Plan B's Personal Command Module.
Commanding user activities in mobile and ubiquitous environments

Katia Leal Algara, Francisco J. Ballesteros Cámara,
Gorka Guardiola Múzquiz, and Enrique Soriano Salvador

Laboratorio de Sistemas
Universidad Rey Juan Carlos
C/Tulipán SN, Mósteles
Madrid, Spain.
{kleal,nemo,paurea,esoriano}@lsub.org

Abstract

Mobile systems introduce the problem of allowing the user to command the system while in the go, despite changes in the availability of computing resources and devices. Ubiquitous systems where the system is supposed to vanish in the background have a similar problem and it is not clear how the user should command such environments. Besides, on both systems there is a new problem: The need for the environment to know parameters about the users, like their location, their preferred I/O devices, etc. Resolving both issues is not trivial, although there are many previous results about how to address them. The biggest problem is that the system is distributed and has to support disconnected operation, which makes complex any solution for this kind of dynamic environments.

We think that by centralizing the control part of the problem at the user, and leaving the rest of the system distributed, both problems become easy. The *Plan B's Personal Command Module* (PCM) described in this paper follows that approach and is as a generic solution for them. Unlike in other approaches, the PCM is not an avatar for the user, a data cache in a user device, or a personal server. The PCM is just a centralized control device that permits both the user to operate on distributed resources, and the system to know what to do with those devices on behalf of the user.

1 Introduction

In a traditional system it is clear how to command the system (eg. by using a shell) and how to let the system know what it needs to know about the users (eg. are they logged in and reachable? What's the audio output device to be used?). We promised that mobile computing would let users move without (much) trouble, and promised as well that ubiquitous environments would support user activities. However, for both environments users still lack a convenient and generic way to let them command the system everywhere and at any time.

For example, consider a typical scenario depicted in figure 1. In (1), the user Gorka is located at Quique's office. Later, he will be moving (2) on his way to Katia's office (3). Consider now that Gorka wants to listen a CD while at Quique's. The devices available in the environment suffice to perform the task. There are speakers, a CD reader, a screen to deploy the player program interface, etc. Note that the systems and networks involved between the devices may be heterogeneous. Later, while moving, Gorka may have only a bluetooth headset available to continue listening music. And, of course, he might decide to stop the playing while he is moving. Finally, when at Katia's (3), Gorka might decide to resume the song using the speakers in the new location.

The problem addressed in this paper is how to let the user control the system in an easy way, and how to let the system know what to do with the devices to obey the user's command. A distributed

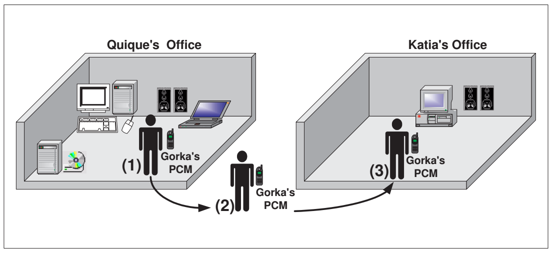

Fig. 1. The PCM let's the user command the system while enabling adaptation to available resources.

solution for the problem is complex, and becomes even more complex because the systems and the networks used are heterogeneous and must usually support disconnected operation. The approach we propose is to place control at the user, with a centralized implementation that governs the behavior of the distributed systems. Our approach is similar to what Google has done regarding file systems [9]. They use a centralized implementation to control their distributed system. The difference in our case is that we want the distributed system to continue its operation while the centralized control is not online (i.e. connected).

Before describing what our approach is, we have to say what our approach is not. It is not an avatar for the user deployed in the distributed system [13], because we want to retain control over the system and might want to issue commands or change preferences even while disconnected. It is not a personal server carried by the user [27], because the user might not be carrying a powerful computer. It is not a user's device carrying some kind of tag or data [2, 4, 6], because we actually want the user to be able to execute programs (to issue requests at least) at any time, even while disconnected.

The rest of the article is organized as follows. Section 2 introduces our approach using the scenario described above as an example. Section 3 describes the Plan B's Personal Command Module.1 In Section 4 we explain how the PCM solves the problems mentioned. Related work is discussed in Section 5. Section 6 presents some conclusions and plans for future work are shown in Section7.

2 One shell for our computing world

The *Plan B's Personal Command Module* (PCM), that we describe in this paper is a tiny program supposed to be always running at a device carried by the user (eg. a programmable mobile phone).

Considering first the problem of how to command the system, users issue requests and control activities through their PCMs, which in the example scenario is always available to the user.

In our scenario, while at (1), the PCM can be used to request a CD to be played. Even if the user is disconnected while moving (2) (i.e. no network available), the PCM is still reachable by the user to request the playing to stop. Of course, it will not stop before the PCM gets connected again, which might happen as soon as the user leaves an elevator, should that be the cause of the disconnection. But what is important is that the user can forget about stopping the previously issued command; The PCM will care.

An important thing to note is that using the PCM to control the system does not mean that its input devices are the ones used to issue the commands. While at (1), the user might use a nearby keyboard. However, while at (2) or (3), the PCM keypad and its voice commands (if any) are the only input means available and there may be no other choice.

Let's consider now the problem of how the distributed system knows what to do to satisfy the user's needs. The PCM interconnects devices provided by the distributed system and may start programs at different computers to use those devices. Once started, the programs continue to execute on their own and the PCM needs not to be connected. However, the PCM can instruct an already started program to stop, and can redirect the I/O devices used by an already started program. That is how we adapt what the system does as dictated by the user's PCM and the environment resources. For example, the PCM may learn that there are speakers at the same location of the user, and it has access rights on them. The PCM can be instructed to redirect the audio output of the previously started player program to such speakers; i.e. to make that playing to follow the user. The particular example is not important here, the point is that the control is exercised over the distributed system by a centralized control.

Also, information about the user (eg. the current location, the preferred audio output device, etc.) is kept centralized within the PCM, although such information may be cached within the distributed system to tolerate disconnections of the PCM. As a consequence, it becomes easy to operate on that information within the PCM, which is necessary to implement services like the playing program that makes audio follow the user. Another consequence, is that when some component or program within 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.

2.1 User activities

The PCM introduces a new abstraction to represent the user requests. This abstraction is used to start, stop, and control any request made by the user. We use the name *Activity* for such abstraction. It does not represent an activity in the real-world sense, but in the systems sense: Some command or set of commands started in response to a user request. This abstraction, like the PCM, is part of a new operating system named *Plan B* [7].

For us, an activity is every task a user wants to perform in the environment. Although the activity abstraction is kept within the PCM, the processing and I/O it might make is not in the PCM but in the surrounding environment. For example, in our scenario, the PCM would contain an activity for ”Playing that CD”, as soon as the user started to reproduce it in (1). However, the player would run at a computer found in the environment, and the I/O devices would come from the environment too.

Continuing the example, when the user was disconnected from the system (2) and requested the player to stop, the request was made to the PCM activity. Later, upon reconnection of the PCM, its activity could connect to the player and request it to stop. Finally, when the user at (3) had another set of speakers (assuming the activity was resumed or never stopped), the activity could connect to the computer running the player and instruct it to use the appropriate audio output device.

To facilitate management of activities, we include also 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 criterium 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 PCM state is lost, due to a device crash, it will learn of already started activities as they are seen in the network. Thus the PCM state is, to some extent, recoverable. Although a crash of the PCM can be very inconvenient for our users.

2.2 Remote access to programs and devices

To let the PCM 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* [3], 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 PCM both for our system and for traditional ones like Windows, UNIX, Plan 9, or Symbian.

To use the PCM with other systems, we need a remote execution interface for those systems if we want the PCM 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[18]. 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 [8] or approaches like Ninja[10]. For example, our X10 service for motion detectors and power switches, which is also exported through the web [3], 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 [3].

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 [16]. We are using BP [3] and 9P [17] 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].

3 Plan B's Personal Command Module

In the previous sections we discussed our approach and introduced the concepts of activity, and session, which are essential to understand the Personal Command Module. In this section we describe the PCM architecture.

The PCM has two different conceptual modules as seen in figure 2:

- ❒ **User Interface**: The front-end for the user of the PCM, to command the system.
- ❒ **System Interface**: The front-end for the system used by the user, to reach the user.

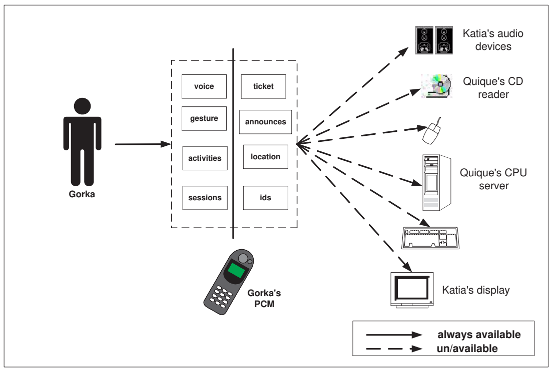

Fig. 2. Personal Command Module's architecture.

[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.

The user can control the system through the *User Interface*. In response, the PCM 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.

Multiple modules in the user interface allow for different input methods, like typed commands, voice, or gesture. This does not means that the processing (eg. voice recognition) must be made within the PCM, 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.

This architecture meets the requirements we placed on it, namely:

① *Distribution*: of resources and execution. The application can use resources from different machines, and the execution of its different processes can be distributed among the different available machines.

② *Dynamicity*: the set of resources to support a specific user activity is not fixed. The PCM can change at any time some of such resources while the activity using them is being executed.

③ *Disconnected operation*: the activity evolves independently from the PCM. As a result, it does not matter whether the PCM is or not connected.

The PCM 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 PCM interferes in such relationship only because of dynamicity or due to a user specified action on the activity. Decisions and requests made by the user while the PCM is disconnected are not lost, they are kept within the PCM 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 PCM. For example, in our current architecture, the PCM 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 PCM for the occasion.

3.1 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*: the PCM has the list of user's activities for the current session.

② *User's sessions*: the PCM provides the list of user's sessions.

③ *Controls for activities and sessions*: the PCM includes a set of controls for all the above. This includes one operation to delete an activity, one to stop the activity, one to resume the activity, one to redirect I/O devices for the activity, and another one to tag an activity for another session. There is no explicit operation to create an activity. Activities are created implicitly as the result of requests given by the user that lead to the execution of new programs. In the same way, the set of controls for the session abstraction includes one operation to create a session, one to delete it, and one to make it current.

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 PCM being navigated, and it may be feasible to perform the recognition even within the PCM device.

Also, the PCM graphical interface may be started at a nearby screen, and the commands may be given using an attached keyboard. The user only has to deploy the interface before using it, and perhaps destroy it before leaving.

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

3.2 System Interface Layer

The system obtains information about the user through the PCM *System Interface*. Part of the PCM state comes from the network and is dynamic. The PCM establishes connections among the different elements participating in a specific activity, that is, the PCM builds a network between programs and I/O devices as figure 3 shows, so that activity evolution is independent form PCM, and vice versa.

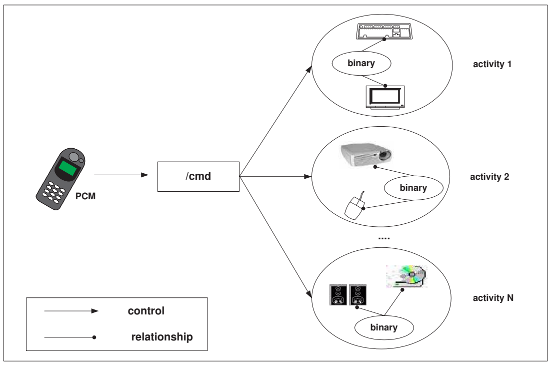

Fig. 3. The PCM shell builds command networks.

As we can see in the figure 3, when the PCM needs to create or control activities, it uses the /cmd service if using Plan B, but might use another remote execution service 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*: the PCM listens to the network to learn of announced devices. Our system announces resources to the network, and it is not complex to make other systems announce their resources too [21]. Besides, the PCM exports the list of preferred I/O devices to the rest of the system.

② *User tickets and ids*: users obtain tickets that latter will be required to let the PCM 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 PCM keeps such information since it's a computing resource located by convention at the user. If the PCM is attached to the internet, all the activities know how to reach the PCM; If it is not, the PCM 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 PCM 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. As we said before, we took the idea from the Google File System [9], that maintains centralized control for its distributed data.

3.3 Discovering, protection, and other issues

We have not discussed issues like resource discovering, since our approach introduces nothing new in that respect. In fact, we use a very naive discovery protocol, but any of the ones known to work, eg. [21, 14], can be used instead.

Regarding protection, naming, remote access, and synchronization, we rely heavily on distributed file system technology to address such issues, as we said before, and our approach is not different in that respect of the one used in Plan 9 to implement a distributed computing system.

4 A complex scenario?

It's early in the morning, so Gorka has the intention of running some of the activities that he usually does in the morning. The PCM has an audio input module, so Gorka only has to say "sessions" to obtain his list of sessions. Then he says "morning" to select the activities grouped under the morning session. Finally, Gorka asks for "coffee", and "newspaper" The PCM tries to start these two morning activities, the first one was previously customized to switch on the power for the coffee machine, and the second wone was customized to download the newspaper to electronic paper [1]. Before leaving home, Gorka starts the "washing machine" activity, to switch on the machine. While on the move, on a disconnected area, he decides to stop the washing machine and stops the activity. Once Gorka arrives at work, his PCM remembers to connect home and stop the washing machine (by switching off its power). The PCM could even provide enough identity, and location information to permit him to enter the private parking. Since Gorka has a class in 10 minutes he wants to review his slides so he starts a new activity for viewing slides selecting the desktop display, and keyboard as the I/O devices to use. Instead of carrying a laptop to the classroom and connecting it to the video projector, Gorka says he wants his slides activity to "follow" him as he moves to the new location. When at the classroom, the PCM controls the viewer activity to use the new graphics interface (the proyector). The PCM joystick (found in the phone input device) is used as the input mouse for the interface.

If we think about how we could implement this using other approaches, we realize that it can be a hard thing to do. The PCM, on the other hand has the advantage of centralizing the implementation of most of it, which makes it a lot simpler. For example, the application we use for viewing slides uses files as I/O devices. Using the PCM, when Gorka requests for a presentation activity at his office, the PCM runs the application and mounts its I/O devices to those selected by the user. Later, when Gorka moves to the classroom and requests this activity to follow him, the PCM can simply (remotelly) mount a different set of files for the given application.

5 Related Work

Systems like InfoPad [26], ParcTab [28], VNC [15], and Roma [25] are server-based architectures that place computing and storage resources on a network of servers. For such systems, it is critical to have good bandwidth and network connectivity between the clients and the servers. They also need the user to be well connected to the rest of the system. Our work is different in that the PCM can still continue working if its network connection for the user is temporally unavailable; The user is still enabled to command the PCM. Besides, some of the systems mentioned provide support just for a specific type of device (eg. VNC supports just graphical I/O). On the other hand, our approach is generic and works for different services and devices.

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 [24] and gestures for mobile devices [19]. The PCM 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.

Systems like Disk-on-key [6], iPod [2], and Pockey [4] are mobile devices that rely on nearby computing infrastructures. They provide access to personal storage when attached or connected to another systems. The main difference with respect to the PCM is that they are simply storage devices, however, we allow the user to command the system through the PCM.

The Personal Server [27] is closer to our approach in that it uses external-computing elements and does not need to be connected all the time. Our approach differs in that we consider the PCM to be a device on its own that can be used to issue commands to the rest of the system. Besides, the program run at the PCM requires significantly less hardware and processing power than what would be required for a personal server.

CAFE [**?**] permits aggregation of devices to perform tasks on behalf of the user. The main difference between the PCM 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.

There are many middleware infrastructures to support active or smart spaces, like Centaurus [11], the Interactive Workspaces Project [5], Gaia's Active Spaces [20, 12], and Aura [23]. Although the PCM has been designed with such kind of environment in mind, we differ in that we consider necessary to allow the user to move, and to operate on resources outside the active spaces. Besides, our system does not require the controlled applications to use any particular middleware.

6 Conclusions

The PCM centralizes control at the user. Such control is exercised over the set of distributed resources. It is the interface to the system for the user, and viceversa. The Activity and Session abstractions are implemented within the PCM, but they are only control for the actual applications that would run outside the PCM, in the resources provided by the network.

7 Current status and future work

We are currently working on the second edition of our system, and have also modified a Plan 9[18] system to include most of the services that our system provides. This includes our remote execution service /cmd, and our graphical interface service. A prototype of PCM is being built, and we plan to demonstrate it at the Workshop.

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. E Inks Electronic Paper. http://www.eink.com/news/index.html.
2. ipod: Apple's MP3 Player. http://www.applet.com/ipod/.
3. Plan B Users Manual Second Edition. http://lsub.org.
4. Pockey: Pocketable Disk Driver. http://pockey.co.kr.
5. Stanford, Interactive Workspaces Project. http://graphics.stanford.edu/projects/iwork/.
6. 'Q' Drive, Agate Technologies Inc., May 2001. http://eiwww.com/products_q.html.
7. F. J. Ballesteros, G. Guardiola, K. Leal, E. Soriano, P. Heras, E. M. Castro, and S. Arévalo. "Plan B: Boxes for networked resources". Submitteed for publication. Also in http://lsub.org, June 2004.
8. 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.
9. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System". In *19th ACM Symposium on Operating Systems Principles*, October 2003.
10. 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.
11. Lalana Kagal, Vladimir Korolev, Sasikanth Avancha, Anupam Joshi, Tim Finin, and Yelena Yesha. "Centaurus: A Infrastructure for Service Management in

Ubiquitous Computing Environments". *Wireless Networks*, 8:619–635, 2002.
12. F. Kon, C.K. Hess, M. Roman, R.H. Campbell, and M.D. Mickuna. "A Flexible, Interoperable Framework for Active Spaces". In *OOPSLA 2000 Workshop on Pervasive Computing*, Minneapolis, October 2000.
13. P. Koutsabasis, J.S. Darzentas, and T. Spyrou J. Darzentas. "Facilitating User System Interaction: the GAIA Interaction Agent". In *32nd Hawaii International Conference on System Sciences - 1999*, volume 5, January 1999.
14. Choonhwa Lee, David Nordstedt, and Sumi Helal. "Enabling Smart Spaces with OSGi". *IEEE Pervasive Computing*, July-September 2003.
15. Sheng Feng Li, Mark Spireti, John Bates, and Andy 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.
16. Chris Locke. "Styx-on-a-Brick". In http://www.vitanuova.com/mkt/press/Styx-on-a-brick.pdf.
17. Plan 9 Users Manual. *9*p. Murray Hill, NJ, 1995.
18. R. Pike, D. Presotto, K. Thompson, and H. Trickey. "Plan 9 from Bell Labs". *EUUG Newsletter*, 10(3):2–11, Autumn 1990.
19. 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.
20. M. Roman and R.H. Campbell. "Gaia: Enabling Active Spaces". In *9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.
21. Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy Campbell, and Klara Nahrstedt. "Gaia: A Middleware Platform for Active Spaces". *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):65–67, October 2002.
22. Enrique Soriano Salvador, Francisco J. Ballesteros, Katia Leal Algara, and Gorka Guardiola Múzquiz. "A Human Centered Security Protocol fo Ubiquitous Environments". Submitteed for publication. Also in http://lsub.org, June 2004.
23. M. Satyanarayanan. "Pervasive Ccomputing: Vision and Challenges". *IEEE Personal Communications*, 8(4):10–17, August 2001.
24. B. Schilit, J. Trevor, D. Hilbert, and T. Koh. "m-Links: An Infrastructure for Very Small Internet Devices". In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, July 2001.
25. 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.
26. Thomas E. Truman, Trevor Pering, Roger Doering, and Robert W. Brodersen. "The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access". *IEEE Transactions on Computers*, 47(10), October 1998.
27. Roy Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. "The Personal Server: Changing the Way We Think About Ubiquitous Computing". In *Proceedings of UBICOMP 2002*, June 2002.
28. Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. "An Overview of the ParcTab Ubiquitous Computing Experiment". *IEEE Personal Communications*, 2(6):28–43, December 1995.