# A Comparative Study of Mobile Agent and Client-Server Technologies in a Real Application

R. B. PATEL
Department of Electronics & Computer Engg.
IIT Roorkee, Roorkee-247667,
Uttaranchal, India,
+91+9412978806
Email: patrbdec@iitr.ernet.in

K. GARG, Senior Member IEEE
Department of Electronics & Computer Engg.
IIT Roorkee, Roorkee-247667,
Uttaranchal, India,
+91+1332-285649

## ABSTRACT

The anticipated increase in popular use of the Internet has created more opportunity in information dissemination, E-commerce, and multimedia communication. It has also created more challenges in organizing information and facilitating its efficient retrieval. From the network perspective, there are additional challenges and problems in meeting bandwidth requirement, as in network management. In response to this, new techniques, languages and paradigms have evolved which facilitate the creation of such applications. Certainly the most promising among the new paradigms is the use of **mobile agents.** In this paper, mobile agent and client-server technologies are applied in an E-commerce application and a comparative study is discussed. We report an implementation of a banking system, whose database may be distributed at different sites on the Internet. A customer can send his/her mobile agent to perform various tasks involved in banking and get back an appropriate result or can use client-server methodology to perform the same task. PMADE is used as the platform to develop these mobile agents. Security checks have been also implemented, as this is an important requirement in E-commerce.

**Keywords- Mobile Agent, Bank Agents, E-commerce**

## 1. INTRODUCTION

Internet technologies are rapidly evolving and modifying the way people interact with each other. The increasing number of virtual market places facilitates trading transactions by bringing together a vast number of potential buyers and sellers [9], [10], [14]. In this context, business interactions are moving toward more dynamic and automated solutions and electronic payment methods play a key role for all forms of online business [5]. Despite the growing deployment of e-banking systems that allow some degree of automation [4], Web interfaces or ad hoc tools still require a large degree of human interactions [6]. Transactions and payment orders [14], for instance, can be performed electronically, but only if a human enters the right code or presses the right button in a specific graphical user interface. Customer self-service channels have evolved to more automated and fully integrated business applications that drive products and services to the consumer.

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)
© CSI 2005

In the near future people, will not physically go to their financial organization branches nor logon to the Internet to deal with their banking tasks. They will delegate the management of their bank accounts, paycheck, investments, insurances, mortgages, loans and credits to their personal electronic financial assistants [7]. The software entities acting on behalf of humans and/or service providers will automate several electronic business and commercial activities such as service advertisement, market trend monitoring, services pricing and negotiation [8].

Applications that need to monitor events on remote hosts, such as, whether a particular bank account's balance has fallen below a threshold, are greatly benefited from MAs, since agents need not use the network for polling. Instead of periodically downloading bank statements, an agent can be sent to quote service to monitor the balance. The agent can inform the user when a specified event occurs.

The expression 'banking services' refers to the set of processes and mechanisms required for enabling agents to make / receive side payments for creating / maintaining / closing bank accounts. While current bank services are very simple and still require extensive testing, they do demonstrate that it is possible to represent a first step towards developing the robust services that would be required to support an effective agent economy. Online businesses, and in particular e-banking, require complex interactions between diverse systems owned by different organizations or individuals.

When building business systems relying on mobile agent technology, the notion of trust has to be redefined for building an appropriate secure framework. This is particularly crucial for bank- specific services, since authentication, non-repudiation, privacy and confidentiality represent intrinsic requirements that need to be satisfied. It is possible to envisage two main levels at which mechanisms are needed.

(1) Authentication of an agent that is willing to access the bank. The banks may define specific policies to be followed when interacting with the bank or with other agents making use of banking services. For instance, in order to access the bank, agents may need to be identified as regular customers or they may be required to introduce themselves and enter specific information.

(2) Specific security mechanism/policies at the level of every different service offered within the bank. For instance, if an agent admitted to the bank wants to know the amount of money in a specific account, the

bank will verify if the agent has the rights to access this information.

This paper presents an implementation of a MA-based banking system whose databases may be distributed at different sites on the Internet, i.e., a virtual organization offering banking services to agents accessing the agent based market place. Customers can send their mobile agents (MAs) [15] to perform various tasks and get back appropriate results. Our aim is to define an open and distributed framework to create an online network in which autonomous and heterogeneous agents can supply and/or provide a variety of services, i.e., integration of agent infrastructures with existing non-agent based environments such as databases, legacy systems, various tools, etc. In this context, the proposed model provides a generic way for agents to make payments to one another. PMADE is used as the development platform which has been developed at IIT Roorkee [1].
The rest of the paper is organized as follows: Section 2 gives an overview of PMADE. Section 3 presents the Architecture and Design of the Banking System. Section 4 discusses an implementation and performance study of the developed system and Section 5 concludes the paper.

## 2. OVERVIEW OF PMADE

Figure 1 shows the basic block diagram of PMADE. Each node of the network has an **Agent Host (AH),** which is responsible for accepting and executing incoming agents. A client called Agent Submitter (AS) [1], submits the agent on behalf of the user to the AH.

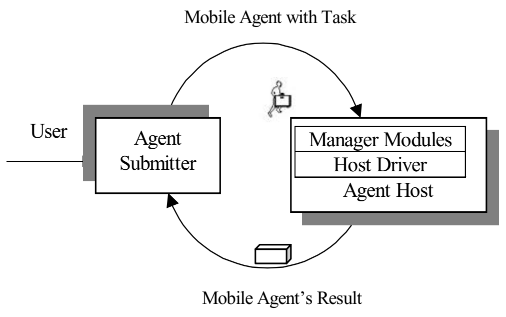


**Figure 1. Block Architecture of PMADE Model**

A user, who wants to perform a task, submits the MA designed to perform that task, to the AS on the user system. The AS tries to establish a connection with the specified AH, where the user already holds an account. If the connection is established, the AS submits the MA to it and then goes offline. The AH examines the nature of the received agent and executes it. The execution of the agent depends on its nature and state. The agent can be transferred from one AH to another whenever required. On completion of execution, the agent submits its results to the AH, which in turn stores the results until the remote AS retrieves them for the user. The AH is the key component of PMADE. It consists of the manager modules and the Host Driver. The Host Driver lies at the base of the PMADE architecture and the manager modules reside above it. It is the basic utility module responsible for driving the AH by ensuring proper co-ordination between various managers and making them work in tandem. Details of the various managers and their functions are provided in [3]. PMADE provides weak mobility to its agents. One-hop, two-hop and multihop agents can be programmed on PMADE [2].

## 3. ARCHITECTURE AND DESIGN OF THE BANKING SYSTEM

In our system the customer (client) dispatches one or more MAs, each with its list of required modes of transaction, amount and potential accounts/banks. The MA visits each bank server in turn to perform the required transaction. If the desired account is present, it processes the transaction on behalf of the user, or it moves to other bank servers. If the transaction is committed, or the bank list is exhausted, the MA returns to the customer with the details of the operations performed.
Three types of agents are implemented. They differ mainly in the different roles they can cover and/or services they can offer during trading transactions.

(a) **Bank Agent (BA)** which acts on behalf of the banking organization. It offers two kind of services:
   (i) Account management service that includes open account, close account and list account information operations.
   (ii) Electronic payment service in which the transfer of funds between two accounts is performed. This entity can therefore be considered as the agent interface of the banking system for the organization toward the external world.

(b) **Customer Agent (CA)** which can be considered as a personal assistant that acts on the behalf of end users and uses the services offered in the agent enabled market, and

(c) **Insurance Agent (IA)** which represents an insurance business offering, mainly, to sell an insurance policy. Credit card based payment is considered.

The last two entities represent agents making use of banking services, i.e., specific possible customers of the banking organization. In the following, we list the scenarios that are developed for defining and verifying the specific mechanisms needed and offered by the different types of agents depicted above. Note that in order to facilitate task decomposition, CAs act mainly as buyers and IAs act essentially as sellers.

**Opening an account:** When CA-X wants to open an account within a given bank it requests BA-Y to open an account including in the request message the information needed. BA-Y will send back to CA-X the result of his demand.

**Closing an account:** When CA-X wants to close an account within a given bank it requests BA-Y to close its account. BA-Y will then verify if the given account belongs to the agent CA-X, and after having performed the required action, BA-Y will send back to CA-X the corresponding action's result.

**Getting information from an account:** When CA-X asks for accounts information, BA-Y receives the query and verifies the ownership of the account and the result is sent back to CA-X. The message is either an error notification or the requested information. If the account identifier is not specified, the query is interpreted as a query for information about the full list of accounts owned by CA-X.

**Credit card based payment:** CA-X stipulates an insurance policy with agent IA-LIC (Insurance Agent with a Life Insurance Company) for an amount $x$. CA-X sends its credit card details to IA-LIC that requests the bank server to transfer amount $x$ from CA-X's account to IA-LIC's account. Both CA-X and IA-LIC have accounts with BA-Y (i.e., the same bank agent managing both accounts) and are informed about the transaction.

The next section gives details of our e-banking application, the architecture of which is shown in Figure 2.

Deadlocks are avoided by making all operations independent of others and having a single control on a tuple. No cyclic dependency occurs and no rollback is required. The MA may be terminated if it stays for a long period on the server or it may itself migrate to another server if it does not get access to the databases.
We have identified four tables in the required database and their design is given in the following sections:

**Ledger Book**: which contains information about every account holder. An entry is made in it whenever a new account is opened in the bank. It is searched by either *Credit Card ID* or by *Account Holder Signature*. The Bank Administrator generates these automatically whenever a new entry is made. A MA is authenticated by these fields before it is forwarded for further operation.

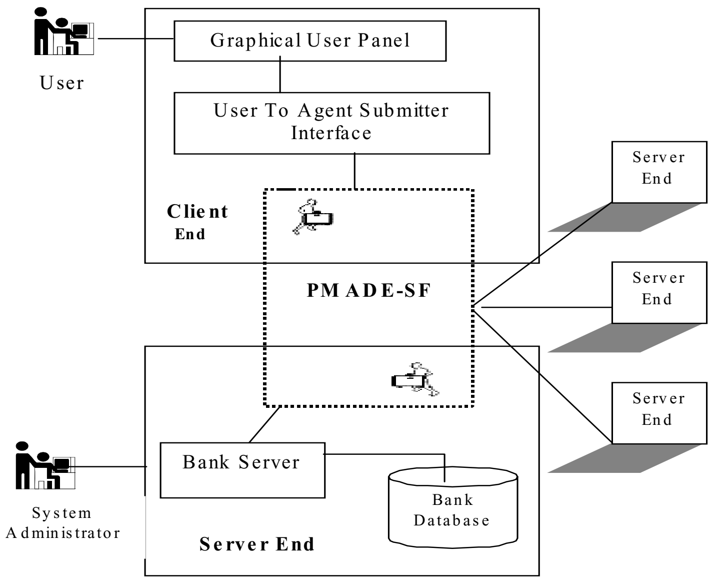


Figure 2. Architecture of e-banking Application

## 3.1 Distributed Database

A distributed database is required to hold all the data pertaining to the banking system, like details of customers, transactions completed, accounts, etc. It may be distributed by fragmentation, or replication, or both. The designed application supports all three types of distributed databases and uses a relational database model.
All updates are made in parallel in all parts of the database. Once a tuple of the database is opened by an entity (MA), no one else can access it (for updates, but reading is permissible) until this entity has released it. Thus at any time, a tuple is under control of only one MA and update operation is done in buffer and its contents periodically incorporated into the database. This feature facilitates parallel operation, i.e., multiple agents (from different clients) can access same database in parallel. This feature supports concurrency control in the database.

**Balance Book**: has an entry for each account holder and contains information about the present balance in it. It is searched and updated by *Credit Card ID* and *Account Holder Signature*. It is updated whenever a transaction takes place, after authentication from the Ledger Book.

**Debit Book**: If the MA comes for debit, a detailed entry is made in this table about that transaction. As a customer can make several such debits, it can have several entries corresponding to each transaction.
**Credit Book**: If the MA comes for credit, a detailed entry is made in this table. As a customer can make several such credits, it can have several entries corresponding to each transaction.
*Credit* and *Debit book* can be used to generate a monthly statement of all the transactions that the customer has made. Also the expenditure pattern of a customer can be studied and special packages may be offered to him. This is an important feature in e-commerce. These records help in back tracking the flow of money in case of any problems later.

## 3.2 Bank Server

This is the central module of the application. All the functionality of the bank resides here. All MAs report to this module, which performs the actual operation such as authentication and transaction on a database and returns an appropriate message to the agent. The architecture is illustrated in Figure 3.

The **Bank administrator** creates new accounts and can check and manipulate the user's account. The customer has to go physically to the server who incorporates this program and ask to open a new account. This module has to be hooked up with the mobile agent system (MAS). A Graphical User Interface (GUI) has been designed. The administrator uses the services of the following:

**New-Account/Customer Information Module**: This module provides services for creating an account, checking and updating the information pertaining to the customer of an existing account. This module returns a *Credit Card ID* and *Signature* for a new account. *Credit Card ID* number is an *AutoNumber* generated by the database. *Signature* is a four-character word randomly generated by the bank administrator using 26 (small letters), 26 (capital letters) alphabets and 10 digits. Thus, we can have more than one million unique signatures.

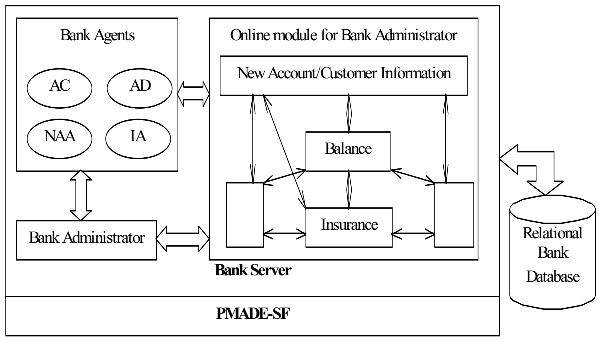


**Figure 3. Architecture of Bank Server**

Minimum opening balance and guarantor criteria has been embedded here, a violation of which gives an appropriate message. A guarantor is required for every new customer is seeking an account. This guarantor should be an existing account holder. His/her *Credit Card ID*, *Name* and *Signature* are also authenticated. Initially, when the database is created, an entry is made in the name of the *Bank* in the *Ledger Book*.
The GUI form for this sub module has thirteen fields for accepting various information from the customer, like his/her full address, opening balance, details of guarantor, etc. Of these thirteen, the customer has to fill eleven fields and two are for *Credit Card ID*, and *Signature*. The module itself displays these when the form has been submitted. All this information is written in the Ledger Book.
We can also search and update information of existing accounts. The GUI is the same as above. First, an account's information has to be searched by given a signature of that account. After matching this signature with the existing one, information is displayed in the same form. Then one can update the information and resubmit the form. It should be noted that Credit Card ID, and signature field cannot be updated, as the bank allots them.

**Balance module** accepts the *Credit Card ID* and *Account Holder Signature* of an existing account and returns the balance in that account from the *Balance Book.* The entire procedure takes place via a form. A GUI has also been provided so that a customer, who has failed to submit a MA, can himself/herself do a transaction.

**Credit and Debit modules**: These modules perform the actual credit/debit into the account. All the above modules display appropriate messages after each operation, informing either successful transaction execution or stating reasons for failure. A GUI with a Help facility, has been designed for ease of use of these modules.

**Insurance Module:** This module supports in the management of insurance policies of the customers.

## 3.3 Bank Agents

The Bank Server has four **stationary** agents called the **Authenticator Credit (AC), Authenticator Debit (AD), New Account Agent (NAA)** and **Insurance Agent (IA),** which provide second level security to the bank database after the authentication**,** by PMADE itself. AC is the stationary agent and helps the customers' agents to perform transactions associated with them. AD provides a similar facility to the customers' agents which want to debit their account. NAA helps in adding customers to the bank while IA supports the management of customer insurance policies.
A customer MA uses the services of these bank agents to perform the required transactions by furnishing the required parameters. Customer MAs do not themselves fetch, search or update the bank database. This is done for enhancing the security of the system and prevents an authorized customer from creating a malicious agent who can manipulate his/her account. The stationary agents perform the actual transaction and inform the customer MAs whether the transaction was successfully completed or reasons for its failure.
It is important to note that the database is opened in read mode only. An agent may be required to find only the balance from an account. Hence, it has direct access to the database. This is done because a third party may want to check the account of a customer. This feature is necessary in a buyers-sellers problem in e-commerce, the seller would first have to check the buyer's account before delivering commodities to him.

## 3.4 Customer Agents (Mobile Agents)

The standard GUI provided by the AS interface is shown in Figure 4. It accepts sixteen parameters: *Login Name, Password, Agent Name* (MASIF standard [13]), *Agent Version, Method Name* (class file name of agent), *Argument List, Itinerary Pattern type* (S: Serial, VS: Virtual Serial, P: Parallel) and *Itinerary Address* (List of Host to be visited), *Services & Location* needs to define when additional class files (other files) are required to agent at remote site, *Agent Packing* (Nested, Patel): Mode of sealing agent, i.e., *Distributed Object* Model or nested, *Base Host*: Name of Router of the Network, where

*Trusted Re-router* is installed and *Last Field* is the location where the agent is initially submitted by the agent owner. The first two fields are for loading the authentication certificate of agent owner from the database maintained at AS and required to authenticate the agent owner on the host whose IP address/URL is given in the Itinerary Address field. This authentication is different from the bank's authentication.

Parameters (*Credit, Name, Signature*, etc) for MA are given in *Argument List* field. As there is only one field provided in this GUI, all the parameters are given in this field separated by '|'. This field is given to the MA, whose name is specified in the *Agent Name* field. Therefore, all the MAs have code for parsing this field to obtain the actual parameters.

A single agent is sufficient for performing the same task at different banks (single code multiple data). When the agent reaches an AH, it clones itself and sends these to all the places where a bank server is running. Hence all the databases, where the customer account exists, are updated simultaneously, allowing concurrency control. This feature makes handling of distributed databases possible. We have identified several MAs for our application. They are:

1. **BFindAgent**: This agent is used to enquire balance in an account. It accepts three parameters, namely, *CID No., Name* and *Signature* from the user. This agent has code which directly accesses the balance table of the bank database. After confirming the *CID No.* and *Signature*, it reads the amount field of the table, stores the result in a variable, and notifies it to the present AH. The host then sends this value to the user.

2. **AgentDebit**: This agent is used for debiting an account. It accepts four parameters *CID No*., *Name*, *Signature* and *Amount. Amount* is the value by which the account has to be debited. On reaching the bank server it calls the AD with these parameters. The AD then performs the actual task of debit and notifies the MA with the appropriate result about either a successful debit or reasons for failure.
3. **AgentCredit**: This agent is used for crediting an account. It accepts seven parameters as follows: *CID No., Name, Signature, Amount, Cheque/DD Number, Cheque/DD Date* and *issuing bank*. After reaching the bank server it calls the AC with these parameters. The AC then performs the actual task of credit and notifies the MA with the appropriate result about either a successful credit or reasons for failing of the transaction.
4. **NewAccountAgent:** The functionality of this agent is the same as New Account/Customer Information module discussed earlier.
5. **QueryAgent** is a general agent, used to find the different schemes launched by banks from time-to-time. It needs no authentication on the bank server because it is allowed to only read the general database, which is open for the public.

## 4. IMPLEMENTATION AND PERFORMANCE STUDY

The database is made in Oracle and the driver used to run it, is Microsoft Access Driver of ODBC (Object Database Connectivity). JAVA has its own database driver called JDBC (Java Database Connectivity). The interface between these two has been provided by JDBC-ODBC Protocol. All queries are made in SQL (Structured Query Language).

Users can interact with PMADE by developing agent application programs which are implemented as Java objects. Users would first need to write a Java class that specifies some action, such as accessing a database on a remote host. Once this Java class is written and compiled, the user can launch the agent program in three ways: (1) via a GUI Agent Launch Wizard, (2) via a command line tool, or (3) using the external API (Agent Programming Interface). The first two mechanisms are provided with PMADE, while the last one requires the user to write a customized launch class, which makes use of PMADE class libraries.

We have implemented the banking application using the following mechanisms:

1. **Serial or Parallel Client-Server** [12]**:** This is based on the traditional client-server paradigm. The client queries and receives replies from each server sequentially or in parallel.
2. **Serial or Parallel MA:** A **single** multihop MA moves from the client to each server sequentially or in parallel to process the information.

The CS implementation consists of a bank server that sends an information brochure on request, from a multithreaded client. The client and the bank server have been implemented using Java RMI [11]. We used various application parameters that influence performance, such as, size of CS messages, size of the MA, number of remote information sources, etc, and performed experiments to study their effect on performance. We used **trip time**, i.e., time elapsed between a user initiating a request and receiving the results, as the metric for performance comparison. This includes the time taken for agent creation, time taken to visit/collect account details and processing time to extract the required information. We have performed experiments to determine:

(a) **The effect of data size on trip time:** The processing delay at the bank server was kept constant and information brochure sizes of 100KB, 200KB, 500KB and 1 MB were used. This was done for different enquiries from 1 to 52 bank servers.

(b) **The effect of bank server processing delay on trip time:** The information brochure size was kept constant at 500KB and the bank server processing time for servicing each request was varied from 10ms to 500ms. The trip time was measured for different enquiries from 1 to 52 bank servers.

Results are shown in Figs. 5 to 7, from which the following observations can be made:

- The performance of the MA based application remains the same for different data sizes while the performance of the CS based application degrades with increase in data size.
- CS implementations perform better than MA implementations for data sizes less than 100 KB.
- MA performs better than CS when the data size is greater than 200KB and number of banks to visit is greater than 6.

- MA performs better than CS for all mobility patterns, for small processing delays (10 ms) and large (500KB) data size.
- Parallel implementations perform better than serial implementations, when the number of banks to visit is greater than 6 and the processing delay is greater than or equal to 500ms.
- Parallel MA performs better than parallel CS for higher processing delays (500 ms) and large (500KB) information brochure size.
- Performance crossover points, i.e., parameter values for which MA starts performing better than CS implementation can be found for a given set of e-commerce application parameters.

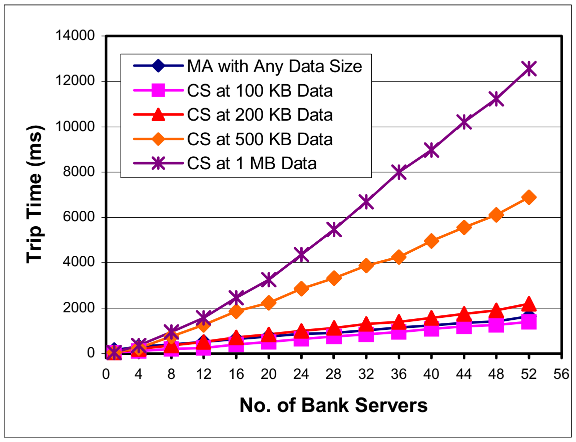


**Figure 5. Effect of data size on trip time in serial MA and serial CS**

## 5. CONCLUSION

Our experiments suggest that CS implementations are suitable for applications where a small amount of information (less than 100 KB) is retrieved from a few remote servers (less than 6), having low processing delays (less than 10ms). However, most real-world e-commerce applications require a large amount of information to be retrieved and significant processing at the server. MA's scale effectively as the size of data to be processed and the number of servers the data is obtained from increases. Scalability being one of the needs of net-centric computing, we find that MAs are an appropriate technology for implementing e-commerce applications.

Parallel implementations are effective when processing delay (greater than 500ms) contributes significantly to trip time. Our experiments also identify performance crossover points for different implementation mobility patterns; this could be used to switch between implementations for performance critical applications. We feel that a complex model employing all the three patterns would result in high performance gain for large-scale distributed applications.

Our experience suggests that mobility patterns play an important role in deciding the implementation strategy to be used for performance critical applications. The selection of a mobility pattern from those feasible for a given application could be based on several criteria such as ease of implementation, performance, availability of technology, etc.

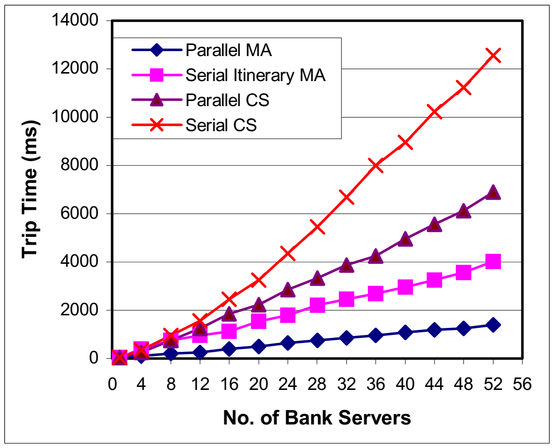


**Figure 6. Trip Time for processing delay of 10ms and information brochure size 500KB**

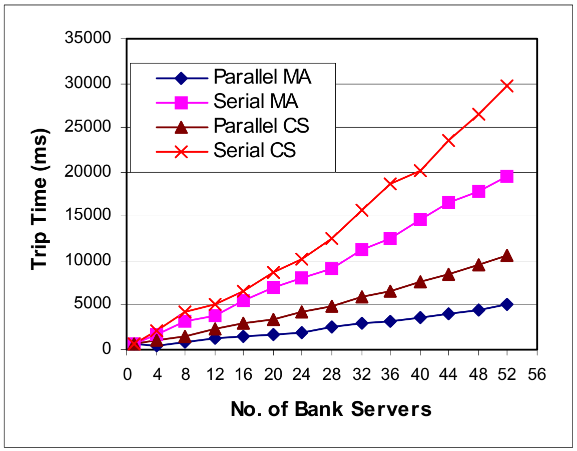


**Figure 7. Trip Time for processing delay of 500ms and information brochure size 500KB**

In future we intend to implement some complex B2B applications using our mobile agents to see how effective the PMADE system is. We would also compare our agent-based banking system with more efficient CS based algorithms for banking.

## REFERENCES

[ 1] Patel, R.B. and Garg, K., "PMADE – A Platform for mobile agent Distribution & Execution," in the Proceedings of 5th World MultiConference on Systemics, Cybernetics and Informatics (SCI2001) and 7th International Conference on Information System Analysis and Synthesis (ISAS 2001), Orlando, Florida, USA, July 22-25, 2001, Vol. IV, pp. 287-293.

[ 2] Patel, R.B. and Garg, K., “Providing Security and Robustness to Mobile Agents on Open Networks,” in Proceedings of the 6th International Conference on Business Information Systems (BIS 2003), Colorado, USA, June 4-6, 2003, pp. 66-74. (Received Best Paper Award).

[ 3] Patel, R.B. and Garg, K., “Mobile Agent Management in PMADE,” in Proceedings of ADCOM 2001, 9th International conference on Advanced Computing and Communications, Bhubaneshwar, India, Dec. 16-19, 2001.

[ 4] Asokan, N., Janson, P.A. Steiner, M. and Waidner, M., “The state of the art in electronic payment systems,” Computer, 30(9): 28–35, 1997.

[ 5] Furche, A. and Wrightsom, G., “Computer Money – A Systematic Overview of Electronic Payment Systems,” DPunkt Verlag, 1996.

[ 6] Xubin He and Qing Yang “Performance Evaluation of Distributed Web Server Architectures under E-Commerce Workloads,” Journal of Parallel and Distributed Computing, 2003.

[ 7] Calisti, M., Deluca, D. and Ladd, A., “An agent-based framework for financial transactions,” in Proceedings of Autonomous Agents 2001 Workshop on Agent-Based Approaches to B2B, May 2001.

[ 8] Nwana, H. S., Rosenschein, J., Sandholm, T., Sierra, C., Maes, P. and Guttmann, R., “Agent-mediated electronic commerce: Issues, challenges and some viewpoints,” in K. P. Sycara and M. Wooldridge, editors, Proceedings of the 2nd International Conference on Autonomous Agents (Agents’98), pages 189–196, New York, May 9–13, 1998. ACM Press.

[ 9] Dasgupta, P., Narasimhan, N., Moser, L.E. and Melliar Smith, P.M., “A Supplier Driven Electronic Marketplace Using Mobile Agents,” First International Conference on Telecommunications and E-commerce, Nashville, TN, Nov. 1998.

[ 10] Dasgupta, P., Narasimhan, N., and Moser, E. L., “MAgNET: Mobile Agents for Networked Electronics Trading,” IEEE Transaction On knowledge and data engineering, 11(4), July/August 1999.

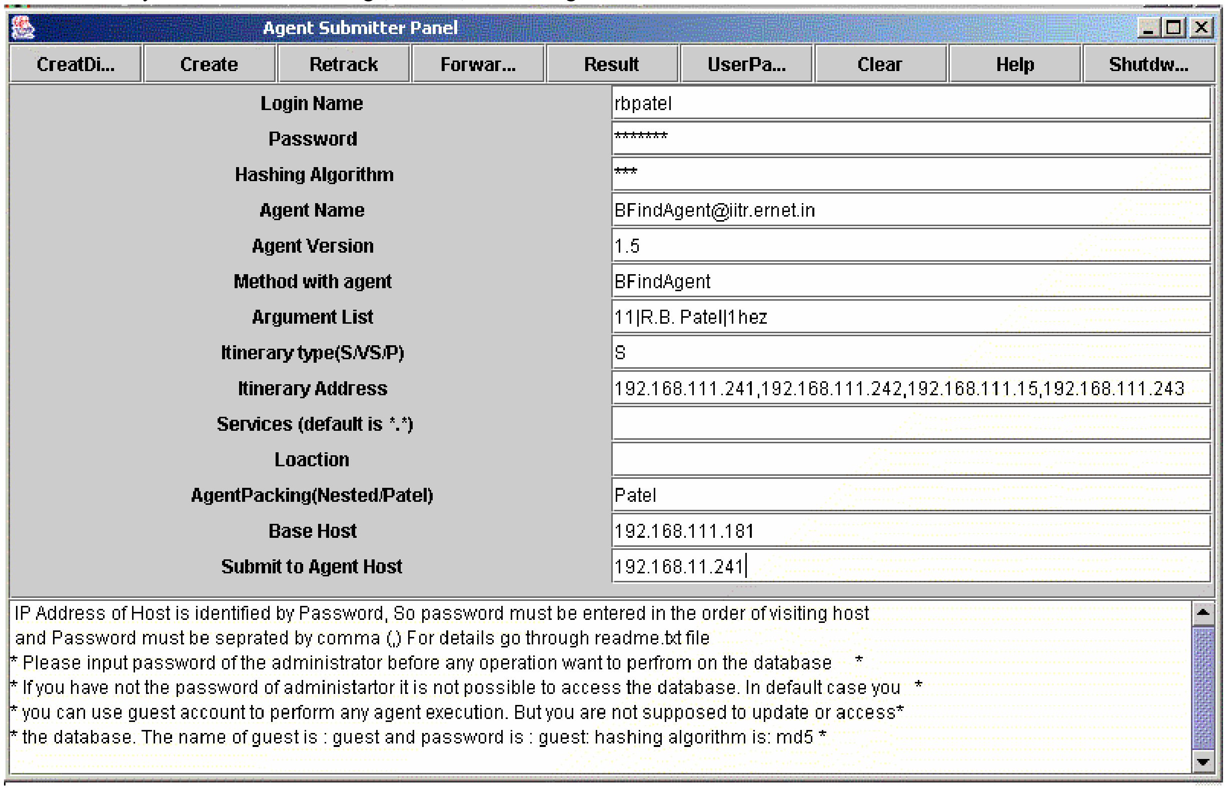
[ 11] Sun Microsystems, “Java Remote Method Invocation–Distributed Computing for Java. White Paper,” March 1998.

[ 12] Tay B.H. and Ananda A.L., “A Survey of Remote Procedure Calls,” Operating Systems Review, 24(3): 68-79, July 1990.

[ 13] MASIF specification is available through http://ftp.omg.org/pub/docs/orbos/97-10-05.pdf

[ 14] Calisti, M., Deluca, D. and Ladd, A., “An agent-based framework for financial transactions,” in Proceedings of Autonomous Agents 2001 Workshop on Agent-Based Approaches to B2B, May 2001.

[ 15] Nwana, H.S., “Software Agents: An Overview,” Knowledge Engineering Review, 11(3): 1 - 40, Sept. 1996, Cambridge University Press, 1996.


Agent Submitter Panel
CreatDi... Create Retrack Forwar... Result UserPa... Clear Help Shutdw...

| | |
|---|---|
| Login Name | rbpatel |
| Password | ******* |
| Hashing Algorithm | *** |
| Agent Name | BFindAgent@iitr.ernet.in |
| Agent Version | 1.5 |
| Method with agent | BFindAgent |
| Argument List | 11\|R.B. Patel\|1hez |
| Itinerary type(S/VS/P) | S |
| Itinerary Address | 192.168.111.241,192.168.111.242,192.168.111.15,192.168.111.243 |
| Services (default is *.*) | |
| Loaction | |
| AgentPacking(Nested/Patel) | Patel |
| Base Host | 192.168.111.181 |
| Submit to Agent Host | 192.168.11.241 |

```
IP Address of Host is identified by Password, So password must be entered in the order of visiting host
and Password must be seprated by comma (,) For details go through readme.txt file
* Please input password of the administrator before any operation want to perfrom on the database    *
* If you have not the password of administartor it is not possible to access the database. In default case you  *
* you can use guest account to perform any agent execution. But you are not supposed to update or access*
* the database. The name of guest is : guest and password is : guest: hashing algorithm is: md5 *
```

**Figure 4. Agent Submitter Interface**