Mobile Agents for Pervasive Computing Using a Novel Method of Message Passing

David Levine, Renjith Thomas, Farhad Kamangar, and Gergely V. Záruba
Department of Computer Science and Engineering,
The University of Texas at Arlington, Arlington, TX-76019

Abstract *- Pervasive computing is an emerging technology that brings new dimensions to distributed computing; it uses a wide variety of smart, ubiquitous devices throughout an individual's working and living environment. Mobile agents are software entities that can migrate between servers or mobile agent environments of the network accomplishing various tasks on the behalf of their owners. The objective of this paper is to describe a test and prototyping environment for experimenting with mobile agents in pervasive environments. A prototype environment for a novel proactive infrastructure is described for mobile agent assisted pervasive computing. In addition, a new message passing algorithm is provided for mobile agent connection establishment and management (CEMA).*
Keywords: mobile agents, pervasive computing, distributed computing, ubiquitous devices

1. Introduction

This paper introduces a new mobile agent communication paradigm to be used in pervasive computing [6] environments. Mobile agents are software entities or programs that can migrate from host to host in a network, at the instance and destination of their own choosing. A host refers to a computationally able ubiquitous node that provides an execution environment for mobile agents. Hosts can vary in size and computational power from a small sensory device to a large server, typical examples include workstations, desktops, laptops, PDAs, and more sophisticated/dedicated devices such as routers or database servers. There are some well-known mobile agent environments in existence today, including IBM Aglets [7] (Java byte-code based), Tcl agents [8] and Telescripts [9] (both script based); all of these methods are interpreted by respective agent servers enabling code mobility.

Pervasive computing is a newly emerging paradigm to provide users with anytime anywhere access to information or computing resources. Pervasive computing enables convenient access to relevant information for users and applications through a class of intelligent and ubiquitous software and hardware entities that have the ability to come alive and become available when and where needed. Ubiquitous entities refer to small, mobile computing devices like handhelds, portable and wearable computers and appliances equipped with intuitive user interfaces to enhance information processing while being accessible by different networking paradigms. Many research groups and projects throughout the world are focusing on various aspects of pervasive computing, e.g., the Oxygen project at MIT [5].

The motivation behind this paper is to extend the mobile agent paradigm to pervasive computing environments, enabling the development of novel ubiquitous applications. One of the features of mobile agents is the asynchronous and autonomous behavior of mobile agents. Mobile agents are generally transport layer and architecture independent depending only on the availability of the execution environment. Mobile agents can be designed to be robust and fault tolerant to dynamically adapt to unfavorable conditions. After careful analysis of mobile agents [12] the authors propose a mobile agent systems architecture that can be used in a pervasive computing environment for ubiquitous devices.

The prototype design of the mobile agent architecture as well as the server for the mobile agent architecture are described in Section 2. Section 3 outlines a novel Connection Establishment and Management Algorithm CEMA for message passing among mobile agents in heterogeneous

networks. Section 4 presents a simulation based performance analysis of CEMA.

2. Prototype Architecture

One of the main research thrusts in the field of pervasive computing is making the computations and functional behavior the system invisible to users while providing smart spaces [6] around ubiquitous devices. The proposed software infrastructure relies on the operating system to provide the execution environment for mobile agents. The agent environment consists of five software blocks as outlined in Figure 1.

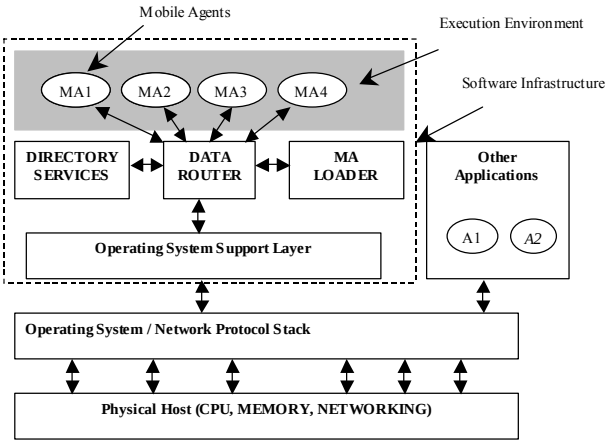

Figure 1. Prototype architecture.

The *Data Router* is the central nerve of the infrastructure. It is responsible for receiving, interpreting and exchanging data between the mobile agents and the outside world. The routing functions are supported by the directory services**.**

Directory Services provide with the addressing lookup for inter-host as well as intra-host data transfer. Directory services implement three directory-tables for communication services: resource table, service table and lookup table:

- The resource table maintains information on neighboring hosts.
- The service table stores information about mobile agents residing in the neighboring hosts as well as the native host.
- The lookup table implements a volatile cache of mobile agent addresses enabling the routing of messages among mobile agents.

The *Mobile Agent Loader* (MA Loader) provides the execution environment for the mobile agents. It maintains a process table with entries for each of the currently running mobile agents in the host, while tracking of the execution state of these agents.

The *Operating System Support Layer* abstracts low-level services to the other blocks in the infrastructure; it maintains information about the host system and provides communication channels and basic input and output primitives.

2.1. Mobile agent architecture

In our prototype, mobile agents inherit all properties of threads; they are able to spawn parallel thread on their own, sleep, suspend, stop and resume like any normal thread. A UML representation of a mobile agent is given in Figure 2. In order to enable migration, mobile agents have to have a serializable interface to suspend parallel behavior for the duration of the migration. Mobile agents should also have a uniform MA interface with a standard framework for mobile agent communication.

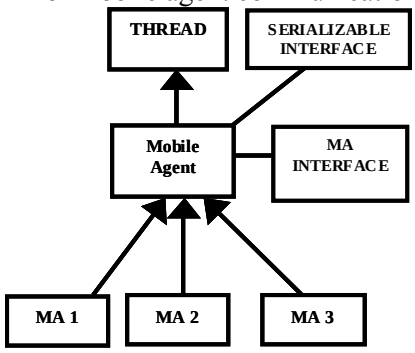

Figure 2. UML representation of mobile agents.

A general mobile agent template is provided based of the previously outlined UML. All other mobile agent instances are inherited from this base template. The basic architecture of a mobile agent is depicted in Figure 3. Mobile agents consist of three modules: functional module, control module and communication module:

- The *functional module* contains the functions the mobile agent should perform; the functions can receive input from the host, or from other agents.
- The *control module* consists of three sub-modules maintaining information about the mobile agent. The ID sub-module is responsible to carry the MA identity. The Peer sub-module maintains a list of peer mobile agents. The Stat sub-module stores the state information of the MA

The communication module maintains a common framework for communication with the software architecture. The MA Interface sub-module allows simultaneous reception and transmission of messages between agents. The input buffer provides buffering for incoming data, while the output buffer does the same for outgoing data. The serializable interface serves the agent in preparing for migration to another host by suspending all parallelism.

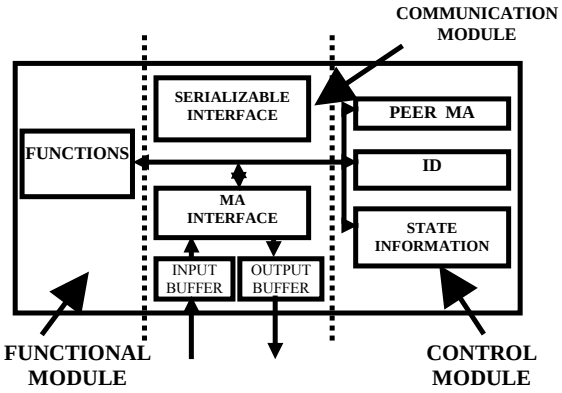

Figure 3. Basic mobile agent architecture.

3. CEMA

CEMA (Connection Establishment and Management Algorithm) is used to establish and manage connections between mobile agents. CEMA ensures that all mobile agents are able to communicate with peer agents even in sparsely connected pervasive networks. Previous research addressing communication among mobile agents has concentrated on fully connected networks, in which any two hosts can communicate with each other directly [10]. While an active message approach [11] uses an agent to route messages on-the-fly, CEMA calculates routes on demand.

3.1. Design components

The network between hosts is represented by an undirected graph $G = (V, E)$ where V is the set of vertices representing the hosts, while E is the set of edges representing a communication link between two hosts. E_{xy} denotes a connection between vertices V_x and V_y implying a full duplex communication channel. C_x represents the set of vertices connected directly to V_x. Communication between two hosts not directly connected with each other requires messages relayed by other intermediate nodes. A sample multihop network is depicted in Figure 4.

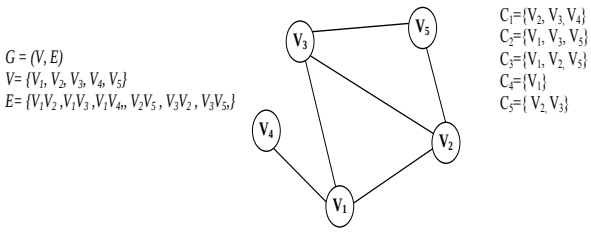

Figure 4. A sample network.

Key parameters of messages relayed from one host to another are: *mode*, *sid* (sender mobile agent id – the unique identifier of the originating MA), *shid* (sender host id – the unique identifier of the originating host), *rid* (receiver agent id – the unique identifier of the destination MA), *ts* (timestamp), *info* (the body of the message), *current* (the unique identifier of the host currently running CEMA on the message). Depending on the *mode* parameter, messages are divided into three categories:

1. **Control messages** (*mode*=0): contain routing information and they dispatched when an agent is launched in the agent loader. When an agent migrates, it will transmit control messages to its peer mobile agents
2. Normal messages (mode=1): contain user data to be relayed between mobile agents.
3. Acknowledgement messages (mode=2): contain acknowledgments for messages..

As described in Section 2, each host maintains three tables in the directory services that help in sending messages. A more comprehensive description is given in the next paragraphs:

1. **Resource table:** is a hash table maintaining information on the physical neighbors of a specific host with unique identifier *hid (host id)*. The *hid*s are used as keys associated with corresponding values (*values*) (communication parameters).
2. **Service table:** is a hash table maintaining information of mobile agents residing in the physical neighbors of hosts. A typical key in a service table is the unique identifier of mobile agents residing at the neighbors while the corresponding values are the unique identifiers of the serving hosts.

3. **Lookup table:** is a hash table maintaining routing information on mobile agents in a host with unique identifier *hid*.

3.2. CEMA Primitives

The following list contains the CEMA primitives:

- CEMA-RESOURCES (*res (hid)*) is called to retrieve the unique identifiers of all the physical neighbors of a host *hid*. It has a complexity of O(N) where N is the number of hosts in the network.
- CEMA-CONTAINS-MOBILE-AGENT-SERVICE (*ser(hid)*, *id*) determines whether there is an entry key for an agent in service table of the host with unique identifier *hid*. It has a complexity of ***O(N)***.
- CEMA-FIND-ADDRESS-SERVICE(*ser (hid)*, *keys*) returns the unique identifier of the host where the mobile agent exists. It has a complexity of ***O(1)***.
- CEMA-CONTAINS-MOBILE-AGENT-LOOKUP (*lookup(hid)*, *keys*) determines if there is an entry for an agent in the lookup table of host with unique identifier *hid*. It has a complexity of ***O(M)*** where *M* is the number of mobile agents in the network.
- CEMA-FIND-ADDRESS-LOOKUP(*ser (hid)*, *keys*) returns the unique identifier of the host where the mobile agent is presumed to exist. It has a complexity of ***O(1)***.
- CEMA-VERIFY-MESSAGE-LOOKUP (*lookup(hid)*,*keys*, n*ewts*) verifies if a message it received is an old message; if so then the message should be discarded. It has a complexity of ***O(M)***.
- CEMA-UPDATE-LOOKUP (*lookup(hid)*,*keys*,*newts*,*level*) is used to add/update entries in the lookup table *hid* when it receives a message from an. It has a complexity of ***O(M)***.

3.3. Description of CEMA

The CEMA algorithm is given in Table 1. CEMA runs in the data router module of each host. It consults with the directory services to determine what function should be performed on each message it receives. The *current* parameter refers to a unique identifier of the host where the algorithm is currently executing. The output of CEMA is the most optimal neighbor to route the message to. The variable *level* refers to the host from where the message was received.

Lines 1–4 initialize the variables that are being used in CEMA. The Boolean variable *found* is set to true when the location of the destination agent *rid* is found in the directory services during the execution of the algorithm. Control messages (*mode* = 0) do not rely on the lookup table (*lookup(current)*) to route from the source mobile agent to the destination agent. Control messages are used to build the lookup table, so they consult only the service table (*ser(current)*) and the resource table (*res(current)*). Line 5 ensures that the lookup table is consulted only if the message is not a control message.

In line 8, CEMA checks whether the message is an old message; if the message is old then it is discarded. When a message is received, only the information regarding the sender mobile agent (*sid*) is updated. It has to be ensured that no entry for *sid* is added in the lookup of the host where *sid* is residing (implemented by lines 11-14). An update is made for the location of *sid* along with the timestamp (*ts*) of the received message in line 15.

When mobile agents are dispatched to a pervasive network the first task they perform is to transmit control messages to their peer mobile agents. These control messages populate the lookup tables initially. The resource tables are populated when the network is formed. The service table is populated when the host receives the mobile agent. Figure 5 shows how the data in various tables are affected by the various events.

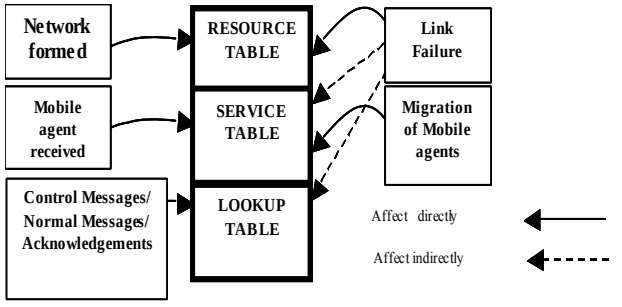

Figure 5. Events affecting tables in a host.

If a message is a control message (*mode* = 0), CEMA first checks if the message is outdated. It then updates the information in the lookup table for the originating agent. The service table is queried to determine the location of the destination agent. If the query is unsuccessful then CEMA will multicast the

Table 1. CEMA algorithm

CEMA(*current, lookup(current), service(current),resource(current), sid ,rid, mode, ts, level)*

```
1   destination ← NULL
2   found ← FALSE
3   continue ← FALSE
4   temp ← NULL
5   if CEMA-CONTAINS-MOBILE AGENT-LOOKUP(lookup(current),rid)) = TRUE and mode ≠ 0
6     then found = TRUE
7          destination ←CEMA-FIND-ADDRESS-LOOKUP(lookup(current), rid)
8   if CEMA-VERIFY-MESSAGE-LOOKUP(lookup(current),sid))
9     then continue = TRUE
10  if continue = TRUE and level ≠ current
11    then if CEMA-CONTAINS-MOBILE AGENT-SERVICE(ser(current), sid) = TRUE
12           then temp←CEMA-FIND-ADDRESS-SERVICE(ser(current), sid)
13         if temp ≠ current
14           then CEMA-UPDATE-LOOKUP(lookup(current), sid ,ts, level)
15  if continue = TRUE and found = FALSE
16    then if CEMA-CONTAINS-MOBILE AGENT-SERVICE(ser(current), rid) = TRUE
17           then found = TRUE
18                destination←CEMA-FIND-ADDRESS-SERVICE(ser(current), rid)
19  if continue = TRUE and found = TRUE
20    then if current = destination
21           then if mode = 1 or mode = 2 or mode = 0
22                  then print "Destination reached"
23                if mode = 1
24                  then send an acknowledgement for the received message
25         else
26           send the message towards destination
27   if continue = TRUE and found = FALSE
28     then for each element in keys in CEMA-RESOURCES(res(current))
29          do if element ≠ level
30  then send the message towards element.
```

message to all of its neighbors, except for the neighbor from which it has received the message.

If a message is a normal message (*mode* = 1), then the lookup table is queried for the location of the destination host. CEMA then proceeds the same way as in the case of control messages. A fallback to search the service table is made only if CEMA fails to find the location of the destination host in the lookup table. Acknowledgement messages are generated for every normal message. Acknowledgement messages (*mode* = 2) are similar to normal messages except that they do not require an acknowledgement

3.4. Link Failure and Migration of Agents

Formally, if MA_z wants to migrate from V_x to V_y then it has to remove itself from $ser(V_x)$ before moving to V_y to ensure proper working of the routing algorithm. After MA_z reaches the destination, it adds/updates an entry for itself in ser (V_y), as well as adds/updates an entry for itself in the service table(ser) of each element in C_y. It also removes all entries in $lookup(V_y)$ referring to MA_z. In the next step MA_z transmits control messages to all of its peer mobile agents. Failure of edge E_{xy} mandates V_x and V_y to remove entries corresponding to V_y and V_x,

from their resource-, service-, and lookup tables. Changing the route of a message if a new link is not handled by CEMA; the path remains undiscovered until a mobile agent migrates and sends out the control messages using (and detecting) the new link.

4. Analysis of CEMA

This section presents a performance analysis of CEMA. The following assumptions are made for the analysis:

1. The communication channel remains duplex at all times
2. All link bandwidths and power sources of devices are infinite.
3. The cost of each link is uniformly one.
4. There is equal delay on all links and there is no loss during transmissions.
5. Graph G remains connected at all times.
6. V_x and V_y are notified if E_{xy} is removed.

In a time interval τ, if n agents are launched then total number of control messages generated is equal to $n*(n\text{-}1)$. If in time internal τ, there are m migrations of mobile agents then the total number of control messages generated due to the migrations is: $m*(n\text{-}1)$. If x number of messages are transmitted in a time interval τ, then the total number of normal messages is x and the total number of acknowledgement messages is x. Therefore in a time interval τ, if n mobile agents are launched followed by m number of migration and they send x messages among themselves, the total number of messages is: $n*(n\text{-}1) + m*(n\text{-}1) + 2x$.

The algorithm has a message complexity of $\boldsymbol{O}(N+M)$ where N is the number of hosts in the network and M is the number of mobile agents in the network. Messages in CEMA traverse the network in a breadth-first fashion; T is a breadth-first generated spanning-tree of nodes from V_x to V_y. The maximum number of hops a message (normal message) can take to reach its destination depends on the number of levels in T (the location of the mobile agent).

The algorithm allows only forward movement of messages and does not allow a message to revisit an edge it has already visited. Thus, the numbers of levels in T denoted by x determine the average number of hops a message takes to reach from a source to a destination. The worst-case scenario occurs when the graph G is a linear graph having a worst-case complexity of $\boldsymbol{O}(|V|)$. In such a case, the number of *levels* in T is equal to the number of edges. The best-case scenario is when all the mobile agents are concentrated on a single host, while in a fully connected G, to reach the destination the message has to take either zero or one hop.

4.1. Experimental setup and results

In order to investigate the performance of CEMA, a simulator was implemented in Java. Two distinct sets of experiments were performed:

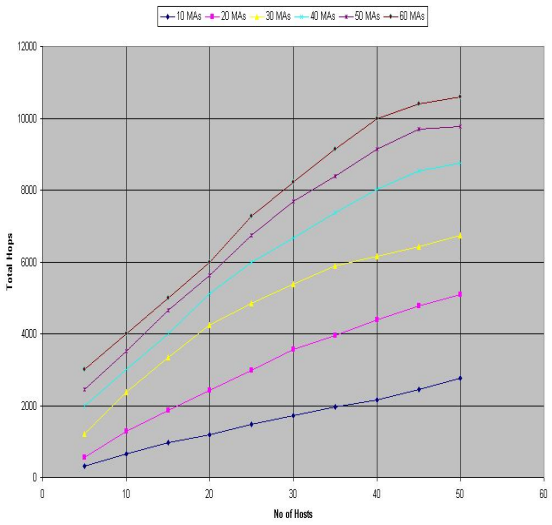

Figure 6. Graph of number of hops vs. number of hosts.

1. The simulator parameter was the population of hosts, n (n =5, 10, …, 50). Six different experiments were performed for each instance of n, changing the number of mobile agents m (m=10, 20, …, 50, 60) (mobile agents are distributed uniformly between nodes). Figure 6 shows the graph for the number of hops versus number of hosts.
2. The simulator was run with a fixed number (20) of hosts keeping m as a factor (m=5, 10, 15, 20, 25, 30). Mobile agents were randomly picked to migrate between devices (with a uniform random distribution of destinations). The number of migrations was varied at 10, 20, and 30 migrations during the transmission of 100 randomly generated normal messages

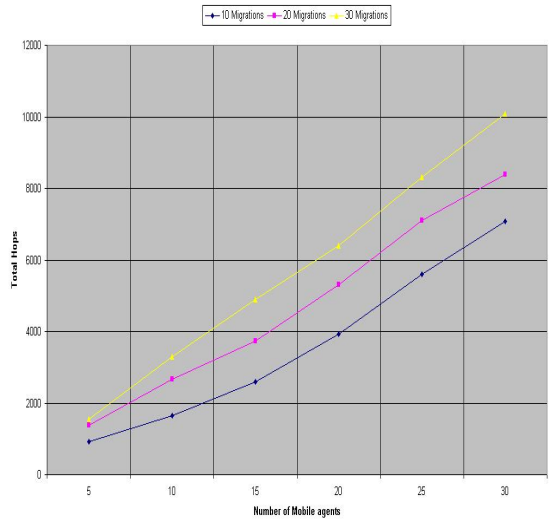

Figure 7. Graph of number of mobile agents vs. total hops.

4.2. Limitations of CEMA

The CEMA algorithm cannot automatically detect link failures. If a link fails the updates to the tables have to be done by an external entity. Work is underway to extend CEMA for these scenarios.

Another limitation of CEMA is that it may lose messages under the following circumstances. Messages can only move forward from the source to the destination due to the timestamp and due to the multicast restriction towards the originator. If a message crosses a bridge, it will not be able to return across the bridge; if meanwhile the destination mobile agent happens to migrate to the other side of the bridge, then the message is lost since it cannot go back across the bridge. This situations cannot be handled by the current algorithm and will be addressed in a subsequent paper.

5. Conclusions

This paper presented the design of a prototype software infrastructure for pervasive computing utilizing mobile agents. Architectural components of the software infrastructure were described as well as a novel algorithm (CEMA) for establishing connections between mobile agents in a pervasive network was presented. A Java based simulator was designed to evaluate the message complexity performance of CEMA. Various experiments conducted by the simulator have shown the scalability of the infrastructure as well as the limitations of the infrastructure. Ongoing work is addressing the limitations of CEMA to design real-life pervasive computing systems.

6. References

[1] A. Silberschatz, J. Peterson, and P. Galvin, "Operating System Concepts, 3rd Edition," *Addison-Wesley Publishers*, 1991.

[2] G.H. Forman, and J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer*, vol. 27, nr. 6, April 1994.

[3] M. Satyanarayanan, "Pervasive Computing: Visions and Challenges," *IEEE Personal Communications*, 2001.

[4] T.C. Agoston, T. Ueda, and Y. Nishimura, "Pervasive Computing in the Networked World," *Proceedings of INET2000*, Yokohama, Japan, 2000.

[5] MIT Oxygen project, 2001. http://oxygen.lcs.mit.edu

[6] A.C. Huang, B. Ling, S. Ponnekanti, and A. Fox, "Pervasive Computing: What Is It Good For?," *Proceedings of the Workshop on Mobile Data Management (MobiDE) in conjunction with ACM* MobiCom, 1999.

[7] D. Lange, and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets," *Addison-Wesley Publishers*, 1998.

[8] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko, "AGENT TCL: Targeting the Needs of Mobile Computers," *IEEE Internet Computing*, vol. 1, nr. 4, pp. 58–67, 1997.

[9] W. Cockayne, and M. Zyda, "Mobile Agents," *Manning Publications*, 1998.

[10] K. Paul, and S. Bandyopadhyay, "Evaluating The Performance of Mobile Agent Based Message Communication Among Mobile Hosts In Large Ad-Hoc Wireless Networks," *Proceedings of the Second ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems* (In conjunction with IEEE/ACM MobiCom'99), Seattle, WA, August 1999.

[11] C. Okino, and G. Cybenko, "Modeling and Analysis of Active Messages in Volatile Networks," *Proceedings of the 37th Allerton Conference on Communication Control & Computing*, Monticello, IL, September 1999.

[12] R. Thomas, "Mobile Agents for Pervasive Computing," *Master's Thesis*, The University of Texas at Arlington, shameless self-advertising, May 2002.