Towards the ubiquitous computer: A mobile agent approach

Carlos Bobed
IIS Department
Univ. of Zaragoza
50018 Zaragoza
cbobed@unizar.es

Eduardo Mena
IIS Department
Univ. of Zaragoza
50018 Zaragoza
emena@unizar.es

Abstract

New advances in mobile computing devices allow us to think about new approaches such as a functional ubiquitous computer. In particular, it would be very useful for users to move from one place to another having all his data and applications available, in the same state, and without doing anything special (a real ubiquitous computer).

In this paper we propose an infrastructure that makes the user independent from a specific computer/location. The intelligent environment detects the location of the user and presents his computer workspace on the closest computer. The system finds the optimal balance between remote and local access to user files and applications, to reduce the use of remote resources whenever possible. This is achieved by using mobile agents that manage the mobility of users. All the data (and applications' state) "move" wherever the user goes, giving him the impression of controlling a computer just by getting close to it. We have done some tests that proof the flexibility of our approach.

1 Introduction

The idea of separating the user from the usual view he has of the computer has been rounding around since Mark Weiser proposed to hide the computer in order to use it without noticing it [18]. The goal of this paper is not to hide computers, but to become independent from a specific physical computer, thus users would work on ubiquitous computers.

Mobility and independence are the ideas behind our proposal. The user should be able to move around just focusing on his work, and the state and all the data the user left in the last computer used must be available transparently in the new one. This virtual ubiquitous computer will instantiate in the nearest-to-the-user "physical" computer (whose computation power would be used).

Nowadays, there are some approaches that implement remote desktops (e.g. VNC [13], RDesktop [1, 8]) or use virtual machines that maintain their states (the Internet Suspend/Resume (ISR) project [7, 16]). However, these systems consume a great amount of network resources: the former has to refresh the image of the remote desktop continually (the local host only has to send keyboard and mouse events), and the latter has to send all the virtual machine state over the net.

To overcome the drawbacks of the previous approaches, we have designed an agent-based layer on top of the underlying OS that provides users with the desired mobility capabilities without any change in the applications. The user will have the same view of the system whatever computer he sits in front of, without having to do anything special to access all his files but to get near the computer. Manual login is not needed as the system detects users' location, by using an indoor-location system based on WiFi [15]: the user logs in automatically if the system decides so according to its predefined rules. Thus, in our system, physical computers are intelligent terminals which

offer themselves to users, who are not aware of the movement of his data and applications (a seamless movement). To achieve this, the system must transfer all the applications' state and keep the way the user works with his computer unaltered.

The rest of the paper is structured as follows. In Section 2, we present the agent-based mobile infrastructure that manages the users' movements. In Section 3, we describe the technological problems that our system must deal with. In Section 4, we overview our prototype. In Section 5, we review the related work. Finally, in Section 6, we draw some conclusions and future work.

2 Mobile infrastructure

Two main requirements have led the design of the proposed mobile infrastructure. Firstly, when the user moves from one host to another, he has to find everything in the same state as it was on the previous host (applications, opened files, resources, configurations, ...). Secondly, the local resources must be used as much as possible, avoiding to become a simple remote desktop application that consumes network resources continually. In this way, the usage of network is reduced and the response time of applications decreases.

To delimit the problem we deal with, we put three restrictions to our system:

1. All the available computers have the same OS and installed applications. This allows our system to move a user session data without taking care of whether an application is available on the target host or not.
2. All the applications keep their state at the moment when the user stops using a computer. This allows the system to transfer only this status information to restore the previous application state; this assumption is based on the existence of specific protocols for this session management such as XDMCP [19].
3. A generic user is used to impersonate real users. The system adapts this generic account depending on the user that takes control of the host computer.

In the rest of the section, we present the behavior of our system, the agent infrastructure proposed to reach our goals, and we describe how the system deals with user location.

2.1 System behavior

Our system detects whether a user is approaching to or going away from a computer and acts in the following way:

1. When the system detects that the user is leaving a computer, it closes his session after saving its state (all the data needed to restore his session). Also, our system stores information about the applications left running in background.
2. When the user gets near of another available[1] computer, the system retrieves the information about the user's files, makes these files transparently available to the new host computer and logs the user automatically in (restoring the previous session as the user left it). Finally, it forwards the display of the running applications the user left, if any, to the new host.

Not all the user files are transferred at once, but incrementally. When the user moves onto a new computer, only the most needed files (session, configuration and some currently opened files) are transferred there; the others are gradually and automatically sent as the user requests access to them. The user is not aware of this background process, so we manage to deceive him and finally have the most accessed files on the new host (theoretically, we would finish having all the files available locally). So, these files can be opened locally and then most of the work is done using the resources of the current computer.

2.2 Agent infrastructure

The mobile infrastructure that manages the movements of the user is based on mobile

[1]Some computers could be being used by other users or access could be restricted for that user.

agents. A mobile agent is a program that executes autonomously on a set of network hosts on behalf of an individual or organization [10]. Mobile agents execute in contexts denominated *places*. A mobile agent is able to pause its execution, travel from one place to another, and, once there, resume its execution.

We distinguish three different kinds of agents: *Guardian*, *MovementManager*, and *UserAgent* (see figure 1). The goals of each one are:

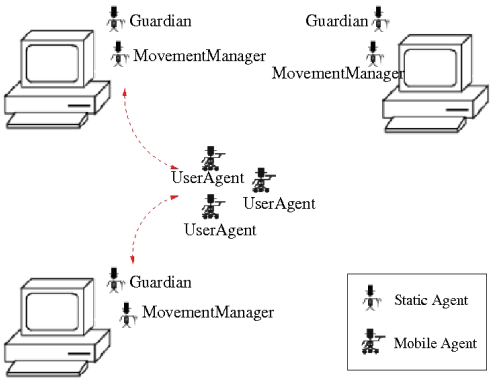


Figure 1: Agents of mobile infrastructure

- The *Guardian* agent manages the resources of the host on which it resides (there is one *Guardian* in each host). It automatically logs the users in and out and keeps the users' resources' information up to date.

- The *MovementManager* agent coordinates the movements of the (*UserAgents*) and the information interchanges across *Guardian* agents. There is one *MovementManager* in each host of our system.

- The *UserAgent* follows the user from one host to another, carrying with it the user's resources information updated. There is one *UserAgent* for each mobile user in our system.

2.3 Managing the location of users

In our prototype, we use a WiFi indoor localization method to acquire the user locations which gives us a precision of two meters, useful enough for our purposes. The localization method is not the goal of this paper and details can be found in [15]; however, any other localization method could be used in our system.

We want the underlying movement infrastructure (described in 2.2) independent from the particular way in which users are located. This is achieved by defining a location layer between these two modules (see figure 2). The location layer receives the user locations from the localization method and generates two kind of events, "$user_i$ is close to $computer_j$"(arrival event), "$user_i$ leaves $computer_j$"(departure event), which alerts the *MovementManager* agent on $computer_j$ when a $user_i$ gets close enough or leaves the $computer_j$, respectively.

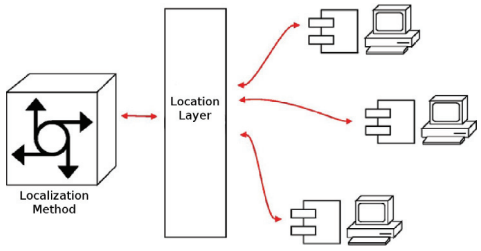


Figure 2: Managing the user location

The rules that guides the decision making process are very simple. A user can be in three possible states: out of range, near, and close to a computer. In our prototype, a user is near to a computer if his distance to it is smaller than 7 meter, and he is close to the computer if it is within 3 meter. A user must remain close to a computer for a predefined time to generate the corresponding event; similarly, he must remain in out of range state for a predefined time to generate a departure event.

When a MovementManager agent on $computer_j$ receives an arrival event, it requests the UserAgent of $user_i$ to travel to $computer_j$

and forwards a login request to the Guardian agent on $computer_j$; this Guardian agent will decide if $user_i$ is able to log in that computer. Similarly, when a MovementManager agent on $computer_j$ receives a departure event, it requests the Guardian agent on $computer_j$ to log $user_i$ out.

3 Technological challenges

There are three main issues we have to deal with: how to make the files accessible from any host and gradually transfer them to gain local access to them; how to make the applications follow the user, and how to keep the users' privacy.

3.1 File accessibility

One of the hardest problems is to have all the user's files available for him at any time but trying to access them locally most of the times. If we do not want to stop the user accessing his files when he moves to another host, the first accesses must be remote. Our main idea is to make the file operations smart enough so that they would be in charge of finding where the needed file resides (we propose a virtual file-system as described in figure 3).

We studied some existing remote filesystems such as NFS (3.0) or AFS [5], but they do not provide local cache for the files (NFS v4.0 is the first NFS version that implements it); some distributed filesystems such as Coda [4] or Intermezzo [3] have been also studied, but the cache they provide is not persistent and they did not relieve us from the use of a "centralized" cluster (we want to have host computers as free as possible).

Figure 3 shows how our virtual file-system works. We have all the data about the files in a special directory tree, which would be shown to the user; at the moment of accessing the files, the modified functions would test whether the file has been already brought to the local computer, or if it has to be accessed remotely. Once a file is transferred to the local hard disk, it has to be tagged as local and have the remote copy erased. Meanwhile, the local

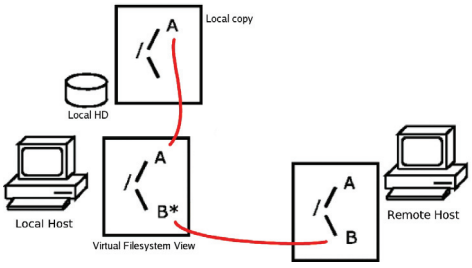


Figure 3: Our FS concept

directories are filled in background so, in the end, almost every access would be local (theoretically, all of them because of the progressive background process).

In figure 3, the file named B is tagged as remote and our filesystem would access it remotely, while the "A" file has been already carried to the new local host and properly tagged as locally accessible (the remote "A" file would be erased by a garbage collector).

3.2 Movement of applications

The applications cannot be made travel from one host to another (we are not talking about just standalone plain processes, but applications that can have resources that can't be moved), but we desire to make the user have that view. We use the X-Windows system as underlying display system due to its separation into XServers and XClients, which gave us more flexibility than other integrated display systems (such as the model that Windows OS uses). We thought of changing the screen which the applications forward its windows into, but this could not be done because of two problems:

- The applications get the value of the environment variable DISPLAY (a variable which tells them where to display their windows) only at the moment they are launched, so any later change of the DISPLAY variable does not affect their read value.

- The information of each window is stored in the server-side of the connection, the XClient accesses to it via an identifier. Even if the application could change the DISPLAY value to the XServer in the new host, the new XServer would not have any idea of the information that is associated to the identifiers that the client presents to it.

We studied the use of a VNCServer to forward the windows to another hosts, but we found different problems. A VNCServer is associated to an XServer, so if we want to keep running all the applications of a user while he moves, we would have to keep the XServer alive.

This would lead us to make a decision between: 1) to keep n XServers running on each host of the system (one XServer for the windows of each of the n mobile users), or 2) to block completely the host to other users once a mobile user has entered it (in order not to mix different users' windows into the same host). Neither of the options are acceptable.

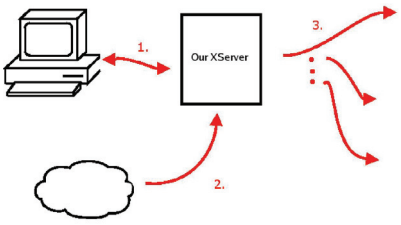


Figure 4: Our modified XServer

We propose a modified server (figure 4) that behaves in three different ways depending on who interacts with it:

1. It is a normal XServer to the local applications, so they could run without any modification.

2. It accepts movement requests from an external party to forward the windows of the applications to the proper display.

3. It behaves as an XClient to the remote XServers which it forwards the windows into.

The different resources in the modified XServer would have to be tagged as owned by the proper user. Having this server, we could make the applications give the sensation of movement to the user, without having to recode them. The Xdmx server[2] behaves in a similar way but it would require to be modified to accept the movement requests and fit exactly our wanted behavior.

3.3 Privacy

Privacy is a very important issue when we deal with user's profiles, files, configurations and autologins. There arose a problem with this matter when we had to manipulate a generic user to make it appear as the same one in different hosts.

Due to this modification of the generic user, all the mobile users are regarded as the same one for a unique host. This could be used by a malicious user to run an application to monitor other users in background (an eavesdrop attack). As the OS sees the application as being launched by the same user (the generic one), it would have the proper rights to access to all the actions of the next users that log in that host. We could control it at XServer level (with our modified one), but not at OS level. There were two options: to kill all the user's processes when he moves (unacceptable for us) or to modify the underlying OS to support a kind of mobile users.

So we propose a simple modification which consists of adding a little tag to the structure of user identification. This little tag could be used for controlling who is the real mobile user that is behind the generic user identity. Now, although its cost, by modifying the kernel of Linux, for example, we could control the access to the resources to support mobile users in a flexible way and we could assure user's privacy.

[2] *http://dmx.sourceforge.net/*

4 Prototype

To implement our prototype, we had to relax some of the restrictions we had imposed to the system.

When we thought about how to manage the user's files the first option was to pack the user's account (limiting it to a determined maximum size) and send it to the new host (just the brute-force solution). The option we have finally implemented is to divide the user's account into two kinds of directories: static or mobile. Mobile directories are packed and travel with the *UserAgent*, while static directories are accessed always remotely. As we see in figure 5, files can be both transparently accessed locally or remotely depending on the class of their parent directory (properly tagged as mobile or static). This gives us the freedom to select which directories are going to travel in the next move.

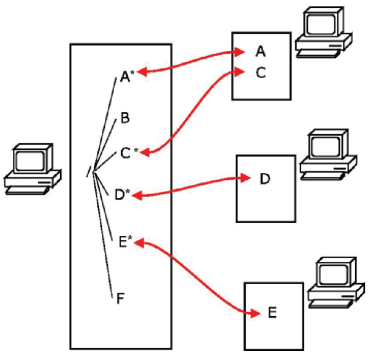
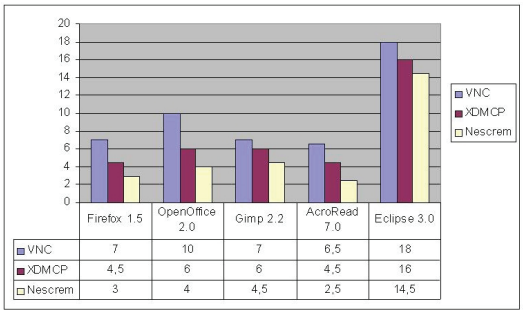


Figure 5: Transparent file access

The prototype (*Nescrem*) runs on Linux and has been implemented in Java, using the Grasshopper Agent platform [6] and KDE as Desktop Manager. Movement of the applications has been delegated to them implementing the XSMP protocol (X Session Management Protocol [19]). Fortunately, almost every application in KDE and Gnome Desktop environments implements it and that allows the system to move the state of the applications just moving the appropriate session files.

The results were more than acceptable. When the user moved to the new host, he found all his files available, the desktop almost in the same state (most of the applications implement the XSMP protocol), and all the work done locally. In figure 6, we show the sum of load and response time (execution time) of some well known applications for remote session protocols (VNC and XDMCP) and the Nescrem prototype[3].



	Firefox 1.5	OpenOffice 2.0	Gimp 2.2	AcroRead 7.0	Eclipse 3.0
VNC	7	10	7	6,5	18
XDMCP	4,5	6	6	4,5	16
Nescrem	3	4	4,5	2,5	14,5

Figure 6: Execution time of different approaches (in seconds)

We now focus on session restoring time and bandwidth usage. Concerning restoring time, we see in figure 7 that our system allows high mobility scenarios (imagine a user moving from one room to another). It must be noticed that the sizes showed in the figure are not the total user's account size, only the transferred files (the system can decide whether to move or not a file dynamically, which allows us to have large user's accounts, but only transfer initially a small part of it). The restoring time is linearly proportional to the size of the part of the account transferred.

The bandwidth test consists of monitoring two consecutive user sessions for 10 minutes. The sessions involved opening the above mentioned applications and doing some default actions (editing or viewing some files, playing some games, and so on). As we can see in figures 8 and 9, our approach consumes less bandwidth than the others. We only consume bandwidth when the user moves or when he

[3]The test is performed in Pentium IV 2Ghz machines 1GB RAM running GNU Debian OS

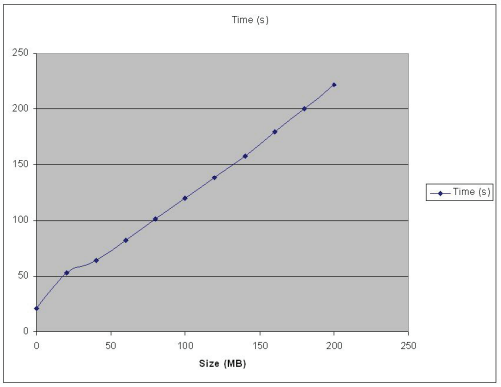


Figure 7: Session restoring times depending on the size of transferred files

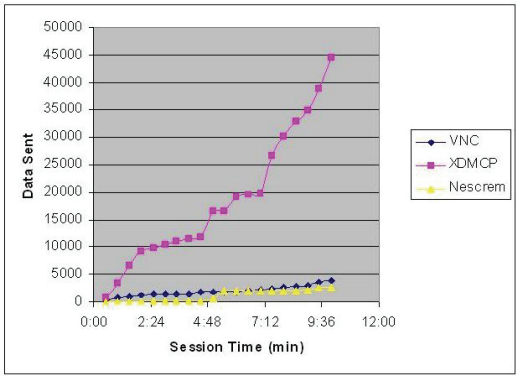


Figure 9: Bandwidth results (data sent in a 10 minutes session)

accesses a file which has not been transferred yet. Both tests (response time and bandwidth usage test) have been performed over an Ethernet 100Mbps network.

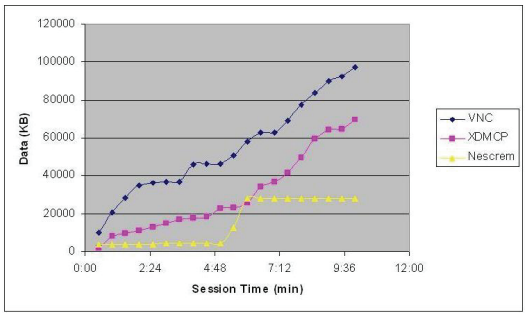


Figure 8: Bandwidth results (data received in a 10 minutes session)

5 Related work

In [7, 16], they present the so-called ISR technology (Internet Suspend/Resume) whose goal is the same as ours is. They work with virtual computers (VMWare machines[4]) by sending all the system state over the Internet when the user moves, which nowadays make their system not feasible to use in some high-mobility scenarios. Their approach is expensive in terms of bandwidth, resuming time (in their best case they have to send an average of 47MB and their response time falls to 45 seconds over a 100Mbps network, and in their worst case all the VM state has to be transferred incurring in a 43 minutes session restoring time at the same network speed) and computational power (everything is executed inside a virtual machine). Our approach is more flexible to the user movement at the cost of loosing the seamlessness requirements. We also use all the computational power of every computer of our system when the user works with it.

With remote desktop technologies (such as VNC [13], Remote Desktop [12] - a opensource client for Microsoft's RDP [8]- or Remote Administrator [11]), the user always uses the same computer. They transform the in-front computer in just a simple terminal to which the user's computer forwards the desktop. Although they manage to show exactly the state of the desktop, the network dependence and the bandwith needed to refresh the remote desktop are high, and they do not really hide the use of the system to the user (he has to know how to work with remote desktops and has to control the movements by hand). Sun

[4] *http://www.vmware.com/products/ws/*

offers SunRay [9], a more advanced remote desktop technology. It is a three layer system that makes independent the running applications from the final client hosts. They allow to have different OS applications being executed and forwarded to the same final client, but it shares the remote desktops drawbacks. Our system allows the applications run locally, avoiding the refreshing costs and response delays, and automatizes the movements.

Google offers a web-based approach with its Google Docs project[5]. They provide some common applications (a word processor and spreadsheet editor) that can be accessed via web-browser. This approach make user change radically his view of the computer; and, what is more, the user does not have all its applications available, having to adapt to the new tools.

Finally, other approaches propose to develop a middleware on which programming the ubiquitous software [2] from scratch, to create a new operating system which would be able to provide us with ubiquitous resources and behavior [14], or even to use mobile tasks [17]. But this was not what we were looking for, we did want to reuse all the existent applications without having to recoding them.

6 Conclusions and future work

In this paper we have presented a system that gives the real feeling of using an ubiquitous computer: the user can access his files, applications and configurations at any time and from any computer.

The proposed technological infrastructure is enough to reach our goals and, although some issues are still open, the results we have reached with the prototype are very satisfactory. In particular, the resuming times we reached make it react very quickly to the user's movements.

We have managed to make the movement of the account and applications of the user invisible. The advantages of our system system:

- The access to the user's files is not always remote or local: it varies dynamically from remote to local as the user moves depending on the rules defined.
- Applications are executed locally.
- A high level of seamlessness in the movements is achieved and the way the user uses the computer is not changed.
- And, finally, we have reduced the resuming time that other approaches reach [16] by adopting some trade-offs.

As future work, we plan to add the notion of domains to our system, different environments in which the user can be (his job, his home, ...); and to study how to use computers with different software configurations (different installed applications, different desktop environments, ...) in our system. We also plan to add more control features to raise the flexibility of the users movements, i.e. allowing the users to specify the computers they wants to use.

References

[1] Application sharing, recommendation t.128. Technical report, Microsoft, PictureTel, Polycom. *http://www.rdesktop.org/docs/t128.zip*.

[2] P. Bellavista, A. Corradi, and C. Stefanelli. Mobile agent middleware for mobile computing. *Computer*, 34(3):73–81, 2001.

[3] P. Braam, M. Callahan, and P. Schwan. The intermezzo filesystem - in o'reilly perl conference 3.0. o'reilly, 1999., 1999.

[4] P. J. Braam. The coda distributed file system. *http://www.coda.cs.cmu.edu/ljpaper/lj.html*.

[5] P. J. Braam. File systems for clusters from a protocol perspective. *http://citeseer.ist.psu.edu/253433.html*.

[6] IKV++. *Grasshopper Programmer's Guide*, 2001.

[7] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, USA, June 2002. IEEE Computer.

[8] Microsoft. Remote Desktop Protocol, http://msdn2.microsoft.com/en-us/library/aa383015.aspx.

[5] *http://docs.google.com*

[9] S. Microsystems. SunRay Overview, *http://www.sun.com/sunray/techinfo/New_SR_WP_12_04.pdf*.

[10] D. Milojicic. Mobile agent applications. *IEEE Concurrency*, 7(3):80–90, 1999.

[11] RAdmin. Remote Administrator, *http://www.radmin.com/*.

[12] RDesktop. A Remote Desktop Protocol Client, *http://www.rdesktop.org/*.

[13] RealVNC. *http://www.realvnc.com*.

[14] M. Román, C. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 73–83, 2002.

[15] J. Royo, E. Mena, and L. Gallego. Locating users to develop location-based services in wireless local area networks. In *I Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI2005), Granada (Spain)*, pages 471–478. Thomson, ISBN 84-9732-442-0, September 2005.

[16] M. Satyanarayanan, M. Kozuch, C. Helfrich, and D. R. O'Hallaron. Towards seamless mobility on pervasive hardware. *Pervasive and Mobile Computing*, 1(2):157–189, June 2005.

[17] J. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. in: Proceedings of 3rd ieee/ifip conference on software architecture, montreal (2002), 2002.

[18] M. Weiser. Ubiquitous computing. *IEEE Computer*, 26:71–72, October 1993.

[19] M. Wexler. X session management protocol. Technical report, X Consortium, Inc., 1993-1994. *http://www.xfree86.org/current/xsmp.pdf*.