Mobile Agent Location Management in Global Networks

R. B. Patel
Department of Computer Science & Engineering
M. M. Engineering College, Mullana-133203
Haryana, India

Nikos Mastorakis
Dept of Computer Science
Military Inst. of University Education / Hellenic Naval Academy
Terma Hatzikyriakou 18539, Piraeus, GREECE

K. Garg
Department of Electronics & Computer Engg,
IIT Roorkee, Roorkee-247667, Uttaranchal, India

Abstract
Mobility management is a necessity in highly dynamic and large-scale mobile agents network, especially in a multi-region environment in order to control and communicate with agents after launching. Existing mechanisms for locating mobile agents are not efficient as these do not consider the effect of location updates on migration time and produce network overload. A location management protocol consists of location updates, searches and search updates. An update occurs when a mobile agent changes location. A search occurs when a mobile agent needs to be located. A search–update occurs after a successful search. This paper presents a hierarchical model for location management of mobile agents in global networks. Three protocols are developed, namely search, update and search-update. The location management technique uses one combination of search, update and search-update protocols throughout execution. Three cases are considered for Update and Search-Update Protocols. Thus nine combinations of location management protocols are generated, from which an agent can dynamically select one as per requirement, to communicate with other agents on the global network. We have implemented these protocols on the PMADE system developed at IIT Roorkee, to evaluate the performance of different protocols for various communication and mobility patterns. Results indicate that performing search –updates significantly reduces the message overhead of location management.

Keywords: Location Management, Mobile Agent, Update Protocol, Search Protocol and Search-Update Protocol.

1 Introduction

A Mobile agent (MA) [5] is a software process, which can move autonomously from one physical network location to another. The agent performs its job wherever and whenever it is found appropriate and is not restricted to be co-located with its client. Thus, there is an inherent sense of autonomy in the mobility and execution of the agent. Agents can be seen as automated errand boys who work for users. MA research evolved over the past years from the creation of many different monolithic mobile agent systems (MASs), often with similar characteristics and built by research groups spread all over the world, for optimisation and better understanding of specific agent issues [5], [3].

As large scale MASs are the next trend following the popularity of MA technology, the collaborations between roaming agents has increased. There should be an efficient locating mechanism (mobility management) for locating MAs as part of the agent communication platform. Most MASs have the following common features:

(a) MAs are launched to complete some tasks. They may roam around the network automatically from host to host. They normally end at the launching point with their results or submit results at the last host in the itinerary [12].
(b) The agent management centre keeps locating these roaming agents so that it can set up communication with them at their current locations whenever necessary.

The second feature given above is actually the function of MAS mobility management. The basic operations associated with mobility management are:

1. A roaming agent updates its location frequently to the central management server (e.g. a directory server).
2. The agent management server refreshes the current location record of the agent in its location database.
3. When there is a request asking for the location of the agent, the management server searches the database and replies with the current location of the MA. Beside these three basic steps, the management server may also process issues such as out-of-date location records. Most

existing MASs have provided partial mobility management, by defining different naming and locating mechanisms.

An agent owner has no control over an agent, after launching it. To give some control or useful information to it, it is necessary to locate the required agent. Further, if any agent wants to communicate with another, it must know its present location. Typically, locating an agent is invoking a function of the form "where_is_agent(*A*) which should return the current location of agent *A*. Researchers have recently proposed many protocols for designing such a function. Various approaches for storing, updating and locating MAs are addressed in [4, 9]. Some communication protocols use broadcasting or multicasting approaches to locate MAS [8].

The ability to locate MAs while they are migrating from one node to another one is of great importance for the development of agent-based applications which have to work in geographically distributed environments [15, 9]. This issue becomes even more important when focus is shifted from distributed application limited in space, to distributed application whose environment is spread all over the Internet. None of the Java-based MA platform provides a comprehensive, effective location management system. In any case these mechanisms are strictly tied with the platform that they are designed for without exploiting existing techniques for searching or locating objects in the Internet. When a global environment such as the Internet is considered, a centralized naming protocol quickly becomes a bottleneck for the system, providing poor performance. Distributed techniques and algorithms are often more effective even if their implementation is more difficult [14], [13].

For example, [16, 17] defined a naming and ATP mechanism; [18] used a middle-agent method and [19] adopted a matchmaking scheme. But none of these systems considered the performance overheads caused by mobility management, especially in case of highly dynamic and large scale MAs. Such systems need to update their locations more frequently, more network resources, more server storage and associated computation on both sides are required. The same happens given large scale of MAs.

Di Stefeno et al [2] have developed a Search-by-Path-Chase (SPC) protocol. They have assumed that a network is divided into regions and then into sites. Two types of registers are used in which an agent has to register or update its location information, whenever it has to migrate from one site to another. This registering and location updates result in time and network overhead. It increases migration time and ultimately total trip time of the agent. SPC protocol mainly concentrates on providing increased interaction efficiency.

Location management is an important issue in MA computing. It consists of location updates, searches and search-updates: An update occurs when a MA changes location. A search occurs when a MA/host wants to communicate with a MA whose location is unknown to the requesting agent/host. A search-update occurs after a successful search, when the requesting agent/host updates the location information corresponding to the searched MA. The goal of a good location management protocol should be to provide efficient searches and updates. The number of messages sent, size of messages and distance the messages need to travel, characterize the cost of a location search and update protocol. An efficient location management protocol should attempt to minimize all these quantities. Hence, a new protocol is required that would generate minimum overhead and be suitable for both global and local area networks.

This paper reports several location management protocols based on a hierarchical tree structure database. It also reports on the results of implementations carried out to evaluate the performance of proposed location management protocols for various call and mobility patterns. PMADE, developed at IIT Roorkee, is used as the development platform [11].

The rest of the paper is organized as follows: Section 2 presents an overview of PMADE. Section 3 reports on a system model for a distributed system with MAs. Section 4 presents proposed location management protocols and Section 5 presents implementation details. Evaluation results are shown in Section 6. Section 7 gives related works and Conclusions are given in Section 8.

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 autonomous Java agents and an Agent Submitter (AS) [10], which submits the MA on behalf of the user to the AH.

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 then 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.

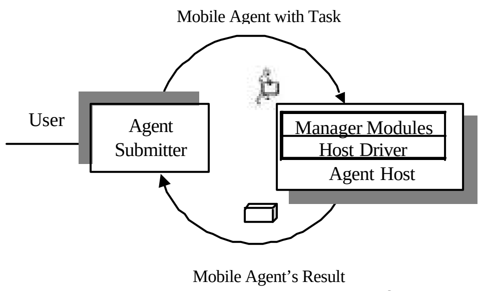

Figure 1. Block Architecture of PMADE

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 managers and their functions are provided in [11]. PMADE provides weak mobility to its agents and allows one-hop, two-hop and multi-hop agents [12].

3 System Model

In PMADE, agent location is based on some assumptions for the distributed environment, as shown in Figure 2. We have assumed that the global network environment is divided into network domains, regions (subnetworks) and agent hosts (local sites). Further, there is a domain management server (DMS) in each network domain which has information about all other DMSs in the global network. It also has information about all the regions in the network domain. It is responsible for maintaining uniqueness of names of regions, which are part of that network and helps to identify the region in which an agent is present.

Each DMS maintains a Domain Agent Database (DAD), for information about the current location of all agents which were created in that domain or transited though it. Every region maintains information about all AHs which are part of that region. An AH can be a member of an existing region or can start in a new region. In each region, a Region Agent Database (RAD) is present at an AH which runs at the gateway of a subnetwork. It contains location information about each agent which was created in that region or transited through it. This host acts as the Agent Name Server (ANS) [35], which manages the RAD. ANS is responsible for maintaining uniqueness of names of all MAs, created in that region. When a new agent is created, the user assigns a name to it by registering in the RAD of its birth region.

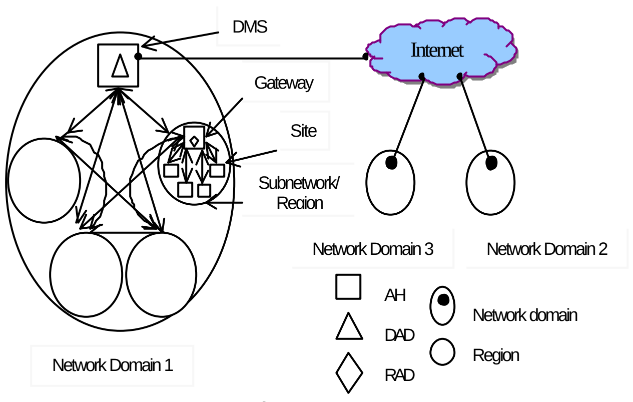

Figure 2. Structure of a Distributed Environment

Each entry of DAD of the form (A, FD, r) represents that agent A can be found in region r of the foreign network domain FD, or it has transited from that network domain or region. Each entry of RAD of the form (A, r, Nil) represents the region r where agent A was found or transited through it. Similarly (A, Nil, AH) represents an agent A which exists in that region at AH. For DAD and RAD, the primary key is the agent name A.

Agent migration from one network domain to another is always accomplished through the DMS. During inter domain migration the agent has to update location information in the DAD of the present domain and register in the DAD of the target network domain.

For intra region migration, it has to update its location information in the RAD of that region. This is an Intra Region Location Update. During inter region migration, the agent has to update the location information in the RAD of present region and register in the RAD of the target region, specifying the host in that region to which it is migrating. Any location protocol for MAs deals with three aspects: **name binding**, **migration** and **location,** each related to a particular phase in the agent's lifetime. We have defined four atomic operations on **DAD** and **RAD.**

- **bind** operation is performed when a name is assigned to a newly created MA, whose birth location is also stored. This operation causes the insertion of a new tuple in the database. As the agent name has to be unique, this operation fails if a tuple with the same name already exists in the database.

- **newloc** operation is performed when the agent changes its location, by migrating to a new one. This operation updates the tuple already present in the database.
- **find** operation is performed when an agent has to be located in order to interact with it. For a given agent name, this operation returns the current location of the agent.
- **unbind** operation is performed when an agent name is no longer used (i.e., the agent has been disposed off). This operation causes the deletion of the relative tuple from the database.

Mobile networks generally comprise of a static backbone network and a wireless network. There are three distinct sets of entities, namely MAs, MHs and fixed hosts. A host that can move while retaining its network connection is called a MH. The static network comprises of the fixed hosts and the communication links between them. Some of the fixed hosts, called base host (BH) are augmented with a wireless interface, and they provide a gateway for communication between the wireless network and the static network [8]. Due to the limited range of wireless transreceivers, a MH/MA can communicate with a BH. The geographical area covered by a region is a function of the medium used for wireless communication. Currently, the average size of a region is of the order of 1-2 miles in diameter. As the demand for services increase, the number of regions may become insufficient to provide the required grade of service. Region splitting can then be used to increase the traffic handled in an area without increasing the bandwidth of the system. A MA communicates with one BH at any given time. The BH is responsible for forwarding data between the MH/MA and the static network. Due to mobility, MH/MA may cross the boundary between two regions while being active. Thus, the task of forwarding data between the static network and the MH/MA must be transferred to the new regions. This process, known as handoff, is transparent to the mobile user [6, 1]. The initiative for a handoff can come from the MH or the BHs. Handoff helps to maintain an end-to-end connectivity in the dynamically reconfigured network topology.

4 Location Management Protocol

A location management protocol is a combination of a search protocol, an update protocol, and a search-update protocol. Only the location management protocols in the absence of a home location server (HLS) are discussed in the paper.

4.1 Logical Network Architecture (LNA)

A Global network consists of MAs, MASs and location servers (LSs). The logical network architecture (LNA) is a hierarchical structure (a tree with H levels) consisting of BHs and LSs. As shown in Figure 3, the BHs are located at the leaf level of the tree. Each BH maintains information about the agents residing in its region. The other nodes in the tree are called LSs. Each LS maintains information regarding MAs residing in its subtree.

Each communication link has a weight attached to it. The weight of a link is the cost of transmitting a message on the link. Let $l[src][dest]$ represent the link between nodes src and $dest$, and let $w(l)$ represent the weight (or cost) of link l. The cost depends on the size of the message, the distance between the hosts (agents), and the bandwidth of the link. For analysis purposes, we assume that, for all l, $w(l)=1$. Essentially, the cost metric is the number of messages sent.

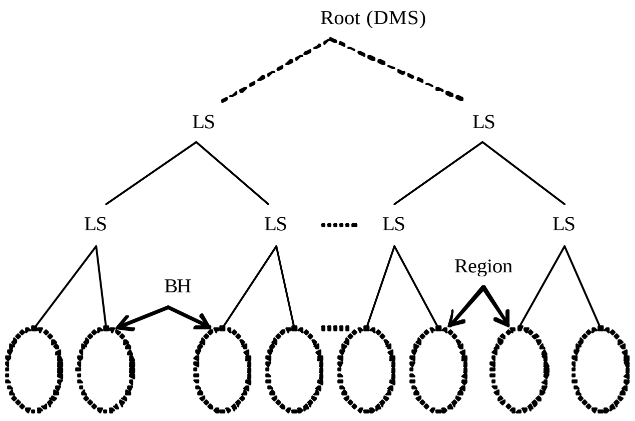

Figure 3. Logical Network Architecture

4.2 Data Structures

There is a unique “home” address for every MA. The home address is the identifier/name of the MA. The “physical” address of a MA might change, but its home address remains the same, irrespective of the agent location [8], [2]. Each LS maintains an address matching table that maps the home address to the physical address of the MAs residing in the subtree beneath it. Thus, the problem of location management basically focuses on the management of the address matching tables.

There is a location entry in a LS corresponding to an agent A, if it is in a region in the subtree under LS. If A moves to a region which is not in the subtree under LS, then the entry corresponding to A is updated at LS. All the nodes maintain location information using 3-tuples which have the following elements:

(i) MA identifier (id) (given by agent naming server),
(ii) Forwarding pointer destination (fp_dest), and
(iii) Time at which last forwarding pointer update took place (fp_time).

Each LS maintains a 3-tuple for each MA residing in the subtree beneath it, and each BH maintains a 3-tuple for each MA residing in its region. The default value of fp_dest and fp_time is NULL. If the fp_dest field of an agent A is NULL in LS L, then, A is not in a region in the subtree under L. Let us suppose that we are using a protocol which uses forwarding pointers for location updates. Let A reside initially in the region r. The BH of region r will have an entry (A, NULL, NULL). Let there be a LS L which maintains information about the agents residing in r. There will be an entry (A, r, NULL) corresponding to A at L. Let A move to a new region r', which is not a part of the subtree of L. Let t be the local time at the BH of r when change of location of A is recorded at BH. Let t' be the local time at L when the change of location of A is recorded at L. Thus, the location information of A will be (A, r', t') at L and (A, r', t) at BH of region r.

Note: The above data structures contain fp_time field to store time. The fp_time entry for a data structure on a node, say v, contains the local time at node v when the data structure was last modified. We will denote this time by t in the following. It should be noted that the correctness of the algorithms does not require the clocks at various nodes to be tightly synchronized.

4.3 Initial Conditions

It is assumed that, initially location information of the MAs is stored in the corresponding LSs, i.e., each LS has the correct location information for all the agents residing in the region in its subtree. Thus, the root LS should have the correct location information of all agents in the system. In Figure 4, nodes LS_{1-7} are LSs, and BH_{8-15} are BHs. There are two MAs A_1 and A_2. In the initial state, agent A_1 is in region 8, and A_2 is in region 12. Initially, the correct location information of agent A_1 will be available at LSs LS_4, LS_2 and LS_1. Likewise, the location information of A_2 will be available at LSs LS_6, LS_3 and LS_1. Thus, the location information of an agent is available at all the LSs located on the path from its current BH to the root.

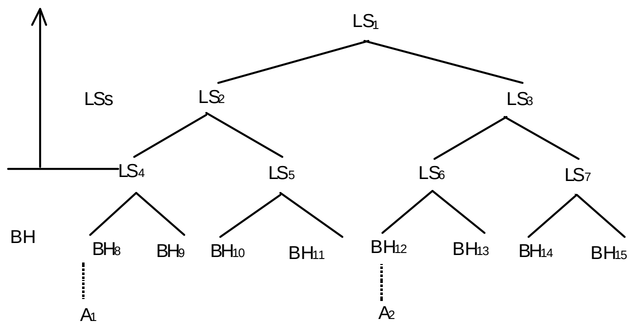

Figure 4. Example of Proposed Network Architecture

4.4 Update Protocol

The protocol for updating the location information at the LSs and the BHs, when a MA moves, is as follows: Let src and $dest$ be the source and destination regions, respectively. Let A be the identifier of the MA. Let t denote the local time at a node when a change in location of A is recorded at that node. The value of t will be different at different nodes. For this protocol we have considered three cases as follows:

Case 1: Single Updates (SU): In this the update takes place only at the BH of the source and destination regions. A forwarding pointer is kept at the source BH. The updated entry at the source BH becomes $(A, dest, t)$. An entry for agent A, $(A, NULL, NULL)$ is added at the destination BH. The location information at the LSs are not updated. The cost of update is zero, because there is no update message being sent.

Case 2: Full Updates (FU): Upon a move, apart from BHs involved (i.e., BH of the source and destination regions), location updates take place in all the LSs located on the path from the BH of the source and destination region to the root. Details are as follows:

Source region:

1. At the BH: For agent A, set $fp\_dest = dest$, and $fp\_time = t$. The updated entry for agent A at the BH becomes $(A, dest, t)$.
2. All LSs on the path from src to the root: The BH of src sends update message to these LSs. Upon receipt of the update message, the LSs update the entry for A to $(A, dest, t)$.

Destination region:

1. At the BH: An entry $(A, NULL, NULL)$ is added for agent A. If there was an old entry for A, it is overwritten by this new entry. At any node, there can be only one entry per agent.
2. All LSs on the path from $dest$ to the root: The BH of $dest$ sends update message to these LSs. Upon receipt of the update message the LSs create an entry. If there was an old entry, it is over written by this new entry.

Therefore, in an H-level tree, the update cost (the cost metric is number of messages) per move is $2(H-1)$. Suppose in Figure 4, agent A moves from BH_8 to BH_{14}. A forwarding pointer to BH_{14} will be kept at BH_8. BH_8 sends update messages to LSs LS_4, LS_2 and LS_1, and these LSs also create a forwarding pointer to BH_{14}. An entry for A_1 will be made at BH_{14}. BH_{14} sends update message to LSs LS_7, LS_3 and LS_1, and these LSs also make an entry for agent A_1.

Case 3: Limited Update (LU): Update in the location information takes place at a limited number of levels of LSs in the tree. Here, updates occur at $m(< H)$ lower levels of LSs on the path to the root. Updates at these LSs are similar to the FU. The LSs at levels higher than m are not updated. Thus, the update cost per move is $2m$. Let the value of m be chosen to be 1. Suppose in Figure 4, agent A_1 moves from BH_8 to BH_{14}. The forwarding pointer to BH_{14} will be kept at BH_8. BH_8 sends an update message to LS_4, and LS_4 maintains forwarding pointer to BH_{14}. An entry for A_1 will be made at BH_{14}. BH_{14} sends an update message to LS_7, who makes an entry for agent A_1.

4.5 Search Protocol

If agent A in region R wants to communicate with another agent $A^{'}$, A has to know the location of $A^{'}$. This requires that agent A search for agent $A^{'}$. As stated earlier, we do not make explicit use of HLSs for searches. The search process in the absence of a HLS is as follows.

1. If the BH of R has no location information for $A^{'}$, it forwards the location query to the next higher-level LS on the path to the root.
2. If the LS does not have any location information for $A^{'}$, it again forwards the location query to the next higher-level LS on the path to the root.
3. Repeat 1 & 2 until a LS which has location information for $A^{'}$ is reached.
4. If the location information (i.e., region identifier, say S) for $A^{'}$ is obtained, the location query is forwarded to the BH of region S. Agent $A^{'}$ will either be in region S or the BH will have a forwarding pointer corresponding to $A^{'}$.
5. If $A^{'}$ is in region S, the search is complete. Else, a chain of forwarding pointers is traversed until BH of the containing agent $A^{'}$ is reached.

4.6 Search-Update Protocol

Location management becomes more efficient if the location updates also take place after a successful search. For example, suppose there is an agent A that frequently calls $A^{'}$. It may be useful to update the location information of $A^{'}$ after a successful search, so that if A calls again, the search cost is likely to be small. The location information update takes place at the BH of the caller agent. Let agent A be the caller at the source, and $A^{'}$ be the agent to be searched at the destination host. Let the location of A and $A^{'}$ be K and $K^{'}$, respectively. The ollowing are the three cases to update location information upon a search.

Case 1: No Update (NU): There are no location updates, the fp_time field of the entry corresponding to $A^{'}$ at the BH on the search path is updated to the current time at the BH. The cost is zero. This is because the update of the time field could be done during the search process itself, and no additional message needs to be sent for this purpose. The update in fp_time is done to avoid purging of the forwarding pointer data at the BHs. The purge protocol is explained in the next section.

Case 2: Jump Update (JU): A location update takes place only at the caller agent's BH, i.e., BH of region K. The entry for A' at the BH of region K is set to (A', K', t), where t is the local time at the BH when the location information is updated. This update cost is 1. This is because only one message needs to be sent from BH of K' notifying the location information of agent A'.

Case 3: Path Compression Update (PCU): Upon a successful search, a location update takes place at all the nodes in the search path. All the LSs on the search path have the entry of A' updated to (A', K', t) where t is the local time at the LS when the location information is updated. All the BHs on the search path including the caller agent's BH have an entry of A' updated to (A', K', t), where t is the local time at the BH when the location information is updated. In Figure 4, let agent A_1 calls agent A_2. Suppose the location information of A_2 is available only at the LS_6, LS_3 and LS_1. Using the search protocol described previously, the search path will be $LS_8 \to LS_4 \to LS_2 \to LS_1 \to LS_{12}$. The location updates take place at LS_4, LS_2 and LS_1, and BH_8. The update cost is the length of the search path that is in this example is 4.

4.7 Purging Protocol

We need to periodically purge the stale forwarding pointers at the LSs and the BH. This should be done in order to (i) save storage space at the nodes, and (ii) avoid storing stale location information. We use a design parameter called Maximum Threshold Call Interval (MTCI) to decide whether to purge a forwarding pointer information or not. Let the current time be $curr\_time$. If fp_time ? $NULL$, and $curr\_time - fp\_time \geq MTCI$ then the entry for the agent is purged from the BH_i. If some other agents in the system which have recently used the forwarding pointer information of BH_i. In the LSs, if $curr\_time - fp\_time \geq MTCI$ for MA, the location entry for that agent is purged.

When SU and LU cases of update protocol are used, the forwarding pointers at higher level LSs do not get updated, and become stale. Thus, these forwarding pointers get purged periodically. However, some of the searches for the agent might reach the higher levels. If the LSs at the higher levels do not have information about the agent, the root has to broadcast to determine the location. To avoid this, the forwarding pointers at the LSs on the path to the root from the current BH must be updated periodically along with purging. The current BH of each MA achieves this by sending a location update message to the LSs on the path to the root.

Note: the fp_time value for an agent residing in the region will be NULL. So we are considering agents which are currently not residing in the BH's region and whose forwarding pointer information is stored at the BH.

5 Implementation and Performance Study

A tradeoff exists between the cost of updates (upon moves and searches) and cost of searches. The parameters that affect this tradeoff are (i) call frequency, and (ii) mobility. This paper evaluates the effects of mobility and call frequency on the cost of updates, search-updates and searches. As stated earlier, the location management protocol is a combination of a search protocol, an update protocol and a search-update protocol. The search protocol is the same for all location management protocols. A total of nine location protocols are obtained using the above protocols for updates and search –updates. We performed simulations to analyze the performance of the proposed location management protocols for various call frequency and mobility values. The location management protocols simulated were obtained by choosing one case of update protocol (say UX, where UX = SU, FU or LMU) and one case of search-update protocol (say SY, where SY = NU, JU or PCU). Thus, the location management protocol obtained is denoted as UX-SY.

5.1 System Model

We assume a binary tree as LNA for the simulations. The height of the tree is H. The number of LSs in the network is $2^{(H-1)} - 1$, and the number of BHs (or the number of regions) is $2^{(H-1)}$. Physical proximity of the region under the same LS is assumed. This will help in determining short and long moves. The height H was chosen to be 10 for the simulations (in existing networks like GSM or

Internet, the height may be small, i.e., 3 or 4. Since a binary tree was assumed for the simulations, we needed to have higher number of levels to have a sizeable number of regions in the network. However, similar performance trends are expected for other networks.) . Thus, there were 512 regions in the network.

The main aim of the work was to develop protocols for efficient searches and updates, i.e., reduce, the number of message due to location updates, without increasing the number of messages required for searches. Since the average message delay is likely to be small compared to the intervals between consecutive calls and moves, we ignore message delays, i.e., the location updates and searches are immediate.

Simulations were performed for two types of environments: (i) arbitrary agent moves and arbitrary caller agents, (ii) Short agent moves and a set of caller agents. In type (i), the agent can move to any location (region) and receive calls from any other agent in the network. This is not necessarily true in real life, but it gives a fair idea of the performance of the location management protocols in such extreme conditions. Type (ii) is closer to real life mobile environments. Agents are expected to make a lot of short moves to nearby destinations, and are expected to receive calls from a specific set of agents (e.g. family, business colleague's agents). It should be noted that we have assumed that the caller agents are immobile.

5.1.1 Call-mobility distribution for type (i)

The time between moves of an agent is assumed to follow an exponential distribution with a mean T_M. The destination region is chosen randomly among 512 regions. The time between calls for a caller agent is assumed to follow an exponential distribution with a mean T_C. The caller agent's region is chosen randomly from among the 512 regions.

5.1.2 Call-mobility distribution for type (ii)

Type (ii) consists of generating calls from a specific set of caller agents and short moves. One option to generate short moves is to put an upper limit on the length of the move, in terms of number of regions, and randomly vary the length of the move within the upper limit. For example, in Figure 4, if we keep an upper limit of 1, agent A_2 will be able to make the next move to region 11 or 13. But, our LNA just assumes proximity of regions which are under the same LS. Thus, a move from 12? 11 is not equivalent to the move from 12? 13.

Instead, we varied the number of levels of LSs, where location information will be updated due to the move, if FU were to be used. The number of levels can be varied between 1 to $(H-1)$. Level 0 is the BH level. The lesser the number of levels affected, the shorter is the length of the move. The probability distribution function of the length of the move in terms of height (number of levels) is $p(h)=\frac{2}{(H-1)(H-2)}*(H-h)$, shown in Figure 5.

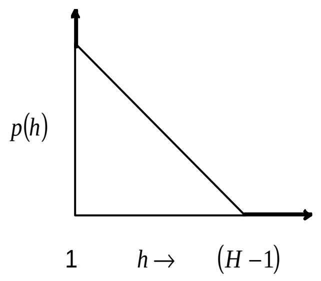

Figure 5. Probability Distribution Function $p(h)$

The cumulative distribution function (cdf) is as follows: $cdf(h)=\sum_{x=1}^{h} p(x)$. We randomly chose a height h based on the given probability distribution region as 2^h. Let the identifier of the current region (i.e., the source region) be $curr$. Knowing the height h and $curr$, one can easily determine the ancestor of $curr$ at level h in the binary tree. Let it be ls. Knowing ls, the set of destination region possible is $\{ls*2^h, ls*2^h+1, ..., ls*2^h+2^h\}$. A destination region is chosen randomly [7]. This is in coherence with the assumption of proximity of regions under the same LS. The time between moves of an agent is assumed to follow an exponential distribution with mean T_M.

In type (ii), for each MA, caller agents were chosen from a specific set of regions. The size of the set was chosen to be 20. The set was chosen arbitrarily and regions were not necessarily neighbouring. The calls always originated from those regions. The time between calls for an agent is assumed to follow an exponential distribution with mean T_C.

Purge was performed periodically at every MTCI units of time. The value of MTCI was chosen to be 10 units of time.

5.2 Cost model

As stated earlier, the cost of transmitting a message over any link is 1. Therefore, the cost metric is essentially the number of messages required for each operation (search, update and search-update). Thus, the cost of an update is the number of LSs which update the location information of the agent. The cost of a search is the number of LSs and BH visited before locating the agent. Cost of a search-update is the number of LSs which update the location information of the agent.

The performance parameter is the aggregate cost, defined as the sum of average update cost, average search cost, and the average search-update cost.

5.3 Experimental Results

Simulations were performed to analyze the performance of the various location management protocols. Results were obtained for the two types of environments, Type (i) and (ii) The values of T_C and T_M were both varied from 1 to 15 units of time. The value of T_C was changed to vary the time interval between two successive calls. The value of T_M was changed to vary the mobility of the agent. For example, $T_C = 1$ and $T_M = 1$ characterizes a communication intensive and ultra mobile environment.

Type (i): The average length of a move was 170, and the average distance of a call was also 170. It was observed that the SU-PCU protocol outperforms all the other protocols for all values of T_M and T_C. Therefore, we have only plotted the curves for SU-PCU. The protocols using FU and LMU suffered due to the high cost of updates upon each move. SU-NU protocol suffered due to very high search costs. Because the caller agents were arbitrary, SU-JU protocol did not perform well in the update upon a successful search was not helping in reducing the search cost.

Figure 6 shows the aggregate cost for the SU-PCU protocol as a function of T_C for different values of T_M. As seen in the figure, the aggregate cost increases, the calls become infrequent, and the agents might have moved to new locations, requiring new searches. Thus, the reduction in search cost by path compression is not very effective. It is also seen that the rise in aggregate cost with T_C is higher for lower values of T_M. Lower the value of T_M, higher is the mobility, and thus the search cost will be higher. At high values of T_M, the difference in the aggregate costs due to different values of T_M is low. This is because as T_M increases, the agent movement reduces. Beyond a point, increasing T_M does not affect the aggregate costs, and the curves converge to a single curve.

Type (ii): The average length of a move was 9, and the average distance of a call was 110. It was observed that the SU-PCU and the SU-JU protocols outperformed all the other protocols for all values of T_M and T_C. In contrast to Type (i), SU-JU performed well, because there is a specific set of caller agents. Thus, the jump update at the caller agent's host is more effective in reducing the search cost, because the caller agent calls the agent again with a higher probability than in Type (i) environment. Figure 7 shows the aggregate cost for the SU-JU protocol and the SU-PCU protocol as a function of T_C for different values of T_M. As can be seen, SU-JU performs better than SU-PCU in high-communication and low-mobility and low-communication and high-mobility environments. In these environments, the search cost for SU-PCU and SU-JU are comparable. Since the search-update cost is same as the search cost for SU-PCU is simply twice the search cost. On the other hand, the average search-update cost for SU-JU is less than or equal to 1. It should be noted that in cases where the caller agent has correct information of the destination agent (host), the search-update cost is zero. Thus, the aggregate cost of SU-JU is lower than SU-PCU. SU-PCU performs better for other values of T_M and T_C because the search cost for SU-JU becomes large compared to SU-PCU.

Figure 8 shows the average search cost for the SU-JU protocol and the SU-PCU protocol as a function of T_C for different values of T_M. As can be seen, SU-PCU has a much lower search cost than SU-JU. The search cost of SU-JU is slightly lower than SU-PCU for high communication and low-mobility environments.

It was noticed that performing search-updates significantly reduced the search and aggregate costs. For the assumed LNA, it is seen that the SU-PCU protocol performs better than the other protocols for most values of T_C and T_M. It is expected that SU-PCU will perform well in other network models too. For models with different costs associated with each link, we expect the other proposed protocols to perform well, and sometimes better than the SU-PCU protocol (for some values of T_M and T_C).

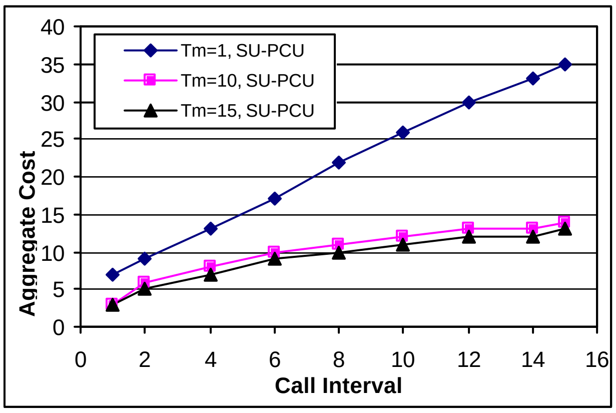

Figure 6. Aggregate Cost for type (i)

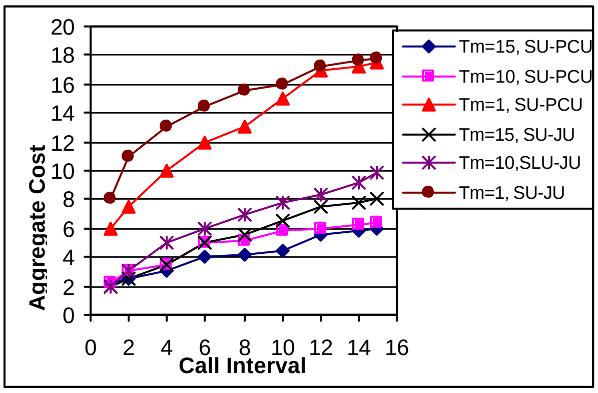

Figure 7. Aggregate Cost for Type (ii)

Note: Tm and T_M are identical

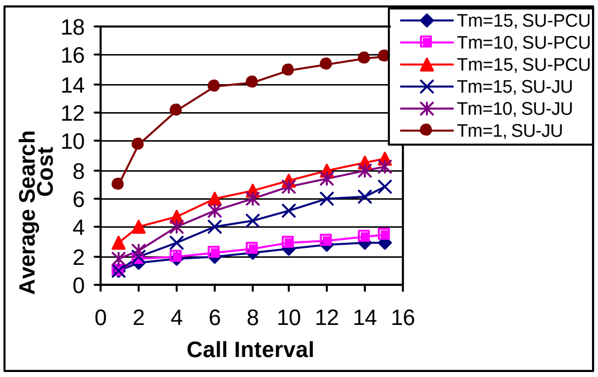

Figure 8. Search Cost for Type (ii)

6 Evaluation Results

Our tests took place in a 10/100 MBps switched LAN that connects 850 workstations and personal computers, and is used by about 500 hundred researchers and students. We ran PMADE equipped with the developed protocols on several P-4, 3 GHz machines. The AS node and agent host nodes have 256 MB main memory, while the LS (Agent Host at the root) has 512 MB. We used the j2sdk 1.4.1 Java Virtual Machine with native thread support.

First, we tested the capacity and performance of our storage backend. The LS (root agent host) was able to hold up to $4*10^6$ entries before the system ran out of memory {Figure 3}. This means that, given an extreme of $8*10^8$ Internet users (*NUA estimates there were more than 605.60 million users online in the Internet on September 2002*, [36]) each running 100 MAs simultaneously, about 20,000 LSs would be required to keep all entries. This is less than 0.0057% of the hosts in the Internet, according to ISC estimates (*ISC estimates there were more than 350,000,000 hosts in the Internet in January 2005*, [37]) at the time of writing.

Next, we let up to eight agents/ASs send requests concurrently. Table 1 gives the response rates we measured in tests with a single agent/AS, sorted by request type. Secured registration was slowest, as could be expected. However, this type of request is required only once per agent. In this test the LS handled about 400 agent lookup requests per second, which includes processing overhead at the AS (ASs start requests parallel threads). Figures 9 and 10 show the response rates we measured for concurrent lookup requests with one to eight agents/ASs. With two or more agents/ASs, the response rate jumps from about 210 requests per second to roughly 332, and remains more or less stable at this mark (with one agent/AS, the agent host has idle time, with two or more it becomes congested). Table 2 shows how response times develop with an increasing number of agents/ASs. With about 2626 agents/ASs, requests take longer than 13 seconds to process, which causes network connections to time out for few agents.

Type	Length	Mean Time	Requests/s	Action of
Lookup	32 bytes	4.7 ms	313	Location search
Registration secured	431 bytes*	11 ms	15	Init
Update	103 bytes*	1 ms	150	Location Update
Register unsecured	103 bytes*	5 ms	270	LS

Table 1. This figure shows the size of request packets, and average processing time of the searching service with one agent/AS, by request type. *The lengths marked which might differ depending on the length of the stored location reference.

Number of Agent Host	1	2	4	8	16	32	64	128	256	512
Number of Agents /AS	1	2	4	8	16	32	64	128	256	512
Response time	301 ms	572 ms	1 sec, 500 ms	2 sec, 3ms	4 sec, 3 ms	6 sec, 6ms	10 sec, 510 ms	21 sec, 21 ms	53 sec	154 sec, 100 ms

Table 2. Agent Response Time (It includes Agent Migration Time, Agent Decryption Time, User/Agent Authentication Time, Result Encryption and Packaging Time).

Number of LSs	1	2	3	4
Agent Migration time when LSs are active (ms)	145.7	147	148.1	150
Agent Migration time when LSs are not active (ms)	140	140	140	140

Table 3. Effect of LSs on Agent Migration Time (size of agent is considered 10.203 KB)

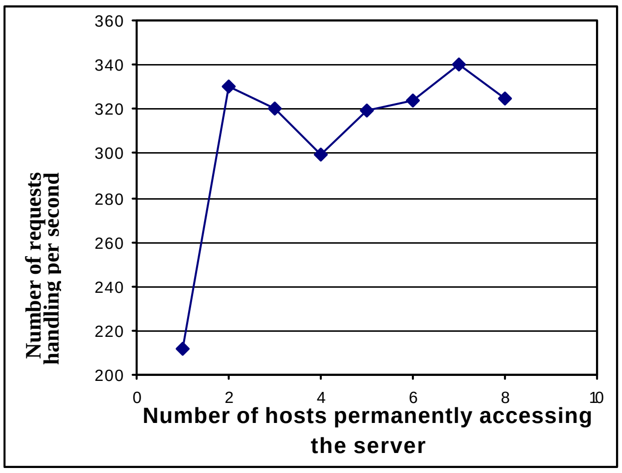

Figure 9. This figure shows the average number of requests that can be handled by the LS, depending on the number of ASs/agents that query the server concurrently. A circle mark represents the mean of a set of 6000 measured values.

We also measured the impact of the location service (search and update) integration on the migration time of MAs in the PMADE. Without location service integration, we measured an average of 140 milliseconds per migration of a simple benchmark agent, compared to with location service (search & update), which we consider tolerable {Table 3}.

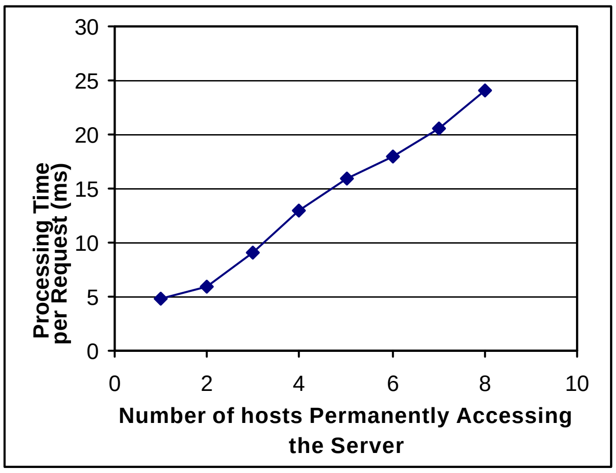

Figure 10. This figure shows how response times of the LS develop with an increasing number of ASs/agents that queries the LS concurrently. A circle mark represents the mean of a set of 6000 measured values.

7 Related Work

In the literature, several approaches [2, 13, 20] described the problems of locating MAs. Di Stefano *et al.* [2] propose the use of LSs, where each server is responsible for all agents in its domain. Each agent has a home server that can be derived from a location-specific part of the agent's name. Whenever the agent enters a new domain, the servers responsible for the old and new domain, as well as the home server are updated. Lookups for agents not in the local domain start at the home server. Lazar et al. [13] used DNS like name solving to find the latest location of a MA in specific domain gateways. One approach [21] used location transparent routing in MASs that merges name lookups with routing. However, these approaches cannot solve the performance problem in case of large scale of MAs. Pitoura [23] suggested an efficient hierarchical scheme for locating highly mobile users.

There are also approaches [22, 20] that use automating directory services for tracking MAs. But these approaches cannot give an optimal solution for the lack of a suitable cost formulation. The problem lies in that they do not consider the performance overheads caused by a single location update operation, O. Shehory [24] proposed a scalable agent location mechanism which considers the overall system overheads. The *Globe* [32] system is a distributed directory designed to support billions of references to mobile objects. However, the authors acknowledge that their hierarchical approach is not scalable enough to fulfil this goal due to the enormous storage demands and relatively large number of requests that must be handled by higher-level directory nodes. In order to overcome these problems, they propose to use the first bits of an object's globally unique handle as the identifier of directory subnodes, which share the load on their directory level. This approach equals the one we chose in order to provide scalability.

Although several major MASs have developed some locating mechanisms [16, 17], they did not consider mobility management for large-scale MAS. So their applications are quite limited. Several other schemes for locating MAs, and routing messages among them were proposed in the past, e.g. [27, 30, 28, 33, 7]. Some of these approaches assume that there is a logical network of connected MASs [27, 7, 30], and routing of agents or messages is done along the edges of this graph. In the case of [7], the graph must actually be a balanced tree. However, any approach that builds on a particular network topology makes sense only if MASs are implemented on the network layer as part of routers. Most of the contemporary MASs are implemented on the application layer, though. From the perspective of the application layer, the Internet is a fully connected graph. Hence, a logical topology that is layered on top of the physical structure of the Internet creates undesired and unnecessary routing overhead. The logical routing may even run counter to the actual physical routing.

Additionally, the approaches described in [27, 7] put the burden of setting up and maintaining the logical structure on administrators; a job that, in our opinion, quickly spirals out of control. In particular, the approach described in [27] is not scalable. Each node in the tree has storage requirements proportional to the number of MAs managed by it, and update rates proportional to the rate of migrations that start or end in its subtree. In particular, the root node has to cope with all of the traffic. Protocols based on forwarding pointers and dynamic for shortening of pointer chains are proposed, e.g. in [29, 31]; they are also used in Mole for the purpose of orphan detection [26]. The disadvantage of this approach is its lack of robustness, a single broken or timed-out link makes the agent unreachable. The Mobile Object Workbench [28] supports a hierarchical directory service for locating objects that moved. Wojciechowski et al [33] use a combination of registering and forward references. Forward references act as a cache. In case of a miss, the central server is asked to forward the message, and the invalid forward reference is updated. The approach let each agent hold a mostly accurate contact list of other agents it knows. An agent's location can be found by consulting these contact lists dynamically maintained by some neighbour agents. This scheme can only work under the assumption of low movement frequency (hence, less updates of an agent's contact list). So, it is not suitable for a highly dynamic scenario. A scalable hierarchical protocol is not only the main concern of our research presented here, we have developed a novel location management protocol in which a set of protocols implemented and an agent dynamically can select as per requirement to establish communication link to the communicating party. Selection of protocol depends on the network topology.

8 Conclusion

In this paper we have presented several location management protocols based on a hierarchical tree structure database. These location management protocols use one combination of search, update and search-update protocols throughout the execution. Simulations were carried out to evaluate the performance of the various location management protocols. It was noticed that performing search-updates significantly reduced aggregate costs. For the assumed LNA, it is found that the SU-PCU (combination of single updates and path compression search-update) protocol performs better than the other protocols for most values of communications rate T_C and mobility T_M. It is expected that SU-PCU will perform well with other network models too. We have also applied these protocols in the real life application implementation developed on PMADE. It is found that overhead generated by them does not affect the actual agent response and migration times.

References

[1] Bar-Noy, I. Kessler, and M. Sidi, "Mobile Users: To update or not to update?" ACM-baltzer J. Wireless Networks Vol. 1, No. 2 175-186, July 1995.

[2] Di Stefano and C. Santoro, "Locating Mobile Agents in a Wide Distributed Environment," IEEE Transaction on Parallel & Distributed Systems, 13(8): 844-864, Aug. 2002.

[3] R. Tripathi, T. Ahmed and N. M. Karnik, "Experiences and Future Challenges in Mobile Agents Programming," Microprocessors and Microsystems, 25(2): 121-129, April 2001.

[4] E. Pitoura and G. Samaras, “Locating Objects in Mobile Computing,” IEEE Transaction on Knowledge and Data Engineering, 13(4): 571-592, 2001.

[5] G. P. Picco, “Mobile Agents: An Introduction,” Microprocessors and Microsystems, 25(2): 65-74, April 2001.

[6] Kimberly Keeton, Bruce A. Mah, Srinivasan Seshan, Randy H. Katz, and Domenico Ferrari, “Providing connection oriented network services to mobile hosts,” in the Proceedings of the USENIX Mobile & Location-Independent Computing Symposium, Cambridge, Massachusetts, August 2-3, 1993, pp. 1-32.

[7] M. Spreitzer and M. Theimer, “Providing Location Information in a Ubiquitous Computing Environment,” in Proceedings of the 14th ACM symposium on Operating systems principles, Asheville, North Carolina, USA, Dec. 05 - 08, 1993, pp. 270-283.

[8] R. B. Patel, “Design and Implementation of a Secure Mobile Agent Platform for Distributed Computing,” PhD thesis, Department of Electronics and Computer Engineering, IIT Roorkee, India, 2004.

[9] R. J. Flower, “The complexity of using forwarding address for decentralized object find,” in Proceedings of the fifth annual ACM symposium on Principles of distributed computing, Calgary, Alberta, Canada Aug. 11 - 13, 1986, pp. 108-120.

[10] R.B. Patel and K. Garg, “A New Paradigm for Mobile Agent Computing,” WSEAS Transaction on Computers, Issue 1, Vol. 3, pp. 57-64, Jan. 2004.

[11] R.B. Patel and K. Garg, “PMADE – A Platform for mobile agent Distribution & Execution,” in Proceedings of 5th World MultiConference on Systemics, Cybernetics and Informatics (SCI2001) and 7th International Conference on Information System Analysis and Synthesis (ISAS2001), Orlando, Florida, USA, July 22-25, 2001, Vol. IV, pp. 287-293.

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

[13] Sashi Lazar, Ishan P. Whereon, Deepinder P. Sidhu, “A Scalable Location Tracking and Message Delivery Scheme for Mobile Agents,” in Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98), Palo Alto, CAUSA, June 17-19, 1998, p.243-249, IEEE Computer Society.

[14] V. Roth and J. Peters, “A Scalable and Secure Global Tracking Service for Mobile Agents,” in Proceedings of the 5th International Conference on Mobile Agents (MA2001), Atlanta, Georgia, USA, G. Picco (Ed.), LNCS 2240, Springer Verlag, 2001, pp. 169-181.

[15] W. S. E. Chen, C.W.R. Leng, “A Novel Mobile Agent Algorithm,” in Proceedings of the first International workshop on Mobile Agents (MA97), K. Tothermel, R. Popecu-Celetin, (eds.) LNCS 1219, Springer Verlag, Berlin, Germany, April 7-8, 1997, pp. 162-173.

[16] Karjoth, G., Lange D.B., and Oshima, M., “A Security Model for Aglets,” IEEE Internet Computing 1(4): 68-77, July-Aug. 1997.

[17] Lange, D.B., and Oshima, M., “Programming and Deploying Java™ Mobile Agents with Aglets™,” Addison-Wesley, ISBN 0-201-32582-9, Aug. 1998.

[18] K. Decker, K. Sycara, and M. Williamson, “Middle-agents for the Internet,” in Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI-97), Nagoya, Aichi, Japan, Aug. 23-29, 1997, pp. 578-583.

[19] D. Kuokka and L. Harada, “Matchmaking for Information Agents,” in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI95), Montreal, Quebec, Canada, Aug. 20-25, 1995, pp. 672-679.

[20] H. Maass. “Location-aware Mobile Applications Based on Directory Services”. Mobile Networks and Applications 3 (1998) 157–173, Baltzer Science Publishers BV, 1998.

[21] W. Van Belle, K. Verelst and T. D’Hondt, “Location Transparent Routing in Mobile Agent Systems Merging Name Lookups with Routing” in Proceedings of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 99) Dec. 20-22, 1999, Cape Town, South Africa, pp. 207-213.

[22] P. Stanski, D. Thompson, M. Nzama, A. Zaslavsky, N. Craske, “Automating directory services for mobile agent tracking,” in IEEE Global Telecommunications Conference

(GLOBECOM ’98), Nov. 8-12, 1998, Sydney, Australia, pp. 1947 –1951.

[23] Evaggelia Pitoura and Ioannis Fudos; “An efficient hierarchical scheme for locating highly mobile users,” in Proceedings of the 7th international conference on Information and knowledge management (CIKM '98), Bethesda, Maryland Nov. 3-7, 1998, pp. 218 – 225.

[24] Onn Shehory. “A Scalable Agent Location Mechanism,” in Proceedings Intelligent Agents VI: Agent Theories, Architectures, and Languages: 6th International Workshop (ATAL’99), Orlando, Florida, Usa, July 15-17, 1999, Lecture Notes in Artificial Intelligence, Intelligent Agents VI, M. Wooldridge and Y. Lesperance (Eds.), pp. 162-172.

[25] Tie-Yan Li, Kwok-Yan Lam, “An optimal location update and searching algorithm for tracking mobile agent,” in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy, July 15-19, 2002, pp. 639-646.

[26] Joachim Baumann and Kurt Rothermel, “The Shadow Approach: An orphan detection protocol for mobile agents,” in Rothermel and Hohl [34], pp. 2–13.

[27] L. Bernardo and P. Pinto, “A scalable location service with fast update responses,” in IEEE Global Telecommunications Conference (GLOBECOM ’98), Nov. 8-12, 1998, Sydney, Australia, pp. 2876–2881, 1998.

[28] Michael Bursell, Richard Hayton, Douglas Donaldson, and Andrew Herbert, “A Mobile Object Workbench,” in Rothermel and Hohl [34], pp. 136–147.

[29] Luc Moreau, “Distributed directory service and message routing for mobile agents,” Technical Report ECSTR M99/3, Department of Electronics and Computer Science, University of Southampton, U.K., Nov. 1999.

[30] Amy L. Murphy and Gian Pietro Picco, “Reliable communication for highly mobile agents,” in Proceeding of the First International Symposium on Agent Systems and Applications, and Third International Symposium on Mobile Agents (ASA/MA ’99), Oct. 3-6, 1999, Palm Springs, CA, USA, pp. 141–150.

[31] Peter Sewell, Pawel Wojciechowski, and Benjamin Pierce, “Location-independent communication for mobile agents: a two-level architecture,” Technical Report 462, Computer Laboratory, University of Cambridge, U.K., April 1999.

[32] M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum, “Locating Objects in Wide-Area Systems,” IEEE Communications Magazine, pp. 104–109, Jan.1998.

[33] Pawel Wojciechowski and Peter Sewell, “Nomadic Pict: Language and Infrastructure Design for Mobile Agents,” in Proceeding of the First International Symposium on Agent Systems and Applications, and Third International Symposium on Mobile Agents (ASA/MA ’99), Oct. 3-6, 1999, Palm Springs, CA, USA, pp. 821–826.

[34] K. Rothermel and F. Hohl (eds), in Proceedings of the Second International Workshop on Mobile Agents (MA’98), Stuttgart, Germany, Sept. 9-11, 1998 Lecture Notes in Computer Science, 1477, Springer Verlag, Berlin Heidelberg.

[35] D.B. Terry, “Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments,” Ph.D. thesis, University of California, Berkely, 1985. Available as UCB/CSD Tech. Rep 85-228 and as Xerox PARC Tech. Rep. CSL-85-1.

[36] http://www.nua.com/surveys/how_many_online).

[37] http://www.isc.org/ds