Tracker: A Universal Location Management System for Mobile Agents*

George Samaras, Constantinos Spyrou, Evaggelia Pitoura+, Marios Dikaiakos

Department of Computer Science, University of Cyprus
CY-1678 Nicosia, Cyprus, <cssamara@cs.ucy.ac.cy>

+ Department of Computer Science, University of Ioannina
GR 45110, Ioannina, Greece, <pitoura@cs.uoi.gr>

ABSTRACT

This paper presents TRAcKER, a distributed "location management" middleware which has the ability to manage the location of mobile agents that travel independently the Internet in search of useful information. A major goal for this middleware is to be flexible and effective, to respond to the demands posed by the environment it aims to assist: an environment that deals with users that are accessing the Internet via static or mobile units of limited resources (e.g., palmtops, handheld devices, WAP-phones) and utilizing mobile agents that are moving "autonomously" and asynchronously around the Internet with small, medium and in many cases high moving frequency. The mechanism we propose is modular, simple, of low overhead and able to serve all the Java based mobile agent platforms. In addition, is quite generic able to dynamically incorporate various location mechanisms.

1. INTRODUCTION

Mobile agents provide many benefits [5,6,24] for the development of new generation Internet systems, such as great capabilities for distributed Internet system programming (otherwise networking programming), in which there is the need for different kinds of integrated information, for example, telemedicine [20], monitoring and notification and encapsulating artificial intelligence techniques, security and robustness [7,10,17,21]. Also the mobile agents paradigm assures satisfactory performance for distributed access to Internet databases, for distributed retrieving and filtering of information, and for minimizing network workload [7,8,10,12,13,17,21,22]. Finally, mobile agents are a software technology that has been proved very effective in supporting the asynchronous execution of user's requests, weak connectivity and disconnected operations, the dynamic adaptation to the various connection modalities regarding user connectivity, etc. [7,10,11,13]. Today the technology of mobile agents can resolve many problems that arise in the context of mobile devices and wireless connectivity, such as limited resources, small computational power, communication interference and the high cost of the wireless connection to the fixed network [9,24]. For example, the agents during their journey can collect useful information from different nodes of the network while transferring at the same time the workload to these nodes, which usually have large computational power. In this way they significantly relief and aid the user which might be using a device with small computational power and limited resources.

The ability to locate mobile agents as they roam autonomously on Internet is crucial for the wider adoption of mobile-agent technologies in a variety of applications [14,15]. Of particular importance is the capability of interrogating a roaming agent to retrieve information of interest, in an almost real-time frame. Nevertheless, most existing Java-based mobile agent platforms (i.e., Aglets [2], Concordia [3], Grasshopper [4] and Voyager [1]) currently do not provide a comprehensive, efficient or effective location management system. The support of these platforms for mobile-agent location ranges from limited to non-existent.

In this paper we present ***TRAcKER***, a distributed "location and tracking management" middleware, which has the ability to manage the current location of mobile agents traveling on Internet. The TRAcKER mechanism is flexible, effective, and meets the requirements that arise in the context of Wide-area networks hosting large numbers of autonomous mobile agents roaming with various frequencies to support the access of users connected through static and mobile devices, possibly with limited resources (e.g., palmtops, handheld devices, WAP-phones). The TRAcKER middleware is a modular, simple, low-overhead system that can be easily integrated with all Java-based mobile agent platforms. Furthermore, it is designed so as to accommodate different agent-location algorithms and different topologies with minimal cost. In this paper we present an implementation and evaluation of the TRAcKER system, where the mobile-agent location services were established upon two location-management algorithms used for GSM systems [14]. Our experiments show that the specific hierarchical schemes suggested in the wireless telephony, and implemented by these two location algorithms, are inefficient and ineffective.

The remaining of this paper is organized as follows: Section 2 presents the location problems encounter when mobile agents roam the Internet. Section 3 presents a study of the most popular mobile agents platforms giving special emphasis on the way these systems provide agent location management support. In Section 4, we describe related work in mobile telephony and link it to mobility of agents on Internet.Section 5 presents the TRAcKER location management mechanism, its design and its features. Section 6 describes how the system TRAcKER can serve any

* This work has been partially funded by Cyprus Research Promotion Foundation.

Java-based mobile agent or moving object platform. In Section 7 experimental results demonstrating the TRAcKER system as well as related insights are presented. We conclude in Section 8.

2. MOBILE AGENT LOCATION MANAGEMENT PROBLEM

To fulfill their assigned tasks, mobile agents move from one node to another [5, 6, 24]. Often, mobility is ad-hoc; that is, the mobile agents move autonomously and asynchronously, without following a predefined route. Thus, even the creator of an agent, needs to keep track of the agent's current location, in order to contact the agent and get access to its data and resources. Furthermore, other agents, called client agents, may need to contact the agent. In this case information retrieval is almost impossible, even if the agent follows the predefined route. The location problem becomes even more difficult when agents need to locate and cooperate with agents in other execution environments. Thus there is a need for a service with the ability to locate any type of mobile agent at any time from anywhere. Such a mechanism (i.e., TRAcKER) must be generic, flexible, independent of the agent platform and able to dynamically accommodate location algorithms to support various moving and invocation patterns. The following scenario demonstrates these issues (Figure 1):

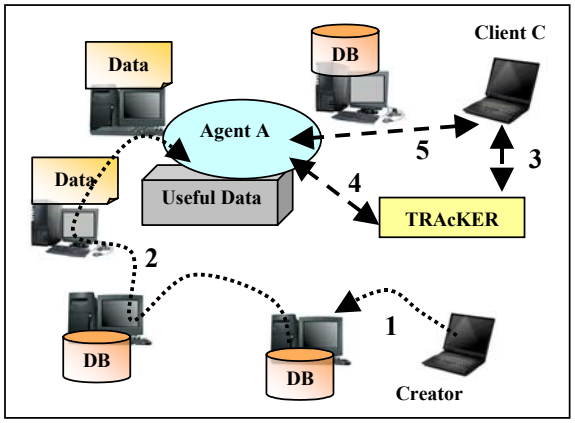

Figure 1: A Specific Scenario of the Problem of Locating Mobile Agents

1. The creator submits agent A to perform some task.
2. The agent moves to different machines and environments.
3. Client C needs to communicate with the agent. The client is unaware of the agent's location therefore it contacts TRAcKER.
4. TRAcKER knows the agent's location and communicates with it in order to inform the client.
5. The client communicates with the agent.

Without TRAcKER, the client would not know the agent's location, so contacting it would be impossible, unless the agent returned back to the client or the creator. The TRAcKER must be able to assist the communication between the client and the moving agent even if the agent has moved in between the client's request and the TRAcKER's response and independently of the speed of the movement. We focus on the performance of the mechanisms in correlation with the moving behavior of agents in the Internet. Our goal is to develop a location architecture that would be adaptable and accommodate a variety of location algorithms and thus able to operate on both low and high mobility, without consequences on its operation and performance.

Two things are mainly needed to effectively attack such a problem:(a) to study and evaluate the current location technology supported by the exiting mobile agent platforms, and (b) to understand the specific requirements and peculiarities in having mobile agents or moving objects roam around the internet.

3. CURRENT STATE OF THE ART: LOCATION MANAGEMENT BY JAVA-BASED MOBILE AGENT PLATFORMS

In our work, we examine the most popular mobile agents platforms (i.e., Aglets [2], Concordia [3], Grasshopper [4] and Voyager [1]) giving special emphasis on the way these systems provide agent location management support. The advantages and disadvantages of the current methodologies are identified and studied. The positive features are effectively adopted while the negative ones are in some sense eliminated. Great attention is given to the performance behavior of the mechanism in correlation with the moving frequencies of mobile objects expected on Internet.

IBM's Aglets. In the Aglets platform [2], for a client system to communicate with an agent, it needs to know the exact location (IP address) currently hosting the agent, along with the agent's identity: Communication between two aglets is conducted via the **proxy** abstraction**.** The creator of an agent maintains in its possession an **original proxy** of that agent. Any request addressing that agent and made through its **original proxy** can be answered normally only if the agent resides locally (Figure 2).

```
AgletProxy originalproxy =
getAgletContext().createAglet(getCodeBase(),
ClassName,args);

URL destination= new
URL("atp://cs126.cs.ucy.ac.cy:434");
AgletProxy originalNewproxy =
originalproxy.dispatch(destination);
Case 1 – Calling the Agent through the "original"
proxy
Message msg=new Message("test");
originalNewproxy.sendAsyncMessage(msg);
Case 2 – Calling the Agent through the "copy"
proxy
AgletID ID = originalproxy.getAgletID();
AgletProxy
copyproxy=getAgletContext().getAgletProxy(destinati
on,ID);
Message msg=new Message("test");
copyproxy.sendAsyncMessage(msg);
```

Figure 2: Creator's communication pseudo code example (Aglets).

If the agent is dispatched explicitly by the client system to a particular location (Figure 3), the dispatcher client can store an original proxy for the new location (originalNewproxy in Figure 3), and through this it can still send requests to the agent at its remote location. Similarly, a client that knows the agent's name and new location, can request a remote proxy for that agent (copyproxy in Case 2 of Figure 3) and use it to communicate with the remotely located agent.

Mitsubishi's Concordia. Inter-agent communication in Concordia [3] is implemented via events. For an agent to be able to raise and listen to events, it has to register with an **Event Manager.** Subsequently, the sender agent connects with the event manager and posts an event with its request. As soon as the event manager and receives a particular event, it notifies all agents, which have registered an interest on that specific event (see Figure 3). Event if an agent has moved, it will still be able to receive requests from the event manager. An agent can register with many event managers. However, in order for one agent to reach another agent, it has to connect with an event manager – in which the recipient agent has been registered to, i.e., it has to know the event manager location.

```
TestAgent originalproxy = new TestAgent(arguments);
Itinerary itinerary = new Itinerary();
itinerary.addDestination(new
Destination("rmi://agentDestination/ AgentReceiver",
"method"));
originalproxy.launch();
EventManagerConnection eventMC = new
EventManagerConnection();
eventMC.makeConnection("agentDestination",false);
EventPost eventQueue = (EventPost) new
EventQueueImpl(this);
eventMC.registerAll(eventQueue);
eventMC.postEvent(new EventRequest(arguments));
```

Figure 3: Creator's communication pseudo code example (Concordia).

Therefore, for a client system to communicate with a roaming Concordia agent, it has to know the exact location of an event manager with which the roaming agent is registered, and the types of events the agent is listening for.

```
ServiceInfo info = this.createService(classname,
codebase, place, arguments);
TestAgentP originalproxy=new
TestAgentP(info.getIdentifier(). toString());
originalproxy.move(destination);
TestAgentP copyproxy = new TestAgentP
(info.getIdentifier(). toString(), destination);
copyproxy.callMethod(arguments);
```

Figure 4: Creator's communication pseudo code example (Grasshopper).

```
TestAgentP copyproxy=new
TestAgentP(info.getIdentifier().toString(), destination);
copyproxy.callMethod(arguments);
```

Figure 5: Client's communication pseudo code example (Grasshopper).

IKV++'s Grasshopper. Grasshopper [4] supports communication with agents in exactly the same manner as Aglets technology (Figures 4 and 5). Therefore, under Grasshopper we still need to know the exact location and the identity of a roaming agent, in order to communicate with it.

ObjectSpace's Voyager. In Voyager [1], there is also the notion of a proxy. Only the creator of the agent owns the original proxy. Any request to the agent, if made through the original proxy, will reach the agent regardless of the agent's location (Figure 6). The difference with Aglets lies in the way the proxy operates in Voyager.

```
ItestAgent originalproxy = (ITestAgent)
Factory.create(classname,new Object[]{});
originalproxy.callAgent();
```

Figure 6: Creator's communication pseudo code example (Voyager).

Any client that wants to send a request to the agent must look up the agent in the name service of the node in which the agent has been registered to (Figure 7) and which he must "somehow" know of [27]. An agent can register to many name services during its trip. The name service gives back to the client the copy proxy through which the client can send its request. The request will reach the agent no matter of its location. The use of a name service creates a centralized location management mechanism. According to this mechanism, all the agents are registered to a specific name service, which they inform about their movements. This way a client can communicate with an agent by knowing this central node.

```
ITestAgent copyproxy =
(ITestAgent)Namespace.lookup("//cs163:8000
/Agent");
copyproxy.callAgent();
```

Figure 7: Client's communication pseudo code example (Voyager).

The problem with Voyager is that the client must know the exact centralized location to which the agent has been registered and its name. Furthermore, being centralized, it is quite inappropriate for distributed systems, not only for reliability reasons but also in terms of performance. Chart 1 shows the degradation on performance when multiple users perform intensive querying.

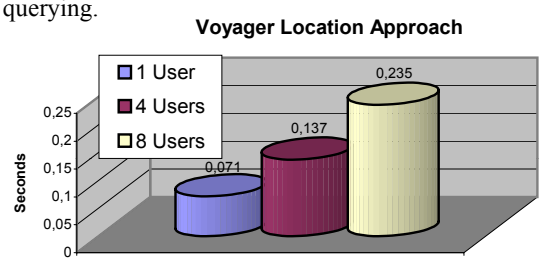

Chart 1: Performance Degradation of Voyager's Location Management Mechanism

3.1 Comparisons and Discussion

Table 1 presents an overview of the four agent platforms regarding agent location management.

MA Platform	Communication with an Agent
Aglets [2]	Agent's Location and Identity
Concordia [3]	Event Manager's Location and event's name in which the agent is registered, or central Event Manager's Location and event's name
Grasshopper [4]	Agent's Location and Identity
Voyager [1]	Agent's Location and Name, or Central node location (or a specific name service) and agent's name.

Table 1: Technologies comparison for communication with an agent.

From this table it is apparent that to communicate with a roaming agent we need to know its location, or the location of a centralized (or the specific) manager (e.g., name service) with which the agent registers its position. Our goal is to relieve the user from the need to know the location of an agent. *Therefore our goal is to provide an effective mechanism capable of managing and supplying the location of agents at any time based solely on their unique id.*

4. RELATED WORK: GSM MOBILE PHONE USERS VS MOBILE AGENTS

Mobility of entities exists in other environments, such as in wireless telephony where mobile users move from one cell to another, while maintaining connectivity [14]. Mobile-phone mobility is supported by a distributed architecture deployed to locate mobile phones. As we will see, these mechanisms are inadequate for mobile agents because:

1. Mobile-phone users move relatively slowly. The need to update location information arises only during hand-offs. This occurs only when a user crosses from one cell to another. The size of a cell is usually quite large. The diameter ranges from 100's meters to a few kilometers. In contrast, a mobile agent changes locations very fast often remaining at a site for a few milliseconds only.
2. The wireless telephony infrastructure is quite structured when compared to Internet. A telephone number can uniquely identify the home location of a mobile phone. The mobile agent id or IP address cannot provide the same information.
3. The size of the wireless infrastructure, measured in terms of cell-numbers,is minimal if compared to the enormous size of Internet , measured in terms of Internet hosts.

These three parameters define the type of the mechanisms/algorithm that can be used in each one of these environments. Obviously what is effective and efficient for GSM phone users is not for the mobile agents environment. For example, the GSM HLR-VLR [14] mechanism cannot be used as is by mobile agents because of the second point mentioned above, while the hierarchical approaches [14] proved to be very ineffective and inefficient for mobile agents[1]. Preliminary experimental results[2] show miss hit up to 16% and very low performance for medium to high moving frequencies, see Charts 2 and 3 (in the charts moving frequency is express in sleep time). Similarly, while a predefined hierarchical location management structure is adequate in the GSM (for example) environment it might not be appropriate for the Internet. In effect, a distributed but more flexible and dynamic infrastructure and more efficient location algorithms might be more appropriate for the mobile agents environment.

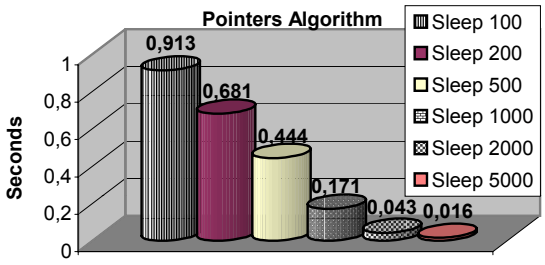

Chart 2: Access Time for randomly locating an Agent.

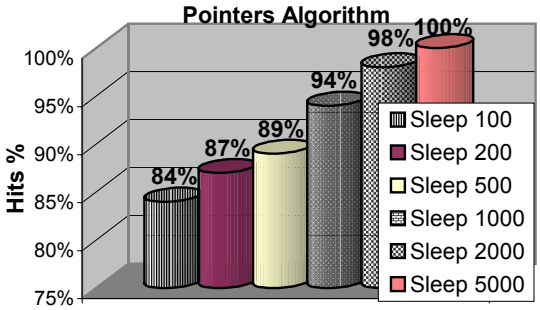

Chart 3: Miss Hits Ratio.

5 THE TRACKERS DISTRIBUTED ARCHITECTURE

We consider mobile agents that roam around the Internet being hosted by various agent execution environments. This implies the need for a distributed mechanism (see Figure 8) that is simple, flexible, scalable, light, dynamically configured and agent-platform independent.

5.1 The Middleware's Architecture

In a nutshell, TRAcKER: (a) enhances mobile agents with the ability to notify the system of their departure or arrival (via a MASIF type of interface) and (b) provides distributed components with the ability to manage and keep track of agent locations. We call such agents and component instances ZoneAgent and ZoneRegistry class respectively.

Effectively, by extending the abstract class ZoneAgent we create a TRAcKER enabled agent. During creation this agent is registered to the

[1] For the definition and implementation of the most popular hierarchical location algorithms, i.e. "Pointers" and "Exact Location" [14] see section 7

[2] The testbed configuration is presented in Section 7

ZoneRegistry of the local node by using the appropriate registration method (namely the *"registry.UpdateHierarchy(Agent's Name, Agent's URL"* see figure 10)". Then the local ZoneRegistry informs its parent about the presence of the new agent. This is done in accordance with the current location algorithm implemented in the TRAcKER system.

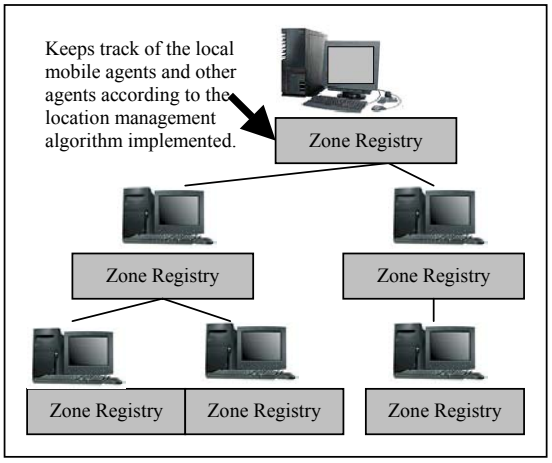

Figure 8: A Network forming a hierarchical TRAcKER-enable Configuration.

During a move of a TRAcKER-enabled agent from one node to another, the agent itself informs the destination-node's ZoneRegistry about the arrival. This is the responsibility of the agent and the appropriate method is part of the ZoneAgent's definition (namely the method *postArrival(),* see figure 10). In turn, the destination node's ZoneRegistry, informs its parent (root direction) about the agent's arrival in accordance with the current location algorithm. As soon as this informing finishes and again in accordance with the current algorithm all the appropriate nodes are informed about the departure of the agent in order to delete it. This is done, for example, by using the ZoneAgent's method *preDeparture(), s*ee figure 10. Now from any node a user can ask the local ZoneRegisrty for the location of an agent by simply using the agent's name.

All the ZoneRegistries are using the same Name and port at all nodes that are part of the TRAcKER tree. This makes it easy for a TRAcKER-enabled agent to find and communicate (i.e., be registered or be de-registered) with the ZoneRegistries. This info is passed via an installation object.

System Components

TRAcKER consists of two agents, the **ZoneRegistry** and the **ZoneAgent**. It also includes an installation agent called the RegistryServer.

The RegistryServer agent is responsible for (a) the activation of the agent execution environment if not already activated. In our case the VoyagerServer since Voyager is the implementation platform and (b) the creation of the ZoneRegistry at the specific site. It also supplies the ZoneRegistry with information about the identity of its father and which port of the node the Voyager Server will observe (i.e., watch for incoming agents or calls). The initial distributed configuration of the system can be decided by the user/administrator and be expanded dynamically as new sites are added to it.

```
public class ZoneRegistry
//(if needed extends the system class)
{
    //Father
    ZoneRegistry father
    //Information about agents
    Data
   //Constructor
   public ZoneRegistry(father URL)
   { FindFather(father URL) }
    public void FindFather(father URL)
   {father=findFather(father URL) }
    public void UpdateHierarchy
    (agent Name, location)
   {  //Update my data  }
    public void RemoveAgent(agent Name)
   { //Remove the agent from my data    }
    public agent URL LookUpAgent(agent Name)
   { //Check my data and return agent's URL}
 }
```

Figure 9: ZoneRegistry Abstract Class.

ZoneRegistry: The ZoneRegistry is responsible for the location management of mobile agents residing locally in the node or in one of its children-nodes. It is also responsible to inform its ancestor (father) in the hierarchy, about these elements (nodes) and of any changes that happen to these elements that are caused by the moves of the agents that reside locally or in any of its descendant nodes. It is also responsible for serving any request regarding an agent's location. The ZoneRegistry via the location mechanism implemented, learns the location of a specific agent and informs the user. The object oriented nature of this agent allows the utilization of various location mechanism via inheritance (see Figure 9 and section 7). By overriding, for example, the "*UpdateHierarchy*" method, the "*RemoveAgent*" method and the "*LookUpAgent*" method we can create the various hierarchical location management mechanisms. In our case study we first implemented the "Pointers"-ZoneRegistry location mechanism then via inheritance the other mechanisms (see section 7).

ZoneAgent: The ZoneAgent is an abstract class (Figure 10), i.e., it cannot exist by itself as an object. It has to be extended and adjusted to the user demands by using object oriented programming. This agent provides the operations related to location management. In particular, it implements the registering and deregistering methods from a node(/ZoneRegistry) to another due to movement. These operations are enabled via the "*postArrival"* and "*preDeparture"* methods. It is important to mention that this task was previously the responsibility of the user. Extending this agent and overriding the methods "*postArrival"* and "*preDeparture"* can materialize the various location mechanisms. Any user agent that needs location management must extend this abstract class.

```
public abstract class ZoneAgent
{ //Agent's name
  Name
 //Local ZoneRegistry
 ZoneRegistry registry
 //Local URL
 my URL

 public ZoneAgent(name)
 {  Name = name
    //Register to the local ZoneRegistry
    registry.UpdateHierarchy(Name, my URL)
 }
 //After the arrival to the new Zone
  public void postArrival()
  {  registry.UpdateHierarchy(Name, my URL) }
 //Before the Departure of the old Zone
  public void preDeparture()
 {   registry.RemoveAgent(Name) }
}
```

Figure 10: ZoneAgent's Abstract Definition.

5.2 Additional Features and Issues

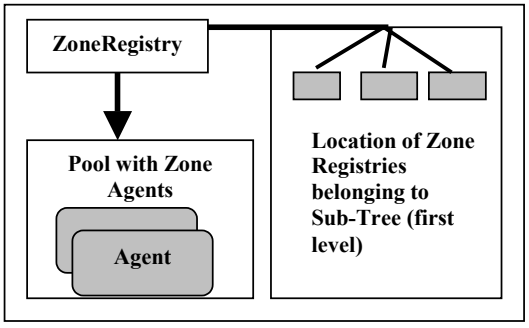

Figure 11: TRAcKER's Structure at Participating Nodes.

5.2.1 Simple

It consists of only three java objects that are also mobile agents. The objects have simple structure that can be easily extended. In fact the only thing needed for an application is instead of extending the system agent (i.e., the abstract class provide by the agent platform) to just extend the ZoneAgent provided by the TRAcKER system. As long as the visiting site is TRACKER-enabled then the agent can be found and interrogated at any time and from anywhere and by anyone.

Making a node TRAcKER-enabled is quite simple as well, see section 5.2.2 below. Figure 11 shows the role of each object in a TRAcKER enabled node.

5.2.2 Scalable

In order for a node to participate in a location aware configuration and benefit from its location management the following steps are needed: (a) install the agent execution environment (in our case is the Voyager platform/server) and (b) install the ZoneRegistry. All the ZoneRegistry needs, in order to be activated and to be added to the Configuration/Hierarchy, is the location of a node, which already participates in the Hierarchy - so the ZoneRegistry can be its child, and the port of the node, which to observe. The user on the other hand in order to create an agent for a particular task – who's location will be managed by TRAcKER–he only has to extend the ZoneAgent abstract class.

On the other hand, the effectiveness of the TRacKER once a number of nodes are added mainly depends on the specific location mechanism utilized by the system.

5.2.3 Dynamic

The components of the TRAcKER system are not just java objects but mobile agents as well. Assuming that the agent execution environment is already installed at the nodes of interest we just need to send the mobile agent ZoneRegistry to these sites. In that manner, the needed infrastructure is configured automatically.

5.2.4 Low Overhead

TRAcKER is quite light-weight. We evaluate whether the overhead of TRAcKER degrades the capacity of the system significantly. We measure the overhead as the difference in the number of mobile agents a particular configuration can host with and without the TRAcKER system. Our experiments indicated that this overhead is small. In particular, utilizing as the location mechanism the "hierarchical exact location" algorithm, which is one of the heaviest ones in terms of message exchange [23], and having the agents move continuously and very often (i.e., sleep time at a node 100 ms) the above overhead measures to only 39 agents (358 agents with TRAcKER vs. 397 without TRAcKER). While these results are very encouraging more rigorous evaluation involving different configurations, location algorithms and metrics is required for tangible results.

5.2.5 Flexibility

Flexibility here is defined in terms of how easily TRAcKER can be utilized with different location mechanisms. The modular and object oriented nature of the agents of the systems allows great degree of such flexibility. In fact, to utilize a particular location mechanism, we just need to extent the agents ZonerRegistry and ZoneAgent appropriatelly. By extending the ZoneAgent and overriding the methods postArrival and preDeparture and by extending the ZoneRegistry and overriding the "UpdateHierarchy" method, the "RemoveAgent" method and the "LookUpAgent" method we can materialize the various hierarchical location management mechanisms. In [23] we have used this system to materialized most hierarchical location mechanisms very effectively. In fact certain mechanisms due to their similarities where extended from others already created. In [23] we used this system to materialized most hierarchical location mechanisms very effectively. In fact certain mechanisms due to their similarities where extended from others already created

A higher degree of flexibility can be achieved via the notion of TaskHandlers [18,19] to replace the current location mechanism with another on-line. Of course this requires for the TRAcKER system to be inactive for the update period. This will also make it easier for us to have different location algorithms active at the same time. Of course this requires the hosts to be divided into regions, each with a tree-like organization

effectively creating a "forest" of TRAcKER hierarchies. Preliminary research has shown this to be quite feasible

6. THE GENERIC NATURE OF THE MECHANISM

We have implemented and tested the TRAcKER system over the Voyager platform (i.e., the various classes are implemented as Voyager mobile agents). Irrespective to that, its implementation is generic and able to support all the java-based mobile agent platforms.

Indeed the TRAcKER mechanism is implemented using the JAVA programming language and is thus portable. The mechanism consists of two classes, the ZoneRegistry and the ZoneAgent. These classes have been deliberately implemented by using pure JAVA and without any relation or dependency to the specifics of any known Mobile Agents Platform, such as Aglets, Concordia, Grasshopper or Voyager etc.

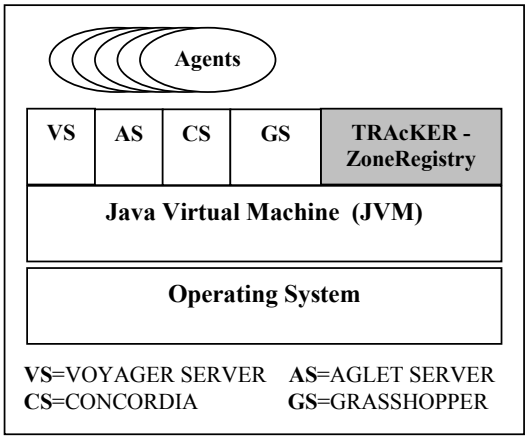

Figure 12: The TRAcKER System as a Middleware Running on Top of JVM.

The only thing needed is for the platform to include a local version of the ZoneAgent class and for any agent that needs location management to be an extension of this class/agent. However, there is a problem with the dynamic installation of the ZoneRegistry. The problem exists because we need this object/agent to run outside of any agent execution environment (see Figure 12). This can be solved as follows. The TRAcKERRegistry can be encapsulated in a configuration agent (e.g., the RegistryServer Agent) of the current platform and be installed by the agent once the agent is delivered to the targeted site. Note that the encapsulating agent could be of any platform.

As a result the TRAcKER mechanism has the ability to manage the location of any mobile agent independent of its platform as long as these agents are extensions of the local version of the ZoneAgent. As an added feature of this approach is that the TRAcKER mechanism runs using an individual JAVA thread, not depending on the local mobile agent platform server(s), therefore the various activities that are used for location management, do not effect the platform's operation.

It is worth noting, however, that while this middleware allows the location of an agent even by an agent of a different platform it does not offer agent-to-agent communication.

6.1 MASIF Compliance

The TRAcKER system is almost (!) compliant with the Mobile Agent System Interoperability Facility (MASIF, also called MAF an acronym of the original proposal Mobile Agent Facility) specification [26,28]**.** The system currently follows a MAFFinder-like Interface, however we expect its newer version to be fully MASIF compliant.

The MAFFinder interface provides methods for maintaining a dynamic name and location database of agents, places and agent systems. The TRAcKER system has been created for Mobile Agent Location Management; as a result it provides these methods only for agents. Contrary to MAFFinder, TRAcKER provides not only the local interface but also flexible protocols governing the cooperation among the various ZoneRegistries (MAFFinders). In addition, TRAcKER solves one other limitation of MASIF; via the notion of ZoneAgent allows greater flexibility in implementing location mechanisms especially those that required the active involvement of the moving agents (e.g., mechanisms based on forwarding pointers [14, 15]).

7. IMPLEMENTATION AND PERFORMANCE RESULTS

We have implemented various location mechanisms [14] within TRAcKER. In this section, we describe the experimental platform and present results for two of these mechanisms, two hierarchical ones.

We present experimental results for these mechanisms and make an analysis to demonstrate:

- The applicability and flexibility of the TRAcKER middleware. For example, in this case study, we first created and implemented one of the two hierarchical mechanisms and then via inheritance the other one. In fact, the ZoneAgent for both hierarchical mechanisms is the same; only the ZoneRegistry needs to be specialized
- Initial results on the performance of location mechanisms for mobile agent architectures.
- That the GSM approaches are not appropriate for the mobile agents environment.

The Location Mechanisms. In all mechanisms, each node (that is, the ZoneRegistry at the node) maintains information about the agents it currently hosts. Depending on the location mechanism, this information is maintained in additional nodes (ZoneRegistries).

We consider two hierarchical architectures: the exact-location and the pointers mechanisms [14]. In both of them, each internal (i.e., non-leaf) node N contains information for the agents hosted at all nodes in the subtree rooted at node N. In the exact-location mechanism, each internal node maintains the exact location of these agents, whereas in the pointers mechanism, it maintains a pointer to its descendant node (child) that has information about the location of the agent.

7.1 Experimental Results

The testbed configuration consists of 10 Pentium III 866MHz workstations with 128 MB RAM running MS Windows 2000. The hierarchy (which was created

dynamically) is the same for all tests (like the tree in Figure 8). For each test, we create 100 mobile agents on various tree nodes (machines) that continuously move from one node to another. Each agent remains on each node for a predefined time interval called sleep time. We varied this interval among the values 100, 200, 500, 1000 and 5000 milliseconds. We perform 100 search (location look-up) requests for randomly selected agents on randomly selected nodes. These search requests are made one at a time, that is we make a search request, receive an answer, verify the answer by communicating with the agent and then continue with the next search request.

For each of the location mechanisms we perform two types of tests.

First Performance Test

Each time we take the "by first time successful search time average" and the "agents finding success percentage"

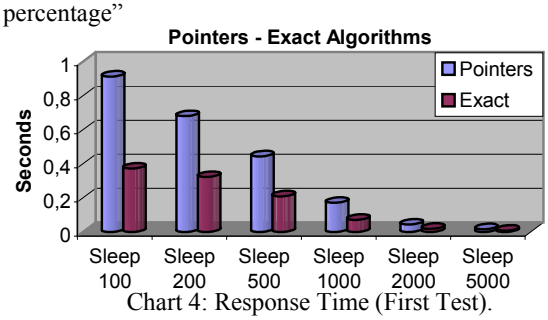

Chart 4: Response Time (First Test).

For the two mechanisms we have two figures. Chart 4 shows the average time of a successful search request for a randomly selected agent. Chart 5 shows the percentage of each run where the agent is located successfully via the first search. That is, we call a miss hit if the first attempt by TRAcKER to locate the agent fails. Note that the system will always return an answer, this answer is consider correct if the agent is indeed at the returned location.

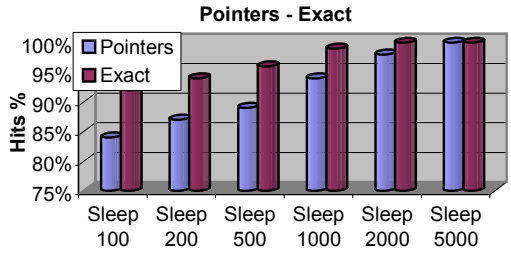

Chart 5: Hit Ratio (successful searches) (First Test).

As we can see from Chart 4, the exact location mechanism is faster than the pointer location mechanism, because less nodes are contacted to locate an agent. For smaller "sleep times" (100, 200 and 500 milliseconds) the agents move more frequently, thus the mechanisms consume more time in updating the TRAcKER tree.

As we can see from Chart 5, the exact mechanism has more hits than the pointer mechanism. For smaller "sleep times" (100, 200 and 500 milliseconds) the agents move more frequently, therefore it is obvious that the mechanisms face greater risk in giving wrong information regarding the agent's location. The two-tier with proxy has better hit ratios because it takes advantages of the Voyager platform.

Second Performance Test

The difference of this performance measurement test with the previous one is the following: instead of taking the "by first time successful search time average", we make as many searches as needed to find the agent or make at most10 unsuccessful searches (which ever comes first). The same node makes the 10 (possibly) unsuccessful searches. We take the average time for these measurements. Chart 6 presents the average time.

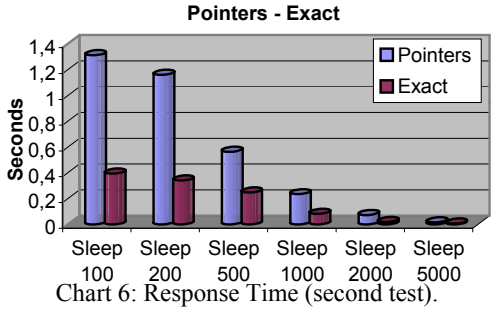

Chart 6: Response Time (second test).

The difference from the first type of test (i.e., comparing Charts 4 and 6) is that in Chart 6 we observe larger time difference between the two hierarchical mechanisms. This is due to the fact that the exact mechanism has better hit ratio (e.g., for sleep time 100, the hit ration is 92% - see Chart 5) which means that it will make more than one effort to find the agent, only for the 8% of the measurements. The pointer mechanism has worse hit ratio (e.g. for sleep time 100, the hit ration is 84% - see Chart 5) which means that it will make more than one effort to find the agent, for the 16% of the measurements.

Discussion

From the experiments, we notice that for high sleep times, namely 5 seconds and above most mechanisms achieve 100% hit ratio. Obviously for environments such as mobile telephony (e.g. GSM wireless networks), where movement is infrequent due to the size of the wireless cell [14] these location algorithms are quite sufficient. However, for Internet-based environments where very often applications require a large number of relocations with sleeping times ranging from low to high to very high (e.g., 200 milliseconds) hierarchical algorithms are inadequate. New types of non-centralized location algorithms are needed for this environment.

8. CONCLUSIONS

In this paper we have evaluated the most popular mobile agents platforms (i.e., Aglets [2], Concordia [3], Grasshopper [4] and Voyager [1]) giving special emphasis on the way these systems provide agent location management support. To remedy the identified weaknesses we designed and implemented TRAcKER, a distributed location management middleware for mobile agents. The main characteristics of the TRAcKER system are (a) simple; it consist of three java

objects, (b) flexible; any hierarchical or chain location mechanism can be easily created, and this was demonstrated by the materialization of two hierarchical location management algorithms (c) scalable; any node can participate in the system by just incorporating just one of these objects, which could be send to it dynamically and in real time, (d) light: the overhead, measure in the number of active agents, of utilizing the architecture versus the opposite is insignificant, and (e) dynamically configured in that the system can be dynamically configures and set up by just sending these objects which are mobile agents at the various sites. In fact our experimental test bed we set up in that manner.

In addition, in demonstrating the various features of the system we also showed the limitations of certain GSM type of location schemes. In fact, the hierarchical schemes used in the wireless telephony, and implemented in our experiments, proved quite inefficient and ineffective.

REFERENCES

[1] ObjectSpace Voyager [tm] Technical Overview. Web Site <http://www.objectspace.com/voyager/ whitepapers/VoyagerTechOview.pdf>

[2] Aglets Workbench, by IBM Japan Research Group. Web site: <http:/aglets.trl.ibm.co.jp>

[3] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young and B. Peet. *Concordia: An Infrastructure for Collaborating Mobile Agents*. Lecture Notes in Computer Science, 1219, 1997. <http:// www.meitca.com/HSL/Projects/ Concordia/>.

[4] M. Breugst, I. Busse, S. Covaci and T. Magedanz. *Grasshopper A Mobile Agent Platform for IN Based Service Environments*. IEEE IN Workshop, Bordeaux, France, May 10-13, 1998.

[5] J. E. White, *Mobile Agents*, General Magic White Paper, www.genmagic.com/agents ,1996.

[6] Colin G. Harrison, David M. Chessm, Aaron Kershenbaum. *Mobile Agents: are they a good idea?* Research Report, IBM Research Division

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

[8] G. Samaras, M. Dikaiakos, C. Spyrou, A. Liberdos, *"Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment"*, The Joint Symposium ASA/MA'99. First International Symposium on Agent Systems and Applications (ASA'99). Third International Symposium on Mobile Agents (MA'99), pp. 50-64, USA, 1999.

[9] George Samaras and Paraskevas Evripidou, Evangelia Pitoura, *"Mobile-Agents based Infrastructure for eWork and eBussiness Applications"*, The eBusiness and eWork Conference, 2000, (to appear).

[10] E. Pitoura and G. Samaras, *"Data Management for Mobile Computing"*, Kluwer Academic Publishers, ISBN 0-7923-8053-3, 1998.

[11] Constantinos Spyrou, George Samaras, Evangelia Pitoura, Evripidou Paraskevas *"Mobile Agents for Wireless Computing: The Convergence of Wireless Computational Models with Mobile-Agent Technologies"*, Journal of ACM/Baltzer Mobile Networking and Applications (MONET), special issue on "Mobility in Databases & Distributed Systems ", 2000. (to appear)

[12] Barron C. Housel, George Samaras, David B. Lindquist, *"WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment"*, Journal of ACM/Baltzer Mobile Networking and Applications (MONET), special issue on "Mobile Networking on the Internet", 3(4): 419-431, December, 1998.

[13] Samaras, G., A. Pitsillides, *"Client/Intercept: a Computational Model for Wireless Environments"*, Proc. 4th International Conference on Telecommunications (ICT'97), Melbourne, Australia, April 1997.

[14] E. Pitoura, and Samaras, G., *"Locating Objects in Mobile Computing"*. IEEE Transactions on Knowledge and Data Engineering Journal (TKDE) 2001.

[15] E. Pitoura and I. Fudos, *"An Efficient Hierarchical Scheme for Locating Highly Mobile Users"*, in Proceedings of the 6th ACM International Conference on Information and Knowledge Management, (CIKM98), November 1998. pp 218--225.

[16] Papastavrou S., G. Samaras, E. Pitoura, *"Mobile Agents for WWW Distributed Database Access"*, IEEE Transactions on Knowledge and Data Engineering (TKDE) 2000.

[17] Papastavrou S., G. Samaras, E. Pitoura, "*Mobile Agents for WWW Distributed Database Access"*, Proc. 15th International Data Engineering Conference, p228-237, Sydney, Australia, March 1999.

[18] Evripidou P., Samaras G., Pitoura E., Christoforos P., *"The PacMan Metacomputer: Parallel Computing with Java Mobile Agents"*, Journal FGCS special issue on JAVA in High Performance Computing, 2001. (to appear)

[19] Christoforos P., Samaras G., Pitoura E., Evripidou P., *"Parallel Computing Using Java Mobile Agents"*, 25th Euromicro Conference, Workshop on Network Computing, September 1999. Also technical report TR-99-7, University of Cyprus, February 1999.

[20] Andreas Pitsillides, George Samaras, Marios Dikaiakos, Eleni Christodoulou, "*DITIS: Collaborative Virtual Medical team for home healthcare of cancer patients*", Conference on the Information Society and Telematics Applications, Catania, Italy, 16-18 April 1999.

[21] M. Dikaiakos, D. Gunopoulos, *"FIGI: The Architecture of an Internet-based Financial Information Gathering Infrastructure"*. In Proceedings of the 1st International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems. IEEE-Computer Society, pages 91-94, April 1999.

[22] Y. Villate, A. Illarramendi, and E. Pitoura, *"Data Lockers: Mobile-Agent Based Middleware for the Security and Availability of Roaming Users Data"*, In CoopIS Israel, September 2000.

[23] Constantinos Spyrou. *"Creation of a Mobile Agents Location Management Mechanism"*, Master thesis supervised by George Samaras, Computer Science Department, University of Cyprus, June 2001.

[24] D. B. Lange and M. Oshima. "*Seven Good Reasons for Mobile Agents"*. Communications of the ACM, 42(3):88-91, March 199

[25] M. Dikaiakos, M. Kyriakou, G. Samaras, *"Performance Evaluation of Mobile-agent Middleware: A Hierarchical Approach"*. In Proceedings of the 5th IEEE International Conference on Mobile Agents, J.P. Picco (ed.), Lecture Notes of Computer Science series, vol. 2240, pages 244-259, Springer, Atlanta, USA, December 2001.

[26] Crystaliz, General Magic, GMD Fokus, and IBM. "*Mobile Agent System Interoperability Facility" (MASIF – specification)*. Available through ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf, November 1997.

[27] Object Space Inc, *"VOYAGER ORB 3.2 Developer Guide"*, 1997-1999.

[28] E. Di Pietro, A. La Corte, A. Puliafito, and O. Tomarchio. *"Extending the MASIF Location Service in the MAP Agent System"*. In IEEE Symposium on Computer Communications (ISCC2000), Antibes (France), July 2000.