ISSN: 2321-7782 (Online)
Volume 3, Issue 4, April 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Mobile Agents for Mobile Computing

Dharti Bobade[1]
Computer Science and Engineering
H.V.P.M COLLEGE of Engineering and Technology
Amravati, India

P. L. Ramteke[2]
Professor
Information Technology
H.V.P.M COLLEGE of Engineering and Technology
Amravati, India

Abstract: ***Mobile agents are programs that can move through a network under their own control, mi-grating from host to host and interacting with other agents and resources on each. We argue that these mobile, autonomous agents have the potential to provide a convenient, efficient and robust programming paradigm for distributed applications, particularly when partially connected computers are involved. Partially connected computers include mobile computers such as laptops and personal digital assistants as well as modem-connected home computers, all of which are often disconnected from the network. In this paper, we describe the design and implementation of our mobile-agent system, Agent Tcl, and the specific features that support mobile computers and disconnected operation. These features include network-sensing tools and a docking system that allows an agent to transparently move between mobile computers, regardless of when the computers connect to the network.***

Keyword: Mobile agent, Agent Tcl, Network sensing.

I. INTRODUCTION

Mobile computers have become increasingly prevalent as professionals discover the benefits of having their electronic work available at all times. Developing distributed applications that make effective use of networked resources from a mobile platform, however, is difficult for several reasons. First, mobile computers do not have a permanent connection into the network and are often disconnected for long periods of time. Second, when the computer is connected, the connection often has low bandwidth and high latency and is prone to sudden failure, such as when a physical obstruction blocks the signal from a cellular modem. Third, since the computer may be forced to use different transmission channels depending on its physical location, the performance of its network connection can vary dramatically from one session to another. Finally, depending on the nature of the transmission channel, the computer might be assigned a different network address each time that it connects. In short, any distributed application that works on a mobile platform must deal with unforgiving network conditions. In this paper we describe a system that uses mobile agents to support distributed applications for mobile computers. An agent is a program that is autonomous enough to act independently even when the user or application that launched it is not available to provide guidance and handle errors. A mobile agent is an agent that can move through a heterogeneous network under its own control, migrating from host to host and interacting with other agents and resources on each, typically returning to its home site when its task is done. We argue that mobile agents are a good paradigm for distributed applications and an excellent paradigm when mobile computers are involved. We briefly describe a mobile-agent system, Agent Tcl that is under development at Dartmouth College and then present a system of support agents that provide network sensing and routing services. These support agents allow an agent to transparently migrate between a mobile computer and a permanently connected machine, or between one mobile computer and another, regardless of when the mobile computers connect to the network. These support agents provide a more general solution to mobile computing than approaches in which mobile agents are used simply to move an application onto a laptop for continued interaction with the laptop's owner.

II. WHY MOBILE AGENTS

Mobile agents are an effective paradigm for distributed applications, and are particularly attractive for partially connected computing. Partially connected devices include physically mobile computers such as laptops and personal digital assistants as well as home and business computers that are occasionally connected to the network over a SLIP or PPP modem connection. All of these devices are frequently disconnected from the network for long periods of time, often have low-bandwidth, unreliable connections into the network, and often change their network address with each reconnection. Mobile agents directly address the _rst two problems, and with low-level support, can handle the third problem without difficulty. A mobile agent, for example, can migrate on a laptop and roam the Internet to gather information for its user. It can access the needed resources efficiently since it moves to their network location rather than transferring multiple requests and responses across the low-bandwidth laptop connection. Since it is not in continuous contact with the laptop, the agent is not affected by sudden loss of connection, and can continue its task even if the user powers down or disconnects from the network. When the user reconnects, the agent returns to the laptop with the result of its travels. Conversely, an application that lives in the network can send a mobile agent onto the laptop. The agent acts as the application's surrogate, interacting with the user efficiently and continuing to interact even in the event of long-term disconnection [TLKC95, JdT+95]. Mobile agents also ease the development, testing and deployment of distributed applications since they hide the communication channels but not the location of the computation [Whi94b]; they eliminate the need to detect and handle network failure except during migration; they do not require the pre installation of application-specific software at each site (although the agent system must be present); and they can dynamically distribute and redistribute themselves throughout the network. Mobile agents move the programmer away from the rigid client-server model to the more edible peer-peer model in which programs communicate as peers and act as either clients or servers depending on their current needs [Coe94]. Mobile agents lead to more scalable applications since work can be easily moved to whichever network location is most appropriate. Mobile agents allow ad-hoc, on-the-y applications that represent would be unreasonable investment of time if code had to be installed on each network site rather than dynamically dispatched. Finally, our experience with agent programming suggests that mobile agents are easier to understand than many distributed computing paradigms.

III. APPLICATION OF MOBILE AGENTS

It can be argued that mobile agents are not an enabling technology since there are few applications (if any) that are impossible without mobile agents [HCK95]. However, the advantages of mobile agents lead to improved performance in many distributed applications, where performance is a matter of network utilization, completion time, programmer convenience, or just the ability to continue interacting with a user during network disconnection. Mobile agents are best viewed as a general tool for realizing arbitrary distributed applications. This view is reacted in the range of applications in which mobile agents are used. Perhaps the most common examples of mobile code are Java applets. Java applets are interactive applications that can be dynamically pulled across the network with a Java-enabled WWW browser[Sun94]. Java applets are not true mobile agents since they migrate only once, before they start executing, and then only when requested by a user. Java applets are a powerful argument for mobile code, however, since most applets would be intolerably slow if they controlled the screen from a remote location. By moving to the local machine, an applet can control the screen efficiently without the need for pre-installation. Applets represent a special case of mobile agents. Mobile agents are much more powerful since they migrate at will. True mobile agent systems include Tel script [Whi94a, Whi94b], Tacoma [JvS95], Mobile service Agents (MSA) [TLKC95], and our own Agent Tcl [Gra95, Gra96]. Tel script agents are currently used for network management, active e-mail, electronic commerce, and business process management. In network management, a Tel script agent might carry a software upgrade onto a machine along with the code to perform the installation; the agent executes the installation code and disappears. In electronic commerce, a Tel script agent might leave a laptop, search multiple electronic catalogs on behalf of its user, and then return to the laptop with the best purchase price. The most visible use of Tacoma is StormCast, a system for distributed weather

simulation in which the volumes of data are so immense as to make data movement impractical. Mobile Service Agents (MSA) have been used primarily in \follow-me" computing in which an application moves to the location of the user. One MSA demo involves electronic conference proceedings. When a user connects his laptop to the conference's machines, an agent is sent to the laptop. The user interacts with the proceedings via this agent and can continue interacting even when disconnected. Agent Tcl has been used primarily in information retrieval applications. One information retrieval application involves searching distributed collections of technical reports; another, medical records [Wu95]; and a third, three dimensional drawings of mechanical parts [CBC96]. The advantages of agents in these retrieval applications is that each distributed collection can provide low-level primitives rather than all possible search operations; an agent can combine the primitivesinto efficient, multi-step searches. With the service agents for mobile computing that are introduced in Section 4, these same applications work unchanged on roving devices. Agent Tcl is also being used in work now applications, in which an agent carries a multi-step task description from one site to another, interacting with the user at each site in order to carry out that user's part of the task [CGN96]. In Section 4, we describe a work now application that involves both axed and mobile computers, and that is supported easily with our mobile computing infrastructure. In this application, an independent traveling salesperson carries a laptop when visiting customers and uses software that helps to select vendors and products and to place orders. Agents represent orders and travel to the corporation's computers where they interact with billing, inventory, and shipping agents to arrange for the purchase. Agents are also used to explore vendor catalogs and search for products that meet the customer's needs. In all cases, the agents can function while the salesperson's laptop is disconnected.

IV. AGENT TCL

Agent Tcl [Gra95, Gra96] is a mobile-agent system that we are developing at Dartmouth College and using in several information-management applications. Agent Tcl meets four main goals: Reduce migration to a single instruction like the Telescript go instruction [Whi94b], allow the instruction to appear at arbitrary points, and once the instruction is called, transparently capture the current state of the agent and transmit this state to the destination machine. The programmer should not have to explicitly collect state information. The system should handle all transmission details, including the possibility of the destination machine being disconnected or having a new network address. Provide transparent communication among agents. The communication primitives should be edible and low-level, but should work the same regardless of whether the agents are on the same or different machines, and should hide all transmission details. Provide a simple scripting language as the main agent language, but support multiple languages and transport mechanisms, and allow the straightforward addition of a new language or transport mechanism. Provide effective security in the uncertain world of the Internet. The architecture of Agent Tcl is shown in Figure 1. The agent server keeps track of the agents that are running on its machine, provides inter-agent communication facilities, accepts and authenticates agents arriving from other hosts, and restarts these agents in their own interpreter. All other services are provided by agents. Such services include navigation, network sensing, and access control. The agents themselves are separate processes executing the appropriate language interpreter. Each interpreter has the capability to capture the agent's state and send the state to a remote agent server .The only language that we currently support is Tcl, although work on Java is underway. Tcl is a high-level scripting language that was developed in 1987 and has enjoyed enormous popularity [Wel95]. Tcl is an attractive agent language due to its simplicity, ease of use, and portability. A set of special commands was added to Tcl to create Agent Tcl. An agent uses these commands to migrate from machine to machine and to communicate with other agents. The most important command is agent jump, which migrates an agent from one machine to another. The agent jump command captures the internal state of the agent, encrypts and digitally signs the state image, and sends the state image to the agent server on the destination machine. The server authenticates the agent and starts a Tcl interpreter. The Tcl interpreter restores the state image and resumes agent execution at the statement immediately after the agent jump. Further details about Agent Tcl can be found in [Gra95] and on our web page.1. Details about Agent Tcl's security mechanisms can be found in [Gra96].

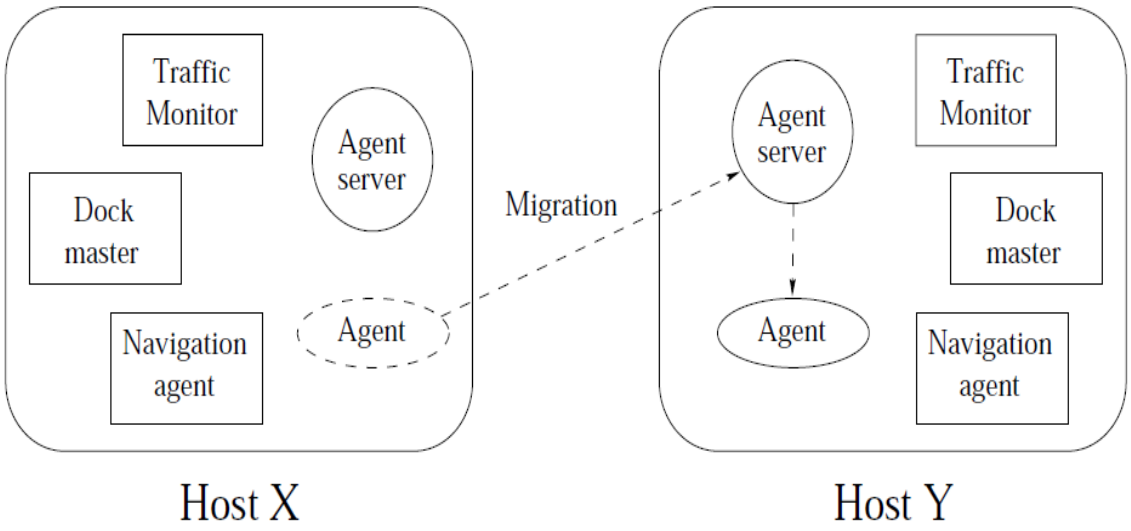

Figure 1: The server-based architecture of Agent Tcl. The agent server coordinates the activities of all local agents and accepts new agents that are arriving from other machines. All other services are provided by specialized agents such them as the dock master, trace monitor, and navigation agents.

V. MOBILE COMPUTING

Mobile agents are an excellent paradigm for implementing distributed applications, particularly in the context of partially connected computers. To be effective, however, the agent system must support disconnected operation in several ways.

» An agent must be able to jump on a partially connected computer (such as a laptop) and return to it later, even if the computer is only connected for brief periods and changes its address upon reconnection.

» An agent must be able to navigate through the Internet to find the services that it needs.

» An agent must be able to sense and react to the network environment, so that it may act autonomously while its user is disconnected.

» An agent must be able to communicate effectively with other agents.

VI. SUPPORTED FOR DISSCONNECTED OPERATION

Unlike traditional client-server computing, agents continue to operate even when the laptop is disconnected. For agents trying to jump into or out of the laptop, however, the traditional approach (try, timeout, sleep, retry, ...) can often fail, particularly if the agent does not happen to retry its jump during a brief reconnection period. To overcome these problems, our laptop docking system pairs each laptop with a permanently connected dock machine (Figure 2). While not all machines act as docks, all machines have a dock-master agent (Figure 1). Consider an agent wishing to jump to a disconnected laptop named D

(Figure 3). To do so, it executes the command agent jump D. When the command completes, the agent will be running on D; the process is transparent. The agent jump implementation attempts to contact D, which fails because D is disconnected. So it then attempts to contact the dock-master agent on the laptop's dock. By convention, the dock for host D is named D dock. Internet host naming allows a single permanently connected machine to have many aliases, which allows one host to act as a dock for many laptops. Once the agent is transferred to D dock, it is not restored into a running agent, but stored on disk under the control of the dock-master at D dock. When D reconnects, its dock- master agent contacts the dock-master at D dock so that all waiting agents can be transferred to the laptop D, where they are restored. In the process, D dock learns of any change in the address for D. Thus, agents trying to reach D will fail to reach it at its old address, jump to D dock, and eventually reach D at the new address.

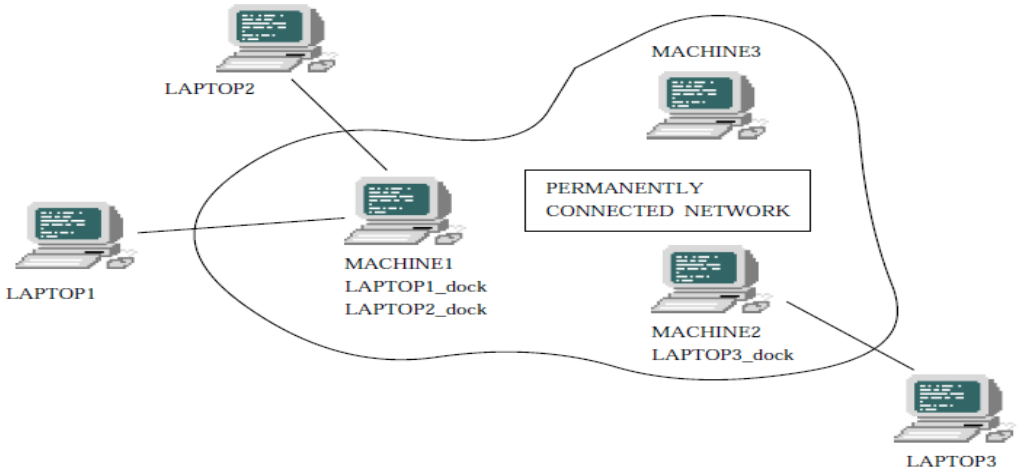

Figure 2: Laptop-docking system

Now consider an agent trying to leave the disconnected laptop D. Again the agent executes agent jump, which detects that the laptop is disconnected, saves the state of the agent to disk, and informs the local dock-master agent. The dock-master continually monitors the network status; when the network is connected, the dock master immediately transfers all waiting agents o_ of the laptop (Figure 3). This scheme has several advantages: the agents leave the laptop as soon as Figure 3: Jumping to or from the laptop possible; agents do not miss any opportunities to leave; waiting agents are saved on disk, where they survive crashes and shutdowns, and do not occupy precious memory and CPU time; and their state is captured and ready for transfer as soon as the network is connected. Thus, agents wishing to jump on or o_ the laptop move quickly as soon as the laptop is connected, minimizing the connection time necessary. Again, the entire process is transparent to the agent. Now consider a more complex case, where an agent's source (host S) and destination (host D) are both laptops (Figure 4). It is easy to imagine that they may never both be connected at the same time, making a direct jump impossible. The agent's state is captured on S, and saved onS's disk until the dock-master detects a network attached to S. At that point S's dock-master attempts to transfer the agent to D; when that fails, it transfers the agent to D's dock (D dock). If D dock is unreachable, perhaps due to a temporary problem in the Internet, the S dock-master tries to transfer the agent to S dock. If S dock is also unreachable, the dock-master will try the entire process again at a later time. If S dock is reachable, the agent is sent to S dock. The dock-master on S dock will periodically attempt to transfer the agent to either D or D dock. The agent may reside at D dock until D connects and notifies the dock-master at D dock of the new location of D. Once at D, the agent's state is restored.

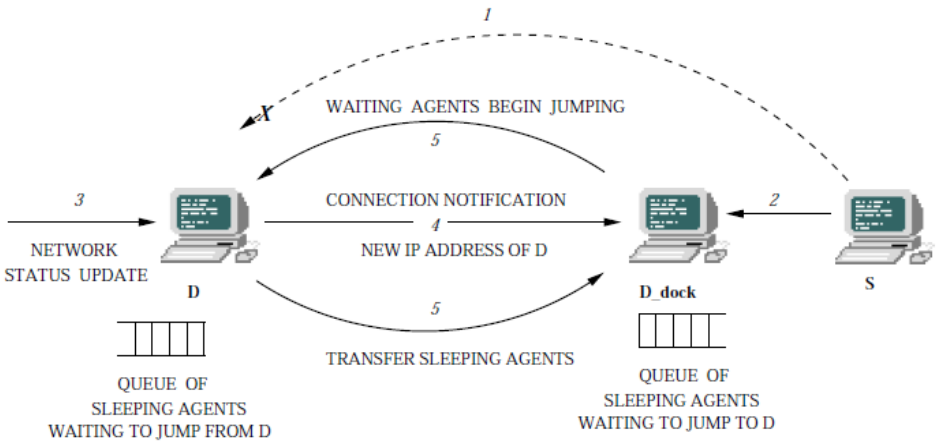

Figure 3: Jumping to or from the laptop

We are extending our laptop docking system to support multi-destination jumps, which are useful when an agent wishes to visit multiple hosts (D1;D2; :::;Dn) but in no particular order. This situation arises when the agent is searching all of the sites for information, or when it needs to visit one of a replicated set of servers. The multi-destination jump allows the agent to travel in a manner most suitable to the present network conditions. The dock-master agent on S _rst tries to transfer the agent to one of the _nal destinations by trying each one in order (D1;D2; :::;Dn). If all the destinations are unreachable, the S dock-master transfers the agent to S dock. The dock- master at S dock periodically tries to reach the destinations until one of the transfers succeeds. S dock does not transfer the agent to a dock Dk dock in order to avoid premature commitment to a destination that may rarely connect, although this issue is a matter for further research. When the agent awakes (returns from its call to agent jump), it knows that it has arrived at one of the destinations. A quick check of the local host name con_rms the particular destination. For agents that desire more control over the jumping process, we provide hooks to allow agents to query the status of the network connection, to request a failure notification rather than being blocked when the jump destination cannot be reached immediately, or to request that the jump go as far toward the destination as possible and then wake up the agent.

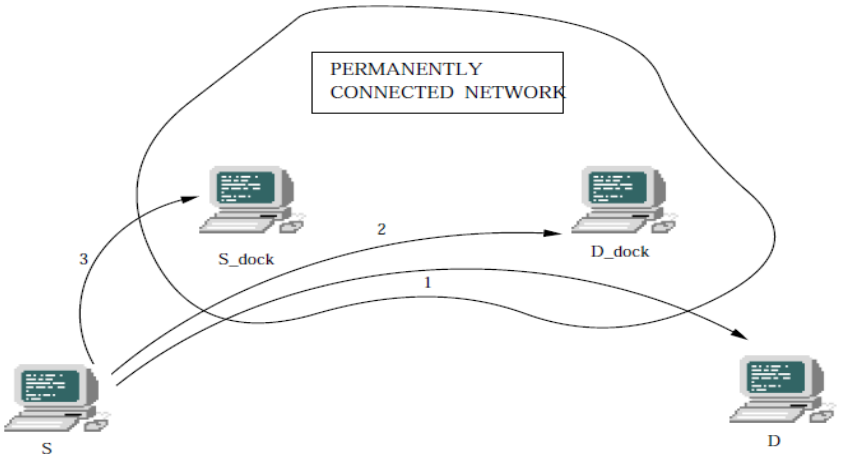

Figure 4: Laptop to laptop jump

VII. Agent Nevigation And Adaption

The world of an agent is dynamic and uncertain. Machines go up and down, the information stored in repositories changes, and the exact sequence of destinations and steps needed to complete an information gathering task often is not completely known at the time that the agent is launched into the world. An autonomous agent is crippled without external state (what the agent can perceive about the state of its world) since it has no way of perceiving and adapting to the dynamic changes in its environment. In this section we describe the sensors that allow an agent to determine its external state and a mechanism that uses these sensors for adaptive navigation.

Network sensing.- Network sensing, at least the ability for a laptop to detect the state of its network connection, is an integral part of our laptop docking system described in the previous subsection. It performs an even more important task, however, when providing agents with information about the expected transit time across the network and about whether a network site is reachable at all. This information enables agents to adapt to changing network conditions. Consider an agent that needs to visit information resources at several sites. A smart agent should be able to adapt to the fact that some sites may currently be unreachable, and to visit other sites _rst. An even smarter agent may be able to plan a sequence of visits given an estimate of the current network delay to each site. Other agents may wish to tailor their behavior to the current bandwidth available, such as the amount or format of the data that they carry with them. We provide a set of network sensing tools that the agents can use to gather information about the status of the network.

» A tool to determine whether the local host is physically connected. This tool \pings" the broadcast address on the local subnet; if there is any response in a short interval, the network is connected. 9

» A tool to determine whether a specific host is reachable; this is just the standard \ping."

» A tool to determine the expected bandwidth to a remote host, so that agents can choose their destination or amount of data based on the bandwidth. Rather than measuring the bandwidth by sending lots of data to the remote host, which would often take as much time as sending the agent itself, we attempt to predict bandwidth from experience. A trace monitor agent at each site tracks information about all recent communications (bytes moved and time required), which is provided by the local agent server. Application agents contact the network monitor to obtain estimates of bandwidth or latency, which are computed from the recorded information. Our trace monitor uses a weighted average of all communications with the requested remote site, weighting recent communications more heavily than older communications. If there are no recent communications with the requested site, the trace monitor may use data from recent communications with \similar" sites, that is, other sites in the same subnet or domain as the requested site.

VIII. CONCLUSION

Mobile computing has proven a fertile area of work for researchers in the areas of database and mobile agents. The inherent limitations of mobile computing systems present a challenge to the traditional problems of mobile agents. As we can see, the amount of research in this area in the last few years have been staggering. However, some problems remain open for research. There is a need for better protocols in the area of data sharing and transaction management, better interfaces, clever algorithms that exploit locality to shape the answers to queries. Undoubtedly, we will continue to see a steady number of research contributions in the future.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their helpful comments to improve this paper.

References

1. S. Pittie, H. Kargupta, B. Park. Dependency detection in MobiMine: a systems perspective. Information Sciences, 155(34): 227243 (2003).
2. H. Kargupta, B. Park, S. Pitties, L. Liu, D. Kushraj, K. Sarkar. Mobimine: monitoring the stock marked from a PDA. ACM SIGKDD Explorations, 3(2): 3746 (2002).
3. F. Wang, N. Helian, Y. Guo, H. Jin. A Distributed and Mobile Data Mining System. Proc. Int. Conf. on Parallel and Distributed Computing, Applications and Technologies (2003).
4. H. Kargupta, R. Bhargava, K Liu, M. Powers, P. Blair. S. Bushra, J. Dull. VEDAS: A Mobile and Distributed Data Stream Mining System for RealTime Vehicle Monitoring. Proc. SIAM Data Mining Conference (2003).
5. Domenico Talia† and Paolo Trunfio, “Mobile Data Mining on Small Devices Through Web Sevices”(2010).
6. Pittie, H. Kargupta, B. Park, ”Dependency detection in MobiMine:a systems perspective” Information Sciences, 155(34):227243 (2003).
7. H.Kargupta,B.Park,S. Pitties, L.Liu, D. Kushraj,K.Sarkar, ”Mobimine: monitoring the stock marked from a PDA” ACM SIGKDD Explorations, 3(2):3746 (2002).
8. M.Adaçal,A.B.Bener,”MobileWebServices:ANewAgentB asedFramework”IEEE Internet Computing, 10(3): 5865(2006).
9. M.Tian,T.Voigt,T.Naumowicz,H.Ritter,J.Schiller, ”Performance Considerations for MobileWeb Services” Computer Communications, 27(11): 10971105(2004).
10. H.Chu,C.You,C.Teng,”Challenges:WirelessWeb Services” Proc.Int.Conf.Parallel and Distributed Systems (ICPADS 04), IEEE CS Press (2004).
11. W.Zahreddine,Q.H.Mahmoud,”An Agentbased Approach to Composite Mobile Web Services”, Proc.Int. Conf.on Advanced Information Networking and Applications (AINA’05), IEEE CS Press (2005).
12. Khaled M. Hammouda and Mohamed S . Kamel, “Hierarchically Distributed Peer-to-Peer Document Clustering and Cluster Summarization”, IEEE Transactions on Knowledge and Data Engineering, Vol. 21(5), pp.681-698(2009).
13. Fenggui, Malek, Adjouadi, Naphtali Rishe “Personalized Approach for Mobile Search”, IEEE DOI 10.1109/CSIE.2009.718(2008).
14. .Hyun-suk,Hwang,Seon-hyunShin,Ki-ukKim,Seok- CheolLee,Chang-sookim,”A Context-aware System Architecture using Personal Informatiom based on Ontology”,5th IEEE Int’(2007).
15. Chih-Hao Liu Jason Jen-Yen Chen,Jhong-Li, Taiwan, “Mobile User Agent with User Ontology for Personalized Web Service Access” IEEE (2010)

