

# An Application Model for Interactive Environments

Guruduth Banavar, James Beck, Eugene Gluzberg,  
Jonathan Munson, Jeremy Sussman, Deborra Zukowski

IBM TJ Watson Research Center  
30 SawMill River Road  
Hawthorne, NY 10532 USA  
+1 914 784 6385

{banavar, jabeck, gluzberg, jpmunson, jsussman, deborra}@us.ibm.com

## ABSTRACT

We are now standing on the brink of a new computing frontier. Advances in computing technology that enable processors, sensors, etc. to be placed anywhere and everywhere are in turn elevating our physical surroundings; Our physical surroundings are becoming annotated, interactive environments. Physical objects will provide services that connect us within an environment, across environments, and into the current world of information. Information and services will be available to anyone who enters an environment, adapting to personal devices as needed. However, such an interactive environment is worthless until we understand how to build and deploy useful applications. In this paper, we introduce a project for a Platform-Independent Model for Applications we call PIMA. PIMA includes a model for writing applications in such an interactive environment and a run-time system.

## INTRODUCTION

It has been said that ours is an information economy. While the statement is compelling, it is not as broadly realized as it might be. An information economy requires that information and resources be interchangeable. Thus far, the information economy has been limited to the virtual world, i.e., that contained in computer databases and web servers.

Pervasive computing can extend information management practices to almost all of our economy. With pervasive computing, we can embed devices directly into any product or artifact of import, allowing them to produce information that can then be managed. The key for delivering on the promise of pervasive computing is the development of a programming paradigm that enables applications to better exploit physically-oriented – or in situ – information, along with virtual information.

Our research is focused on providing a Platform-Independent Model for Applications that we call PIMA. We propose a high-level architecture for enabling information-enriched environments to be morphed into interactive environments, based on distributed services. The model extends the current service-based models because it emphasizes user “portals”. That is, a user can use any device as a portal into all of the services embedded in the

environment (provided that the user has the proper access). To fully support this model, applications must be capable of being run on any device that enters the environment. Therefore, applications must be able to be customized to the available resources of the portal device(s). PIMA provides the model, language and run-time support to build and execute such applications.

## RELATED WORK

The original, and founding, project in pervasive (or ubiquitous) computing was the ParcTab effort at Xerox PARC [8]. In the ParcTab project, pervasive applications could travel with a mobile user. The project concerned itself with measuring the value of pervasive applications. Since that time, several other projects concerning pervasive computing have begun, including Portolano[2], Future Computing Environments[5], Oxygen[3], Iceberg[4], etc. All of these projects are picking up where ParcTab left off. The initial efforts are focused at enabling the environment and directly accessing services within that environment. None of these projects address application development in the pervasive computing area.

Other research related to our work is the development of systems for application development in a device independent manner, often called User Interface Management Systems (UIMS). Examples include the work reported in [9] and the UIML System at Virginia Tech [1]. These systems have historically targeted desktop environments and have emphasized consistency of style across devices. Our goal is to enable pervasive devices to perform user tasks in an interactive environment. We are not as interested in maintaining consistent style as in automatically adapting the user interface to best leverage the capabilities of the device.

Our work leverages service frameworks such as CORBA[6] and JINI[7]. These frameworks support pervasive services by allowing transparent service distribution and service discovery. These frameworks serve as a basis for our view of a pervasive environment.

## AN EXAMPLE INTERACTIVE ENVIRONMENT

Museums traffic in information. As such, they are prime candidates to take part in the information economy.

However, museums have failed to capitalize fully on the value of the information they possess. A museum that serves as an interactive environment could better leverage its information base to increase the number and overall satisfaction of its visitors.

The museum becomes an interactive environment by doing three things. First, every artifact in the museum is augmented with an embedded processor that delivers content for the piece. Second, each exhibit is enhanced by a background service that provides access to ancillary information about the artist/inventors displayed. Finally, the museum provides a service that enables visitors to the museum to remotely share their visits with others. Part of the underlying infrastructure is proximity awareness, which couples a visitor's location in the museum with the information they receive.

When visitors arrive at the museum, they are offered rental of museum access devices. Alternatively, the museum could register visitors' personal devices within the environment. For a small additional fee, they are offered the option of projecting their visit to an external device, perhaps used by a spouse or a child's classroom. Note that while the museum may control which device is used internally, multiple device support is always needed for remote access.

Visitors are free to wander throughout the museum. When they look at a displayed work, information specific to that work is shown on their device and, if desired, projected to the remote viewer as well. Visitors can interact with the environment to access background information on an exhibit, based on where they are. Visitors can also communicate with remote visitors (e.g., with a chat room). Thus, remote visitors can ask to see more historical details or offer suggestions to the visitor what next to see. The user interface, shown on the visitors' devices, is tailored to the application preferences of the visitor.

In this simple scenario, we see how information can be used to enhance a visitor's experience. In addition, the sharing of the experience is, in a sense, an advertisement that can help expand the visitor base of the museum. If the remote visitors are interested in some of the exhibits, they may be more likely to visit the museum in the future.

### AN APPLICATION MODEL

Our PIMA work is based on the assumption that interactive environments are supported by a services-based distributed architecture. Further, we assume that users will interact with services using whichever devices are proximate to them. As mentioned previously, our work emphasizes enabling user tasks as opposed to creating full-fledged (esthetically optimized) application user interfaces.

The high-level PIMA architecture is shown in Figure 1. In the figure, an interactive environment provides access to in situ information and background services, some of which provide access to information and services from other environments. Client devices, or "portals", run an

application front end that the user needs for his/her interaction. A front end includes a device-specific actualization of the user interface and logic support for management of the user interaction. A user's device may host a composite front end comprised of multiple application front ends. In addition, some of the services may be migrated to a device to allow for disconnected operation, provided that the device has ample resources to host the service.

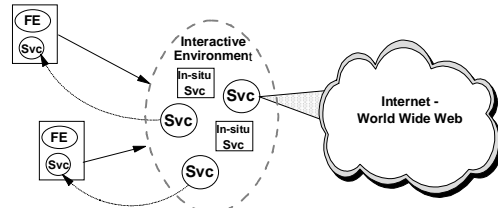


Figure 1

This high-level architecture sets the stage for PIMA contributions in both design- and run-time. The design-time emphasizes technologies to ensure that the front ends are adequately platform independent, i.e., that the front ends can effectively use available device and environment resources. The run-time emphasizes adapting and loading application front ends to devices, and ensuring that the front ends can interact with the environment services. In addition, the run-time provides some specialized services, e.g., capability monitoring, disconnection support, and checkpointing. The rest of this section briefly describes the design-time language and run-time support included in PIMA.

### Design Time

An application to be run in this pervasive world should not make undue assumptions about the devices upon which it will run or the environment services it will use. A developer must design an application front end that sufficiently describes the user interaction in an abstract manner to allow the front end to be run on most current or future devices. Additionally, a developer must describe the services needed by the application in a sufficiently abstract manner to allow the application to run in diverse environments. One goal of PIMA is to provide a special language and design-time tool to simplify this otherwise formidable task.

Figure 2 shows the different aspects of design-time requirements of an interactive application. The application is composed of two, generally orthogonal pieces: 1) a device-independent logical definition of the application, including a description of the user interaction; and 2) a realization of the user interaction that is circumscribed by device characteristics.

The device-independent user interaction is based on a set of abstract interaction elements that capture the intent of a user when he/she interacts with the front end. These elements do not depict physical resources, such as buttons or text fields.

Rather, they depict interaction intent, such as activation or editing. These interaction elements are then automatically mapped to a supported toolkit.

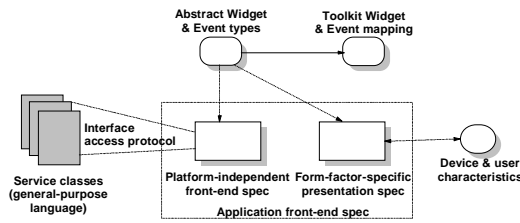


Figure 2

To build an application, the developer partitions the application into a front-end and a set of services. The services can be implemented with any programming language that supports the PIMA interface access protocol. The front-end definition consists of the declaration of interaction elements and a set of event handlers that process user- or environment- initiated actions. These handlers may call methods on the services using the access protocol.

In addition, the developer may provide a set of presentation templates that assist the device in presenting the declared front end to the users. These templates are written with some knowledge of the target devices, perhaps in some general classification.

**Run-Time**

The PIMA run-time supports application deployment and execution tailored to interactive environments. Figure 3 identifies the key portions of the run-time.

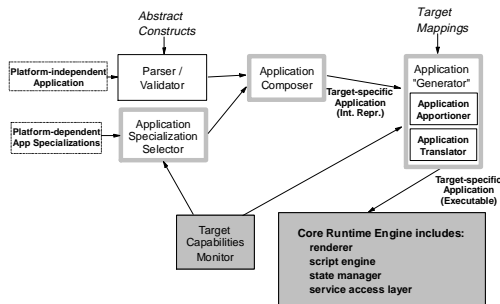


Figure 3

As shown in Figure 3, the application composer uses a parsed version of the platform-independent application and a device specific template to build a representation of the application specialized to the device (or devices) currently interacting with the user. This representation is then apportioned among the devices and translated to a form that the device can execute, including rendering the user

interface. This representation is then sent to the execution engine on each device. A monitor run-time service is also shown, which responds to both changes in the capabilities of device and environment resources and to user-initiated changes in the device portal set. In such situations, the application executable may be rebuilt to match the new execution environment.

**STATUS AND FUTURE WORK**

Presently, we have created an initial language specification and have implemented a simple PIMA run-time. We are using this as a basis to explore application apportionment, state management, and automatic synthesis of user interface representation. We are also investigating the notion of composable applications and cross-device application support.

**ACKNOWLEDGEMENTS**

We gratefully acknowledge the contributions of Kinichi Mitsui and Shinichi Hirose, especially to the run-time components.

**REFERENCES**

1. Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., Shuster, J.E., UIML: An Appliance-Independent XML User Interface Language. Available at <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>.
2. Borriello, G., et. al. Portolano: An Expedition into Invisible Computing. Available at <http://www.cs.washington.edu/research/portolano/>.
3. Dertouzos, M., The Future of Computing. Available at <http://sciam.com/1999/0899issue/0899dertouzos.html>.
4. Katz, R., et. al. The Iceberg Project. Available at <http://iceberg.cs.berkeley.edu/>.
5. Georgia Tech, Future Computing Environments. Available at <http://www.cc.gatech.edu/fce/projects.html>.
6. OMG, CORBA 2.3.1 / IIOP Specification. Available at <http://www.omg.org/library/c2indx.html>.
7. Sun Microsystems, Inc. JINI Connection Technology. Available at <http://www.sun.com/jini/>.
8. Weiser, M., et. al. The Xerox PARCTAB. Available at <http://www.ubiq.com/parctab/parctab.html>.
9. Wiecha, C. and Boies, S, Generating UserInterfaces: Principles and Use of its Style Rules. In Proceedings of the Third Annual ACM SIGGRAPH Symposium on User Interface Software and Tech., pp. 21-30, Oct 1990.