JOURNAL OF AERONAUTICS AND SPACE TECHNOLOGIES
JULY 2003 VOLUME 1 NUMBER 2 (21-32)

A TWO-LEVELED MOBILE AGENT SYSTEM FOR ELECTRONIC COMMERCE

Ozgur Koray SAHINGOZ
Air Force Academy
Computer Engineering Department
Yesilyurt, Istanbul, TURKEY
o.sahingoz@hho.edu.tr

Nadia ERDOGAN
Istanbul Technical University
Electrical-Electronics Faculty
Computer Engineering Department, Ayazaga
80626, Istanbul, TURKEY
erdogan@cs.itu.edu.tr

Abstract:

Electronic commerce technology offers the opportunity to integrate and optimize the global production and distribution on the supply chain. Computers of various corporations, located throughout the world, communicate with each other to determine the availability of components, to place and confirm orders, and to negotiate delivery timescales over the Internet. Software agents help to automate a variety of tasks including those involved in buying and selling products over large-scale networks like the Internet. This paper presents a two-leveled mobile agent system for electronic commerce based on mobile agents, using the publish/subscribe protocol for registration and transaction processing. In a large-scale and dynamic environment, there can be any number of buyers and suppliers at any time. In this system, suppliers can connect, register or unregister to the system at any time, thus preserving the dynamic structure of the system. It not only simulates real commercial activities by buyers, agents and suppliers, but also provides an environment for parallel processing. The latter is particularly important as more shops (sites) can be searched in real time to provide buyers with better choices. Meanwhile, if the number of mobile agents is very large and their dispatching is processed in a serial way, it can become a bottleneck that affects the efficiency as a whole

Keywords: E-commerce systems, Multi-level agent systems, Publish/subscribe paradigm.

1 INTRODUCTION

The number of businesses and individuals through the world who are discovering and exploring the Internet is growing dramatically. Through the past decades we have seen an increasing rate of globalization of the economy and, thereby, also of supply chains. Products are no longer produced and consumed within the same geographical area. Even the different parts of a product may, and often do, come from all over the world. This creates longer and more complex supply chains, and therefore it changes the requirements within supply chain management. The Internet is a cheap, open, distributed, and easy to use environment that provides an easy way to set up shops and conduct commerce at any place [1].

The advances of web technologies such as the Internet, HTML, Java and XML have greatly pushed the development of electronic commerce. Today, many electronic shops publish their product catalogue on the Internet, offering a wide variety of goods. More importantly, consumers are turning to the Internet for such information as well as to purchase their goods.
Electronic commerce is a domain where agent technologies are well suited. The search and retrieval of product information is crucial in every e-commerce system. While some systems provide the user just with a fixed selection of products, others offer a wide variety of products with different characteristics. In this case, the system must provide means to efficiently and accurately search and retrieve product information according to the user's desired product characteristics.

To enable the deployment of dynamic e-commerce environments the European and the US Agentcities initiatives [2], combined with international initiatives in FIPA [3], aim to create a global open information-exchange environment where dynamic services from geographically distributed organizations can be deployed, tested, interconnected and composed. Examples of such dynamic services are: B2B (Business to Business) dynamic value chain creation, dynamic pricing through trading exchange, automatic discovery of business partners, advertising and marketing.

Accompanied by the growth of e-commerce, rapid responses to changes in demand and customer preference, and the ability to exploit new technologies are becoming critical. As information on the Internet becomes more dynamic and heterogeneous, 'software agents' are thought of as the new building blocks for a new Internet structure.

As pointed out by Rodrigo [4], future e-commerce models will enhance current models by using mobile agents. On one hand, in our real life, people can turn to a few agents or agencies for buying something such as an air ticket, or renting a house. They can choose a satisfactory one from multiple plans provided. On the other hand, the mobile agent scenario offers us more flexibility and agent-oriented modeling capability to apply the buyer / agent / supplier model of real commercial activities to the building of electronic marketplaces. In addition, it can also provide an environment for parallel processing over distributed sites to achieve greater efficiency.

In this paper we propose a framework for large-scale electronic commerce system that uses the publish / subscribe paradigm and exploits mobile agent technology extensively. It not only supports activities of buyers and suppliers, but also facilitates parallel computation. The latter is especially important because by using parallel computation more suppliers can be searched in a shorter time to provide buyers with better choices in their decision-making. We present a platform that uses the publish/subscribe mechanism for dynamic utilization of the system, and mobile agents as mediators between buyers and suppliers.

In a large-scale and dynamic environment, there can be any number of buyers and suppliers at any time. Suppliers can connect, register or unregister to the system at any time, thus preserving the dynamic structure of the system. The architecture that we propose is based on the publish/subscribe paradigm which supports many-to-many interaction of loosely coupled entities as depicted in Figure 1. We define the set of services that need to collaborate with the publish/subscribe infrastructure to address the dynamics of mobile environments.

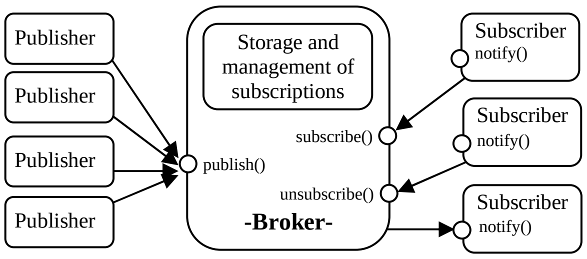

Figure 1. Publish/Subscribe model

The rest of the paper is organized as follows. In the next section, we present mobile agents and the mobile computing paradigm. Section 3 describes some significant electronic commerce systems with references to related works. Section 4 introduces the computational model of the two-leveled mobile agent system. Our conclusions and directions for future work are presented in Section 5.

2 MOBILE AGENTS

The term mobility is used to indicate a change of location performed by the entities of a system. Starting from simple data, the mobility has had an evolution that has led to the movement of execution control, code and execution environment. In the first step of the evolution, we find the mobility of files, for example with the FTP protocol. The next step was the remote procedure call (RPC): in this case, the execution flow is involved in the movement of data between Client and Server. A client calls a remote procedure like a local procedure by sending the necessary parameters and receives the results, as depicted in Figure 2.

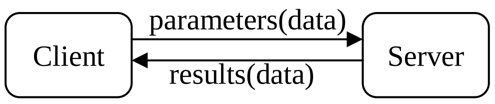

Figure 2. Remote Procedure Call (RPC)

Even though the idea was quite simple and it extended local procedure call, this new model had a great impact in computer science. Then, there was the idea to move code. A piece of program is sent to another machine and is executed there, as shown in Figure 3. This is called remote evaluation (REV)[5].

In the paradigm of remote evaluation, a computational component, a client, has the know-how in order to execute a task, but it lacks the necessary resources, which are suited on a different site. The computational component, therefore, sends the know-how which is needed to fulfill the task, to a computational component, a server, which resides on the same site as the resource. The server executes the task using the know-how received from client. The results of the task are delivered back to client.

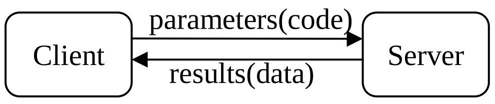

Figure 3. Remote Evaluation (REV)

In the code on demand (COD) paradigm, the computational component, client, has local access to the resources, but does not know how to execute the task. Thus, it contacts a computational component, server, on a different site, which provides the know-how. The client loads the know-how from the server and executes the task locally, as shown in Figure 4.

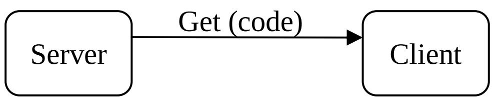

Figure 4. Code on Demand (COD)

Finally, active entities became able to change the environment where they are executing. Thus, the natural evolution of mobility resulted in code mobility. A particular example of mobile code is represented by mobile agents [6]; these are software objects, with data and code, that can be transmitted over the net or can autonomously migrate to a remote computer and execute automatically on arrival. An agent is an active software entity that shows several degrees of autonomy, since it has to take decisions and to carry out jobs without the direct participation of the user. A mobile agent is an autonomous entity with the capability of roaming among nodes in a network-aware fashion to find the needed resources and services, as depicted in Figure 5.

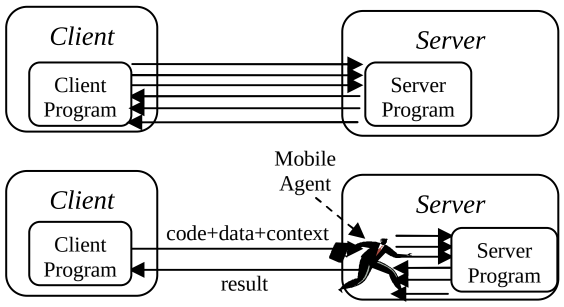

Figure 5. Mobile Agent

Mobile agents are mobile, flexible, autonomous, dynamic and efficient. When encapsulated within a task, a mobile agent can be dispatched to a remote host by the original host. After executing and accomplishing its tasks at the remote host, it can bring the results back by returning to the original host or send them through a message. The mobile agent approach is also suitable for deploying parallel processes over distributed sites on the Internet. The tasks can be decomposed and encapsulated into multiple mobile agents. Every mobile agent can run independently to accomplish its task. Thus, a set of mobile agents can run in parallel on distributed hosts so that the whole task can be completed in a shorter time.

2.1 Kinds of Mobility

Mobile agents have been advertised as an emerging technology/paradigm that provides means to design and maintain distributed systems more easily. Three kinds of mobility have been identified [7,8]:

- *weak mobility*: the dynamic linking of code arriving from a different site. Most of Java based agent systems like Aglet[9] and Mole[6] are weak mobile.
- *strong mobility*: the movement of the code and of the execution state of a thread to a different site and the resumption of its execution on arrival. Systems such as Telescript [10] provide strong mobility by using a dedicated language interpreter to capture and resume the process' execution state.
- *full mobility*: the movement of the whole state of the running program including all thread's stacks, namespaces and other resources. This is a generalization of strong mobility where migration is completely transparent. Full mobility is provided by LOCUS distributed operating system [11]. Full mobility is necessary if process migration is used, for instance, in load balancing where migration has to be completely transparent.

2.2 Mobile Agent Models

Mobile agent models can be classified into five groups [12]; Itinerary Agent, Shuttle Agent, Serial Agent, Virtual Parallel Agent and Serial Parallel Agent.

In the first model, an itinerary agent is created by the master consumer-agent with the addresses of a list of targets. It migrates and visits all targets on its list one by one. When migrating to the next target, it carries the data accumulated from all previous targets including the current one. After it has visited all the targets, it returns to the consumer and carries back the whole data.

Similarly, in the Shuttle Agent model, there is a master consumer-agent and a single worker agent. The worker agent is dispatched to a target in the list maintained by the master consumer-agent. When it has read the data, it returns to the upper-level agent, carrying the results. After sending the data to the master consumer-agent, it migrates to the next target, repeats the process until it has visited all targets, and returns all the data.

In the third model, similar to the second one, the master consumer-agent dispatches a worker agent to a target. When the worker agent obtains the data, it sends it back by a message and then destroys itself. Having gotten the result, the master consumer-agent will dispatch a new worker agent to the next target until all targets are visited.

In the Virtual Parallel Agent model, the master consumer-agent is created at the client end and it starts multiple threads. Each thread locally reads data located in a remote HTTP server. The thread stops when it has finished reading the data and has acquired the results. This model resembles to virtually parallel searching activities in a more efficient way. Obviously, in this model, no worker agent is dispatched.

The fifth model benefits from the parallel data access. In this model, the master consumer-agent dispatches multiple worker consumer-agents one by one to all targets that should be visited. These worker agents can access local data in parallel and send their results to the master consumer-agent in succession, but the dispatch process is serial.

2.3 Mobility Support in Java

Java programming language is strongly network-oriented and provides support for the mobility of code, in the form of the dynamic class loading and applets. In addition, it permits to implement a form of weak mobility by serializing objects and sending them to another JVM via sockets or Remote Method Invocation. The serialization mechanism allows maintaining the values of the instance variables, but it cannot keep track of the execution flow. Several weak mobility systems based on Java have been implemented, both by academic researchers and by enterprises. When the serialized object arrives at the destination JVM, it is de-serialized and is reactivated by invoking a specified method. The choice of that method may vary from different mobile agent systems: for example, it can be the run method or the agent can specify it as a parameter of the go statement.

Java-based mobile agent systems realize an agent by using one or more threads. As the official JVM from SUN does not support a strong kind of agent mobility, to implement it, the JVM code should be modified in order to extract the Java stack and the program counter of the thread(s) to be moved. In particular, they should be collected and sent along with the serialized form of the agent. With regard to the program counter, it is an internal variable of the JVM that can be easily accessed, transferred or restored at the destination node with a light modification to the JVM. The main difficulty is related to the Java stack.

Java based mobile agents inherit the computer independent feature from Java programs and hence provide a platform-independent integration of heterogeneous databases and data sources.

3 ELECTRONIC COMMERCE SYSTEMS

An electronic commerce system covers any form of computerized buying and selling, both by customers and from company to company. These systems are distinguished by their implementation (or lack thereof) of the six stages of the consumer buying behavior model [13]. 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.

AuctionBot [14,15] is a well-known experimental Internet auction server developed at the University of Michigan. Its users can create new auctions by choosing from a selection of auction types and specifying its parameters. Buyers and suppliers can then bid according to the auction's multilateral distributive negotiation protocols. 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 [16] 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) [17] is a prototype for a virtual marketplace targeted towards items that can be transferred over the Internet (such as information). It consists of Java-based trader agents (buyer and supplier 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 inter-agent messages.

MAGNET (Multi AGent NEgotiation Testbed) [18] is an experimental architecture developed at University of Minnesota to provide support for complex agent interactions, such as in automated multi-agent contracting, as well as other types of negotiation protocols. Agents in MAGNET negotiate and monitor the execution of contracts among multiple suppliers. A customer agent issues a Request for Quotes for resources or services it requires. In response, some supplier agents may offer to provide the requested resources or services, for specified prices, over specified periods. Once the customer agent receives bids, it evaluates them based on cost, risk, and time constraints, and selects the optimal set of bids that can satisfy its goals.

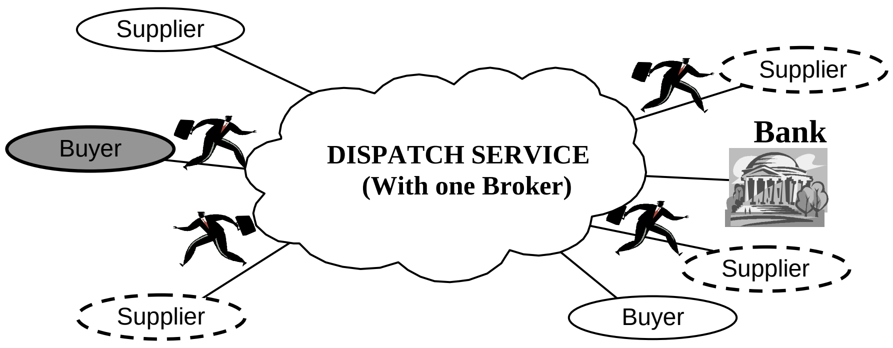

Figure 6. Infrastructure of the system

eNAs (e-Negotiation Agents) [19] and FeNAs (Fuzzy eNAs) [20] are prototypical intelligent trading agents developed at CSIRO to autonomously negotiate multiple terms of transactions in e-commerce trading. The agents can engage in integrative negotiations in the presence of limited common knowledge about other agents' preferences, constraints and objectives through an iterative exchange of multi-attribute offers and counter-offers. Fuzzy eNAs can also flexibly negotiate with fuzzy constraints and preferences.

The F/eNAs environment can consist of many autonomous trading agents representing buyers and suppliers that can engage in concurrent bi-lateral negotiations according to a number of user-selected negotiation strategies.

The work of Papastavrou and Wang showed the advantages of applying the mobile agent approach to parallel processing over distributed databases or data sources [21, 22]. A mobile agent can decompose its tasks to multiple sub-mobile agents and dispatch them to distributed sites simultaneously in order to let them work in parallel. Hence, mobile agent technology is naturally suitable for deploying parallel and distributed computation. Its performance is comparable to, and in some sense outperforms the current approach via expensive network and slow network, such as the wireless network or dial-up network.

We have developed a new model for an e-commerce system, which is based on a two-levelled mobile agent structure that uses the serial parallel mobile agent model, as defined in the previous section. In the system that we present, different from most of e-commerce systems described above, any buyer or supplier can easily join or leave the system using the properties of publish/subscribe paradigm. The computational model of the system is explained in detail in the next section.

4. TWO-LEVELED MOBILE AGENT SYSTEM

Our work consists of a framework for a large-scale electronic commerce system that uses the publish/subscribe paradigm and exploits mobile agent technology extensively. It not only supports activities of buyers and suppliers, but also facilitates parallel computation by running mobile agents on suppliers concurrently. The system has an extensible architecture, as depicted in Figure 6, and provides all the services, which are essential to agent-based commercial activities. Our electronic commerce system involves three actors. *Buyers* are looking for purchase services from suppliers. *Suppliers* or sellers offer the services or products and a *Dispatch Service* facilitates communication between buyers and suppliers. It also includes a communication infrastructure, transfer of goods, banking and monetary transaction along with an economic mechanism for brokered buyer-supplier transactions.

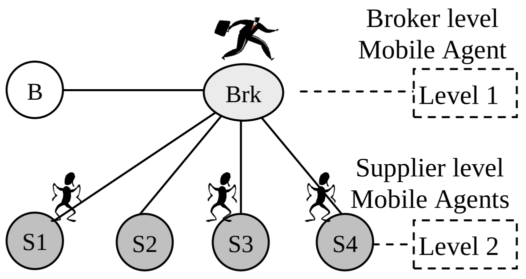

Figure 7. A Two-Leveled Mobile Agent Model

The system consists of mobile agents that belong to two different levels of execution and responsibilities, as shown in Figure 7, Broker level Mobile Agent (BMA) and Supplier level Mobile Agents (SMA). A BMA is created by a Buyer Agent and is sent to the Broker. This BMA creates SMAs and sends them to suppliers in order to search their databases, to select among products, and to negotiate with the supplier (if necessary). The system does not include only a single mobile agent that visits every supplier one by one. Instead, we send a replica of the mobile agent to each of the suppliers concurrently, and thus make use of parallel processing. This model of parallel computation is especially important as more suppliers can be searched in a shorter time to provide buyers with better choices in their decision-making.

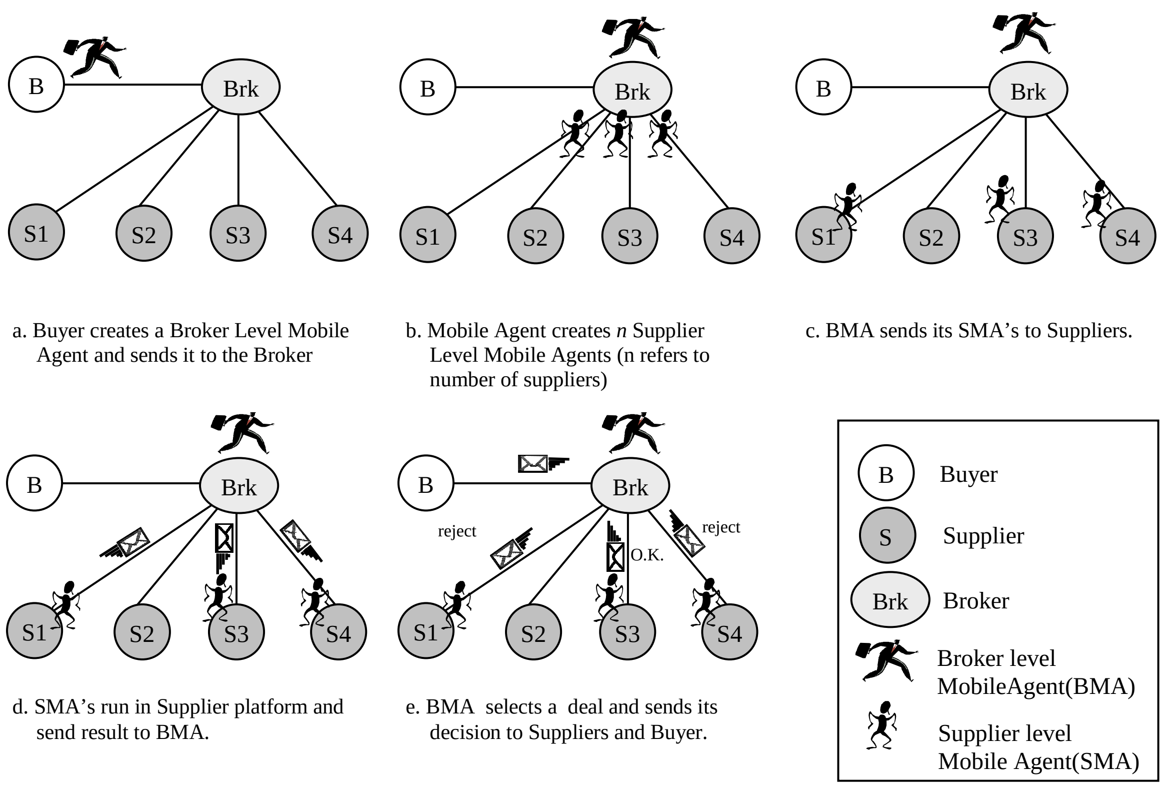
B
Brk
S1
S2
S3
S4
a. Buyer creates a Broker Level Mobile Agent and sends it to the Broker
b. Mobile Agent creates *n* Supplier Level Mobile Agents (n refers to number of suppliers)
c. BMA sends its SMA's to Suppliers.
d. SMA's run in Supplier platform and send result to BMA.
reject
O.K.
e. BMA selects a deal and sends its decision to Suppliers and Buyer.
Buyer
Supplier
Broker
Broker level MobileAgent(BMA)
Supplier level Mobile Agent(SMA)

Figure 8. Buying a product from a Supplier in Two-Leveled Mobile Agent System

In many of e-commerce systems, a buyer (or the system) has a fixed number of suppliers, which are initialized with the system at start up. When a new supplier is to be added, it has to be registered manually by supplying its address and the necessary parameters. In a large-scale and dynamic environment, there can be a varying number of buyers and suppliers at any time. In a dynamically changing electronic marketplace, a system should have the ability to adapt itself to this dynamic world. To meet this requirement, we have designed an architecture that utilizes the publish/subscribe paradigm for registration and dispatching operations, to increase efficiency and effectiveness of the procurement process in terms of costs, quality, performance, and time for both buyers and suppliers.

A user who wants to buy or sell a product creates an agent, gives it some strategic direction, and sends it off into a centralized agent marketplace. Mobile agents of the system proactively visit suppliers and negotiate with them on behalf of their owners. Each agent's goal is to complete an acceptable deal, subject to a set of user-specified constraints such as a desired price, lowest acceptable price, and a date by which to complete the transaction.

The execution flow of the procurement process is as the following:

a. There are some buyers and suppliers, which have subscribed to system with their production and services. The number of buyers and suppliers in the system can increase or decrease at any time.
b. When a user wants to buy a product, he has to make a request from the Buyer subsystem. A Buyer Agent gets a request, creates a Mobile Agent, sets its necessary instance variables and sends it to the Broker as shown in Figure 8.a.
c. When this Mobile Agent (we call it Broker level Mobile Agent (BMA)) arrives at the Broker, it checks the KnowledgeBase of the Broker, selects the suppliers, which produce the requested products, and creates a new Supplier level Mobile Agent (SMA) for each of the selected suppliers. Thereafter, BMA sends each of these agents to them, as shown in Figure 8.b and Figure 8.c.
d. Each SMA searches the product database of its supplier, negotiates with the supplier agent and sends results back to the BMA, as depicted in Figure 8.d.
e. After all results are collected from the SMAs(or the specified timeout period has expired), BMA selects the best dealing one and sends an approval message to this selected SMA and demands it to buy the product and then to destroys itself. BMA sends rejection messages to all other SMAs and they destroy themselves. BMA also sends a message that reports the negotiation and its result to the Buyer Agent, as depicted in Figure 8.e.

Banking: In order for an agent-based marketplace to become anything more than an experimental platform, it has to be able to communicate with existing banking and financial services. When an agent needs to make a payment, it sends a request to its bank to withdraw funds, and receive a secure wrapper, called a check. Before sending out a check, the bank verifies the existence of sufficient funds in the agent's account.

We will now describe in more detail the main components of the system, its subsystems, and explain their functionality and discuss the major design decisions.

4.1 Buyer Subsystem

To request a purchase order from the system, a buyer has to initialize a Buyer Subsystem on its machine. Buyers have to know the address (URL) of the broker agent that they will connect, just like the URLs of well known web sites (i.e. Yahoo!, Alta Vista or Excite).

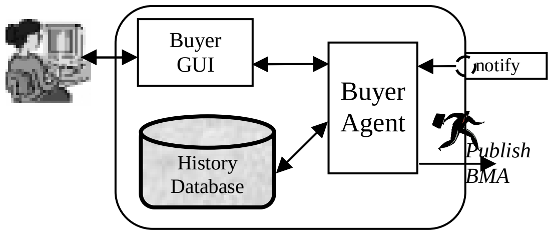

Figure 9. Buyer Subsystem Architecture

The Buyer Subsystem consists of three main components as shown in Figure 9.

a. A History Database, which contains result reports of past procurements.
b. A Graphical User Interface (GUI) which is used for interaction (i.e. getting a new procurement process from a buyer, or searching from the History Database) with human users.
c. A Buyer Agent, which acts on behalf of the user.

As several transaction scenarios are possible, allowing users to generate Broker level Mobile Agents with different behavioral characteristics increases the flexibility of the system. A human user interacts with the Buyer Agent via a *Buyer GUI* module. In the beginning of a transaction, the user supplies the necessary information (i.e. name, maximum price, required quantity and required delivery date of the product etc.). The buyer GUI allows users to control and monitor the progress of transactions, and to query past transactions from the *History Database*.

Buyer Agent is the main process of the Buyer subsystem. Buyer Agent is a stationary agent that is created during the initialization step of the subsystem. It is responsible for offering an interface to end users for inputting query tasks and communicates with the Broker level Mobile Agents it creates to accomplish its task.

Buyer Agent is created with the basic capability to perform routine and simple tasks. Tasks that are more complicated may involve human instructions but once instructed, the agents cache them for future use. This can be achieved by using learning methodologies on Buyer Agent. The function of a Buyer Agent is to search for product information and to perform goods or services acquisition. When a buyer agent receives a purchase request from a user, it creates a *Mobile Agent* to search for product information and to perform goods or services acquisition in the system. Buyer Agent specifies the criteria for the acquisition of the product and dispatches the BMA to the broker. When this BMA reaches the best deal, it sends a result report to Buyer Agent and this information is added to the History Database.

To generate a Buyer Agent in the Buyer subsystem, the Int_Buyer interface shown in Figure 10 has to be implemented to provide uniformity and scalability.

```
public interface Int_Buyer extends Remote
{
public byte[] downloadClass(String file_name)
        throws RemoteException;
// For Downloading necessary class by Broker

public void notify(ResultReport res_rep)
        throws RemoteException;
// For getting result reports from the BMAs
}
```

Figure 10. Source code of Int_Buyer interface

There are two methods in the interface. The "downloadClass" method is used to download the classes necessary for the execution of Broker level Mobile Agent (BMA) and/or Supplier level Mobile Agent (SMA). As we can only use the dynamic class loading feature of Java RMI, sometimes we need to use these classes on the local machine (i.e. our agent can use a different object in its execution,S if we want to send this agent to another supplier). The "notify" method is used to get Result Reports from BMA.

```
public interface Int_BMAgent extends Remote
{
public void run() throws RemoteException;
//Get published data (request) from the Buyer Agent

public String getName() throws RemoteException;
//Gets subscription information from the Suppliers

public String getOwner() throws RemoteException;
//Gets unsubscribe request from Suppliers

public String[] getClassNames()
        throws RemoteException;
//For Downloading necessary class from Suppliers

public ResultSet searchKBase(String sql)
        throws RemoteException;
//For Downloading necessary class from Suppliers
}
```

Figure 11. Source code of Int_BMAgent interface

To create a BMA Buyer Agent has to implement the Int_BMAgent interface, as depicted in Figure 11.

BMA also implements a “Runnable” interface. Therefore, we can run it as a “thread”, concurrently with other BMAs and Broker Manager processes. In the “run” method, Buyer Agent has to define the task of the BMA and how it can perform this task. The “getName” method returns the unique name of the BMA and “getOwner” method returns the address of the (Owner) Buyer Agent. The “getClassNames” method returns the necessary class names, which should be loaded by Broker Agent and/or Supplier Agent. These classes are loaded through a call to the “downloadClass” method of Buyer Agent. The “searchKBase” method is used for selection of Suppliers from the KnowledgeBase of the Broker.

4.2 Supplier Subsystem

A supplier has to initialize a Supplier Subsystem on its machine to join the system. When a supplier system is created for the first time, it subscribes to the system providing its address and the names of its products. If a supplier starts or ends delivering a product, it again subscribes or unsubscribes its productS respectively.

Every supplier agent has to know the address of the Broker so that it can make a connection. Supplier agent subscribes to the broker by sending its product definitions and waits for buyers to make requests for his products.

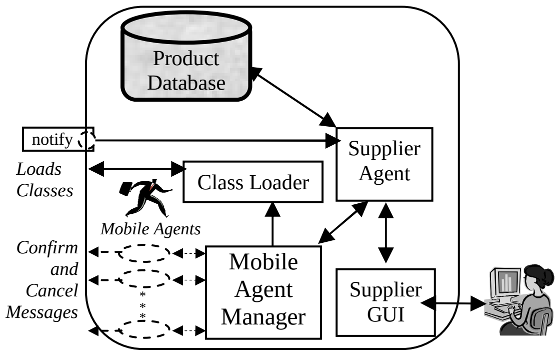

Figure 12. Supplier Subsystem Architecture

The Supplier Subsystem includes five main components as shown in Figure 12.

a. Product Database contains the services and product details of the supplier.
b. Graphical User Interface (GUI) which is used for interaction (i.e. inquiring about past and current transactions from the agent, or a user may also specify selling strategies through this module) with human users.
c. Mobile Agent Manager controls and coordinates the incoming SMAs.
d. Class Loader downloads the necessary classes from the Broker.
e. Supplier Agent acts on behalf of the user.

Supplier Agent is a stationary agent, which is responsible for offering an interface to end users to enter information, control the operations, and input query tasks. Supplier Agent processes purchase orders from buyer agents and decides how to execute transactions according to selling strategies specified by the user. Since organizations differ in the products they sell, a supplier agent should be customized before it is placed online.

```
public interface Int_Supplier extends Remote
{
public void Notify(SMAgent sma)
        throws RemoteException;
// For getting notification(Supplier Mobile Agent)
// from a Broker level Mobile Agent
}
```

Figure 13. Source code of Int_Supplier interface

Supplier Agent implements the Int_Supplier interface depicted in Figure 13, which contains a single method used for getting SMAs from the Broker.

Each SMA on the supplier side can read access the Product Database according to its interests. It determines whether the required quantity is already in the inventory and thus available to offer, and then makes negotiations (if necessary). If so, the supplier agent gives an immediate quotation to the BMA and sends a result message.

If a BMA cancels a reservation, the supplier agent may impose a reservation fee for the quantities of reservation reserved and bill to the buyer 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 therefore reservation fees should be negligibly small.

4.3 Dispatch Service

The Dispatch Service plays an important role in cyberspace. It is a logically (also physically in our system) centralized party which mediates between buyers and suppliers in a marketplace. The main component of the Dispatch Service is the “broker”. Broker is useful when a marketplace has a number of buyers and suppliers, when the search cost is relatively high or when trust services are necessary. The inner structure of Broker is shown in Figure 14.

The client-server paradigm is often not sufficient to solve satisfactorily the problems that may arise during the system life cycle. Often one wants to add a supplier or a buyer to the e-commerce system after its design-time. In the system that we present, the broker implements the publish/subscribe paradigm in which purchase events are published and made available to the supplier components of the system through notifications.

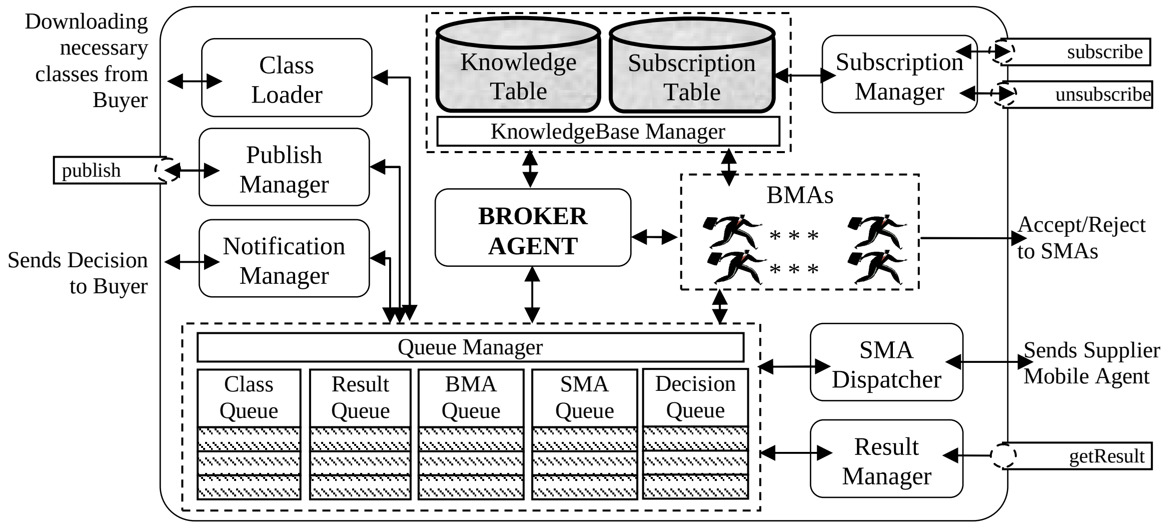

Figure 14. Inner structure of the Broker

Two important tables can be manipulated by KnowledgeBase Manager of the Broker.

a. Knowledge Table keeps statistical information about suppliers, buyers and products in the system, according to message transactions between buyer and supplier agents.
b. Subscription Table keeps simple information, such as names of services or products which are delivered by suppliers. It also contains supplier id and password pairs for authorization of suppliers.

The *Queue Manager* has access to five major queues in the Broker. These queues are used for execution of manager modules concurrently.

a. *Class Queue* contains the names and addresses of the classes that are required to be downloaded by the BMAs.
b. *BMA Queue* contains the serialized form of incoming BMAs from Buyers.
c. *SMA Queue* contains the serialized form of outgoing SMAs to Suppliers
d. *Result Queue* contains incoming results from the SMAs.
e. *Decision Queue* contains the decision data of a BMA, that is to be sent to Buyer Agent.

The broker consists of seven manager modules, which run in parallel in the system. These are *Class Loader, Publish Manager, Notification Manager, Subscription Manager, SMA Dispatcher, Result Manager and Broker Agent.* The tasks of these managers and the execution flow of a procurement process in the Broker is as follows:

a. Subscription Manager gets subscription messages from suppliers via *subscribe* and *unsubscribe* methods, verifies their supplier_id and passwords with the Subscription Table, and continues with the necessary actions.
b. Publish Manager receives the serialized forms of incoming BMAs from the Buyer Agent via publish method and adds them to the BMA Queue. It also gets names and addresses of the necessary class via the same method and adds them to the Class Queue.
c. *Class Loader* downloads classes in the Class Queue from Buyer Subsystem through a call to the “downloadClass” method of the Buyer Agent.
d. Broker Agent creates and activates a thread for each BMA in the BMA Queue.
e. Each BMA has the authority to read and write both Knowledge Table and Subscription Table. A BMA searches the suppliers’ information in these tables, then creates a SMA for each supplier and thereafter puts the serialized form of each SMA into the SMA Queue.
f. SMA Dispatcher sends each SMA in the SMA Queue to its target supplier.
g. Each SMA returns the result of its search and negotiation activity to the Broker via *getResult* method. These results are received by Result Manager and inserted into the Result Queue.
h. BMAs examine the results in the Result Queue carefully and make a decision. After reaching a decision, a BMA creates a decision report and puts it into the Decision Queue.
i. Notification Manager Sends each decision in the Decision Queue to its target Buyer.

Broker Agent is the main component of Dispatch System. It has control over all operations in the system. It evaluates an incoming message and generates lists of target suppliers by using the information in Knowledge Table and Subscription Table.

Broker agent provides a platform for incoming mobile agents to run and create SMAs and dispatching them to the necessary suppliers. It also supports the registering and dispatching operations in accordance with publish/subscribe paradigm.

```
public interface Int_Broker extends Remote
{
public void publish(String address, SMAgent sma,
     String[] classNames) throws RemoteException;
//Get published data (request) from the Buyer Agent

public void subscribe(String sup_address,
        String sup_id, String pass,
        Subscription sub) throws RemoteException;
//Gets subscription information from the Suppliers

public void unsubscribe(String sup_id, String pass,
        Subscription sub) throws RemoteException;
//Gets unsubscribe request from Suppliers

public byte[] downloadClass(String file_name)
        throws RemoteException;
//For Downloading necessary classes for Suppliers
}
```

Figure 15. Source code of Int_Broker interface

A Broker Agent has to implement Int_Broker interface depicted in Figure 15. There are four main methods in the Int_Broker interface. The “publish” method is used to get Broker level Mobile Agent from Buyer agent with address of Buyer Agent. The “subscribe” method is used for subscription of Suppliers with address, unique supplier id, password and subscription (contains product definitions) information. “unsubscribe” method is used for unsubscription operation of the suppliers with supplier id, password and subscription information. The “downloadClass” method is used to download classes necessary for execution of Supplier level Mobile Agent.

```
public interface Int_SMAgent extends Remote
{
public void run() throws RemoteException;
//Get published data (request) from the Buyer Agent

public String[] getClassNames()
        throws RemoteException;
//Gets subscription information from the Suppliers

public ResultSet searchProductDB(String sql)
        throws RemoteException;
// Gets unsubscribe request from Suppliers
}
```

Figure 16. Source code of Int_SMAgent interface

SMAs are created by BMAs in the Broker side of the system. To create a SMA, a BMA has to implement the Int_SMAgent interface, depicted in Figure 16. SMA implements a “Runnable” interface therefore we can run it as a “thread”, concurrently with Supplier Manager processes. In the “run” method, BMA has to define the function of BMA and how it can perform this task. The “getClassNames” method returns the necessary class names, which should be downloaded by Supplier Agent. These classes are downloaded by calling the “downloadClass” of the Broker Agent. The “searchProductDB” method is used to select products from the Product Database of the Supplier. Our system includes only one Broker. It is possible to enhance the system by allowing several Brokers to be present (after some minor modifications in he Broker Agent structure), connecting them with different topologies and thus increase the scalability of the system. This approach is included in our current studies.

4.4 Stages of a Mobile Agent

When a mobile agent (Broker level Mobile Agent or Supplier level Mobile Agent) is sent to a remote host to accomplish a specified task, the whole process can typically be decomposed as follows:

4.4.1. Distributing Agents.

In this stage, the master agent (Buyer Agent or Broker level Mobile Agent) should first create the mobile agent (BMA or SMA) which will be sent. During its creating process, some arguments and behavioural parameters are encapsulated into the mobile agent, including its task and the address of the master agent to return the results. The code for accomplishing the task should also be included in the mobile agent. After the mobile agent has been created, the master agent will dispatch it to the remote host (broker or supplier) by using RMI messaging. Generally, the time for this stage depends on the bandwidth, traffic state of the network, the size of the mobile agent. The dispatch process is mainly a network-dependent job.

4.4.2. Completing tasks.

If the dispatched mobile agent successfully reaches the remote host, it begins to execute and to access local data to accomplish its task. Due to the characteristics of the task, the mobile agent can communicate with local stationary agents, such as Broker Agent or Supplier Agent, or access local data, such as files or database, directly.

Since the mobile agent approach is well suitable for deploying parallel processing over distributed data resources, a mobile agent can be assigned a simple task so that it has a small size and can visit only one remote host to accomplish its task. A mobile agent can also be assigned a set of tasks that should be accomplished by visiting a set of remote hosts. If these tasks are semantically dependent and should only be finished in a specific order, dispatching one mobile agent is essential and good enough that it can migrate in an itinerary pattern. Otherwise, if these tasks are semantically independent and the number of remote hosts that should be visited is large, these tasks should be distributed to multiple mobile agents so that each mobile agent has only one relatively simple task that it will not take a long time to accomplish it. Thus, the master agent can get all the results in a short time since these dispatched agents can execute in parallel over different processors. In this case, as in our system the end user can easily get a large set of quotations for his/her desired products in a very short time.

4.4.3. Reporting Results.

When a BMA or SMA has accomplished its task, it should return the results. It can either dispatch itself to the origin host carrying its results or send the results back through a message. The latter way can be faster since the former way should send back both the results and the code of the mobile agent. Therefore we prefer the second one. This approach is necessary when the network is partially connected or the connection is dynamically changed, where the autonomous migration of a mobile agent can help to choose different route for returning.

5 Conclusions and Directions for Future Work

In this paper, we have introduced a general framework for a Two-Leveled Mobile Agent based E- Commerce systems including mechanism for a communication infrastructure based on publish/subscribe paradigm. The system relies on the utilization of mobile agents as mediators between buyers and suppliers. The publish/subscribe protocol allows participants to join and leave the system dynamically, extending the flexibility and adaptability of the system. By using a two-leveled agent model, we have also made use parallel computation to enhance performance. This is especially important as a larger number of suppliers can be searched concurrently in a shorter time to provide buyers with better choices in their decision-making. As future work, we have plans to change the API to conform to a standard Agent Communication Language (ACL) like KQML[23] or FIPA ACL[3].

References

[1] Guanghao Yan, Wee-Keong Ng and Ee-Peng Lim. Toolkits for a Distributed, Agent Based Web Commerce System, In Proceedings of the International IFIP Working Conference on Trends in Distributed Systems for Electronic Commerce (TrEC '98), Hamburg, Germany, July 1998.

[2] Agentcities Web, http://www.agentcities.org.

[3] Foundation for Intelligent Physical Agents, http://www.fipa.org.

[4] T. D. Rodrigo and A. Stanski, “The evolving future of agent-based electronic commerce,” in Electronic Commerce: Opportunity and Challenges, eds. S.M. Rahman and M. S. Raisinghani, Idea Group Publishing: Hershey, USA, 2000, pp. 337–351.

[5] James W. Stamos and David K. Gifford. Remote Evaluation. ACM Transaction on Programming Languages and Systems, 14(4):537-565, October 1990

[6] J. Baumann, F. Hohl, K. Rothermel and M. Straber, Mole - Concepts of a Mobile Agent System. The World Wide Web Journal, 1, 3, pp 123-137, 1998

[7] G. Cugola, C. Ghezzi, G. Picco, G. Vigna, "Analyzing Mobile Code Languages", Mobile Object Systems, Lecture Notes in Computer Science, No. 1222, Springer-Verlag (D), pp. 94-109, February 1997.

[8] M. Hohlfeld and B. Yee. How to Migrate Agents. Available at http://www.cs.ucsd.edu/~bsy

[9] Lange, D.; Oshima, M.: "Mobile Agents with Java: The Aglet API." In "Special issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents." Baltzer Science Publishers, 1998.

[10] J.E.White. Telescript Technology.: The Foundation for the Electronic Marketplace. White paper, General Magic, Inc., Mountain View, CA, 1994

[11] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The LOCUS distributed operating system. Proceedings Ninth Symposium on Operating Systems Principles, Bretton Woods, New Hampshire, October 1983, pages 49-70.

[12] Y. Wang and K. L. Tan. "A Study of Building Internet Marketplaces on the Basis of Mobile Agents for Parallel Processing", World Wide Web: Internet and Web Information Systems, 5, 41–66, 2002

[13] R. Guttman, A. Moukas and P. Maes, “Agent-mediated electronic commerce: A survey," Knowledge Engineering Review, vol. 13, no. 2 (June 1998) 147-159.

[14] AuctionBot URL : http://auction.eecs.umich.edu.

[15] P. R. Wurman, M. P. Wellman and W. E. Walsh, "The Michigan Internet AuctionBot: A configurable auction server for human and software agents," Proceedings of the Second International Conference on Autonomous Agents, Minneapolis, MN (May 1998), pp. 01-308.

[16] A. Chavez and P. Maes, “Kasbah: An agent marketplace for buying and selling goods," Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK (April 1996), pp. 75-90.

[17] M. Tsvetovatyy, B. Mobasher, M. Gini and Z. Wieckowski, "MAGMA: An agent-based virtual market for electronic commerce," Applied Artificial Intelligence, vol. 11, no. 6 (September 1997), pp. 501-523.

[18] MAGNET: Mobile Agents for Networked Electronic Trading. http://alpha.ece.ucsb.edu/ ~pdg/magnet/

[19] Kowalczyk R. and Bui V. “On Constraint-based Reasoning in e-Negotiation Agents”. In F. Dignum and U. Cortes (Eds.) Agent Mediated Electronic Commerce III, LNAI (2000), Springer-Verlag, pp. 31-46.

[20] Ryszard Kowalczyk, Van Anh Bui (2000). “On Fuzzy e-Negotiation Agents: Autonomous negotiation with incomplete and imprecise information”. DEXA Workshop on e-Negotiation, UK, 2000.

[21] S. Papastavrou, G. Samaras, and E. Pitoura, “Mobile agents for WWW distributed database access,” in Proceedings of 15th International Conference on Data Engineering (ICDE’99), Sydney, Australia, March 23–26, 1999, pp. 228–237.

[22] Y. Wang, K. C. K. Law, and K. L. Tan, “A mobile agent based protocol for distributed databases access,” in Proceedings of 2000 IEEE International Conference on Systems, Man, and Cybernetics (SMC’2000), Nashville, TN, 8–11 October 2000, pp. 2028–2033.

[23] Finn, T., Labrou, Y. and Mayfield, J. “KQML as an agent communication language. In: Software Agents.”

[24] Bradshaw, J. (Ed.) AAAI Press/MIT Press. ISBN 0-262-52234-9 (1997).

BIOGRAPHY

Ozgur Koray SAHINGOZ

Ozgur Koray SAHINGOZ is a Research Assistant in Computer Engineering Department of Air Force Academy, Istanbul. He received BSc degree from Computer Engineering Department of Bosphorus University, Istanbul, and MSc degree from Computer Engineering Department of Istanbul Technical University, where he is currently working for his PhD degree. His research interests lie in the areas of object oriented programming, artificial intelligence, parallel and distributed computation, and e-commerce.

Nadia ERDOGAN

Nadia ERDOGAN is an associate professor at Computer Engineering Department of Istanbul Technical University. She received BSc and MSc degrees from Electrical Engineering and Computer Engineering Departments of Bosphorus University, Istanbul, respectively, and PhD degree from Computer Engineering Department of Istanbul Technical University. Her current research interests are in the areas of parallel and distributed systems, parallel programming, object oriented programming and distributed multi-agent systems.