Integrating PDAs into Distributed Systems: *2K* and PalmORB

Manuel Román *, Ashish Singhai **, Dulcineia Carvalho, Christopher Hess, and Roy H. Campbell

Department of Computer Science
University of Illinois, Urbana, IL 61801
E-mail: {mroman1,singhai,dcarvalh,ckhess,rhc}@cs.uiuc.edu
Web: *http://choices.cs.uiuc.edu/*

Abstract. In this paper we describe an application model for seamless mobile data access using handheld devices and wireless links. We go beyond the current model in which handhelds are used as smart organizers augmented with stripped down versions of popular desktop programs. Instead, we propose to integrate handheld devices seamlessly in a distributed computing environment. Componentized applications, adaptable middleware frameworks, and standardized protocols play a significant role in this new paradigm. We also describe an implementation within this paradigm using PalmORB, a CORBA client for the 3Com Palm devices.

1 Introduction

The current distributed system organization is machine centric. Users have different accounts, passwords, and software on different machines. This organization has sufficed for the computers to users ratio of one or less. However, with a larger computers to users ratio, afforded by smaller devices and cheap wireless connectivity, the existing model breaks down. The challenge is to provide the users with a seamless image of a distributed system irrespective of the device used to access the system. Previous attempts to provide a consistent system image using clusters (e.g., NOW [1]) and distributed file systems (e.g., NFS [6]) are applicable only for LAN based distributed systems.

The *2K* distributed system, currently under development at the University of Illinois proposes a user centric organization of a distributed system. Users view the system as a collection of resources that they can use regardless or their location.

In this paper, we consider the issues related to incorporating handheld devices in a distributed system. We describe the challenges in doing so and our solutions to them.

1.1 The Issues We Consider

Small mobile devices are restricted in terms of bandwidth, CPU and the amount of memory and storage availability. Thus they cannot be treated as workstations

* Supported by a Fulbright-Ramon Areces Fellowship
** Supported by an IBM Graduate Fellowship

or PCs when integrating them in a distributed environment. Each scenario must be analyzed independently in order to customize the system and achieve the best possible performance.

All the issues we will discuss in this paper can be applied to any small mobile device (Palm Computer, HandHeld, wearable...). However, for the sake of concreteness, we will focus on the Palm Pilot only. To refer to all of them as a set we will use the term PDA (Personal Digital Assistant).

Current PDAs cannot be considered as mere schedulers or devices to store telephone numbers anymore. They offer enough functionality to be used in sophisticated environments. This paper describes an infrastructure that integrates PDAs in distributed environments based on the *2K* operating system. Thus, instead of using PDAs as isolated entities, users will be able to use them as an enabling bridge to a collection of distributed resources and services.

One of the more interesting integration issues is how to offer a consistent view of a distributed system from a PDA. If we only take into consideration the screen size, it seems clear that it is difficult to offer the same interaction interface in a PDA than in a 21 inch monitor. However *2K* OS offers an object oriented vision of every single resource and service. This allows us to consider the interaction interface offered to the user as an object container. Therefore, in a big powerful monitor we represent every resource as a graphical icon whereas in a Palm Pilot we could simply use a list of names. In this way, the concept of interacting with components (objects) is preserved, no matter which device is used. However, the way those components are presented to the user is likely to change according to the device being used (the approach extends for example to a voiced controlled PDA that includes no screen). The interaction interface offered to the user is therefore polymorphic as it offers the same functionality through specialization of the interface according to the device.

One of the design features of *2K* is that the operating system resides in a computer network and not in a specific machine. Therefore, it is possible to access the operating system from different devices and locations while keeping a consistent view of the system. This is achieved by using object oriented and component concepts. For example, a user can log in to a login component of the system which may reside in a disconnected PDA. When the PDA is connected to the network, the operating system services are instantiated in the device being used. The state associated with the user attributes is kept in the distributed environment. Sessions may be transfered from one device to another. For example, it will be possible to start a videoconference in a desktop and transfer the videoconference to a PDA. The transition from the desktop to the PDA must be smooth and must not require any user intervention. The system will automatically adapt and synchronize itself to the new environment. If the PDA is disconnected, the session should be reestablished automatically on reconnection.

There is a wide range of PDAs, and each one of them can greatly vary in terms of resource availability. This requires an infrastructure able to deal with every different kind of device. Therefore we propose the use of adaptable proxies, which will alleviate the PDA from the execution of computation intensive software. The decision about what should be done at the PDA and what at the proxy should

be determined according to the hardware capabilities of the PDA. Moreover, it should be possible to modify that decision dynamically, by monitoring techniques.

1.2 Organization of the Paper

Section 2 introduces the *2K* distributed operating system which is the enabling infrastructure. Section 2.1 provides a more detailed description of *2K* environments which are fundamental for the integration of PDAs into the *2K* distributed system. Next section, section 3, describes the challenges associated to integrating HandHeld devices into a distributed system. Section 4 explains the design and implementation of PalmORB which is the core component that provides PDAs with CORBA capabilities. Section 5 presents two applications: PalmShell, a *2K* shell for PalmPilots (section 5.1) and Video Streaming Proxies (section 5.2) which brings video to the PalmPilots by using the dynamic architecture of *2K*. Section 6 offers some information about the performance of PalmORB and PalmShell. Section 7 describes two related approaches, and finally, section 8 concludes the paper and describes our future work.

2 Brief Overview of the *2K* System

2K is a distributed operating system currently under development at the University of Illinois. It offers a reflective architecture that can be modified on the fly. The operating system is built on top of CORBA and the reflective behavior is offered at the ORB level. The ORB that we are using is a modified version of TAO [12] called dynamicTAO [11], which is built on top of the ACE toolkit [4]. The ACE toolkit runs on top of several platforms, therefore, *2K* can run on every platform where ACE has been ported. The main platforms we use for development are Windows NT and Solaris, and we are also working on the design of a customized microkernel called Off++ [2]. *2K* is based on the principle of *What You Need Is What You Get*, which means that the system can dynamically adapt itself to the requirements specified by the user, thus providing a customized execution framework for every application. The main issues associated with *2K* are: architectural awareness, adaptability and network and user-centrism.

Users become active entities within the system and specify what resources they want and how they want to use those resources (QoS requirements). All this information is stored persistently in an object called an environment, which is associated with the user's instantiation within the system.

Network centrism plays a key role in *2K*. The network becomes a single pool of resources managed by *2K* according to the QoS specified by users. Dependencies between resources (local and distributed) are reified by the Component Configurator [8], a *2K* fundamental core component.

A *2K* user is unique within the whole system, it can access to *2K* from different platforms (Windows NT, Solaris, PalmOS) and during a session, resources from different machines can be used. A dynamic security policy mechanism based on the Cherubim project [3] can be attached at runtime to specified dynamicTAO ORBs.

2.1 *2K* Environments

A *2K* environment is a container of components, devices and configuration parameters and provides an execution context for users within the *2K* distributed system. The environment is responsible for managing groups of components as a collective entity, and for facilitating the interface of the components with the *2K* operating system.

Environments are persistent objects that are created and managed by the *2K* Environment Service. When a new environment is created on behalf of a principal, the Environment Service locates a profile of that principal describing default components and devices, as well as resource requirements for that environment. Typically, principals have default profiles that can be customized depending on the location of the principals, their role and the components they execute.

When an environment is created the system must assure that the resource requirements of the profile are satisfied. Moreover, as components execute, their potentially variable QoS requirements must be understood and satisfied by the system. This is accomplished by the interaction with the Resource Manager [13] and the QoS Monitor [14], two of the *2K* core components.

As activity in the environment proceeds, the own environment contacts all the components and devices it requires, either by reusing existing components or by creating new ones.

3 Challenges in Integrating a Hand Held Device in a Distributed System

One of the major challenges is to discover all the new possibilities offered by the integration of PDAs in distributed environments.

The main use of a PDAs nowadays is personal information management. However, with the incoming era of wireless communications, it will be possible to introduce the concept of "Everything, everywhere, anytime computing". Users will be able to access every single resource from every possible device and at any moment.

There are many issues that must be studied carefully in order to provide a minimum computing environment. Connecting to some kind of network is crucial in order to guarantee resource availability, but currently, in most of the cases, the bandwidth offered for PDAs is low. Therefore, it is a priority to offer some kind of optimized transport protocol or use proxies that minimize the communications required with the PDA. By using the adaptation capabilities of *2K*, we can address some of the limitations of the PDAs by locating computation intensive tasks of the user's in PDA accessible self-adaptable proxies.

Users are registered within the *2K* environment, not within a particular device or domain. This translates into a single login mechanism. As it will be explained in next sections, when the user logs in, the login mechanism will contact the Environment Service. This will receive the request and will grant or deny the user to start the session.

The PDA caches the state of the user's environment. This caching allows disconnected operation mode and in most cases reduces the amount of data that the distributed system sends to the PDA each time a log in session is started.

An important issue when offering a unique system view is to find some kind of application user interface that can be used from different architectures and OSs. The ideal would be to provide a single interface instead of one for each platform. We are studying the feasibility of using XML to describe the application interface, and different data style sheets, one for each platform.

Mobile computing combined with distributed environments offer a large number of benefits. However it also introduces some interesting challenges. When working in a distributed system, security becomes an issue of particular importance. It affects several aspects: login authentication, information encryption, privacy, key sharing and proxies. Confidentiality must be guaranteed by protecting users' data from other users. Since it is likely that PDAs will use proxies in some situations, a security delegation mechanism will be required. The initial authentication mechanism sends information about the user to the Environment Service. This information must be protected to avoid a third party intercepting the information and using it later to impersonate the original user. Finally, there is another level of security that should be considered and that is related to the device itself. Because of the size of PDAs they can be easily stolen or lost. In both cases, there must exist a mechanism to guarantee that nobody but the legitimate user can initiate a session or access confidential data stored in the device.

Using the *2K* operating system and the Cherubim project [3] developed by the Software Research Group of our CS department, we can deal with user validation. By attaching the Cherubim Security strategy to a running dynamicTAO ORB, we can allow or deny the execution of methods on objects, according to the identity of the principal issuing the request. Besides user validation, the Software Research Group is also working on other security aspects that we plan to include in *2K* and PDAs applications.

4 Design and Implementation of the PalmORB

Our *2K* Distributed Operating System is based on CORBA. Therefore any device capable of generating IIOP requests can take benefit of the services that *2K* offers. This was the reason that lead us to the development of PalmORB, a small client side ORB for Palm Pilots that offers a subset of the CORBA functionality.

Most of the existing ORBs (commercial and freely available) are not suitable for embedded systems. In most cases, their monolithic design restricts their applicability to PDAs. Their size is too big to be ported to a memory limited device item, offer features that will never be used in a PDA and in some the cases, the client and server side functionality are combined in a single library.

In the case of a PDA, the execution environment greatly varies from the one of a PC or a Workstation. PDAs are not likely to use all the features offered by a standard ORB. Therefore, offering the whole CORBA functionality by default implies, at least at first glance, a waste of memory for properties that will rarely or never be used.

PalmORB was designed having in mind the restrictions of the platform on which it was going to be used. It has been implemented by using the freely available source code from SUN, though the code has been modified to offer only what is required. The current implementation is CORBA 2.0 compliant (at least the functionality provided) and follows the GIOP 1.0 protocol (the IIOP implementation).

All the server side functionality has been removed, and at the client side, only the Dynamic Invocation Interface and the ORB interface is provided. We decided to remove the server side functionality because we consider that the Palm Pilot will be used mostly as a client that uses CORBA objects' services. We do not discard, however, adding minimal server side functionality in the future.

Our current version of PalmORB cannot be customized on the fly. However, by taking benefit of *2K*, we can customize the remote server to optimize the interaction between the PalmPilot and the server.

4.1 Implementation Overview

PalmORB is implemented in C++ by using CodeWarrior Release 5 for Palm Pilot. It has 6000 lines of code and uses around 50Kb of memory. If we consider a PalmPilot III which has 2Mb of RAM, the percentage of memory required to enable CORBA capabilities is 2.5%.

PalmORB runs on any PalmOS compatible architecture with the network library. Current implementation is a static library though in the future we will change it into a dynamic library.

We have been testing PalmORB by sending requests from a PalmOS application to a CORBA object running on a Windows NT machine and another one running on a Unix machine. For the tests we have been using the PalmOS emulator (version 21d25) as well as a Palm III device connected to Internet through a PPP connection by using a serial cable (figure 1).

Our final objective is establish a wireless connection between a Palm Pilot and a PPP server.

5 Sample Applications

This section describes two applications. The first is based on the PalmORB and in the services provided by *2K*. The second one does not use PalmORB, but takes benefit of the adaptive and architecture awareness capabilities inherent to *2K*.

5.1 PalmShell

The PalmORB library introduced in the previous section, allows palm devices to interact with CORBA objects. On the other hand, *2K* offers a set of adaptable, customizable CORBA based services and resources. However, there is a gap that must be filled to connect both sides.

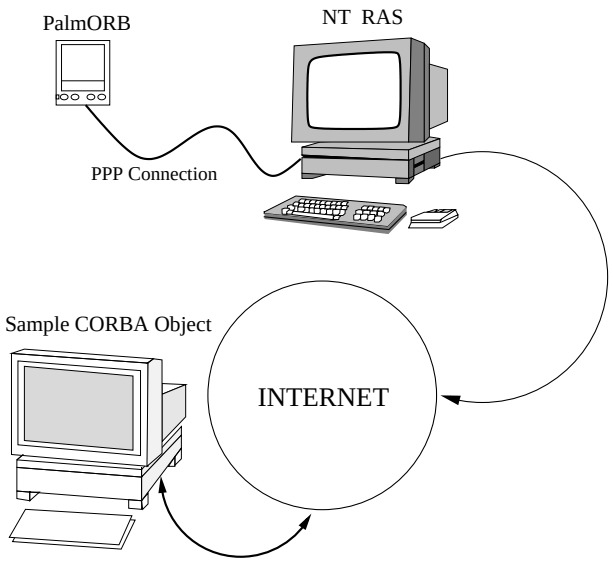

Fig. 1. PPP Connection

We introduce here the PalmShell, an application that allows palm devices to be integrated in *2K*. PalmShell creates a *2K* environment customized for the PalmPilot, based on the user's profile that is stored in the distributed system.

Figure 2 illustrates all the components involved in the application as well as all the steps required to establish a *2K* session.

Whenever a device starts a session, it must contact the environment service to request the activation of the user's environment (1). When the request gets to the Environment Service, the security strategy checks whether the principal that is issuing the request is authorized to invoke that method on that object (2). If the request can proceed (the user has been authenticated), the environment service contacts the profile server (3) and retrieves the profile associated to the user (4). Based on the information contained in the profile, the environment service activates an instance of an environment in a *2K* host (5) (PalmPilot will interact with it remotely) and returns its reference to the PalmShell (6). If the user logged from a PC, for example, the environment could have been instantiated in the PC. The Palmshell will retrieve the list of components contained in the environment and show it to the user (7). The user can add new components to the environment by contacting the implementation repository (8), which will return a list of different categories available (9). When the user selects a category, PalmShell sends a request to the implementation repository, asking for all the components that belong to that category.

The PalmShell displays a list with all the components stored in the environment. The user can then interact with those components by means of CORBA requests. Since the user can dynamically add and remove components from the

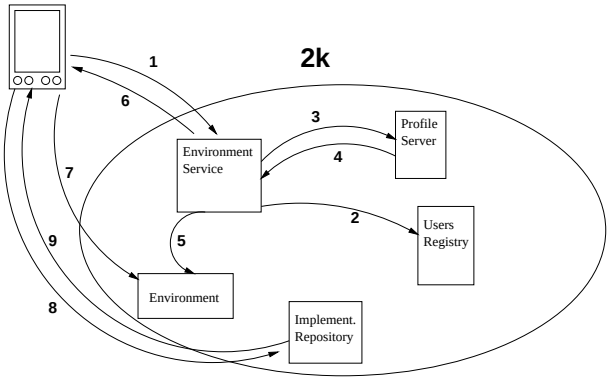

Fig. 2. PalmPilot Login

environment, there is an important issue to be considered: how can the user interact with those components? The problem is that we do not know the programmatic interfaces of those objects, neither what their user interface should look like. The first issue, the one related to the programmatic interfaces of the objects, can be solved by using the interface repository. It contains information about the methods offered by interfaces as well as the parameters required. For the second issue, we are considering two different approaches. On one hand we are working on downloading native applications to the Palm Pilot (which can be obtained from the implementation repository). This solution makes it possible for the user to create customized graphical interfaces. However, these graphical interfaces will be specific for PalmOS devices. The second approach will use XML pages to define the graphical interfaces associated to the components. These XML pages will contain information about the graphical interface, information about different methods associated to the object, and information about how to issue dynamic method calls.

The advantage of this last option (XML pages) is that the component creator will have to define the XML page once, and different filters (data style sheets) will be responsible for making the appropriate changes for every different device.

Figure 3 shows the screenshots of the current PalmShell. The leftmost picture is the login screen. Once the username and password have been introduced, the Environment Service is contacted and if the user is validated, it returns the Environment IOR see the middle form on figure 3. The rightmost form is the Object Browser that allows choosing new components to be added to the environment from a *2K* Implementation Repository.

5.2 Video Streaming Proxies

As mentioned earlier, proxies are used in the *2K* environment to alleviate some of the computational burden from palm devices. Since the processing power of these devices is quite limited, certain computations may be performed at the proxy and

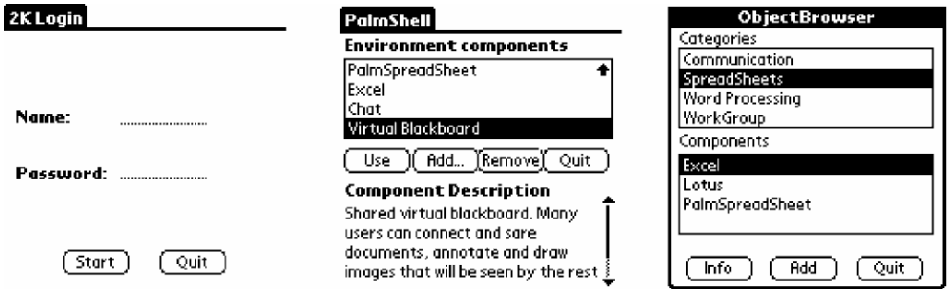

Fig. 3. PalmShell

the results sent to the PDA. We have used this approach in streaming video to the Palm Pilot. The decompression of video frames requires significant amounts of CPU cycles and memory and is better off-loaded to a proxy. In addition, frame size and rate should be reduced to accommodate the small screen size and limited bandwidth to the device. The reduced color depth of the Palm Pilot also provides room to reduce the number of bits sent. Palm Pilots currently only support 4 colors. Therefore, there is no need to send frames with full color; the extra unusable color information should be removed.

We have implemented a video proxy that filters video streams on the fly, allowing the Palm Pilot to view the video sequence. These streams may be stored on disk or may be a live feed. Our proxy currently filters MPEG-1 video and system streams and transforms them to compressed bitmaps.

The proxy first decodes the MPEG stream using a modified version of the MPEG Software Simulation Group's *mpeg2play* [10]. A small decoding engine was created with an object-oriented interface. The interface allows different data sources to be easily plugged in, allowing the decoder to read from the network, files, buffers, etc. For example, to send MPEG files stored on disk, a simple file input stream can be plugged in. The proxy decodes frames into an array of 8 bit pixels and reduces it to 2 bits using a simple lookup table. This produces a grey-scale version of the frame. Next the frame is reduced in size if necessary. We currently reduce images by factors of 2 in both dimensions. For example, a 160x120 image can be reduced to 80x60. This is accomplished by averaging blocks of pixel values. Although frames of 80x60 are small, it is still possible to clearly view the image. The Palm Pilot screen is actually able to view 160x120 frames. However, if a greater frame rate is desired, scaling of the images is necessary. During testing, we were able to view 3 frames per second at 80x60 quite well. The "ghosting" effect that is inherent to the Palm Pilot LED screen was not noticeable.

After the image has been scaled down, it is runlength encoded and sent. The Palm Pilot is responsible for receiving the compressed bitmap and decoding it. The decoding algorithm is simple and can be done quickly. The amount of compression depends on the characteristics of the particular frame, but was found to be good for most of the frames tested.

The proxy can filter the stream further if necessary. For example, it could drop frames if it finds that too much data is being sent. Since the frames are all bitmaps, any frame could be dropped without disrupting the other frames. This is in contrast to MPEG frames which depend on each other. The side effect of removing inter-frame dependencies fits well into our model of streaming video to the Palm Pilot.

Different policies in the proxy may be employed when streaming video to a disparate set of devices. For example, streaming to a desktop workstation on a LAN may require no filtering at all, while if the workstation is connected via a dial-up line, some degree of filtering may be required to compensate for the limited bandwidth. This allows different devices to view the same video stream, with the quality adjusted appropriately for the particular device.

Fig. 4. PalmMPEG

6 Performance of Our System

In this section we will discuss the performance of PalmORB and PalmShell. For information related to the *2K* distributed operating system, check [9].

PalmShell is implemented in C++, has around 1300 lines of code and uses 10Kb of RAM. If we consider a PalmPilot III with 2Mb of RAM, the percentage required for the PalmShell is 0.5%. If we add also the PalmORB which will allow the integration of the PalmPilot into the *2K* distributed system, the total percentage of memory required is 3%.

To initiate the session in a PDA, the PalmShell contacts a Name Service and obtains the references for the Environment Service and the Implementation Repository. Then it sends a request to the Environment Service and obtains the reference of an Environment that will be associated to the user. PalmShell sends a request to

the Environment and retrieves the list of components stored in the environment. For this particular test, we will assume that four components are returned. Each component returned consists on a name and an IOR. The amount of information received during the log in is around 2Kb, without counting the CDR extra stuff.

The average time required to establish a log in session in a Palm Pilot is 1.95 seconds (average time after loging in ten times). From the users point of view, this is the time it has to wait since it enters the login and password and clicks on start, until the shell with the list of components is displayed on the screen.

To calculate the overhead we used a Palm III device and established a PPP connection with a Linux machine using a serial cable at 38400 bps. We started two CORBA objects, the Implementation Repository running on a NT box and the Environment Service on a Solaris machine running Solaris 2.7. The Environment object was instantiated by the Environment Service in the same Solaris machine.

7 Related Work

There are some projects that focus on some of the issues discussed in this paper. The ones that will be discussed in this section are: Adaptive Middleware Proxy and the TACC programming model [5], and Rover [7].

The first project, AMWP (Adaptive Middleware Proxy), provides an infrastructure for developing applications for "thin" clients, that is, clients with limited hardware resources. The infrastructure provides a set of resources for developing proxies, which alleviate those thin clients from computation expensive tasks. The main idea behind this approach is to offer some building blocks and programming interfaces for developing TACC modules (Transformation Aggregation Caching and Customization). These modules preprocess data before sending it to the client. Therefore, the amount of work that the client has to do on the data is minimized. Examples of Transformations are filtering, re-rendering and encryption. Aggregation refers to collecting and collating data from several sources. Caching is used to store post-transformed data, so it does not have to be transformed everytime. Finally customization is related to the configuration of shared services, so that every client gets exactly what it needs. Different TACC modules execute independently and the output of one of them can be redirected to the input of another one. They can also be easily changed or updated. However, specific aspects of a TACC module cannot be dynamically customized; instead they whole module must be replaced by a new one that provide the required functionality.

The Rover Toolkit is an object oriented platform that offers a set of tools to develop mobile applications. It is mainly based on two fundamental ideas: relocatable dynamic objects (RDOs) and a queued remote procedure call (QRPC).

Relocatable dynamic objects can be loaded dynamically into client devices from server computers. Therefore, the execution environment characteristics (bandwidth, properties of the client device, etc.) will dictate which RDOs must be sent to the client and which ones must remain in the server. This decision can be modified later on according to how the execution evolves.

Disconnected operation mode is supported by means of the queued remote procedure call. When the system detects a network disconnection, it stores incoming

RPC requests in a log file; as soon as the connection is reestablished, the log file is replayed

The infrastructure offered by *2K* is similar to the Rover approach at the dynamic adaptation level. However the idea introduced in *2K* of extending the whole OS to mobile devices is new. Neither AMWP nor Rover contemplate that possibility.

8 Conclusions and future work

The infrastructure introduced in this paper extends the OS to every kind of device, providing a unique and seamless image of the system. The user can interact with the system from different locations, scenarios and by using different devices. The *2K* distributed operating system creates a customized execution context for each user. This context is defined in *2K* as an Environment, and it is persistent and independent of the device being used (workstation, PDA, PC).

Using CORBA as the standard *2K* underlying object bus provides a uniform way for interacting with every single service and resource offered by the system. This feature is exploited by PalmORB to offer sophisticated services to the Palm Pilots.

Adaptable *2K* proxies are useful tools for alleviating the hardware limitations of PDAs. They can provide users with functionality similar to the one offered in Workstations or PCs.

The *2K* distributed operating system introduces the concept of "Everything, everywhere, anytime computing", which introduces new possibilities in the field of information accessibility.

As part of our future work we plan to introduce several changes as well as new services and features. We will port our PalmORB to other PDA platforms though we will introduce changes in its design. We are interested in breaking the PalmORB into different components. These components will be added, updated and removed dynamically according to what is required by the execution environment. Our experience with a reflective ORB (dynamicTAO) has proved to be a useful mechanism to cope with dynamically changing environments. We will also redesign the Palm Pilot shell to add the new functionality. By adopting this approach there will not be any difference, from the user's point of view, when accessing local Palm Pilot applications and remote *2K* components.

References

1. Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW Team. A case for networks of workstations: Now. In *IEEE Micro*, February 1995.
2. Francisco J. Ballesteros, Fabio Kon, and Roy H. Campbell. A Detailed Description of Off++, a Distributed Adaptable Microkernel. Technical Report UIUCDCS-R-97-2035, University of Illinois at Urbana-Champaign, August 1997. Also available at `http://choices.cs.uiuc.edu/2k/off++`.

3. Roy Campbell and Tin Qian. Dynamic agent-based security architecture for mobile computers. In *Second International Conference on Parallel and Distributed Computing and Networks (PDCN'98)*. IASTED, December 1998.
4. Schmidt Douglas C. The ADAPTIVE Communication Environment. In *Proceedings of the Sun User Group Conference*, San Jose, California, December 1993.
5. Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network sevices. In *Proc. 1997 Symposium on Operating Systems Principles (SOSP-16)*. ACM, October 1997.
6. Network Working Group. Nfs: Network file system protocol specification. In *rfc 1094*. March 1989.
7. Anthony D. Joseph and M. Frans Kaashoek. Building reliable mobile-aware applications using the rover toolkit. In *Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking*, November 1996.
8. Fabio Kon and Roy H. Campbell. On the Role of Inter-Component Dependence in Supporting Automatic Reconfiguration. Technical Report UIUCDCS-R-98-2080, Department of Computer Science, University of Illinois at Urbana-Champaign, December 1998.
9. Fabio Kon, Ashish Singhai, Roy H. Campbell, Dulcineia Carvalho, Robert Moore, and Francisco J. Ballesteros. 2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments. In *Proceedings of the ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*, Brussels, Belgium, July 1998.
10. mpeg2play. `http://www.mpeg.org/MPEG/MSSG`.
11. Manuel Román, Fabio Kon, and Roy H. Campbell. Design and Implementation of Runtime Reflection in Communication Middleware: the *dynamicTAO* Case. In *Proceedings of the ICDCS'99 Workshop on Middleware*, Austin, TX, June 1999.
12. Douglas Schmidt. Tao overview. `http://siesta.cs.wustl.edu/~schmidt/TAO-intro.html`.
13. TwoK Research Team. Resource management in 2k (document in preparation). `http://choices.cs.uiuc.edu/2k`, 1998.
14. Duangdao Wichadakul. Qos model for the 2k project (document in preparation). `http://choices.cs.uiuc.edu/2k`, 1998.