuotation method, to obtain the price and delivery deadline for the required quantity of each component to be procured. The supplier software may respond with a quotation or with a request for time to quote. If the offer made by the current supplier improves the best offer recorded in the bestYet object, the mobile agent reserves the component with the current supplier, creates a record for the reservation, and updates bestYet.

Having solicited an offer for all of the component parts to be investigated at this location, the buyer's mobile agent determines whether any reservations have been made or any requests are still pending. If so, it creates a temporary surrogate agent at the supplier's site and passes the list of reservations and pending requests to that surrogate agent. The surrogate agent maintains the reservations and messages received from the buyer's stationary agent. The mobile agent then dispatches itself to the next supplier on its itinerary.

When the mobile agent finally returns to the buyer after visiting the suppliers, the bestYet object holds either the best quotation obtained or the buyer's initialization of that object. The mobile agent reports its reservations to the buyer's stationary agent and then disposes of itself.

The pseudocode for the buyer's mobile agent is shown in Fig. 9.

4.3 The Supplier's Site
As Fig. 10 and Fig. 11 show, the buyer's mobile agent contacts the supplier's stationary agent on reaching the supplier site.
For each request made by a mobile agent, the supplier’s stationary agent invokes the checkAvailability method to determine the response to the mobile agent. The checkAvailability method uses the product identifier of the item corresponding to the current supplier to search a catalog of products to determine whether the required quantity is already in the inventory and thus available-to-offer. If so, the supplier’s stationary agent gives an immediate quotation to the buyer’s mobile agent. Otherwise, the supplier’s stationary agent asks the buyer’s mobile agent for time to quote. If the buyer’s mobile agent agrees, the supplier’s stationary agent asks the buyer’s mobile agent for time to quote. If the buyer’s mobile agent agrees, the supplier’s stationary agent needs extra time to quote. If the buyer’s mobile agent agrees, the supplier’s stationary agent needs extra time to quote. If the buyer’s mobile agent agrees, the supplier’s stationary agent needs extra time to quote.

The catalog at the supplier site is stored in XML-annotated files [13], [26], which correspond to the leaf nodes of the DAG. For example, a cd player might be found under the pathname /music/cd/player, or under the pathname /electronics/audio/portable/cd/player. Such items are cross-linked to maintain consistency of

Fig. 9. Pseudocode for the buyer’s mobile agent.
information. Representation of the inventory in this manner enables efficient searching of items.

The supplier’s stationary agent must also handle responses from the buyer’s mobile agent or the temporary surrogate agent left behind by the mobile agent. There may be requests to reserve components, cancel reservations, or confirm purchases. The supplier’s stationary agent may also receive responses from the planning software, which it passes to the buyer’s mobile agent or the surrogate agent. If the surrogate agent confirms a purchase, the supplier’s stationary agent will negotiate settlement with the surrogate agent, using some form of electronic settlement protocol. If the surrogate agent cancels a reservation, the supplier’s stationary agent may impose a reservation fee for the quantity of components reserved and the time that they have been reserved, and bill the surrogate agent. Cancellation penalties discourage competitors from making malicious reservations that freeze stock. Because the buyer’s mobile agents are fast, reservations are typically held for a short time and reservation fees are small. The amount of time that the surrogate agent is allowed to remain at the supplier’s site may depend on the quantity of components reserved and on the duration of the reservations.

Basic security is provided during establishment of the commercial relationship between the buyer and the supplier [8], without which the buyer’s mobile agent would not be admitted to the supplier’s site. At the supplier’s site, the Tahiti server provides quite good protection against improper access by the mobile agent to code, data and files on the host site. The buyer’s mobile agent is permitted read access to the XML catalog of available products, but cannot change that database and cannot access, for example, the file of pending orders. Limits on an aglet’s size and execution time protect against denial of service attacks.

The pseudocode for the supplier’s stationary agent is shown in Fig. 11.

4.4 Multilevel Supply Chains

In a realistic multilevel supply chain, it is likely that the various suppliers may need to contact their own suppliers before they can respond to a request. An interesting complication arises when several suppliers approach the same supplier of a raw material or subcomponent that is needed by all of them.

Consider, for example, a buyer that needs 100 components and approaches three suppliers to quote for those 100 components. Each of those suppliers approaches a common supplier for a necessary subcomponent. Unfortunately, the supplier of the subcomponent can provide only 100. If the supplier of the subcomponent is unaware that all three enquiries correspond to a single original request, and that only one batch of 100 subcomponents is actually required, it is likely that the supplier will decline two of the requests for 100 subcomponents, eliminating two of the potential suppliers. With multiple subcomponents, it is possible that
essential subcomponents will be locked-up by different suppliers and that no supplier will be able to complete its bid.

If, however, the buyer had been seeking 300 components and had split the total requirement among the three suppliers, then there would be a real need for 300 subcomponents and the availability of only 100 would be a problem. Further complications arise because the buyers may not wish to disclose their intentions to the potential suppliers too early in the negotiation.

For MAgNET, we have devised a codeword strategy that solves many of the more common cases of false contention in the supply chain. The buyer issues with its request a codeword that contains two parts: the first part encodes a buyer and the second part encodes a potential purchase. As derivative procurements spread through the supply chain, the requests acquire one or more additional codewords at each step in the supply chain. As Fig. 12 shows, buyer A requests 100 units from supplier B using codeword A-13. For clarity, in this example, the first part of the codeword is the name of the buyer. In practice, the first part of the codeword will conceal the identity of the buyer. When B requests 100 subcomponents from supplier F, B adds an additional codeword, so that B’s request carries the codewords A-13, B-77. Confronted with two or more contending procurements, a supplier can compare the codewords to determine whether the procurements are indeed distinct. Two sets of codewords present no resource conflict if the first codeword of each set is the same and the first difference occurs in the first part of a codeword. They
present a true resource conflict if the first codeword is different or the first difference occurs in the second part of a codeword.

Thus, if the buyer approaches three suppliers for quotations for the same 100 components, all three requests will carry the same codewords, allowing the suppliers of the subcomponents to respond to all three. If, instead, the three solicitations aim to obtain all 300 components, then the codewords issued will carry the same first component and a different second component. The suppliers of the subcomponents will now know that these are independent procurements.

Fig. 12 shows a buyer A who, intending to obtain 100 components, solicits 100 from two suppliers B and C, 40 from another supplier D, and 60 from a fourth supplier E. Suppliers B and C receive the same codeword from buyer A. Suppliers D and E receive two codewords from buyer A. The first of these codewords is the same as that issued to suppliers B and C to indicate that no conflict exists between this procurement and that requested from B and C. The second codewords issued to D and E have identical first parts and different second parts, which indicates that both sets of subcomponents are required and the same subcomponents cannot satisfy both solicitations.

5 EXPERIMENTAL RESULTS

We have conducted two simple experiments. The first experiment measured the effect of aglet size on the time required to move an aglet to a remote host and back again. The second experiment determined the time required for an aglet to complete a multihop round-the-world itinerary.

The configurations for the first experiment are shown in Fig. 13 and Fig. 14. Servers A, B, and C are located within the campus network of the University of California, Santa Barbara. Whereas servers A and B are co-located in the same domain (ece.ucsb.edu), server C is located in a different domain (engr.ucsb.edu) one hop away. Server D is a publicly accessible Tahiti server hosted by Osaka University in Japan (maintained by Takeshi Shirai). Server A is the dispatching server and, therefore, is the focal point of our measurements. Server B is used to provide a reference measurement for comparison.

The buyer is assumed to be located at server A and the suppliers at servers B, C, and D. The buyer’s stationary aglet on server A is responsible for spawning the buyer’s mobile aglet, dispatching it on \( n \) trial runs and processing the timestamps it returns at the end of every run. The dispatched mobile aglet completes a round-trip between server A and one of the other servers (either B, C, or D), registering timestamps corresponding to its departure from server A and its subsequent return. The mean round-trip time \( t_{\text{mean}} \) is computed as

\[
    t_{\text{mean}} = \left( \sum_{i=1}^{n} t(a)_i - t(d)_i \right) + n
\]

where, for the \( i \)th trial run,

- \( t(a)_i \) is the timestamp registered by the aglet prior to departing from server A.
- \( t(d)_i \) is the timestamp computed by the aglet immediately upon returning to server A.
- \( n \) is the number of trial runs made by the aglet between a given pair of servers.

The number \( n \) of trial runs made by the aglet was 100.

By computing both timestamps with respect to the clock at server A, we avoided the complications of calculating offsets due to clock skew arising from unsynchronized clocks at the two servers between which the mobile aglet travels. Trial runs to the three different destination servers were performed concurrently to eliminate bias on any specific run (due to varying network conditions, time of day, etc). Concurrency was achieved by having three identical stationary aglets on server A, each responsible for spawning and monitoring the mobile aglet for a particular destination server.
The first phase of the first experiment consisted of dispatching a small aglet (100 lines of code) from server A to each of the other servers. As expected, the round-trip time was the smallest for server B (299 ms) on the same local-area network and the largest for server D (2,105 ms) across the world in Japan.

The second phase of the first experiment consisted of varying the size of the aglet by adding code in increments of 100 lines. The intent was to increase the size of the code to be carried by the aglet, rather than to increase the amount of processing at the remote server. Because we were interested in the travel time of the aglet, we discounted processing time. The round-trip time as a function of aglet size is shown in Fig. 15. Each data point was obtained by taking the mean over 100 trial runs.

The first experiment yielded a number of interesting results. Whereas mobile aglets traveling to the remote server in Japan showed a nearly two-fold increase in round-trip time as the size of the aglet increased, the mean round-trip time still remained within a reasonable 4 sec. On the other hand, mobile aglets traveling within a relatively local area showed negligible change in round-trip time with increasing code size. An interesting observation was the inability of the server to load aglets exceeding a certain size (2,300 lines of code). Although future versions of the Aglet Software Development Kit may not be so limited, the existing limitation is deemed acceptable because, in our architecture, the aglets contain about a thousand lines of code on average (far below the 2,300-line limit).

The configurations for the second experiment are shown in Fig. 16, and include a public server hosted by the University of Genoa in Italy (maintained by Fabrizio Guidici) in place of the reference server in the previous experiment. As before, the buyer is located at server A, with suppliers located at servers B, C, and D.

In the second experiment, the buyer’s mobile aglet was provided with an itinerary that specified the addresses of the four servers and also the order in which they were to be visited. The buyer’s stationary aglet on server A dispatches the mobile aglet on its trip, processing the data sent back by the mobile aglet from each destination. The mobile aglet traverses the servers in order, registering timestamps at each server immediately upon arrival and just prior to departure. While hosted on a server, the mobile aglet sends timestamps $t_d$ and $t_a$ back to the stationary aglet, $t_d$ being the timestamp corresponding to its departure from the previous server and $t_a$ the timestamp corresponding to its arrival at the current server. This enables the stationary aglet to compute the time $t_{hop}$ taken by the mobile aglet to complete the hop between those two servers. By making the dispatching server the last stop on the itinerary, $t_{roundtrip}$ is computed by using the first $t_d$ and the last $t_a$ reported by the mobile aglet.
The geographical distribution of servers is shown in Fig. 17. A complication of using timestamps in the experiment is that the clocks at the various servers may not be synchronized. To eliminate the bias introduced by clock skew, two identical stationary aglets were created at the dispatching server, each monitoring the actions of one mobile aglet. The two mobile aglets were then provided distinct itineraries consisting of the same servers but requiring a clockwise traversal by one mobile aglet and a counterclockwise traversal by the other. The two \( t_{\text{hop}} \) values thus obtained for each hop were then averaged to eliminate the skew. The two mobile aglets were dispatched at the same time by their respective stationary aglets.

The important results obtained from this second experiment are highlighted in Fig. 17, and the details are given in Fig. 18. The round-trip time shown in this case takes into account both the travel time of the mobile aglet and the processing times at the various servers. It is worth noting that server C, running Linux on a Pentium processor, takes approximately twice the processing time as the other servers and contributes an eighth of the round-trip time calculated. If processing time is eliminated, the effective time for the mobile aglet to travel once around the world is approximately 7 sec. Thus, if there are two buyers looking for the same deal, one buyer would require a window of at least 7 sec to preempt the other successfully.

### 6 Related Work

MAgNET is an electronic commerce system developed using the infrastructure provided by IBM’s aglet technology. Java’s support for mechanisms such as class loaders, remote method invocation, reflection, and object serialization has spawned a new generation of agent-based systems aimed at both the infrastructure and the applications. We now discuss alternatives to IBM’s aglet technology, and also compare MAgNET to existing electronic commerce systems.

#### 6.1 Mobile Agent Platforms

In addition to IBM’s Aglets, the primary contenders in the mobile agents arena are General Magic’s Odyssey, ObjectSpace’s Voyager, and Mitsubishi’s Concordia. All of these systems are Java-based (JDK1.1 compliant) systems that promote agent migration with state preservation and that provide an agent “context” or server to host mobile agents. However, the systems are geared toward different applications and, consequently, exhibit diverse approaches to handling messaging, persistence, security, and agent management.

General Magic’s Odyssey [18] makes extensive use of the Java Remote Method Invocation (RMI) and reflection mechanisms. While RMI is the default, implementations using IIOP and DCOM are also provided, making Odyssey the only system to support all three protocols. A downside of RMI is that it necessitates the running of an rmiregistry, in addition to the Odyssey server, on each participating machine. An Odyssey agent is defined by its allotted set of tasks and its dynamic itinerary. Being a Java thread, an Odyssey agent cannot preserve its execution state during migration. Consequently, a dispatched agent needs either to restart execution upon arrival at the destination or to ensure completion of a task prior to departure. While Odyssey provides basic agent functionality, more support for agent-to-agent communication, agent persistence, and multiple servers would be desirable.

Objectspace’s Voyager [22] is an agent-enhanced Object Request Broker (ORB) for distributed computing. The current release promises a wealth of functionality in the form of dynamic RMI-CORBA compatibility, multicast messaging, integration with existing databases, etc, aimed at agent-based enterprise applications. Voyager exploits Java’s RMI mechanism in conjunction with object serialization, providing tools both to support agents and to build them. The core of Voyager lies in “vcc,” a tool capable of taking an existing Java class that implements serialization, and of dynamically creating a mobile “virtual object” that
mirrors it. This requires some complex coding on the part of the developer to ensure that the classes are compatible with the vcc functionality. Voyager supports the functionality required by MAgNET, and much more, making it rather complex and heavyweight for our purposes.

Mitsubishi’s Concordia [23], [24] is a relatively new technology, intended specifically for enterprise computing. Like Voyager and Odyssey, Concordia relies heavily on Java RMI. However, it is more modular in design, with specialized “manager” objects to oversee the different system functions. These include an agent manager to supervise the communication infrastructure, a persistence manager to maintain and monitor state for all agents in transit through the network, an interagent communication...
manager to oversee collaborations between agents and to perform multicast event notifications, a directory manager to provide a naming service, etc. Concordia also provides an agent tool library with components for building Concordia agents. Like Voyager, Concordia provides more functionality than MAgNET requires, and is rather heavyweight.

After evaluating these agent systems, we elected to use IBM’s Aglets for several reasons. Aglets provide a freely available and lightweight implementation of mobile agents. The Aglet Software Development Kit supports the concept of multiple servers per host and gives developers sufficient information to build their own customized servers. Aglets provide the minimal agent functionality required by MAgNET including agent cloning, agent dispatch and retraction, agent persistence and activation, dynamic itineraries, synchronous and asynchronous messaging, and some elements of security. An important factor in choosing IBM’s Aglets is that the agent model has some similarity (event handling, initialization, etc) to the Java applet model, and presents a familiar appearance to Java users.

### 6.2 Electronic Commerce Systems

Electronic commerce systems range from simple online shopping services to more complex infrastructures that provide a wider range of services. These systems are distinguished by their implementation (or lack thereof) of the six stages of the consumer buying behavior model [11]. These stages are need identification, product brokering, merchant brokering, negotiation, purchase and delivery, and product service and evaluation. We describe several existing electronic commerce systems below.

BargainFinder [4], [5] is essentially a database search engine that queries several online music stores for the best deal on CDs and cassettes. Merchant brokering is restricted to retailers who subscribe to the service, and there are limitations on the service they receive. Jango [9], [12] is similar to BargainFinder, but allows price-based comparison shopping. It exploits the services of the Excite search engine, thus removing limitations on the merchants that it can access. Both systems are intelligent search engines that collect information from suppliers and hold it at a central site; actual purchase transactions require browsing at the supplier’s site. Unlike MAgNET, neither of them supports assembly of products from subcomponents or acquisition of products spread over the entire supply chain.

In contrast, Firefly [4], [10] (which is derived from Ringo [20]) is a mail-order system that provides automated collaborative filtering. This implies that the system trains itself to recognize trends in the users’ preferences that enable it to make recommendations to the users, based on the purchases of other users with similar profiles. Unlike MAgNET, the system is limited to a single supplier and provides no room for negotiations.

AuctionBot [3], [25] is a generic auction server that allows suppliers to auction products by specifying the acceptable parameters for the sale. Buyer agents and seller agents can conduct negotiations with AuctionBot, enforcing user-specified parameters to control the bidding. AuctionBot provides an API that allows a user to create agents that are primed with customized bidding strategies. The agents are dispatched to, and operate at, the single auction server; they do not move from supplier to supplier.

MIT Media Lab’s Kasbah [7] is an online World Wide Web marketplace for buying and selling goods. A user creates a buyer agent, provides it with a set of criteria and dispatches it into the marketplace. The criteria include price, time constraints and quantity of merchandise desired. Users can select one of several price decay functions for goods they are attempting to sell. Supplier agents post their offers on a common blackboard and wait for interested buyer agents to establish contact. A buyer agent filters the available offers according to the user’s criteria, and then proceeds to negotiate a deal. Both buyer and supplier agents operate within the single marketplace server and are dispatched by the buyer or supplier to that server; they do not move from server to server.

The Minnesota AGent Marketplace Architecture (MAGMA) [21] is a prototype for a virtual marketplace targeted toward items that can be transferred over the Internet (such as information). It consists of Java-based trader agents (buyer and seller agents), an advertising server that provides a classified advertisement service, and a bank that provides financing support and payment options. Independent agents can register with a relay server that maintains unique identifiers for the agents and routes interagent messages.

Unlike the systems described above, MAgNET is not based on an auction house server or advertisement board server. Such a server is likely to offer only a restricted range of products. The objective of MAgNET is to allow direct interactions between arbitrary buyers and suppliers for products in which they have a mutual interest, in order to cover the entire supply chain seamlessly.

Xpect [2] is a generic framework for electronic commerce implemented using an object-oriented language, called the Coordination Language Facility (CLF). The system supports customer agents that issue requests, provider agents that display catalogs of items in stock, banker agents that oversee transaction payment options, and broker agents that coordinate interactions between the other agents. Xpect accepts a user’s request, searches catalogs for the item, supervises the financial transaction, and notifies the customer of the status of shipping and delivery. An interesting aspect of XPect is that both the customer and the providers must be clients of the banker by prior registration.

Like MAgNET, the Defense Acquisition System for High-performance Services (DASHER) [19] focuses on the procurement aspect of electronic commerce. It provides a common service infrastructure that supports distributed applications and federations of loosely affiliated software components. DASHER intertwines multiple procurement services, which are controlled asynchronously during operation, using a federated systems approach. DASHER relies predominantly on e-mail, fax, and voice mail for communication among different sites.

The Agent Development Environment (ADE) toolkit [16] is an integrated development environment for designing, developing, debugging, simulating and deploying agents. ADE has been used for building applications in the areas of
manufacturing scheduling, manufacturing process control, network information filtering and network load balancing. It has also been applied to the development of intelligent supply chain agents for the software procurement process using electronic commerce. MAGNET, in contrast, aims specifically at the buyer/supplier supply chain, but is not limited to the procurement of software.

The Metabroker system [6] is a generic framework for creating electronic brokers that provides commonly required functionality and support for popular communication protocols and data formats. Specialist brokers are created by adding the necessary business logic to support the area of speciality of the broker. Unlike MAGNET, the broker acts as an agent of the supplier, rather than of the buyer. None of the systems cited here handles the deep supply chain used by MAGNET to acquire items that are not currently in the supplier’s inventory. The duality of a site as both supplier and buyer, integration of MAGNET with planning software and the fan-out of orders over the entire supply chain distinguishes MAGNET from these other electronic commerce systems.

7 Conclusions and Future Work

Automation of the global production and distribution supply chain through electronic commerce technology offers the opportunity for greatly increased efficiency, for reduced inventories, waste and spoilage, and for increased responsiveness to changing consumer preferences and advances in technology. The challenge is to make it work. Our initial experiments with Java mobile software agents (aglets) have been very encouraging. In MAGNET, aglets have operated on a variety of platforms with excellent performance. Our experiments have shown that a simple aglet can travel from California to Japan and return to California in about 2 sec. They have also shown that an aglet can make a round trip from California to Japan and back to California in only 9 sec. These are acceptable delays in the face of Internet transactions. Certainly, the global scope of the supply chain is no longer a hindrance to the automation of electronic commerce.

We do not know whether a pull model (buyer’s mobile agents visiting suppliers) or a push model (supplier’s mobile agents visiting buyers) is more appropriate. We also do not know whether it might be more appropriate for the buyer’s or supplier’s agents to become semipermanently resident on the computers of established partners in the supply chain. We do not know how the negotiations between buyer and supplier should be conducted, and we do not know how much information a mobile software agent must carry with it to conduct a meaningful negotiation. We do not yet know how to integrate buying, selling, planning and logistics to allow a single procurement to spread automatically through multiple levels of the supply chain, and we do not know how to exploit the integration of software systems through the supply chain to monitor a procurement after the initial purchase decision. Certainly, this is an important and exciting area of research for the future.

ACKNOWLEDGMENTS

This research has been supported by QAD Inc. through the State of California MICRO Program under Grants No. 97-122 and No. 98-107.

REFERENCES

Prithviraj Dasgupta received his MS degree in computer engineering from the University of California, Santa Barbara, in 1998. He is now a PhD candidate in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. His dissertation research focuses on mobile agents and their application to e-commerce. He is a student member of the ACM and the IEEE.

Nitya Narasimhan obtained her MS degree in computer engineering in 1996 and is now a PhD degree candidate in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Her research interests include mobile agents, distributed object computing, security, and fault tolerance in distributed systems. She is a student member of the ACM, the IEEE, and the IEEE Computer Society.

Louise E. Moser received a PhD degree in mathematics from the University of Wisconsin, Madison. She is now a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. She has served as an area editor for the IEEE Computer Society’s Computer magazine and as an associate editor for the Society’s IEEE Transactions on Computers. Her research interests include distributed systems, computer networks, and software engineering. She is a member of the ACM, the IEEE, and the IEEE Computer Society.

P. Michael Melliar-Smith received a PhD degree in computer science from the University of Cambridge, England. He is now a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. His research interests include fault-tolerant distributed systems and high-speed communication networks and protocols. He is a member of the ACM, the IEEE, and the IEEE Computer Society.