UbiTerm: A hand-held control-center for user's activity mobility

Katia Leal, Francisco Ballesteros, Enrique Soriano, Gorka Guardiola
Laboratorio de Sistemas, Universidad Rey Juan Carlos
C/Tulipán SN, Mósteles, Madrid, Spain.
{kleal, nemo, esoriano, paurea}@lsub.org

Abstract

Current approaches handling user's activity mobility address the problem by imposing system decisions instead of user's indications in an attempt to reduce user's distraction. At the same time, these traditional approaches introduce uniformity on the environment, thus preventing users to take full advantage of the computational resources found nearby. We describe an alternative architectural model that better solves the problem of supporting user's mobility, reducing user's distraction, and respecting user's preferences. One of the key features of our solution is that users can directly control their activities. Thus, the Plan B's Ubiquitous Terminal (UbiTerm) provides activity control commands to users. In addition, to take appropriate actions when adapting the environment to user needs, the UbiTerm uses a context information mechanism. Finally, by using a programmable service for the remote execution of applications, the architecture can exploit local capabilities, provide support in different platforms, and fit activities to user's preferences.

1. Introduction

One of the most important issues introduced by ubiquitous computing [28] is to support user's mobility, and so user's activity mobility. The ideal architecture model would exploit local capabilities, provide support in different platforms, and fit activities to user's preferences. Moreover, it would support user's mobility reducing to the minimum user's distraction, and respecting user's preferences to the maximum.

We have classified current approaches to user mobility in three different technologies, none of which fully achieves the goals pointed out. First, we have mobile solutions. Basically, these techniques consist in carrying user's data, and possibly computing, on a mobile device. A second approach are server-based architectures that place computing and storage on a network of servers. A third approach consists in middleware infrastructures to support active or smart spaces. In this schema standard applications are ported or wrapped to conform a particular middleware, and installed in all spaces.

At least there are two problems with these solutions. One is that they impose uniformity on the environment, so users can not take full advantage of the computational resources found nearby. The second is that even though some of these approaches have included abstractions representing the activity of the user in an ubiquitous environment, they do not provide users with the possibility of controlling those abstractions. Instead, in some cases they infer user's intentions in order to transfer user's activities to the right place. In other cases, the image of the activity of a user is migrated as soon as the user arrives at the new location. As a result, user's activity mobility follows system decisions instead of user's indications.

The approach we propose is to place control at the user, with a centralized implementation that governs the behaviour of the pervasive environment. Thus, user's activity mobility obeys user's directives instead of system decisions. The key features of the architecture we propose are the following: first, it provides *activity* control commands to users. Second, the activity abstraction represents a generic human action, so it is not tied to a particular application or system. Third, the use of a context information mechanism permits to take appropriate actions when adapting the environment to user needs. Forth, a programmable service for the remote execution of applications in different platforms allows users to exploit environment resources. Moreover, the combined use of the context information mechanism, and the remote execution service, permits us to deal with heterogeneous environments.

In this paper we will see that our architecture has several advantages. By providing a unique point from which to control all the user activities, users gain the domain of the sys-

tem without being forced to make configuration tasks. By representing user's activities as generic human actions, the architecture can identify several applications implementing those actions, thus providing support across different platforms. The architecture can also notify the user that is not possible to support the activity. By using a context information mechanism the environment information is always up to date, so the architecture can accommodate to dynamically changing resources. With this environment information the architecture can decide if it is possible or not to provide a configuration capable to support the activity. In addition, it can search for the configuration that better matches user's preferences. Finally, by using a programmable service for the remote execution of applications, the architecture can exploit local capabilities, provide support in different platforms, and fit activities to user's preferences.

The rest of the article is organized as follows. Section 2 illustrates how the UbiTerm architecture works using an scenario in which an activity follows its user to a new location. In Section 3 we describe the proposed architecture. Section 4 explains the Plan B's Ubiquitous Terminal structure. Related work is discussed in Section 5. Section 6 presents some conclusions, and the plans for future work.

2. How does the UbiTerm Architecture work?

To illustrate how the UbiTerm architecture achieves its goal of supporting user's activity mobility, we describe a simple scenario of an activity following its user to his current location. We will focus on the interactions among the components forming the UbiTerm architecture that we describe in Section 3.

Let's consider the following scenario. Katia is located at Quique's office, and has Quique's (verbal) permission to use his devices. In 10 minutes there is a talk at Quique's office, and Katia has to present it. In the last moment Katia remembers that she was working in the presentation before she left her office. If she wants to continue that activity at Quique's office, because she is new in the current environment she will be forced to do a lot of configuration work: enter the system (which one?), find the corresponding application that edits the file format, and possibly to move the (file) presentation to the current system.

Now, look at the figure 1. In this case Katia carries her mobile phone which is at the same time her UbiTerm. When Katia enters Quique's office she asks for permission to him. Then, their UbiTerms exchange information to establish the corresponding permission level based on human trust[22]. As you can see, none of them have to make configuration or administration tasks such as the one that consists in creating a new user for Katia in the different systems at Quique's office.

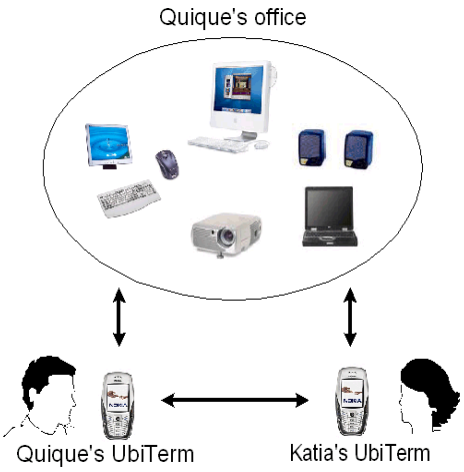

Figure 1. Katia's UbiTerm adapts her activities to Quique's office resources.

Next, when Katia remembers that she has not already finished the presentation, she asks her UbiTerm to move her 'presentation' activity to the current location. As we will see, to facilitate the interaction with the UbiTerm, this will be provided with different modules supporting typed commands, voice and gesture. This activity is within the UbiTerm because it was created when Katia was at her office. Instead of asking explicitly for a concrete application for editing presentations, Katia asked for the human action. Thus, her UbiTerm used an activity-binary(system) table that translated the activity into its corresponding binary or binaries. For example, this 'presentation' activity entry produced the following list: Open Office Impress(Linux), Power Point(XP), Power Point(Mac). Once the UbiTerm knew the possible binary-system combinations he had to look for a Remote Execution Service providing one of those binaries. Thus, Katia's UbiTerm used the Context Information Mechanism to look for a Remote Execution Service located in the same place as Katia was. Also, the UbiTerm could also have looked simply for a Remote Execution Service providing that binary, and have programmed it to use available I/O resources at Katia's office. Katia (*who*) provided her current location (*where*) to her UbiTerm, for example, office 127 . Thus, the UbiTerm used the Context Information Mechanism to consult available resources (*what*) at office 127. In addition, if it was possible the UbiTerm used the favourite I/O devices for Katia to operate the application.

Now, the UbiTerm has to reproduce the 'presentation' activity with Quique's office resources. At least, the UbiTerm can do it in two different ways. First, it can use activity current Remote Execution Service to change the set of I/O

resources of the running application. Second, it can also use activity current Remote Execution Service to close the running application, and look for another Remote Execution Service providing the binary at Katia's current location. In order to decide what to do, the UbiTerm considers different aspects, such as performance. As a result, instead of moving back to her office to finish the presentation, Katia tells her UbiTerm to move the presentation to her current location. Once the activity is running in the new space, Katia can forget her UbiTerm till the moment she needs to control again her activities.

3. Plan B's Ubiquitous Terminal

The Plan B's Ubiquitous Terminal idea comes up out of the necessities of helping users to perform tasks ubiquitously, but satisfying two important issues. First, reduce to the minimum user's distraction. Moreover, while users are performing their activities they do not have to be disturbed, forgetting about their UbiTerms till the moment they need to control again their activities. Second, user's preferences have to be respected to the maximum. Furthermore, the environment heterogeneity do not have to be a problem when exploiting available resources.

In a pervasive environment there are many different ways of disturbing users. For example, by forcing them to make configuration or administration tasks when they manage computing resources in a new environment or because the resources change. To reduce this kind of user's distraction, almost all available approaches try to infer user's intentions in order to transfer user's activities to the right place. However, if the system fails in its predictions the solution will become the problem. Because we do not act always in the same way, the system will take wrong decisions many times, thus introducing more distractions to the user than it is trying to avoid. Our solution reduces user's distraction by using a context information mechanism to obtain information about the environment, so our architecture can accommodate to dynamically changing resources without forcing users to make configuration or administration tasks. In addition, in our solution is the user the one who decides what to do with every activity, when, and where. Although we introduce some degree of distraction to the user because of controlling activities, it is minimal. As a result, user's activity mobility follows user's indications, and not system decisions.

In the Systems Lab at Rey Juan Carlos University we are working in a solution to this problem based on the *UbiTerm* (Ubiquitous Terminal). The idea is integrate the tiny program implementing the UbiTerm in a mobile device (eg. a programmable mobile phone) users carry with them, so the UbiTerm is always available. However, we have to say

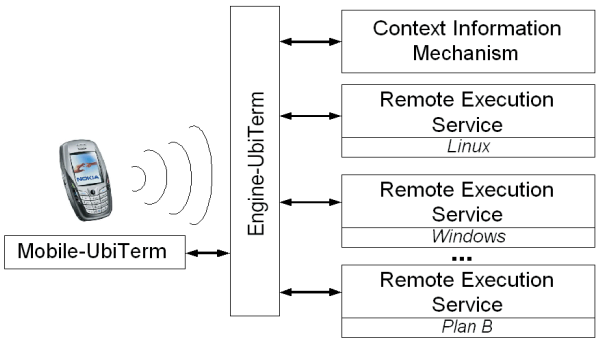

Figure 2. Plan B's Ubiquitous Terminal's Architecture Components.

that the UbiTerm is just a control device, not a server. In fact, the UbiTerm is like a terminal in the sense that it provides commands to permit users control their activities: create, start, stop, kill... To facilitate the interaction with the UbiTerm, this will be provided with different modules supporting typed commands, voice, or gesture. When a user wants to perform an activity, instead of asking explicitly for a concrete application, he asks for the human action.

The UbiTerm uses an *activity-binary-system* table that translates every activity into its corresponding binary or binaries. For example, a 'navigate' activity entry will produce the following list: mozilla-Linux, firefox-XP, links-Plan 9. Once the UbiTerm knows the possible *binary-system* combinations it has to look for a remote execution facility (`/cmd` [1] in Plan B) providing such a binary. So, the UbiTerm uses the context information mechanism [10] to look for a remote execution service providing that binary, and programs it to use available I/O resources in the current location according to user's preferences.

Once the activity is in the UbiTerm, the user can control it using the set of commands provided by the UbiTerm. Thus, when the user moves to a different location, he can ask the UbiTerm to move an activity to the current location. In this case, the UbiTerm reproduces the activity with current available resources. Again, the UbiTerm uses the remote execution service to change the set of I/O resources of the running application; It can also use the remote execution service to close the application; Next it can look for another remote execution service providing the binary at user's current location. The UbiTerm considers different aspects, such as performance, in order to decide what to do. Once the activity is running in the new space, the user can forget the UbiTerm till the moment he needs to control again his activities.

The UbiTerm is part of an operating system that provides

the context information mechanism, and the remote application execution service. This new research operating system that we built is named Plan B [3]. You can also refer to [5] both for a description of Plan B and to see how we have incorporated its mechanisms into Plan 9 [17] to get a system that adapts to changes.

As we can see in figure 2 there are four components types in the UbiTerm architecture. The *Mobile-UbiTerm* provides the front-end for the user to control the activities, and for the system to reach the user. Second, the *Engine-UbiTerm* embodies the politics to solve Mobile-UbiTerm requests (user's activity commands). These two components form the *UbiTerm*. Third, the *Context Information Mechanism* provides information on the physical context (who, what, where). Fourth, the *Remote Execution Services* embody a programmable service for the remote execution of applications in different platforms. Although we have included the Context Information and the Remote Execution as components of the UbiTerm architecture, they are Plan B's mechanisms that provide a generic interface to the whole system. So, they are separate components and can form part of different architectures. An environment has one instance of each of the types: UbiTerm and Context Information Mechanism. Also, each environment may have several Remote Execution Services.

Although information about the user (eg. the current location, the preferred audio output device, etc.) is kept centralized within the UbiTerm, such information may be cached within the distributed system to tolerate disconnections of the UbiTerm. As it is centralized it becomes easy to operate on that information within the UbiTerm. Another consequence is that when some component or program of the distributed system wants to know something about the user, there is a centralized place where to look at—and the user retains control on that information.

3.1. User activities

The UbiTerm introduces a new abstraction to represent the human action that the user is performing in the system. We use the name ***Activity*** for such abstraction. Although the activity abstraction is kept within the UbiTerm, the idea is that the processing and the set of I/O devices to operate the activity are not in the UbiTerm, but in the surrounding environment. For example, the UbiTerm will contain an activity 'Listen to X' as soon as the user starts to reproduce it. However, the player will run at a computer found in the environment, and the I/O devices will come from the environment too.

Continuing the example, when the user requests the player to stop, the request is made to the UbiTerm activity. If the UbiTerm is disconnected from the system, it will wait upon reconnection to connect to the player and request it to stop. Finally, when the user is back at the office, he will have another set of speakers. Assuming the activity was resumed or never stopped, the UbiTerm will connect to the computer running the player and instruct it to use the appropriate audio output device.

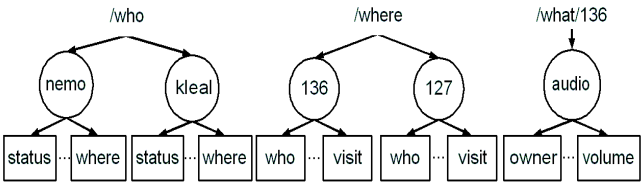

Figure 3. Our context hierarchy. Used to update and to use context information.

To facilitate management of activities, we also include a ***Session*** abstraction. We have used the term session because we think it matches well with its functionality, but it is not a session in the traditional sense. Our session is simply a tag that permits the user to group activities according to some criteria only meaningful to the user. For example, one user can create a session 'work', and a session 'home' to group activities created while at home or at work. But another user might create sessions project' and 'documentation' to group activities according to the kind of work. One of the sessions is considered "current", and all the activities created are tagged for that session. The user can add more tags (i.e. attach the activity to more than one session) if it's desired.

If the UbiTerm state is lost, due to a device crash, it will learn of already started activities as they are seen in the network. Thus the UbiTerm state is, to some extent, recoverable. Although a crash of the UbiTerm can be very inconvenient for our users.

3.2. Context Information Mechanism

The context-awareness "framework" we use is simply a set of directories. Our main file server includes a series of directories where context information is to be placed, so that no other file system needs to be mounted to access context. Nevertheless, we also have small file systems to keep context for mobile devices and people. They are similar to ram based file systems used for temporary storage. Such file systems are handy while outside of our smart space; i.e., while disconnected at home or while we are in the subway.

Figure 3 shows a typical context hierarchy. Users have a directory (one for each user) for their relevant context information. Each piece of context for a user is represented by a file in its context directory. For example, `/who/user/where` is a file that contains the last known

location for *user*, and `status` is a file that contains a descriptive string about the user status (eg. `busy` or `idle`). Each place has also a directory that contains its context information. As examples of context information for places we can mention the file `who`, which contains one line per user known to be at the space, and also `visit`, which contains either `yes` or `no` depending on the answer to *"Are there humans present in the space other than its owner(s)?"* Finally, things (including devices and services) also have their own context directory. For example, the context directory for an audio device includes a `owner` file containing the device's owner; there is also a `volume` file containing the device output volume level desired by the owner.

The context information stored is not assumed to be accurate, it is as accurate as the tools used to extract it from somewhere else in the system. The set of tools used to extract, merge, and use context information is still growing. It includes tiny shell scripts as well as more complex C programs. It is important to note how we can also use simple programs like the UNIX `echo` or the Windows *Notepad* to update our context information or to turn off the lights.

Users and space administrators are free to run whichever tools they see that fit to extract and use context. The different tools work together using the file system to exchange information. This works very much like the UNIX environment did time ago, by combining simple programs to perform complex tasks. Therefore, we do not have to use one solution for all the problems, and users can customize how the system extracts and uses context on their behalf.

3.3. Remote Execution Service

To let the UbiTerm control the programs and devices distributed in the environment, it needs some kind of interface to operate on them. In Plan B we have a remote execution facility, */cmd* [1], that allows us to start remote programs and to dictate which I/O devices must be used by those programs at any time. Programs may be started with a fixed set of I/O devices (and their I/O will not follow the user), or they may be started with a dynamic set of I/O devices to follow the user. The description of this service is beyond the scope of this paper, because here we focuss on how to apply the UbiTerm both for our system and for traditional ones like Windows, UNIX, Plan 9, or Symbian.

To use the UbiTerm with other systems, we need a remote execution interface for those systems if we want the UbiTerm to start programs on them. Fortunately, such service is usually available on most systems, and can be easily built otherwise. However, we also need a way to plug the appropriate I/O devices to those programs. Such devices and the programs using them may reside at different machines.

What we do is to export devices as files, and provide remote access for such files, borrowing the idea from the Plan 9 Operating System [17]. Distributed access to files is a well-known technique, and it allows the programs we start to operate on remote devices.

By using Plan 9 for daily work and by experimenting with our research prototype, we have gained experience with this approach. By carefully choosing how to model resources as files, access to devices and resources can be even more simple than using other approaches like distributed object systems [6] or approaches like Ninja[9]. For example, our X10 service [4] for motion detectors and power switches, which is also exported through the web [1], uses files to represent each sensor and each power switch. A simple write of "on" or "off" can set a switch to the desired state. A simple read can let a program know the status of a switch or a motion detector.

At first sight, it may be argued that the lack of a type system would be a problem for this approach, but experience says it is not. For simple interfaces, a write of an incorrect request for a device would usually return an error for the write operation ("bad control request"). Complex interfaces do not use a single file, but a hierarchy that can include several directories. We have found that what is important is to carefully chose which files to service, and what do they represent. As a more complex example, the graphical user interface service of Plan b uses a hierarchy of files to represent a hierarchy of graphical components like buttons, menus, etc., as said in [1].

Using files to access devices makes issues like authentication, access control, concurrent access, and communication to become mostly solved. The part that must be addressed is how to employ a dynamic scheme suitable for multiple file clients and servers that might not share an authentication service. We discuss elsewhere [22] how we address this issue.

As a final remark, this approach is very portable and makes it easy to interoperate with most systems. All that is required for a system to export a service is a little server program to export the device considered as if it were a set of files. And this program can be really tiny, as demonstrated by the server used to export the sensors and actuators from a Lego Mindstorm Brick [15]. We are using BP [1] and 9P [16] as the file system protocol, but any other could work. Since most devices know how to exchange or remotely access files, things get even easier[1].

[1]Note that the same set of (virtual) files can be exported through different protocols at the same time. For example, a Nokia 6600 may use bluetooth to share a file to export its keypad, and at the same export the file using 9P on a TCP stream.

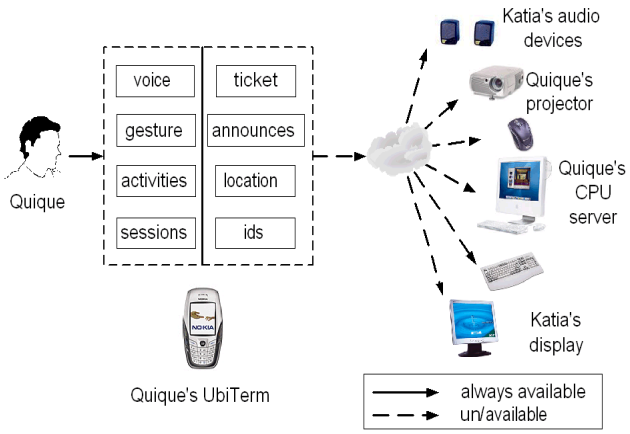

Figure 4. Ubiquitous Terminal's Structure.

4. UbiTerm Structure

The UbiTerm has two different conceptual modules as seen in figure 4:

- ❒ **User Interface**: for the user of the UbiTerm to command the system.
- ❒ **System Interface**: for the system used by the user to reach the user.

The user can control the system through the *User Interface*. In response, the UbiTerm would use modules from the system interface to perform the appropriate actions. In a similar way, the *System Interface* provides user information to the rest system, and it uses the user interface when help from the user is necessary. Both interfaces are built out of different modules, as depicted in the figure 4.

Multiple modules in the user interface allow for different methods, like typed commands, voice, or gesture. This does not mean that the processing (eg. voice recognition) must be made within the UbiTerm, the module could arrange for a program running at a nearby machine to perform the processing. The two other modules in the user interface, Activities and Sessions, implement the corresponding abstractions.

The modules in the system interface are there mostly to provide information for the rest of the system, and there is usually one module for each kind of that information.

The UbiTerm establishes a control relationship among the different resources and programs participating in a given activity. Data paths go only between the application and the devices. The UbiTerm interferes in such relationship only because of dynamic or due to a user specified action on the activity. Decisions and requests made by the user while the UbiTerm is disconnected are not lost, they are kept within the UbiTerm so that the user could forget about them.

Regarding the policies involved, like where to execute a program, we follow simple ones. They can evolve once experience is gained with the UbiTerm. For example, in our current architecture, the UbiTerm would try to execute an application on the first machine known by it that has the requested program and is willing to execute it. If the user wants a particular machine to execute such program, he can select the machine in the UbiTerm for the occasion.

4.1. User Interface Layer

The *User Interface* is the shell for the user to control the system. It has a set of modules that, for the moment, consider this functionality:

① *User's activities*: list of user's activities for the current session.

② *User's sessions*: list of user's sessions.

③ *Commands for activities and sessions*: the UbiTerm includes a set of controls for all the above.

Activities and sessions can be operated like a telephone directory in a mobile phone. The user may navigate the list of sessions, select one, and reach the list of activities for the session. Then select an activity and delete it.

However, to make it more convenient to use, we consider other input modules including voice commands, and gesture modules. The set of commands to be recognized by the voice recognition module depend on the part of the UbiTerm being navigated, and it may be feasible to perform the recognition even within the UbiTerm device.

Finally, we must remember that for applications started, the UbiTerm controls which I/O devices they should use, but data paths go from the devices to the applications without passing through the UbiTerm. This means that while the user maintains a given location, a more convenient set of I/O devices can be used.

4.2. System Interface Layer

The UbiTerm establishes connections among the different elements participating in a specific activity, that is, the UbiTerm builds a network between programs and I/O devices as figure 5 shows, so that activity evolution is independent form UbiTerm, and vice versa.

As you can see in figure 5, when the UbiTerm needs to create or control activities, it uses the `/cmd` service if using Plan B, but might use another remote execution service

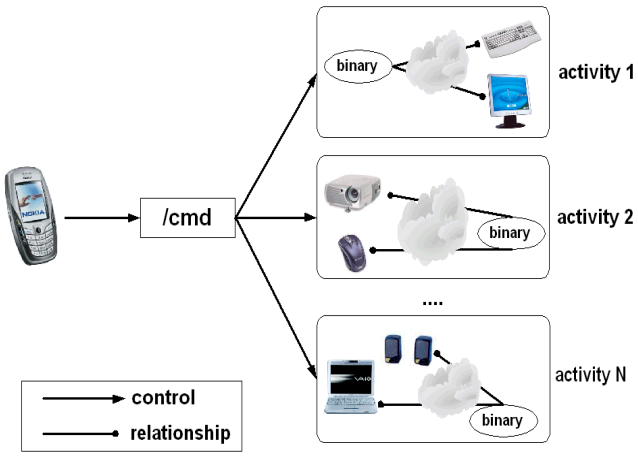

Figure 5. The UbiTerm builds a command network.

otherwise. When not using the rest of our system, the remote execution service must provide some means to control remotely which "files" (ie. devices) are used by a program. For example, when using Plan 9, we provide means to remotely mount a given set of files for a particular program at a specified place. The next time the program uses (opens) those files, it would be using the new device. On other systems like UNIX or Windows, this can be done by placing a proxy file system between the application and its files (which would achieve the same effect), or by interposing a dynamic library that intercepts calls for files whose names we know that refer to devices that we may want to redirect.

The System Interface provides the following services:

① *I/O device settings*: list of user's preferred I/O devices.

② *User tickets and ids*: users obtain tickets that latter will be required to let the UbiTerm authenticate to resources, and to authenticate other programs to access the user's resources, as said elsewhere [22].

③ *User location*: the system usually needs to know the user location, the UbiTerm keeps such information since it's a computing resource located by convention at the user. If the UbiTerm is attached to the internet, all the activities know how to reach the UbiTerm; If it is not, the UbiTerm has to update its caches through the rest of the system. (There is one per machine with commands running on behalf of the user).

As you can see, the UbiTerm centralizes critical user information, which does not have to be necessarily bad. In fact, having a single node centralizing control data and algorithms that would be hard to distribute, vastly simplifies our design and enables users to dispose of the information to interact with the system at any moment. We took the idea from the Google File System [8], that maintains centralized control for its distributed data.

5. Related Work

Mobile solutions consist in carrying user's data, and possibly computing, on a mobile device. Among these, the Personal Server [26] is closer to our approach in that it uses external-computing elements and does not need to be connected all the time. However, every time users move between locations they have to predict what data they will need to store it before leaving. This last one is not convenient when you are only planning to move to the next office. In addition, the use of external-computing elements is restricted to a host infrastructure using Windows XP, and to the set of I/O devices attached to that host. In general, these solutions impose uniformity, so users can not take full advantage of local available resources. In contrast, with the UbiTerm users do not need to store anything, and can deal with heterogeneous environments. Thus, the UbiTerm respects to some degree user's preferences.

Personal servers, running at phones and PDAs, suffer from the inherent difficulty of accessing data through small displays. Some systems try to overcome these limitations, see for example mLinks [21] and gestures for mobile devices [18]. The UbiTerm differs in that it is designed to control the set of external computing elements found in the network, but it is just a control device, not a server.

CAFE [13] permits aggregation of devices to perform tasks on behalf of the user. The main difference between the UbiTerm and such systems is that we do not require the devices and applications to be programmed with any particular middleware or component technology. Moreover, we do not assume that devices will be always connected through an homogeneous (wireless) network and CAFE seems to do so.

Systems like InfoPad [25], ParcTab [27], VNC [14], and Roma [24] are server-based architectures that place computing and storage resources on a network of servers. Again, these client-server approaches impose uniformity on the environment. Thus, they restring users to use the set of I/O devices attached to the host-platform capable of running the client-side. Moreover, for such systems it is critical to have good bandwidth and network connectivity between the clients and the servers. Our work is different in that the UbiTerm can still continue working if its network connection for the user is temporally unavailable or low-bandwidth; The user is still enabled to command the UbiTerm.

There are many middleware infrastructures to support active or smart spaces, like Centaurus [11], the Interactive Workspaces Project [2], Gaia's Active Spaces [19, 12], and Aura [20]. In these schemas standard applications are ported

or wrapped to conform a particular middleware, and installed in all spaces. We differ in that our system does not require the controlled applications to use any particular middleware. Moreover, the action of the UbiTerm is not limited to the smart space where the controlled applications are being deployed. Although the work done in [7, 23] have included abstractions representing the activity of the user in an ubiquitous environment, they do not provide users with the possibility of controlling those abstractions. Instead, they impose system decisions to user's indications when moving user's activities. Unlike them, our main goal is provide always control to users. Thus, with the UbiTerm user's activity mobility obeys user's directives instead of system decisions.

6. Conclusions and Future Work

We have described an architecture that solves two problems that arise when supporting user mobility. First, we provide a tool that permits users indicate control actions over their activities, while reducing user's distraction to the minimum. Thus, user's activity mobility follows user's indications instead of system decisions like in other approaches. Second, the combined use of a context information mechanism, and a remote execution service, allows to respect in some degree user's preferences when adapting on-going activities to changes in the physical environment. Moreover, we can exploit available resources in heterogeneous environments without imposing uniformity.

We are currently working on the second edition of our system, and have also modified a Plan 9[17] system to include most of the services that our system provides. This includes our remote execution service `/cmd`, and the context information mechanism. A prototype of UbiTerm is being built. In the near future we will be integrating more devices and services into our system, so that it could be used for our daily work to gain experience from the user's perspective.

References

[1] Plan B User's Manual Second Edition. http://lsub.org.

[2] Stanford, Interactive Workspaces Project. http://graphics.stanford.edu/projects/iwork/.

[3] F. Ballesteros, G. Guardiola, K. Leal, E. Soriano, P. de las Heras, E. M. Castro, and S. Arévalo. "Plan B: Boxes for network resources". *Journal of the Brazilian Computer Society, special issue on Adaptive Software Systems*, 10(1):31–42, July 2004.

[4] F. Ballesteros, G. Guardiola, E. Soriano, and K. Leal. "Traditional Systems can Work Well for Pervasive Applications. A Case Study: Plan 9 from Bell Labs Becomes Ubiquitous". In *PerCom, 3th IEEE International Conference on Pervasive Computing and Communications*, pages 295–299, Kauai Island, Hawaii, USA, 8-12 March 2005.

[5] F. J. Ballesteros, K. Leal, G. Guardiola, and E. Soriano. "The Design and Implementation of Plan B 3rd edition. A dynamic distributed computing environment". GSYC Technical Report 2004-05, Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos, 2004.

[6] J. Bresler, J. Al-Muhtadi, and R. Campbell. "Gaia mobility:extending active space boundaries to everyday devices". In *24th International Conference on Distributed Computing Systems Workshops, 2004*, volume 23, pages 430–433, March 2004.

[7] D. Carvalho, R. Campbell, G. Belford, and D. Mickunas. "Definition of a User Environment in a Ubiquitous System". In R. M. et al., editor, *CoopIS/DOA/ODBASE 2003*, volume LNCS 2888, pages 1151–1169. Springer-Verlag, 2003.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google File System". In *19th ACM Symposium on Operating Systems Principles*, October 2003.

[9] S. D. Gribble, M. Welsh, R. Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao. "The ninja architecture for robust Internetscale systems and services". *Computer Networks. Special issue on Pervasive Computing*, (35), 2000.

[10] G. Guardiola. "CUROCO: a distributed architecture for the dynamic generation, composition, and use of context in highly dynamic and heterogeneous environments". In *1st Middleware Doctoral Symposium*, pages 287–289, Toronto, Ontario, Canada, October 2004.

[11] L. Kagal, V. Korolev, S. Avancha, A. Joshi, T. Finin, and Y. Yesha. "Centaurus: A Infrastructure for Service Management in Ubiquitous Computing Environments". *Wireless Networks*, 8:619–635, 2002.

[12] F. Kon, C. Hess, M. Roman, R. Campbell, and M. Mickuna. "A Flexible, Interoperable Framework for Active Spaces". In *OOPSLA 2000 Workshop on Pervasive Computing*, Minneapolis, October 2000.

[13] R. Kumar, V. Poladian, V. Poladian, A. Messer, and A. Messer. "Selecting Devices for Aggregation". In *Fifth IEEE Workshop on Mobile Computing Systems & Applications*, 2003.

[14] S. F. Li, M. Spireti, J. Bates, and A. Hopper. "Capturing and Indexing Computer-based Activities With Virtual Network Computing". In *Proceedings of the 2000 ACM Symposium on Applied Computing*, volume 2, pages 601–603, Como, Italy, March 2000.

[15] C. Locke. "Styx-on-a-Brick". In http://www.vitanuova.com/mkt/press/Styx-on-a-brick.pdf.

[16] P. . U. Manual. *9*p. AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[17] R. Pike, D. Presotto, K. Thompson, and H. Trickey. "Plan 9 from Bell Labs". *EUUG Newsletter*, 10(3):2–11, Autumn 1990.

[18] A. Pirhonnen and S. Brewster. "Gestural and Audio Metaphors as a Means of Control for Mobile Devices". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, April 2002.

[19] M. Roman and R. Campbell. "Gaia: Enabling Active Spaces". In *9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.

[20] M. Satyanarayanan. “Pervasive Ccomputing: Vision and Challenges”. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[21] B. Schilit, J. Trevor, D. Hilbert, and T. Koh. “m-Links: An Infrastructure for Very Small Internet Devices”. In *Proceedings of the 7th Annual I*nternational Conference on Mobile Computing and Networking (MOBICOM), July 2001.

[22] E. Soriano. “SHAD: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems”. In *1st Middleware Doctoral Symposium*, pages 294–298, Toronto, Ontario, Canada, October 2004.

[23] J. P. Sousa and D. Garlan. “Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments”. In *3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29–43, August 2002.

[24] E. Swierk, E. Kiciman, N. Williams, T. Fukushima, H. Yoshida, and M. Baker. “The Roma Personal Metadata Service”. *Mobile Networks and Applications*, 7(5), September-October 2002.

[25] T. E. Truman, T. Pering, R. Doering, and R. W. Brodersen. “The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access”. *IEEE Transactions on Computers*, 47(10), October 1998.

[26] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. “The Personal Server: Changing the Way We Think About Ubiquitous Computing”. In *Proceedings of UBICOMP 2002*, June 2002.

[27] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, and M. Weiser. “An Overview of the ParcTab Ubiquitous Computing Experiment”. *IEEE Personal Communications*, 2(6):28–43, December 1995.

[28] M. Weiser. ”The Computer for the 21st Century”. *Scientific American*, 265(3):94–104, September 1991.