A System Based on Mobile Agents to Test Mobile Computing Applications*

Sergio Ilarri\textsuperscript{a}, Eduardo Mena\textsuperscript{a}, Arantza Illarramendi\textsuperscript{b}

\textsuperscript{a}IUS Dept., Univ. of Zaragoza, María de Luna 1, Zaragoza, 50018, Spain
Phone: (+34) 976 76 23 40, Fax: (+34) 976 76 19 14

\textsuperscript{b}LSI Dept., Univ. of the Basque Country, Apdo. 649, 20080 San Sebastián, Spain

Abstract

The interest in mobile computing applications is on the rise and they are expected to be a great source of revenue in the forthcoming years. However, not only developing wireless data services but also testing them is a real challenge. In fact, in many circumstances the use of tools to test these services is the most convenient choice.

In this article, a system that can be used to test mobile computing applications in a distributed environment is presented. The proposed system, based on mobile agents, presents a range of interesting advantages, such as: 1) It allows to test a real wireless data service on a simulated environment; 2) it can be extended to adapt it to the simulation needs; 3) it allows mixing real and simulated elements transparently to the tested data service; 4) it allows distributed testing, which increases the scalability and accuracy of the results; and 5) it is capable of testing services where the mobile devices execute (mobile) agent-based applications. Experiments that highlight the flexibility and accuracy of the proposed system and experiments testing real applications using the system are also shown.

Preprint submitted to Elsevier 11 February 2009
1 Introduction

The attractiveness of computing services that are available anywhere and any-time has given rise to an intense research and development of mobile computing applications. However, there is a great pressure over providers of wireless data services to be the first in providing a new service. Therefore, a mechanism that allows developers to test their prototypes in a timely and non-expensive fashion is required. Testing mobile computing applications in a real environment is an overwhelming task (sometimes, as time-consuming as developing them), mainly due to the distributed and dynamic nature of the underlying mobile network infrastructure. Besides, deploying a data service in a real network for testing purposes is not usually an option. Among other reasons, the following can be highlighted:

- Many users with real wireless devices following certain behaviors could be needed to test how the data service performs in that situation. Besides the effort required, it would be impossible to replicate the same situation.
- Developing data services using a real, large-scale wireless network can imply a high cost in wireless communications (e.g., GSM, GPRS) and mobile

* This work was supported by the CICYT project TIN2007-68091-C02-02. The authors are grateful to Yolanda Villate for testing the Locker Rental Service using the proposed testing system. We also thank the anonymous reviewers for their useful comments.

Email addresses: silarri@unizar.es (Sergio Ilarri), emena@unizar.es (Eduardo Mena), jipileca@si.ehu.es (Arantza Illarramendi)
devices.

- Some extensions to the real network functionality could be required. For example, simulating that certain extra information (e.g., context-aware information) is sent as part of the usual signaling traffic in the network could be needed.
- The data service will probably present flaws during the testing phase, which will affect adversely the normal operation of the real network.
- Mobile users should not experience the overload of testing new applications on the real network.
- The double task of simulating a data service for testing purposes and then implementing it for operation on a real network should be avoided.

Taking into account the previous considerations, a system has been developed, based on mobile agent technology (Ibarri et al., 2006b; Lange and Oshima, 1999; Milojicic et al., 2000), for the simulation of wireless infrastructures composed of proxies (data service providers) and moving objects (data service clients), that can be used to test wireless data services (Ibarri et al., 2004; Mena et al., 2002). In the proposed testing system, mobile agents play an important role. Mobile agents are software components that execute on an execution environment (traditionally called place) provided by a certain mobile agent platform\(^1\), and can autonomously travel from place to place (within the same computer or between different computers) (Milojicic et al., 2000). Mobile agents present a range of unique advantages (such as autonomy, flexibility, and effective usage of network bandwidth (Lange and Oshima, 1999))

\(^1\) Mobile agent platforms are usually implemented in Java, among other reasons because of its portability. In these cases, a \textit{place} is a Java process that provides certain services (e.g., communication and transportation of agents to other \textit{places}).
which makes them very attractive to wireless (Spyrou et al., 2004), pervasive (Cardoso, 2002), and distributed computing in general. In a traditional client/server architecture, a client requests a server to perform a certain task. As opposed to this approach, a mobile agent can move to a computer that has the required resources and perform the task itself. Therefore, mobile agents can save remote data communications (e.g., they can process all the data locally, on the computers that hold such data).

The proposed testing system deals with mobile computing environments that can include both real and simulated moving objects and proxies. Indeed, it is able to hide the true nature (real or simulated) of moving objects and proxies in a scenario. In this way, data services will not be aware that some of the moving objects and proxies they deal with are not real but simulated. So, the system avoids the need for developing a data service prototype that deals with a simulated environment and that later must be adapted to work on a real wireless network. Instead, it is advocated to develop the final data service directly, test it in a simulated (or partially simulated) environment until it is ready, and finally execute the same data service in the real-world wireless environment. With this approach, the developers of wireless data services can build their own scenarios and easily test the suitability of certain design decisions. Summing up, the main features of the testing system are:

- **Plug and simulate.** A real data service can be tested on a totally or partially simulated environment.

- **Use of mobile agent technology** to simulate objects that move among areas served by different proxies. Mobile agents are the software counterparts of real moving objects, and they avoid non-intended delays caused by remote communications in a (distributed) simulation.
• **Support for distributed testing**, which allows to increase the parallelism and scalability. The tested application is not aware of the number of computers used for the simulation: The tester can use one, two, or many computers depending on his/her needs.
• **Simulation of moving objects that execute agent-based applications.** Applications based on (mobile) agents, which are considered very useful for mobile computing, can also be easily tested.

The structure of the rest of this article is as follows. In Section 2, we describe the basic mobile environment infrastructure considered. In Section 3, we explain how moving objects and proxies are simulated in the developed system. In Section 4, we describe the mechanism used to simulate moving objects that execute multiagent applications. In Section 5, we explain how the testing system can be used. In Section 6, we evaluate experimentally the suitability of the proposed testing system, analyzing its accuracy and flexibility. In Section 7, we present some related works. Finally, Section 8 summarizes the main conclusions.

## 2 Mobile Environment Infrastructure

In this section, the main elements in a mobile environment infrastructure are briefly described. Looking beyond the details of a specific wireless technology, the generally accepted mobile environment infrastructure (Pitoura and Samaras, 2001) is considered, which is composed of mobile hosts (MHS) and base stations (BSs). Each BS (or Mobile Support Station, MSS) serves all the mobile hosts within its coverage area or cell. The communication between a mobile host and the BS that provides it with coverage is wireless (e.g., using cellu-
lar network technology or a wireless LAN), and the communication among BSs is wired. Thus, BSs allow the communication between mobile hosts and fixed hosts (FHs) connected to the Internet. In accordance with the previous description, wireless ad-hoc networks are not considered in this work. From the point of view of data services for mobile environments, the following two elements are distinguished:

- **Moving objects** are entities provided with a wireless device that allows them to access the network. For example, a car equipped with a portable computer connected to the Internet or a person carrying a PDA (Personal Digital Assistant) connected to a wireless hotspot are examples of moving objects. As moving objects move, the available data services that they can access might change, as different mobile data services might be available in different geographical areas. Moving objects can have attached devices such as GPS receivers or other sensors that feed them with context information. Similarly, they are capable of executing software, such as a web browser or a route guiding system that shows the location of the moving object on a map. Depending on their features, moving objects can have different functionalities.

- **Proxies** are computers that provide moving objects under their proxy area with mobile data services. For this task, there is some process, called Proxy Server in this article, running on a certain communication port, where it can receive service requests from objects within its proxy area. In a cellular network, a proxy area could be the union of the coverage areas of one or more base stations. It should be noted that a more specific term could be used to designate a proxy depending on the service it provides: for example, a proxy that provides location information is usually known
as a Location Server (see the Mobile Location Protocol Specification by

Therefore, in order to simulate environments to test wireless data services,
how these two elements can be simulated must be considered.

3 Simulation of Moving Objects and Proxies

In this section, it is described how moving objects and proxies are simulated
in the proposed system. A sample scenario with real and simulated moving
objects and proxies is also described.

3.1 Simulated Moving Objects

A simulated object is the representation of a moving object in the testing sys-
tem. Simulated objects can be inspired by real moving objects, although they
could have different features. For example, in a certain scenario, simulating
that a real stationary object is moving could be required.

A simulated object will be implemented as a mobile agent (Ilarri et al., 2006b;
Lange and Oshima, 1999; Milojicic et al., 2000) that will behave as the sim-
ulated object is required to behave, which includes mobility and any other
behavior. Basically, the simulation of a moving object requires the execution
of three tasks: simulating the changes of proxy area, simulating the object’s
movements, and simulating the object’s functionality\(^2\). These three tasks are

\(^2\) It should be clarified that the proposed system does not aim at emulating the
explained in more detail in the following. An example of moving object that simulates a user using a web browser will be shown in Section 5.2.

3.1.1 Simulation of the Changes of Proxy Area

The proposed testing system benefits from the resemblance between moving objects and mobile agents to simulate a change of proxy area by a moving object as a trip of the mobile agent that simulates that moving object, from the old proxy to the new proxy\(^3\), as shown in Figure 1. Thus, mobile agents will always interact locally with their proxies in the same way that moving objects interact with theirs. This is a way of distributing the simulation load across several computers that, as an additional benefit, avoids the introduction of artificial delays in the simulation (caused by real network delays among proxies). So, no remote communications between the mobile agents and their proxies are required. In other words, a communication between an object and its current proxy is represented in the testing system as a local communication between the corresponding simulated object (i.e., the mobile agent) and the proxy where it resides. An alternative solution where the mobile agent does not execute on its proxy would require the use of the network to support the communications between the simulated object and the proxy. Moreover,

\(^3\) It is assumed that not only simulated proxies but also real proxies allow mobile agents to move there (i.e., they provide an execution environment for them); in this way, it is possible to build scenarios with simulated objects and real proxies at the same time.
sometimes a service of a real proxy is not available remotely and the only way to get access to it, apart from using a real device under its coverage, is sending a mobile agent to that proxy; as an example, see in Section 7 the text relative to (Satoh, 2002).

![Diagram of real situation and simulation](image)

**Fig. 1.** Handoff of a moving object: real and simulated scenarios.

In a real situation, moving objects discover dynamically the network infrastructure by using signals received from nearby proxies (i.e., each moving object knows which proxy provides it with coverage). As mentioned at the beginning of Section 3.1, a mobile agent will implement the functionality of a simulated moving object. Such a mobile agent is initialized with a *Proxy Catalog* (proxies in the scenario and their coverage areas). During the simulation of the object’s movements (see Section 3.1.2), the mobile agent will query the Proxy Catalog to keep track of the proxy that would provide the object with coverage if it were real. When the mobile agent detects that the simulated object that it represents leaves the area of the current proxy and enters the area of a different proxy, the mobile agent moves there. Therefore, as indicated before, the mobile agent is always on the proxy that provides coverage to the simulated object; if the simulated object is out of coverage, the corresponding mobile agent will remain on its current proxy until the object gets coverage, and then
it will move to the new proxy.

It should be stressed that the movement of a mobile agent is only expected to take a small amount of time (e.g., agents in the mobile agent platform SPRINGS (Ilarri et al., 2006b) need less than one second to travel even in a very highly overloaded LAN with 500 mobile agents moving and communicating simultaneously with no stop). Moreover, most mobile agent platforms are implemented in Java, whose class loading mechanism provides code caching. Therefore, the code of an agent will only be transmitted to a certain node once and will be later reused by other agents of the same class that arrive there. The small delay that the trip of a mobile agent may incur is not significant, and so using mobile agents is beneficial, according to the experiments presented in Section 6.1.

It is interesting to highlight that, when a change of proxy area occurs, the previous proxy will only notice that a moving object under its area has stopped using its services. Similarly, the new proxy will only notice that a new moving object under its area starts using its services. The mobile agent should communicate with the Proxy Server on its current proxy in the same way that a real moving object would. In this way, from the point of view of the tested data service, there would be no difference between a real and a simulated object.

3.1.2 Simulation of the Object’s Movements

A mobile agent begins its execution on the proxy that provides coverage to the initial location of the simulated object that such an agent represents. Then, the mobile agent can simulate, for example, that it follows a predefined path by processing a function that computes the next location from the current
location, speed and direction. The obtained location could be used for a variety of purposes (e.g., to simulate that the object communicates its location to its proxy) or simply to detect when a change of proxy area occurs.

It should be noted that the use of mobile agents allows creating objects with the required behavior, such as: 1) simple simulated objects that follow a pre-defined trajectory; 2) objects whose movements are controlled remotely (e.g., a GPS simulator\(^4\) is shown in Fig. 2, that allows to control a simulated object remotely by turning the knob on the left and the speed selector on the right); or 3) objects that react to the environment by changing their behavior (e.g., they move slowly or fast depending on traffic conditions).

![GPS Simulator](image)

**Fig. 2.** GPS simulator to control simulated moving objects manually.

\(^4\) The authors of this article thank SailSoft (http://www.sailsoft.nl) for providing the Visual Basic source code of *GPSSIMUL*. Some of the code was adapted to use it in the developed system.
3.1.3 Simulation of the Object’s Functionality

The different functionalities of the simulated object (provided to real objects by hardware appliances or software applications) that are interesting for testing are simulated. If possible, each behavior will be represented by the real code of the application to test. If some behavior cannot be used without changes (e.g., the moving object is supposed to access a database and there is no real database available for testing), then it will be implemented as a thread of the mobile agent that will simulate that behavior. For example, a mobile user browsing the web with his GPS-enabled wireless device can be simulated by a mobile agent with two threads (see Fig. 3): One would be a GPS simulator that computes the current location based on the (wanted) user’s trajectory, and the other would request web pages following the (wanted) user access pattern. If a location-dependent service must be tested, another thread could be in charge of communicating the user’s current location to his/her proxy. It should be indicated that each thread should be encapsulated in a software agent, as explained in Section 4, but it is preferred to keep the description simple here.

![Diagram](image)

(a) (b)

Fig. 3. A simulated object (a) and a real object (b) communicating with the same proxy.

Mobile agents can simulate the behavior of almost any program or appliance attached to real mobile devices. Which features of these elements are simu-
lated depends on the goals of the simulated scenarios (only interesting features should be simulated). When one of the interesting functionalities is the execution, on the moving object, of an application based on agents (or, similarly, when the moving object executes several threads), some challenges arise due to the difficulty of packaging these resources as a mobile agent for simulation (see Section 4).

3.2 Simulated Proxies

A simulated proxy is defined as a certain conceptualization of a proxy. Simulated proxies can be inspired by real proxies, but they can also have different properties. For example, to test certain data services, a real proxy could be required to have a different coverage area. A simulated proxy will be implemented by creating a place (execution environment for mobile agents, as explained in Section 1) with a Proxy Agent on a fixed computer. The Proxy Agent is a static agent that will provide the wanted functionality of a Proxy Server. As an example, a proxy that provides location-dependent web pages (Acharya et al., 1995) will be shown in Section 5.2.

These two elements, the place and the Proxy Agent, can reside on any computer on the fixed network (real proxies are wired connected). The URLs of such fixed computers are stored in the Proxy Catalog (see Section 3.1.1) to identify the simulated proxies. URLs of real proxies are also stored in the Proxy Catalog, in order to allow the simulated objects to be aware of the real network infrastructure; otherwise, they could not detect the real proxies.

In Fig. 4.a, an example of a Proxy Catalog is shown (simulated proxies are
those with dashed areas), where the following information is stored for each proxy in the sample scenario: its identifier, its URL (IP address and communication port on which requests from moving objects can be received), its location, and its coverage area. It should be noted that both real and simulated proxies are identified similarly by a URL. Different proxies can be simulated on different or on the same fixed computer (as shown in Fig. 4.b). When a simulated moving object needs to communicate with a proxy, it first obtains the URL of such a proxy by querying the Proxy Catalog, and then it communicates with it as it would in a real environment. So, the access to the data service is independent of whether the proxy is real or simulated.

Fig. 4. Proxy Catalog example: simulated scenario (a) and real scenario (b).

In order to simulate communication delays, the ProxyAgents can be parameterized to delay adequately when communicating with simulated objects (to simulate wireless network delays) and with other proxies (to simulate fixed network delays). They can also simulate temporal disconnections from their moving objects, if needed. For more details about the simulation of delays, please see Section 5.1.
The complete functionality of a proxy cannot be simulated because, for example, a simulated proxy cannot provide coverage to real moving objects. However, a scenario where a simulated (therefore non-existing) proxy detects real moving objects can still be defined, as long as those real objects are under the coverage of a real proxy. This scenario could be interesting due to three main reasons: 1) to simulate that real objects change of proxy area as they move within the coverage of a single real proxy, 2) to simulate the existence of areas without coverage inside the coverage area of a real proxy, and 3) to test with real objects new functionalities not provided by real proxies.

To detect real objects, the Proxy Agent of the simulated proxy sends Proxy Assistant Agents to the real proxies whose coverage area intersects with the area of the simulated proxy. An example is shown in Fig. 5, where Proxy4 is not real (it is simulated on a certain fixed computer) and Proxy3 is the one that actually provides coverage to the real moving object shown in the figure. Proxy Assistant Agents send to their Proxy Agent information about moving objects entering the area of the corresponding simulated proxy: Every Proxy Assistant Agent obtains this information by interacting with the Proxy Servers on their corresponding real proxies. For this, a Proxy Server must support at least queries that retrieve information about the objects within its area. The testing system can simulate the availability of any information about the real objects (even if this information does not exist in the real world as, for example, the Proxy Assistant Agent could generate it randomly or according to some other strategy). In Fig. 5, the Proxy Assistant Agent

---

For example, it could be interesting to simulate that certain information about the objects (e.g., their locations) are stored by the proxies, which may not be the real case.
on Proxy3 sends the interesting data of the real moving object to the Proxy Agent of the simulated Proxy4 (which executes on a certain fixed computer).

![Diagram of a simulated proxy "detecting" a real moving object.]

**Fig. 5.** A simulated proxy “detecting” a real moving object.

### 3.3 Simulation of a Sample Scenario with Real and Simulated Objects and Proxies

In this section, a sample scenario that could be used to test a certain wireless data service is shown. In Fig. 6.a, as in Fig. 4, simulated proxies are those with a dashed coverage area (Proxy2, Proxy5, and Proxy6), and the rest of the proxies (Proxy1, Proxy3, and Proxy4) are real; the simulated objects are also shown (policeCar1, policeCar2, policeCar3, policeCar5, policeCar15, policeman1, policeman2, policeStation3, and policeStation19). It should be noted that, for the sake of generality, static objects are also considered as moving objects if they are interesting for the tested data service (e.g., if they can be clients of the service). The real scenario is presented in Fig. 6.b, where some real objects appear that were not in Fig. 6.a (car15, car38, policeCar4, and policeStation2). The three simulated proxies have been implemented as *places*
(with a Proxy Agent) on two fixed computers. The simulated objects have been implemented as mobile agents on the places of the proxies (real or simulated) that provide coverage to them in the wanted scenario; e.g., policeCar5 is represented by a mobile agent executing on the real proxy Proxy1, and policeCar15 is simulated by a mobile agent on the simulated proxy Proxy6 on the second fixed computer.

Fig. 6. Simulation example: test scenario (a) and real scenario (b).

The proposed testing system allows mixing real and simulated proxies and moving objects transparently. The following situations can be distinguished (see Table 1):

- **A real object with a real proxy.** A real object interacting with a real proxy requires no simulation, and both can be part of a partially simulated scenario.

- **A real object with a simulated proxy.** It is possible to simulate proxies that provide coverage to real objects (whenever they are under the coverage of
Table 1

Mixing real and simulated moving objects and proxies

<table>
<thead>
<tr>
<th></th>
<th>Real proxy</th>
<th>Simulated proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real object</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>As long as a real proxy detects it too</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulated object</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>A mobile agent on the real proxy</td>
<td>A mobile agent on the computer that simulates the proxy</td>
<td></td>
</tr>
</tbody>
</table>

any real proxy, as explained in Section 3.2).

- A *simulated object with a real proxy*. A simulated object is represented by a mobile agent, which executes on the *place* of its real proxy.
- A *simulated object with a simulated proxy*. This case is like the previous one, but with the mobile agent executing on the *place* of the simulated proxy.

It is also possible to simulate that a real moving object follows a trajectory different from the real one. This would be equivalent to defining a simulated object whose features and behavior are inspired by such a real object. For example, it is possible to simulate that a certain wired device is a moving object that follows a trajectory or that it accesses a data service on a certain proxy: A mobile agent can be created to represent such a device on the wireless network. More importantly, the wired device itself could actually access the data service by interacting (e.g., following the orders of a user) with the corresponding proxy through its mobile agent in the simulation. For example, it can be simulated that a desktop computer is a moving object which retrieves from its current (real or simulated) proxy information about other nearby (real or simulated) moving objects; this information can then be used by the desktop computer (e.g., to show it to a user). In this way, it is possible to access wireless services using a fixed computer.
4 Simulating Multiagent Applications Executing on a Moving Object

In this section, the strategy proposed for the simulation of moving objects that execute multiagent applications is described. From the point of view of a simulation, among the most complicated applications that moving objects can execute are those based on mobile agents. Mobile agents have been proposed as a very interesting technology to build applications for mobile environments (Cardoso, 2002; Spyrou et al., 2004). Therefore, it is highly interesting to consider how the testing system could handle this case. As explained in Section 1, mobile agents can travel across computers. Thus, they can decide to interrupt their execution on its current moving object, travel to another computer or moving object (where the execution is resumed), and come back in the future even if the original moving object is then under the coverage of a different proxy. Therefore, the simulation of moving objects becomes a difficult task if there is an interest in simulating that they are executing multiagent applications. If a simulated moving object is required to execute an application composed of agents, because it is interesting or needed for the tests, then such an application should theoretically execute inside the mobile agent that simulates the moving object, which is obviously not possible.

Alternatively, the functionality of each agent could be simulated by using a new thread within the mobile agent that simulates the moving object. This approach is very costly because: 1) The behavior of the multiagent application must be re-implemented using threads; and 2) some of the agents could be mobile and require to perform movements to other computers or devices to resume their execution there, which cannot be simulated easily without
using mobile agents (e.g., they could need to access real data on those remote computers/devices). Instead, a solution based on reusing the original code of the multiagent application is advocated, which allows to perform a simulation as close to reality as possible with a minimum effort. Agents that should execute on the simulated moving object will be reused directly, and a certain coordination mechanism will allow them, when needed, to travel attached to the mobile agent that simulates the moving object where they reside.

One additional advantage of this approach is that, in the simulated scenario, mobile agent applications “running” on simulated moving objects will behave exactly as expected. Thus, some of those agents could even move to real computers outside the simulation scenario, access a real database, and come back to the simulation scenario with the results. With this approach, the time required to prepare a simulation is greatly reduced.

In this section, it is analyzed how to realize this plug-and-simulate proposal while keeping the original application’s mobile agents (which will be called Internal Agents, IAs) unaware of the real or virtual nature of the moving object where they execute (which will be termed Moving Object Agent, MOA). It should be stressed that the approach described in this section also applies, similarly, to moving objects hosting static agents or threads. Even if the agents are static in the original application code, they are needed to be mobile in the simulation. Thus, an IA must move with its MOA, which can be more easily implemented if the IA is a mobile agent. Similarly, each thread must be encapsulated in a mobile agent, so the simulation can benefit from the coordination protocol described in this section; an example of a moving object with an internal thread that simulates a user using a web browser will be shown in Section 5.2.
4.1 Real and Simulated Scenario

As shown in Fig. 7, when a moving object containing agents is simulated, several mobile agents must be used in the simulation. The mobile agent that is used to simulate the moving object, that is called Moving Object Agent (MOA), and the agents used in the original multiagent application, that are called Internal Agents (IAs). The equivalence between real and simulated scenarios is summarized in Table 2.

![Diagram](image)

(a) (b)

Fig. 7. Moving objects executing multiagent applications: real scenario (a) and simulated scenario (b).

Table 2

<table>
<thead>
<tr>
<th>Real</th>
<th>Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moving object (e.g., a PDA)</td>
<td>Mobile agent MOA</td>
</tr>
<tr>
<td>Place on moving object</td>
<td>Place on proxy</td>
</tr>
<tr>
<td>Agents on moving object</td>
<td>Mobile agents IAs</td>
</tr>
</tbody>
</table>

When an MOA moves to another proxy (to simulate the change of proxy area by the simulated moving object), its associated IAs must move too, in the same way that agents executing on a real PDA will remain on it when that PDA moves geographically. That is, the MOA has to coordinate itself with its IAs when it needs to travel, making sure that they move with it as well.
To achieve this goal, extending the functionality of the agents in the original application code in order to allow the MOA to coordinate with them in the simulation scenario is required (see Section 4.2). For that purpose, a set of classes has been defined, that provide such agents with the required features (see Section 5.1) in such a way that their original functionality will not be disturbed (they will be unaware of being running in a simulated scenario).

4.2 Coordination Protocol

The different stages of the coordination mechanism developed in the proposed testing system to simulate moving objects running multiagent applications are the following (see Fig. 8):

![Activity diagram for the coordination protocol.](image)

(1) Detection of the need for traveling. The MOA computes the next location of the moving object that it represents (e.g., from a predefined trajectory)
and realizes that it implies a change of proxy area.

2) Request of movement. The MOA requests its IAs to travel to the destination proxy.

3) Packaging for the journey. The IAs get ready for the journey by waiting until the current transaction (defined in this article as a sequence of operations that cannot be interrupted by a journey) ends.\(^6\)

4) Traveling. The MOA and IAs move to the new proxy. The order in which they travel is not important, they can travel in parallel.

5) Notification of arrival. After traveling, each IA sends an acknowledgement to its MOA indicating that the trip was performed correctly. A notification of arrival could fail if the MOA is still traveling to the destination; in such a case, the notification will be retried until it gets through.

6) End of Travel. When the MOA arrives at the destination, it waits until it receives notification of arrival from all its IAs. Then, the travel is considered finished. In case the MOA or any IA finds some problem traveling, the movement will be retried as many times as needed; the simulation fails only if the movement has not succeeded after a predefined timeout.

7) Resume Execution. The MOA sends a notification to its IAs for them to resume the execution, and it will also continue its own execution.

A situation where an IA departs from its moving object and comes back later (see Fig. 9 for a sample scenario, where an internal agent travels from one simulated moving object to another) is implemented more easily with a mo-

\(^6\) It should be noted that the termination of the main thread of an autonomous agent cannot be forced (e.g., the agent could be communicating remotely with another at that moment). Instead, the MOA requests an IA to travel and the IA's thread will detect the pending request and perform the trip as soon as it can.
bile agent platform that provides location transparency (Stefano and Santoro, 2002), as SPRINGS (Ilarri et al., 2006b) does. Thus, such a platform automatically keeps track of the proxy where each MOA is executing and ensures that a communication eventually succeeds even if the target agent is traveling to another computer. Using other mobile agent platforms that do not provide location transparency, solving the simulation issues that arise in these cases would be more challenging. Regarding the steps 5 and 6 above, it should also be stressed that some mobile agent platforms, such as SPRINGS, manage failed communications and trips transparently (retrying automatically if needed).

Fig. 9. Example of an internal agent that moves to other simulated moving object and comes back later.

5 Using the Testing System

This section shows how the testing system can be used. First, the main classes in the system and how they can be adapted to specific simulation requirements
are described. Next, an example shows how existing code can be re-used for testing.

5.1 Classes in the System

In Fig. 10, the main classes of the system are shown (the core classes are shaded). The most important ones are the three classes which correspond to the need of implementing the functionalities of proxies (see Section 3.2) and MOAs/IAs (see Section 4.1):

![Class Diagram](image)

**Fig. 10. Class diagram for the testing system.**

- **ProxyAgent** provides the functionality needed by any simulated proxy (see Section 3.2). In our implementation, it basically allows a moving object to register and de-register from the proxy, keeps information about the objects registered, and provides logging facilities.

- **InternalAgent** provides the functionality of an IA, that is, it implements the elements needed by the coordination protocol described in Section 4.2. It should be noted that the agents and threads executing on a moving object are implemented as mobile agents for simulation purposes (see Section 4).

- **MovingObjectAgent** provides the basic functionality of an MOA. This includes the implementation of the coordination protocol explained in Sec-
tion 4.2 as well as the simulation of movements (according to a certain mobility model, as indicated below) and handoffs. See Section 3.1 for details about the simulation of moving objects and Section 4.2 for an explanation of the coordination protocol needed to allow the simulated moving object “to execute” mobile agents.

Another couple of classes in the diagram that are worth mentioning are indicated in the following:

- **MobilityModel** is an abstract class which represents a mobility model (Camp et al., 2002; Schindelhauer, 2006) for a *MovingObjectAgent*. Two mobility models considered as part of the system are a trajectory-based mobility model and a random mobility model (represented in the figure by the classes *TrajectoryMobilityModel* and *RandomMobilityModel*, respectively).

- **DelaySimulator** provides the functionality needed to simulate network delays. Several subclasses are provided by default (e.g., *GSMSimulator* and *GPRSSimulator*) to simulate delays in specific network environments. An instance of *DelaySimulator* is associated to a *ProxyAgent*, in order to delay communications having its proxy as destination. This class implements a delaying method parameterized with the size of the data communicated (in KBs) and the name of the source proxy (this second parameter is not applicable in the case of wireless communications from a moving object to a destination proxy). Such a method must be called whenever a communication is susceptible of being delayed.

In Fig. 10, an additional class appears: *SpringsAgent*. In most mobile agent platforms, a programmer implements a mobile agent by specializing a specific class provided in the API of the platform. In the current implementation
of the system, the mobile agent platform SPRINGS (Ilarri et al., 2006b) is used, that provides a base class *SpringsAgent* that must be extended by the classes in the system. Depending on the mobile agent platform used (Trillo et al., 2007), the name of this class could be different (e.g., *MobileAgent* in the mobile agent platform Grasshopper and *Aglet* in Aglets); also depending on the platform, a different base class may need to be considered for a Proxy (e.g., *StationaryAgent* in Grasshopper), as it is a static agent. According to the experience of the authors with different mobile agent platforms, only syntactic changes are usually required (the name of the agent class, the name of the main method of the agent, the method the agent must call to move to another *place*, etc.), although this can also depend on the main features of the platform (e.g., if it is based on remote calls or message passing).

The tester of a data service should extend the classes *MovingObjectAgent* and *ProxyAgent* to define the moving objects and proxies needed for his/her specific simulation tests (and reusing the original code of the data service to test). For example, a subclass of *MovingObjectAgent* representing a simulated car could add the required code to communicate periodically, to its proxy, information about its surroundings (e.g., number of nearby cars); a subclass of Proxy could receive this information and process it in order to compute the number of cars within a certain area. In this way, the developers of wireless data services can test features that are not even available in a real network. The tester of the data service must also make sure that each agent/thread that should execute on a simulated moving object extends the class *InternalAgent*.

It can also be interesting to specialize the *DelaySimulator* class to implement the simulation of network delays in the wanted conditions. In particular, it is possible to integrate the proposed testing system with a low-level network
simulator. For example, in the context of this work, some tests have been carried out with the popular network simulator ns-2 (http://www.isi.edu/nsnam/ns/). It is possible to define a network in ns-2 (defining the nodes, links, bandwidth, latencies, error rates, etc.) and interact with the simulation from Java code, in order to obtain the delays to apply to the network communications. Finally, the tester of the data service can also implement mobility models suited to his/her needs by subclassing MobilityModel.

As a summary, Table 3 provides some guidelines to consider to test a mobile data service with the proposed testing framework, along with an assessment of the difficulty of the different tasks. An example of use of the testing system, that shows how the basic classes in the framework can be extended, is presented in the next section.

5.2 Use Case: Testing Location-Dependent Web Proxies

This section illustrates, with a simple example, how existing code can be reused in the proposed testing system. The scenario presented in this section is motivated by the idea of location-dependent web pages (Acharya et al., 1995), whose contents depend on the location of the user. The existing code that will be reused is Lobo (http://lobobrowser.org/), a web browser implemented in Java. Although Lobo is open-source, no modification to the original code will be performed. The goal is to simulate moving objects that execute the Lobo browser to access location-dependent web pages. Regarding proxies, the following classes are defined:

(1) A class SimpleURLRewriterWebServer is implemented. An instance of
### Table 3

**Guidelines for using the testing system**

<table>
<thead>
<tr>
<th>Need/Problem</th>
<th>Solution</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) The tester needs to simulate some special functionality on the proxies.</td>
<td>Specialize <code>ProxyAgent</code>. Implement, if needed, other auxiliary classes.</td>
<td>Low. It only depends on the difficulty of implementing the required functionality (if not implemented already); its integration is straightforward.</td>
</tr>
<tr>
<td>2) The tester needs to simulate some special functionality on the moving objects.</td>
<td>Specialize <code>MovingObjectAgent</code>. Specialize <code>InternalAgent</code> also if an additional thread of behavior is required (see row 4). Implement, if needed, other auxiliary classes.</td>
<td>Low. It only depends on the difficulty of implementing the required functionality (if not implemented already); its integration is straightforward.</td>
</tr>
<tr>
<td>3) The tester needs to reuse an application based on mobile agents.</td>
<td>Specialize <code>MovingObjectAgent</code>. Each mobile agent in the existing application must also extend <code>InternalAgent</code> and attach itself as an internal agent of the corresponding <code>MovingObjectAgent</code> (that must create such an internal agent), checking with a certain frequency if a trip is required by the coordination protocol. In case balancing is not possible (because the agent already extends a class that is not a superclass of <code>InternalAgent</code>), wrapping can be used instead (see Section 5.2 for an example).</td>
<td>Middle/High. Some modifications in the original code may be required to check periodically the need of traveling and clean the state of the agent before a trip (e.g., close open files). This may be difficult if the original agent performs long transactions of work that cannot be interrupted by a trip, in which case the transaction should be divided in smaller tasks. Otherwise, it is simple.</td>
</tr>
<tr>
<td>4) The tester needs to simulate moving objects with different concurrent behaviors.</td>
<td>Encapsulate each concurrent behavior (thread) in an <code>InternalAgent</code> and proceed as indicated in the row above.</td>
<td>Middle/High. See the row above.</td>
</tr>
<tr>
<td>5) The tester needs to reuse an application, based on mobile agents, implemented in a platform different from the one used in the testing system <em>(currently, SPRINGIS)</em></td>
<td>A version of the testing system implemented using the same mobile agent platform is needed.</td>
<td>Middle. It is usually easy to re-implement the testing system with a different mobile agent platform, as its functionalities can be wrapped in appropriate classes using the new platform. It can be more difficult if the platform to use considers a different architecture or communication model (e.g., if synchronous calls have to be emulated via message passing).</td>
</tr>
<tr>
<td>6) The tester knows nothing about mobile agents.</td>
<td>A basic knowledge of mobile agents is required. The tester will benefit from learning how a platform is deployed, how agents are created, how they move, and how they communicate. Moreover, he/she needs to understand that a trip of an agent implies a change of execution environment (so the local resources are not accessible anymore and the values of static variables may be lost), and that the agent’s variables must be <code>Serializable</code> or marked as transient.</td>
<td>Low. These are basic aspects and easy to learn. No in-depth knowledge or extensive experience is required.</td>
</tr>
<tr>
<td>7) The tester encapsulates code in a mobile agent but it fails.</td>
<td>Probably something clashes with the agent platform (e.g., the code tries to set a <code>SecurityManager</code> and the platform does not allow it to be changed); Avoid the conflict.</td>
<td>Low. These problems are not expected to be frequent and the solution is usually easy, as long as the original code can be modified or provides a flexible API.</td>
</tr>
</tbody>
</table>
this class is initialized with two parameters: a port number and a URL country extension (e.g., “es” for Spain, “it” for Italy, “us” for the United States, etc.). It creates a thread that listens to the communication port specified (using a socket) to read URL requests (e.g., a request “GET http://www.google.com HTTP/1.0”) from clients (web browsers using that instance of SimpleURLRewriterWebServer as HTTP proxy). When a request is received, it changes the country extension as configured (e.g., if a request for “http://www.google.com” is received and the extension set is “it”, then the URL is changed to “http://www.google.it”). Then, it opens a socket to that URL and read the contents of the corresponding web page, which is returned to the client that initially performed the request. Thus, it implements an HTTP proxy that automatically returns web pages to the browser as they would be received if the URLs were typed in the corresponding country (many web servers today perform this country-based redirection automatically if the URL typed has a “com” extension). As the instance of the SimpleURLRewriterWebServer class can be configured with any extension, it is possible “to simulate” that the Lobo web browser is executing in a different country by setting its HTTP proxy as the URL and port of the corresponding SimpleURLRewriterWebServer instance.

(2) A class ProxyLocDepWebPages is implemented, that extends ProxyAgent to customize the desired behavior of the proxies in the testing system. The basic structure of this class is shown in Figure 11. Basically, it creates an instance of SimpleURLRewriterWebServer whose parameters are determined by the name of the proxy (for simplicity, the mapping between the proxy name and the port and extension is hard-coded, but a more flexible approach can be easily adopted) and adds a method getHTTP-
proxyPort that will allow a moving object to query the port of the HTTP proxy they should connect to. It should be noted that class SimpleURLRewriterWebServer is an auxiliary class used to implement the wanted functionality of proxies. However, it can also be used in isolation to set up an HTTP proxy that can be used from any web browser.

Regarding moving objects, the following classes are defined:

1. Lobo is a program, so a class LoboClient is first implemented to simulate a user using the Lobo web browser. This class simply navigates to different URLs stored in a text file, waiting a certain amount of time between page requests. Thus, this is a simple class that makes use of the Lobo Browser API to automate navigation.

2. A class called LoboClientWithoutFiles is implemented, which extends LoboClient with a method called readURLs which returns a vector with the URLs in the file and is able to process these URLs using the information contained in the vector instead of using directly the file. This class is needed because using the functionality of LoboClient from a mobile agent implies that it is not possible to rely on a local file. Thus, when the mobile agent moves to another computer, the file is not accessible anymore.

3. A class called LoboClientWithoutFilesIA extends InternalAgent to implement the behavior of a user navigating periodically through a set of URLs. It encapsulates an instance of LoboClientWithoutFiles, which performs most of the work. A call to the method travelIfNeeded, implemented by InternalAgent, is performed from time to time (when a transaction finishes, according to the explanation in Section 4.2) to make the internal agent follow its MOA (if needed). The main method of the agent is called
import ...; {Package with the definition of ProxyAgent and java.rmi.RemoteException

public class ProxyLocDepWebPages extends ProxyAgent {
    private transient SimpleURLRewriterWebServer _surw = null;
    private int _portHTTPProxy = -1;
    private String _extensionToPut = null;

    public ProxyLocDepWebPages(ProxyAgentParameters parameters) throws RemoteException {
        super(parameters);
        initialize();
    }

    public void initialize() throws Exception {
        if (_proxyAgentParameters.proxyName.equals("ProxyInSpain")) {
            _portHTTPProxy = 49999;
            _extensionToPut = "es";
        } else if (_proxyAgentParameters.proxyName.equals("ProxyInItaly")) {
            _portHTTPProxy = 5000;
            _extensionToPut = "it";
        } else {
            throw new Exception("Unknown Proxy!");
        }

        _surw = new SimpleURLRewriterWebServer(_portHTTPProxy, _extensionToPut);
    }

    public Integer getHTTPProxyPort() {
        return new Integer(_portHTTPProxy);
    }
}

Fig. 11. Basic structure of ProxyLocDepWebPages.

main in SPRINGS and it specifies the life of the agent. It should be noted that implementing the extra functionality as an internal agent is not required: Instead, it is possible to implement the extra functionality in a subclass of MovingObjectAgent. However, the extra functionality has been considered here as an additional thread of the moving object, and therefore is implemented as an internal agent. The preDeparture method is automatically called in SPRINGS when the agent is about to move to another context.\textsuperscript{7}

\textsuperscript{7} Methods of SpringsAgent that are implemented by InternalAgent, such as pre-Degarture, postDeparture, postArrival, preArrival, etc., may be overridden by the implemented subclasses, but in that case the new implementation of the method should include a call to the corresponding method in the superclass.

32
import ...

public class LoboClientWithoutFilesIA extends InternalAgent {
    private LoboClientWithoutFiles _loboClientWithoutFiles = null;
    private String _nameOfMUA = null;
    private Vector _urls = null;

    public LoboClientWithoutFilesMovingObject(String nameOfMUA) throws RemoteException {
        super();
        _nameOfMUA = nameOfMUA;
        _loboClientWithoutFiles = new LoboClientWithoutFiles();
    }

    public void preDeparture(String originContext, String targetContext) {
        super.preDeparture(originContext, targetContext);
        _loboClientWithoutFiles.dispose(); /* Destroy the GUI */
    }

    public void setHTTPProxy(Integer port) {
        _portHTTPProxy = port;
        _loboClientWithoutFiles.setHTTPProxy(_portHTTPProxy);
    }

    public boolean mainTask() throws Exception {
        if (first execution of mainTask) {
            _urls = _loboClientWithoutFiles.initialize(_nameOfMUA);
            attachToMUA(_nameOfMUA);
            return true;
        } else {
            if (GUI not initialized) _loboClientWithoutFiles.initializeGUI(_nameOfMUA)
                for (each url in _urls and while (super.getNeedOfTraveling() == false)) {
                    _loboClientWithoutFiles.processURL(url);
                    sleep(_loboClientWithoutFiles.MILLISECONDS_BETWEEN_REQUESTS);
                }
            return false;
        }
    }

    public void main() {
        boolean end = false;
        while (end == false) {
            end = mainTask();
            if ((end == false) and (super.getNeedOfTraveling() == true)) {
                travelIfNeeded();
                end = true;
            }
            /* The local agent's copy ends here (it has traveled!). */
        }
    }
}

Fig. 12. Basic structure of LoboClientWithoutFilesIA.

(4) A LoboClientWithoutFilesMOA class extends MovingObjectAgent to add
the behavior of browser navigation using Lobo. The basic structure of this
class is shown in Figure 13. Basically, the ProxyAgent is queried about
its HTTP proxy when a handoff is performed, and the port number is
communicated to the internal agent LoboClientWithoutFilesIA. To com-
municate with another agent, the syntax of SPRINGS for remote calls is used, which allows a location-transparent communication by just specifying the name of the target agent (in this case, both agents –the MOA and the IA– will be on the same computer). Moreover, SPRINGS identifies the target method using Java reflection, and so any of the agent’s methods can (by default) be invoked remotely.

```java
import ...; {springs.context.*, package with the definition of MovingObjectAgent, etc.)

public class LoboClientWithoutFilesMOA extends MovingObjectAgent {
    ...,
    public void main() {
        try {
            if (internal agent not created yet, then createInternalMAgent();
        super.main();
        } catch (Exception e) {...}
    }

    private void createInternalMAgent() {
        _internalAgentName = this.getName() + _SUFFIX_FOR_INTERNAL_AGENT;
        LoboClientWithoutFilesMovingObject internalMAgent = null;
        try {
            internalMAgent = new LoboClientWithoutFilesIA(getName());
        } catch (Exception e) {...}

        while (not success creating the internal agent) {
            try {
                Context.RMIImpl.instance().createAgent(internalMAgent, _internalAgentName);
                success = true;
            } catch (Exception e) {print error and retry...}
        }

        public void handoffToProxy(String proxyName, ...) throws Exception {
            super.handoffToProxy(proxyName, ...);
            Object[] args1 = {};
            Integer portHTTPProxy = (Integer) callAgentMethod(proxyName, "getHTTPproxyPort", args1);
            if (portHTTPProxy != null) {
                Object[] args2 = {portHTTPProxy};
                callAgentMethod(_internalAgentName, "setHTTPProxy", args2);
            }
        }
    }
```

Fig. 13. Basic structure of LoboClientWithoutFilesMOA.

As explained, no changes in the original code are performed. Only the new functionalities required (an HTTP proxy to perform URL redirections) need
to be implemented. Then, a GUI allows the developer to define and execute scenarios consisting of moving objects and their trajectories and proxies and their proxy areas. For both moving objects and proxies, the corresponding implementing class (that must be a subclass of MovingObjectAgent or ProxyAgent, respectively) can be specified in the GUI, such that the needed instance is built (using Java reflection) when the tester decides to run the testing system. Figure 14 shows a scenario consisting of two cars under the coverage of different proxies and the Lobo browser associated to each object at a particular time during the execution of the testing system (car1 is in Spain and car2 is in Italy). Both cars request the same URL (http://www.nokia.com) but the contents of the web pages shown by the two browsers are different. It should be noted that the Lobo browser appears on the screen associated to the terminal where the proxy that provides coverage to the moving object is executing. Therefore, a user on that terminal could interact directly with the Lobo browser. Depending on his/her purposes, the tester could not require the Lobo browser to appear on the screen.

6 Evaluation of the System

In this section, the proposed testing system is evaluated. For this purpose, four experiments are carried out. The first one proves the benefits of a simulation approach based on mobile agents to distribute the testing load efficiently over a set of computers and minimize the communication overhead. The second one shows an example that indicates how the testing system can be used to test applications based on mobile agents easily, thanks to the agent coordination protocol proposed. Finally, the last two experiments prove the suitability of the
Fig. 14. Snapshot during the testing of moving objects with Lobo browsers.

The proposed testing system to evaluate pre-existing data services. The computers used in the simulations are Pentium IV 3.6 GHz, 2 GB RAM, with Linux 2.6.14, connected through a LAN.

The proposed testing system has been implemented using Java and the mobile agent platform SPRINGS (Ilarri et al., 2006b). Java is used because: 1) as explained in Section 1, most mobile agent platforms are implemented in Java; and 2) many wireless data services to test will be probably implemented in Java. Thus, Java also offers many facilities to develop applications for distributed and mobile computing; portability, safety and robustness are the main reasons to use Java for wireless development, according to http://developers.sun.com/mobility/getstart/. However, mobile data services implemented in a different programming language could also be pos-
sibly evaluated by the system by using, for example, JNI (Java Native Interface) to enable the interaction between Java code (of the testing system) and application code (of the mobile data service to test) written in another programming language.

6.1 Evaluation of the Simulation Approach Based on Mobile Agents

In this section, the goal is to measure the accuracy of the proposed simulation strategy (based on mobile agents), by comparing it with other alternative approaches. Simulating that the proxies are able to detect the locations of moving objects within their coverage is wanted in this experiment. In the simulation, each proxy stores the locations of such objects, which are received every second (considering the objects’ trajectories) from the mobile agents that represent them.

The accuracy of a simulation is measured in this test in terms of the difference between the expected (real) locations of the objects (i.e., the locations of the objects in the real scenario) and the locations known to the underlying infrastructure (i.e., the locations received by the proxies) over time. Although measuring the location error is only significant in a scenario where the objects communicate locations, it indicates whether the simulated objects are able to communicate with the proxies as quickly as required by the scenario that must be tested. In other words, the location error in this experiment indicates how close to reality the simulation performs.

To perform this test, the network infrastructure shown in Fig. 6 has been slightly modified in order to minimize the areas without coverage and consider
only simulated proxies (see Fig. 15). To avoid the problem of objects getting out of coverage (it is not possible to compare a real location with a location that is undetectable by the network infrastructure), the objects are constrained to move within a rectangular area (marked in Fig. 15) that is fully within coverage; a change in the direction of movement is forced for objects that are about to get out of that area.

Fig. 15. Network infrastructure considered for accuracy evaluation.

For this experiment, 1500 objects moving at 60 mph are created. First, pre-defined trajectories are generated according to the following strategy: Every object is initialized with a random location and orientation, and the orientation will change randomly (with a 30% probability) every five seconds to simulate erratic and unpredictable movement patterns. These trajectories are considered the real data for the tests, and therefore they are stored on files for later comparison with the simulation results.

Three alternatives that could be used to simulate a moving object are compared: 1) using a mobile agent that represents the moving object (as it is
advocated in this article); 2) using a thread that communicates the location of its moving object using remote invocations; and 3) using the IBM Location Transponder (http://www.alphaworks.ibm.com/tech/transponder, published on April 2002), a real-time location data reporter which paces the transmission of location updates to match the time instants when such updates should be generated\(^8\). For each of these possibilities, two cases are considered: 1) All the proxies in the network infrastructure are simulated on the same computer, and 2) the proxies are simulated using five different computers in a LAN (one computer must host two proxies). Each experiment is carried out several times and, for each time instant, the average location error per object (in meters) is computed; the results for 100-second tests are shown in Fig. 16. The similarity of the results obtained in the different experiment repetitions (not shown in the figure) proves that the number of repetitions is enough. The following conclusions are drawn:

![Graph showing average location error over time](image)

**Fig. 16.** Accuracy of the simulation using different strategies.

1) *Distributed simulations are useful.* As shown in Fig. 16, the worst results are obtained when a single computer is used to simulate all the

---

\(^8\) With the Location Transponder, remote communications are also used. However, it implements a deadline-based strategy for location updates with the goal of providing a better accuracy.
moving objects and proxies (it becomes a bottleneck). In this situation, the experimental results show that there is no significant difference between using mobile agents, remote communications, or the IBM Location Transponder. Therefore, only one line is shown for this case in the figure.

(2) *Mobile agents allow to perform distributed simulations with the best accuracy.* In Fig. 16, it can be seen that there is a significant improvement in the simulation when mobile agents are used in a distributed environment. On the one hand, they allow to distribute the simulation of the moving objects. On the other hand, a mobile agent simulating a moving object can communicate locally with its proxy, avoiding remote communications. This explains that, in a distributed environment, the approach based on mobile agents outperforms the strategy based on remote communications and the one using the IBM Location Transponder (both represented by a single line in the figure, due to the similarity of the results obtained).

It should be stressed that although the execution of a mobile agent is stopped during a trip (and resumed once the movement to another computer has finished), the additional delay introduced is small and clearly pays off. Thus, many delays will be saved because the simulated moving object will interact locally with its proxy.

These experimental results prove the benefits of the simulation approach considered, based on the use of mobile agents to distribute the simulation efficiently over a set of computers. It should be stressed that the simulation time (the total time needed to perform a simulation) is independent of the strategy adopted. This is because a simulation is carried out considering the real time scale, as the goal is to evaluate a real data service executing it as it would be executed in a real environment. However, the proposed approach based on
mobile agents and distributed simulations achieves an accuracy clearly superior to the other alternatives (i.e., it performs a simulation much closer to reality). Thus, it minimizes the communication overhead, which is important when distributed simulations are performed (Kumar, 1992), and distributes the overload efficiently over the set of computers involved.

6.2 Evaluation of the Coordination Protocol: Testing an Application Based on Mobile Agents

The flexibility of the proposed system enables its use to evaluate applications based on mobile agents. In this experiment, a sample scenario is shown to illustrate that a mobile agent-based application can be easily evaluated. This experiment evaluates the correct functioning of the proposed agent coordination protocol (see Section 4.2).

A scenario where different moving objects (mobile devices) obtain information on different topics that they want to share with their peers (e.g., sports news or stock quotes) is considered. When an object obtains new data about a certain topic (e.g., by querying computers located on the fixed network), it broadcasts the availability of new data on that topic to the moving objects located within a certain target area around its location (see Fig. 17.a). In this way, any other object within the area (interested in the topic) can query the reporting object (target object) to get the relevant data. On the interested object, there is a mobile information agent in charge of retrieving these data. For querying, two possibilities can be considered (see Fig. 17.b):

- The mobile information agent moves to the object holding the data. On
arrival in the target object, it analyzes and filters the data on the interesting topic. Then, it comes back only with a portion of the data (the data that it considers relevant to the interested object).

- The mobile information agent retrieves all the data on the interesting topic from the target object through a remote invocation. Then, the data obtained are analyzed and filtered.

![Diagram](image)

Fig. 17. Sample application and scenario: broadcasting new data available (a) and requesting relevant data (b).

A test has been carried out in order to evaluate the convenience of each of these approaches. The *ProxyAgent* class was specialized to add the capability of keeping track of the objects within the target areas; in this way, a moving object can query the set of objects it must notify. *InternalAgent* and *MovingObjectAgent* (see Section 5.1) were also specialized to implement the wanted behavior. In the scenario, a wireless bandwidth of 40 kbps and a random delay of up to one second affecting wireless communications is considered (to simulate unreliability). Additionally, it is assumed that the radius of a target area is half a mile and that 100 KB of new data on the interesting topic can be retrieved each time a target object is queried. Only a certain percentage of those data will be really relevant to the interested object. In the experiment, this percentage is modified and the average *query time* (time elapsed since the
appropriate actions to retrieve the remote data are started until those data are available on the interested object) with both strategies (a mobile agent and remote querying) are compared. For each percentage of relevant data, the experiment was repeated several times, obtaining similar results. The average results are shown in Fig. 18, where it can be observed that:

![Comparison of query time using mobile agents and remote communications](image)

Fig. 18. Comparison of the query time when using mobile agents and remote communications.

- Using a mobile agent is beneficial in most cases. Thus, it can perform data filtering on the target objects, reducing the amount of data communicated over the network.
- Only when the amount of data that it filters out is low (more than 90% of data on the topic must be communicated), the extra overhead of the wireless transmissions of the mobile agent does not pay off. Even in that case, the difference is minimal.
- Obviously, the query time with the remote approach is independent of the percentage of relevant data, as all the data on the topic must be transmitted anyway over the network (filtering is performed only on the client side).

It should be noted that, in the scenario presented in this section, different situations may arise. For example, an internal agent can move from one moving
object to another; when coming back (with the results retrieved) to its original moving object, that object may be under the coverage of a different proxy (and therefore the computer hosting the mobile agent that represents it may be a different one). Moreover, an internal agent may also move due to a request from its current MOA (an MOA must travel with its IAs), whether the internal agent is on its original moving object or visiting another. All these situations are managed transparently in the testing system, as explained in Section 4.

6.3 Evaluating Real Services Using the Testing System

In this section, the suitability of the proposed testing system is evaluated by using it to test two pre-existing wireless data services: a locker rental service (Villate et al., 2002) and a location-dependent query processor (Ilarri et al., 2006a).

6.3.1 Testing a Locker Rental Service (LRS)

This service offers its users the possibility of keeping their data in a secure and safe space (located on a proxy) called locker (Villate et al., 2002). In summary, the use of lockers provides wireless device users with the following advantages. Firstly, lockers alleviate the device exposure problem (wireless devices are more vulnerable and fragile than stationary devices, because they can be easily stolen, lost or damaged). Secondly, data stored in a locker are available for the agents at the proxy, even when the wireless device is disconnected, thus providing a solution for the availability problem (wireless devices might stay disconnected for long periods of time). So, specific tasks are carried out in the fixed network, with data stored in a locker, instead of on the user
device, in this way relieving the media problem (wireless communications are often unstable, asymmetric and expensive). Thirdly, lockers can follow user movements, moving from proxy to proxy, but always residing close to the current location of the user, also relieving the media problem. Finally, due to the use of agent technology in their implementation, lockers constitute an autonomous and auto-managed storage space.

Real data have been obtained in a scenario where a user residing in the USA travels to Spain, concretely to San Sebastian and then to Zaragoza. When the user arrives in San Sebastian, his associated locker travels there too. Consequently, while the user is at San Sebastian, file requests imply access to the local proxy. However, when the user arrives in Zaragoza, the LRS considers that it is better not to move the locker there (since there is not a high wired communication delay between Zaragoza and San Sebastian). Access times from Zaragoza to San Sebastian are similar to local access times, as wired communication costs are insignificant with respect to the (much higher) wireless communication costs (a GSM network is used).

The previous scenario has been simulated using three fixed computers to represent the proxies in the USA, San Sebastian and Zaragoza, respectively. The corresponding ProxyAgents simulate different delays among them (delays between those three geographical places using the fixed network) and the wireless network delay with the user device. The user device was represented as a mobile agent that implements a simulated moving object traveling through the three proxy areas. The modules of the data service have been used with no significant change.

---

9 With the exception that they use the Proxy Catalog of the simulated scenario.
In Figure 19, the X-axis shows the user requests at each location (ten accesses to files of about 190 KB) and the Y-axis shows the accumulated time of interaction between the user device and the locker (in seconds). It can be seen that the times are very similar whether they are obtained by simulation or not. Thus, by using the testing system, the results are similar to those obtained by running the data service using real proxies in different countries.

![Graph showing differences due to the use of the simulator.](image)

**Fig. 19.** Locker Rental Service: differences due to the use of the simulator.

6.3.2 **Testing a Location-Dependent Query Service (LOQOMOTION)**

**LOQOMOTION (LOcation-dependent Queries On Moving ObjecTs In mObile Networks)** is a software architecture, based on mobile agents, whose goal is the processing of location-dependent queries in distributed environments (Ilarri et al., 2006a), i.e., queries for which the answer depends on the locations of objects. It deals with contexts where not only the user issuing the query can change her/his position, but the objects involved in the query can move as well. An example of location-dependent query is: “show me the taxi cabs within two miles around my current position”. These queries are considered as **continuous queries**, i.e., queries whose answers must be updated continuously.
as opposite to instantaneous queries in which only a single answer is obtained for a query.

In Figure 20, it is shown the precision of LOQOMOTION when simulating the scenario of Figure 6 in different ways: 1) all the elements (proxies and objects) are simulated on the same fixed computer (the proposed testing system is used on a single computer); 2) one fixed computer is used to simulate each proxy (the proposed testing system is used in a distributed environment with six fixed computers); and 3) an alternative to the proposed testing system that tries to replicate a real scenario where each element is an independent device: no mobile agents and one fixed computer for each element are used, i.e., six fixed computer for proxies and eleven fixed computers more for moving objects (represented as processes communicating with proxies using remote calls). In these three simulations, each moving object communicates GPS location data to its proxy, and each proxy stores GPS location data of moving objects under its coverage. Objects move at 30 mph and the answer is refreshed every five seconds.

Fig. 20. Precision of LOQOMOTION for different simulation strategies.

The results shown in Figure 20 focus on the location of a certain moving object
retrieved by a query for a one-minute test. The precision of LOQOMOTION (in terms of location error) over time in the three simulations is very similar\textsuperscript{10}. Therefore similar results are obtained when using the proposed testing system (avoiding using many hardware resources) with respect to using an approach closer to reality (independent devices and more hardware cost).

7 Related Work

In this section, other interesting works are reviewed. The three works that are considered more related to the proposal in this article are the following:

- The \textit{integrated Mobile Computing Environment and simulation testbed MCE} (Rajagopalan et al., 1995) bears some similarities with the work presented in this article. Thus, it stresses the importance of minimizing code changes during the evaluation of algorithms, protocols and applications, so that they can be tested on the simulation testbed and then used directly on a real network. A distributed simulation infrastructure is also considered. In particular, host mobility is simulated by checkpointing and process migration in a UNIX environment, whereas in the proposal in this article mobile agents are used in any heterogeneous environment with Java. Unfortunately, the approach presented in (Rajagopalan et al., 1995) is not evaluated. Moreover, the focus is more on testing protocols and algorithms (e.g., routing algorithms) than on the evaluation of data services for mobile computing. Therefore, issues such as the simulation of moving objects executing applications based on mobile agents are not considered.

\textsuperscript{10}The imprecision incurred around time=25 seconds is due to a handoff of the monitored moving object, which in this case decreases the accuracy of LOQOMOTION.
• In (Satoh, 2002, 2003), mobile agents move among access points in a wireless network to emulate the movements of mobile devices. In this way, a mobile agent can use the service provided by its access point in the same way that a mobile device would. However, the goal is not to simulate mobile environments, not even moving objects. Instead, packaging an application as a mobile agent that travels across access points is proposed to avoid having to move physically the mobile device to access a wireless data service. Moreover, the issues that arise when the application itself is based on (mobile) agents are not studied.

• In (Kubach et al., 2001), both the mobility and information access patterns of mobile users are simulated, providing an environment for evaluating mechanisms to obtain location-dependent information. As opposed to the approach presented in this article, only a certain behavior of the moving objects is simulated: their requests to query location-dependent information. Moreover, evaluating the real data service on a simulated environment is not considered.

In the rest of this section, some other relevant works in three related areas are mentioned:

• Simulation of Mobile Networks. The goal of the proposed system is to allow developers to test high-level data services and not to evaluate network protocols. Therefore, this work is orthogonal to those dealing with the simulation of wireless networks at a protocol level (Kojo et al., 2001; Samfat et al., 1995; Zeng et al., 1998). In fact, such works could be used to improve the simulation of the proposed system at that level of detail (e.g., as mentioned in Section 5.1, it is possible to integrate it with ns-2). This category includes also works that, to evaluate network protocols, not only simulate
mobile networks but also the movements of the users; for example, (Samfat and Molva, 1997).

- *Simulation of Moving Objects.* These works are also complementary to the one presented in this article, as they can be used to simulate movements of objects in the proposed system:
  
  - On the one hand, several works propose different mobility models for moving objects (Camp et al., 2002; Schindelhauer, 2006), such as random walk, for different contexts. Moving objects in the proposed testing system can be programmed with any wanted behavior. Therefore, they can consider a predefined trajectory or any other mobility model (see Section 5.1).
  
  - On the other hand, and based on the previous models and/or other simulation requirements, several projects aim at generating location data for benchmarking purposes; e.g., (Brinkhoff, 2002; Hu and Lee, 2005; Myllymaki and Kaufman, 2003; Pfoer and Theodoridis, 2003; Saglio and Moreira, 2001; Theodoridis et al., 1999). These works focus mainly on the simulation of moving objects as generators of spatio-temporal data for performance evaluation, rather than on testing the functionality of high-level data services. As opposed to these works, the system proposed in this article allows the simulation of other behaviors of the moving objects (e.g., their interactions with proxies). Moreover, the possibility of mixing real and simulated objects is not considered in these works, and distributed simulations are not supported either. Nevertheless, these works can also be considered complementary to the proposal in this article: They can be used to simulate the trajectories followed by the moving objects in the testing system.

- *Use of Software Agents in a Simulated Environment.* Some works consider simulations to evaluate mobile agent systems (Li et al., 2004; Liotta et al.,
2002; Uhrmacher et al., 2000; Uhrmacher and Kullick, 2000); these approaches allow to reuse the original agent code in the simulation, which is similar to the goal of reusing the original data service code in the proposed testing system. In the context of multiagent systems, there are also works that highlight the importance of distributed simulations (as in this article) to avoid overloading a single centralized computer with simulation tasks (Ewald et al., 2006; Riley and Riley, 2003; Uhrmacher et al., 2000; Uhrmacher and Kullick, 2000). Finally, it is interesting to indicate some works that also use mobile agents for simulation purposes (Stone et al., 1996; Wilson et al., 2001). In (Stone et al., 1996), mobile agents are used to reduce the network traffic (they interact with their local proxy) and distribute simulation tasks for multiplayer games over a network. In ABELS (Wilson et al., 2001), mobile agents are used to link efficiently independent simulations that can be executing on different computers (as opposed to the work in this article, wherein mobile agents are core elements of the simulations).

Finally, it is interesting to indicate a recent seminal work (Spieß et al., 2008) whose goal is to evaluate the performance of an executable process description modeled with an extension of BPEL (Business Process Execution Language), proposed to take into account situations where some services are offered by devices in a ubiquitous environment. A straightforward simulation approach is proposed, where the devices are replaced with a web application server that introduces “virtual costs” for service invocations.
8 Conclusions

Due to the difficulty of testing wireless data services in a real environment (e.g., many real wireless devices managed by many persons within the coverage of a real wireless network could be needed), a system for testing these services has been developed. The main benefits of the proposal presented in this article are:

- It allows the developers of wireless applications to test their services with minimal changes in the application code. The system can be easily extended by the service tester to implement moving objects and proxies suited to his/her needs (reusing the original code of the data service).
- Moving objects are simulated using mobile agents, and proxies are simulated as execution environments for mobile agents (places). The use of mobile agents increases the accuracy of the simulation and allows accessing (real) services not available remotely.
- Several computers can be used to simulate proxies and moving objects easily, which allows to increase the parallelism and accuracy of the simulation. The data service is not aware of the number of computers used.
- It also allows the simulation of objects that execute multiagent applications. A suitable coordination protocol provides the synchronization needed between the original application agents and the agents used in the simulation.
- The proposed testing system is able to hide the true nature (real or simulated) of moving objects and proxies in the wireless network. So, it allows developers to test a real data service in (completely or partially) simulated wireless environments.
A sample use case has been presented, that illustrates how the system can be extended. Moreover, the system has been tested experimentally regarding two different aspects. On the one hand, its main components have been evaluated the simulation approach based on mobile agents and the agent coordination protocol proposed to test applications based on mobile agents. These tests have proven that distributed simulations using mobile agents achieve the best precision and that the agent coordination protocol enables the use of the system to simulate moving objects that execute applications based on mobile agents. On the other hand, the whole testing system has been evaluated by using it to test real data services, showing that the results of the tests are reliable. Finally, the developed system could be integrated with a low-level network simulator such as ns-2

References


