Challenges: An Application Model for Pervasive Computing

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

IBM T. J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532

ABSTRACT

The way mobile computing devices and applications are developed, deployed and used today does not meet the expectations of the user community and falls far short of the potential for pervasive computing. This paper challenges the mobile computing community by questioning the roles of devices, applications, and a user's environment. A vision of pervasive computing is described, along with attributes of a new application model that supports this vision, and a set of challenges that must be met in order to bring the vision to reality.

1. INTRODUCTION

Pervasive computing is maturing from its origins as an academic research area to a commercial reality. This transition has not been a smooth one and the term itself, pervasive computing, still means different things to different people. For some, pervasive computing is about mobile data access and the mechanisms needed to support a community of nomadic users. For others, the emphasis is on "smart" or "active" spaces, context awareness, and the way people use devices to interact with the environment. And still others maintain a device-centric view, focusing on how best to deploy new functions on a device, exploiting its interface modalities for a specific task.

Pervasive computing encompasses all of these areas, but at its core, it is about three things. First, it concerns the way people view mobile computing devices, and use them within their environments to perform tasks. Second, it concerns the way applications are created and deployed to enable such tasks to be performed. Third, it concerns the environment and how it is enhanced by the emergence and ubiquity of new information and functionality.

Today, pervasive computing is more art than science. It will remain this way as long as people continue to view mobile computing devices as *mini-desktops*, applications as *programs* that run on these devices, and the environment as a *virtual space* that a user enters to perform a task and leaves when the task is finished. This paper challenges the mobile computing community to adopt a new view of devices, applications and environment. Specifically, our vision can be summarized in three precepts:

- A device is a portal into an application/data space, ***not*** a repository of custom software managed by the user.
- An application is a means by which a user performs a task, ***not*** a piece of software that is written to exploit a device's capabilities.
- The computing environment is the user's information-enhanced physical surroundings, ***not*** a virtual space that exists to store and run software.

A new application model is needed to support this vision. This paper describes the attributes of such a model.

2. TODAY'S SCENARIO

Albert uses his PDA as the main repository for his personal information management, or PIM, data. Yesterday morning, the batteries on his PDA died while he was walking over to Betty's office for a meeting. Of course, his PDA synchronizes with his laptop, but he did not have his laptop with him, just his mobile phone. So, he was stuck looking at the PIM data that he stores on his phone. This information is much less likely to be complete, since it is more tedious to enter data on his phone. *Why can he not run the same program on his phone as his PDA?* Sure enough, the number he was looking for was not there. *Why is his PIM information spread across so many devices, some of which cannot speak to one another?*

When he got to Betty's office, he was a little early. The secretary gave him some new batteries for his PDA. He figured

*Email: {banavar, jpmunson, jsussman}@us.ibm.com.
†This work was done while authors were with IBM Research. Current email: jebeck@ix.netcom.com, eugene@newyork.usa.com, deborra@zedak.com.

MOBICOM 2000 Boston MA USA

To appear in the proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)

he would take the opportunity to print out his daily calendar. The offices are wired for a common printing service, so this should not have been a problem. He had printed his PIM data from his laptop many times, but the version of this application is different on his PDA. He spent an inordinate amount of time searching before he found the device's printing capabilities. *Why does the program have to be different on the different devices, instead of adapting itself to the device?* As it turned out, his PDA has no way to discover services, so it could not access the printer. *Why can some devices access some services, and not others?*

During the meeting with Betty, Albert complained to her about his morning experience. She showed him a new PIM program that she uses, which looked better to him. He would have liked to start using it immediately, but he could not. *Why not have the program live in the environment, so that it is immediately available for use?*

He got home that night, and used his laptop to upgrade his PIM program. He saw that this program uses a web-based location service to provide context awareness. He subscribes to a GPS service. *Wouldn't it be nice if the context awareness could be provided by any location service?*

Given his problems from earlier that day, Albert decided to synchronize all of his PIM information. It had been a while, since he always finds the synchronization process painfully complex. Sure enough, there was a number which he had updated in two different places, and he could not remember which was correct. *Why is the fact that there are multiple copies of the information exposed to him?* Needless to say, Albert had a frustrating day!

3. A NEW APPLICATION MODEL

The scenario in the last section illustrates that there are many things people would like to do with their mobile devices that are not supported today. These problems are not confined to today's devices or networking technology or programming standards and APIs. Improvements in each of these areas would surely help, but the problems are much broader.

We believe that the problems are rooted in the notions people have of computing devices, applications, and the environment. As mentioned in Section 1, we believe that these notions need to change fundamentally, and an application model needs to emerge that supports these changed notions.

To model the applications that we are envisioning, it is necessary to consider the life-cycle of an application. This life-cycle can be divided into three parts: *design-time*, *load-time* and *run-time*.

Design-time is when the developer creates, maintains and enhances the application. At load-time, the system composes, adapts and loads the application components into an application instance on particular hardware devices. At run-time, the end-user invokes the application and uses its functionality. The system provides an environment in which the application can run, and adapts the application to variations in this environment.

In this section, we present a new application model from the perspectives of application design-time, load-time and run-time. For each perspective, attributes of the model are described along with a set of challenges. We show how the attributes of the new model support the precepts "device as portal", "application as task" and "physical surroundings as computing environment".

3.1 Design-Time

Imagine building an application that fits the three precepts that are the basis of this challenge paper. If "devices are portals," then the application should not be written with a specific device in mind. The developer should not make any assumptions about the screen size or device capabilities, or even that there is a screen at all (for example, an application may be run using a voice synthesizer and a phone). The user interface of the application must not include any information specific to a device or set of devices. Instead, the application front-end should be device-neutral.

If applications are to be device-neutral, then the developer should not start with the presentation and then fill in the underlying logic. The task logic should not be secondary to the user interaction. The user interface definition should not include a rigid decomposition of the interaction. Rather, the decomposition of the user interaction should be driven by the definition and structure of the tasks. The application description should capture the purpose of the user interaction at a high level.

If the environment of an application is to be context aware, then the developer should not make assumptions about the services that are available. Services that the application needs in order to run should not be explicitly named, but rather specified in an abstract manner. Furthermore, there may be services available to the application at run-time that are not known or available to the developer at design-time, but may be useful for the task. Applications should be able to use such services. When appropriate, the designer can abstractly specify optional services that, if present at run-time, enhance the application.

3.1.1 Programming Model

As described above, the programming model must allow for the description of abstract user interfaces and abstract services. The structure of the program should be described in terms of tasks and subtasks. The granularity at which these tasks are presented to the user is a load-time issue, and therefore the relationship among the tasks must be rich enough that the user interface can be actualized at the various granularities. We call this relationship *navigation*, as it specifies how the user will navigate the sub-tasks that make up the application. The challenges for this programming model are:

- Identifying abstract interaction elements. These abstract interaction elements must capture user intent, not device mechanism. That is, base elements of user interaction must abstract away the differences in the devices. For example, an application running on a device with a GUI may offer a button for the user to

perform some action; on a voice-activated device the same action may be performed via a spoken command.

- Specifying an abstract service description language. Application logic may use existing services or infrastructure, as well as service instances unanticipated by the designer. A means is needed to express the expected function of a service, allowing for different services to provide this function when the application is running. This must allow for services to be declared optional as well.

  In the scenario above, Albert's PIM, instead of having a location service built into it, would specify a requirement for a location service, using a abstract service-description language. This requirement could be satisfied by any location service instance in Albert's current environment.

- Creating a task-based model for program structure. The application should be delineated into tasks and subtasks. A task includes the abstract interaction and the application logic, including the use of the services. The structure is used by the system to generate device specific "presentation units"; e.g., screens.

  For example, in a PIM calendaring application, user authentication is one task, browsing the appointments for a day is another task, and entering a new appointment is another task. On large-screen devices, the browsing and appointment-entry tasks may be presented on the same screen, whereas on small-screen devices, these tasks may be presented on separate screens.

- Creating a navigation model. The navigation specifies what causes a task to begin and end (e.g., a user action), and what tasks precede and follow it. This information is complementary to the task structure, and is used by the system to automate the flow of the "presentation units" when the application is running.

3.1.2 Development Methodology

The purpose of a development methodology is to take the developer through a step-by-step process of realizing the application from a set of requirements. An ideal methodology for building an application is to focus on the user task, rather than the user's interaction with an interface on a specific device in a specific environment. This methodology would allow a programmer to build an application by answering questions such as:

- What task does the user want to accomplish? If the task is a composite of many subtasks, how are these defined to assist the user in his/her overall task?
- What is the "flow" through the tasks? How does each task begin? How does it end? How does one subtask initiate another in a dynamic framework?
- What is the user interaction for each task? What user actions are needed to perform the task? How are user actions a reflection of user intent?
- What information does the user need to perform the task? Where does this information come from?
- What logic does the system perform for each (sub)task? Is it possible for the (sub)task logic to adapt itself to a given environment?

By answering these questions, the programmer will have specified an application at a high level of abstraction. The concrete results of these questions make up the implementation of the application. Given the programming model explained above, the implementation will be made up of a task structure annotated with navigation flow, an abstract user interface for each task, and scripting logic that details the task function.

The major challenge here is to build a development environment that supports the above methodology. This methodology is not captured by current programming tools. Certain parts of the methodology, such as navigational flow, may lend themselves to visual interfaces, whereas others such as scripting logic may not.

3.2 Load-Time

An application model derived from the basis of the three precepts requires a more dynamic load-time approach than is traditionally supported. To realize the concept of "device as portal," devices must dynamically discover what applications are available, and the system must adapt the applications to the device resources available. An application must be specified in terms of its requirements, the device must be described in terms of its capabilities, and some mediating algorithm must be used to negotiate a match between these competing constraints.

To realize the concepts of "application as task" and "physical surroundings as computing environment", the system must be dynamic at load-time. That is, the tasks that a user wishes to perform may depend on the physical surroundings. Such tasks are enabled by contextual services. The system must, therefore, be able to discover and compose the services that are available in the physical environment, in order to perform desired tasks. This is in contrast to today's model, where applications are loaded onto a device manually from a CD or other storage medium, and managed by the user rather than the system.

3.2.1 Dynamic Discovery

Applications and services live in the surrounding physical distributed environment. Discovery mechanisms allow a mobile device to dynamically identify and enumerate the applications and services in its local vicinity.

The major challenge posed by dynamic discovery is the definition of a service adaptation layer. A standard definition is needed, to both hide the differences between heterogeneous service frameworks and to maximize the use of legacy code.

In our scenario, dynamic discovery is needed both for Albert to print his schedule in Betty's environment—his PDA should 'discover' her office's printing service—and for his PDA to switch from one location service to another.

3.2.2 *Requirements and Capability Negotiation*

At load-time, a device needs to negotiate with a server that hosts applications and services for several reasons. First, the device may not have all of the resources needed to run some of the applications and services. The set of available software needs to be pruned so that only the hostable functions are presented to the user. Second, application performance is a concern, so it may be desirable to split the execution burden between the device and available servers. This split, which we call *apportioning*, uses information about the currently available resources and the resource demands of the application. Some of the challenges related to negotiation are:

- Modeling device characteristics and application requirements. The characteristics that are relevant for differentiating between devices must be codified, and a metric for each of these characteristics must be developed. The application requirements must be specified in the same terms.
- Developing negotiation protocols. Such protocols are necessary for a device to ascertain what subset of applications and services can be hosted within the bounds of its resource limitations.
- Incorporating fast and efficient apportioning algorithms. Loading is not necessarily a one-time function. Instead, the load-time mechanisms may be re-invoked when changes in the device or physical environment warrant re-apportioning.

3.2.3 *Presentation Selection, Adaptation and Composition*

A good user interface must exhibit qualities such as consistency and style, which are difficult to quantify and synthesize. Indeed these qualities are subject to human taste. These qualities are embodied differently on devices with different interface modalities and form factors (e.g., a graphical input device versus one with a speech interface). Thus, it may be desirable to have multiple abstract representations of the application interface, one for each combination of interface modality and form factor. These will most likely need to be generated by human designers, perhaps through semi-automated tools. The challenges for the load-time system are:

- The system needs to support dynamic selection of an appropriate application interface from a set of available interfaces, based on the device's resources and form-factor. The presentation selected in this manner will be specific to an interface modality and form factor. Further adaptation may be necessary for the characteristics of a particular device.
- The system needs to seamlessly integrate the applications and services found in the environment. This involves composing the functionality (e.g., a discovered map application should be able to use a discovered GPS service) as well as the user interface (e.g., if a service is discovered for controlling a VCR, the interface needs to be integrated and displayed along with the interfaces of other discovered services). The composition is subject to the constraints and resource limitations of the device and the composition restrictions of the discovered entities.

In the scenario above, Albert wanted the same PIM application on his cell phone as the one on his PDA—not two different PIM applications accessing the same data. For example, a PIM application may offer the function of querying for an individual's manager or other members of the individual's work group. This function should be available whether the PIM's interface is presented on Albert's PDA or his cell phone—appropriately adapted to the device's capabilities.

3.3 Run-Time

To realize the precept "device as portal," the run-time must monitor the resources for the currently active portal, or portal set, and appropriately adapt the application to those resources. In addition, the run-time must respond to changes initiated by the user. For example, the user may choose a different set of portal devices.

To realize "application as task," the run-time must allow a user to initiate and perform a task in an uninterrupted manner, despite changes in the environment and portal devices. The run-time should support handoff of task context from one environment (e.g., office) to another (e.g., car), possibly through a disconnected state. The key to supporting a task-oriented application is that a user's access to the task be continuous.

To realize "physical surroundings as computing environment," the run-time must be able to take advantage of services provided by the environment and the physical resources available within it. The run-time must handle unexpected failures, such as exhausting batteries or a service crash. Existing failure detection and recovery mechanisms may need to be re-examined for their applicability in this new paradigm.

3.3.1 *Monitoring and Redistribution*

The application model proposed in this paper requires the run-time to detect changes in the resources of any portal device or environment hosts that participate in application execution. Resource changes include changes in available network bandwidth, introduction of new devices into the environment, introduction of new users and/or applications, etc. In response to detected changes, the run-time must initiate a reapportionment and/or relocation of application components. The challenges introduced by this monitoring and redistribution include:

- Non-obtrusive re-apportioning. Resource changes may impact the user's interaction with the application. However, some changes may be transient and unseen by the user. Transient resource changes should be recognized as such and should not impact the application. When changes are significant and long-lived, the application should be automatically re-apportioned, with minimal impact on the user.

- User initiated re-apportioning. The user may initiate re-apportionment of the application. Reasons for re-apportionment may range from anticipated change in the connectivity of devices to a mobile user entering the proximity of new devices. In the latter case, the user should be given a choice of whether to use the new devices or not.

3.3.2 Disconnection

One of the resources that must be considered when making apportionment decisions is the communication network. If the network connection between client and server is detected to degrade via run-time monitoring, the apportioner may react by migrating code from the server to the client to reduce the application's demand for communication. In this way, a running application can react to dynamic changes in the quality of the network connection.

For some applications and devices, it may be feasible to migrate the entire application to the client in order to accommodate brief, sporadic network disconnections. However, this approach is not viable for devices with limited resources, or for sudden, unanticipated network disconnections. Because of this, explicit support for disconnected operation needs to be added to the model. In other words, the model needs to be augmented in order to bridge the desire of using a device (along with its accompanying resource limitations) as a portal while minimizing the impact of network disconnections.

The major challenge in this area is automating disconnection and reconnection as much as possible. The run-time should prepare for disconnection without a user's intervention whenever possible (e.g., automatic migration as described above). For those scenarios where user intervention is needed, there should be a natural way for a user to prepare for disconnection, minimizing overall task disruption (e.g., hoarding, as in [14]). While disconnected, a framework should be provided to automate tasks such as queuing network requests. A user should be able to reconnect to an application within the user's current environment. Reconnection in both the original and transitory environments should be supported.

3.3.3 Failure Detection and Recovery

Many existing failure detection and recovery techniques may be applicable to pervasive environments. However, they may need to be modified to better serve the particular requirements of these environments. Some of the challenges here include:

- Adapting checkpointing strategies. In the application model discussed here, the device is a user's portal to an application that runs in an environment. This is a model previously unexamined by traditional fault tolerance research. Requirements on the type and timing of checkpointing may be different from the current state of the art.
- Understanding disconnection. The distinction between failure and disconnection is often blurred in conventional systems. In this application model, the distinction is important. Disconnections should not be treated as failures, as they are part of the expected specification of the environment.

4. A GLIMPSE OF TOMORROW

Consider the scenario of Section 2, but within the new environment of tomorrow presented in this paper. At design-time, tomorrow's developer would realize that PIM information management is separate from the PIM front-end, and would therefore create a service for information management. This would allow the user to have a single PIM service which is part of the environment, immediately eliminating the synchronization problem. When the user updates a phone number, that phone number is the same regardless of the device through which it is accessed.

The application is no longer thought of as a selling tool for a device. Instead, the application is built to be run on any device. Therefore a single, consistent, view of the task of accessing personal data is supported by all devices.

Tomorrow's developer would also realize that part of the task of accessing personal information might be to get a hard copy. Because the printer is also seen as a service, run-time discovery of the printer is easily enabled. If the personal data of different people is linked on the network, then that information is available to all of the authorized devices in the environment. Similarly, the map program which is linked to the PIM program (to provide context awareness of schedules) is described in an abstract manner, and therefore the conversion from the web-based location service to the GPS-based location service is transparent to the user.

Finally, because the PIM program is managed by the environment, if the user "finds" a new, improved version, he/she can easily update the application on any device without hassle or delay. From that point on, this device can use the updated application. Indeed, the concept of "upgrading" software may quickly become anachronistic!

5. RELATED WORK

The model proposed in Section 3 is not as revolutionary as it might at first appear. Its roots can be found in several mature technologies. We believe that there is a natural evolutionary path to realizing this model. Indeed, parts of this model are being realizing in current work in pervasive computing.

In this section, we present some existing technologies that provide the underpinnings of the application model. We also present related pervasive computing efforts. Both common and missing elements of each technology will be discussed in relation to the proposed model.

5.1 Foundation Technologies

5.1.1 User-Interface Management Systems (UIMS)

UIMS efforts identified the need to divorce the user interface from the rest of the application logic. Examples include the work reported in [21] and the UIML System at Virginia Tech [1]. In these systems interaction front-ends are tailored to allow users to perform tasks as best supported by the devices. The application model proposed in this paper

expands this goal beyond the UI, to handle the heterogeneity of other device capabilities such as compute power, network bandwidth, and available services.

5.1.2 Client-Server Computing Model

The client-server model was introduced as a way to share applications and data within an organization. Applications were generally developed for a specific set of platforms, but they were divided to better accommodate resource consumption. Those parts of the application that were best supported by centralized servers (e.g., data access and memory/compute intensive portions) were included in the server piece of the application. Those that more closely interacted with users were included in the client piece. Standard protocols, such as sockets and RPC [19], were developed for communication among the pieces.

The client/server division is often statically decided at design time. Such a division may not yield the best performance over the full range of network conditions or the full range of client devices, which may vary in processing power. The client is typically assumed to be constantly connected to the server, especially for the "thin-client" variation of the model. Thus, in a mobile environment, the model must be enhanced to accommodate sporadic disconnections, and caching is needed to buffer this effect from applications (e.g. [14]). Furthermore, the model supports heterogenous platforms, especially by the acceptance of the standard protocols. However, the application must be recoded to each platform, making it extremely costly and complex to develop and maintain the application code base.

5.1.3 Java™ Computing Model

The Java computing environment [3] alleviates the re-coding problem described above, in that it enables device-independent code that can be shared across platforms. Applications are written to a common Java platform consisting of a Java Virtual Machine (JVM) and a set of standard libraries. However, some devices are not able to support this standard platform. While there are some efforts to define Java subsets, this reduces the platform independence of Java.

Another problem is that Java uses a least-common-denominator approach for user interfaces. The toolkits Java provides for building interfaces, such as AWT [22] and Swing [8], assume user interface elements that are common to all currently supported platforms. This leads to a less than ideal environment for creating high-quality user interfaces, adapted to the capabilities of individual devices. That is, such interfaces cannot recognize and utilize device-specific resources available on some devices, such as a scroll wheel or hard buttons. Additionally, Java widgets contain presentation information and maintain assumptions about the underlying structure of the user interface. Because of this, user interfaces written in Java are not portable across devices with different form factors, not to mention interface modalities. Consequently, this approach is not suitable for the type of device-specific rendering described in Section 3.

5.1.4 Web Technologies

The World Wide Web has moved applications away from generalized GUIs towards a more information based interface. Users browse global (virtual) information. They are able to initiate server actions. These interactions are enabled by the use of a browser model that provides a consistent and uniform user experience across heterogeneous clients. However, much like Java, information on the web is authored for presentation on specific platforms, and usually cannot take advantage of the resources available on different devices.

Though the web model offers promise for pervasive applications, it needs to be augmented to address new concepts, such as context-awareness and intermittent access. More complex server actions need to be supported – users need not only to initiate server actions but to respond to "unexpected" server information (called push services). Web-based "applications" are evolving from browsable content to interactive applications with broader user interactions via graphical widgets. To enable such user interactions, technologies such as Java applets [12] and JavaScript [10] have been added to the web model. While such technologies do expand the capabilities of the browser, in some regards they are a step backward. For instance, JavaScript is not device independent, violating the reason for using a web browser in the first place.

5.1.5 Service Technologies

The emergence of distributed object models within the confines of the client/server computing model (e.g., CORBA [17], DCOM [16] and Java's Remote Method Invocation (RMI) [20]) set the stage for service frameworks and related discovery protocols. The subsequent introduction of service frameworks such as SLP [11] and Berkeley's Ninja project [6] are extending the role of the wired Internet and are enabling the discovery and use of functionality based on context. These approaches allow a software service to be discovered dynamically based on attributes supplied by the client.

The emergence of Sun's Jini technology [4] has enhanced the concept of service frameworks. Jini provides a common framework for registering available services and answering client look-up requests. The combination of downloadable Java code and Java's RMI model allows a discovered service to be loaded dynamically and then executed either locally on the client, on a service provider, or any combination of the two. However, Jini relies on the use of a central server which acts as a broker; it registers services on behalf of service providers and answers look-up requests on behalf of clients. Because of this, Jini requires a connected network.

More recently, service frameworks have been introduced for use over small proximity wireless networks, such as SDP over Bluetooth [13] and the MOCA service framework [5] for ad-hoc networks. The aim of these approaches is to provide transient access to context-sensitive information and functions, as well as gateway access to the larger context of the wired Internet. Consequently, these approaches allow a mobile device to dynamically discover and adapt its functionality to changes in the user's environment. These approaches support the application model of this paper.

5.2 On-Going Pervasive efforts

Perhaps the seminal project in ubiquitous computing was the ParcTab [2, 15] effort at Xerox PARC. The roots of

the application model described in Section 3 can be found there. In the ParcTab project, much attention was given to an application traveling with the user, and being accessible from mobile devices. This is an example of devices acting as portals into an information space and lends credence to the proposed vision of pervasive computing.

Unfortunately the ParcTab project ended before it could realize its potential. At that time, applications were often custom coded, and the project focused on the utility of pervasive applications rather than application development and an accompanying application model.

Recently, other efforts have begun under the designation "Invisible Computing." They appear to pick up where the ParcTab effort left off. These approaches are, by and large, in harmony with the application model described here. Two sample projects, Portolano [9] and Oxygen [7] are discussed below.

5.2.1 Portolano

The Portolano Project at the University of Washington focuses on three main areas: Infrastructure, Distributed Services and User Interfaces. Portolano addresses a particular research area, that of data-centric routing. Such routing facilitates automatic data migration among applications on behalf of a user. Data becomes "smart", and serves as an interaction mechanism within the environment.

Portolano's view of horizontal integration is synonymous with our view of composition that we presented in Section 3.2. Though Portolano proposes an infrastructure based on mobile agents that may appear to differ from our model, agents are an implementation option of our model. That is, agents interact with an application and the user, and applications must be developed to utilize the agents. The service deployment model of Portolano is similar to our view of how applications and services are deployed into an environment.

The largest apparent divergence between Portolano and our model concerns the role of user interfaces. Portolano emphasizes invisible, intent-based computing. The intentions of the user are to be inferred via their actions in the environment and via their interactions with everyday objects. Devices are still portals into the environment. However, their tasks are implicitly defined. Our model allows the devices to explicitly run tasks. In either case, the portals capture user input, and reflect that input to the application.

5.2.2 Oxygen

Oxygen is another approach to Invisible Computing, being pursued by MIT. The emphasis is on understanding what turns an otherwise dormant environment into an empowered one. Users of an empowered environment shift much of the burden of their tasks to the environment.

The Oxygen project is focusing on eight, environment-enablement technologies. The first is a new mobile device, the H21, which relies on software to automatically detect and re-configure itself as a cell phone, pager, network adapter or other type of supported communication device. The H21 is a good example of a mobile device that acts as a portal.

The second and third technologies are the E21, an embedded computing device used to distribute computing nodes throughout the environment, and N21, network technology needed to allow H21s and E21s to interact. These provide some of the load- and run-time requirements described in Section 3.2.

The final five technologies underlying Oxygen are all aimed at improving the user experience: speech, intelligent knowledge access, collaboration, automation of everyday tasks, and adaptation of machines to the user's needs. Inherent in these technologies is the belief that shrink-wrapped software will disappear as an application delivery mechanism. More dynamic mechanisms will be used instead. This reflects the load-time attributes described in Section 3.2.

The technologies underlying Oxygen are complementary to the application model described in Section 3. Our model also specifies design-time infrastructure, which would enable the development of applications for use in conjunction with Oxygen.

6. RESEARCH PLAN

The PIMA project at the IBM T.J. Watson Research Center is taking an incremental, evolutionary approach to implementing the application model described in Section 3. PIMA is progressing along a number of concurrent research thrusts. Each thrust is rooted in existing technology, and is gradually introducing new functionality. In this section, we describe our research plan.

Our research is based on a number of assumptions. One assumption is that an underlying services-based distributed architecture is a part of interactive environments. Another assumption is that although certain devices may be best suited for certain tasks, users will interact with applications and services via whatever devices are handy at a given time. The emphasis is on task enablement, rather than support for device-specific applications such as high-end word processors or games.

We describe below each of our concurrent research thrusts in three steps: the current status, the next steps, and the eventual goal.

- Design-time Environment: Currently, the developer uses a programming model to statically specify the application interface, any scripting logic, and the interface to back-end services. However, developing a device-independent application is inherently more complex than developing a device-specific one. In the next steps, the developer's burden will be alleviated by inferring as much as possible about the designer's intention, generating design-time artifacts where appropriate and providing realistic defaults. Eventually, the goal is to build a development environment that supports a comprehensive methodology and allows designers to manage the added complexity.
- Device-specific Rendering: Currently, device-specific rendering is performed via static specification of the mapping from the device independent application to the toolkit and the form factor of particular platforms.

While this is adequate for adapting the presentation of individual elements of a user interface, it is inadequate for handling differing form factors and interface modalities. In the next steps, capability negotiation is being introduced as a means to select, at load time, an interface specification from a finite set of specifications. Eventually, the goal is to investigate the use of automatic synthesis to generate device-specific renderings of an abstract interface specification.

- Distributed Services: Currently, networked-based services are assumed, such as those supported by SLP. However, applications and services should be associated with a local environment. In the next steps, service frameworks and discovery mechanisms that are appropriate for use over small-proximity, ad-hoc networks will be introduced. Eventually, the goal is to develop and deploy a service adaptation layer that allows uniform access to services hosted in various frameworks.

- Application Apportioning: Currently, a capability negotiation session is being used in conjunction with an algorithm such as [18] to dynamically apportion an application at load-time. However, the resource environment of a mobile device is hardly static. In the next steps, run-time monitoring and service migration mechanisms will be introduced to dynamically vary the application apportionment between client and server at run-time. Eventually, the goal is to introduce caching and automatic checkpointing to support sporadic network disconnections and application recovery following a client device failure, respectively.

- Application Adaptation: Currently, applications are written to accommodate two classes of services: essential services, which must be present for the application to proceed; and optional services, which, if present, allow the application to provide additional functionality. However, the application's execution environment cannot be known statically at design-time. In the next steps, composition techniques will be introduced to allow the functions of applications and services to be chained together and enable the integration of their respective user interfaces. Eventually, the goal is to develop the interfaces and mechanisms needed to allow an application to identify and use a service at run-time that was unanticipated when the application was written.

The above research plan takes us from well-understood current technologies towards realizing the application model envisioned in this paper. However, addressing all the issues within this broad research agenda is a challenge for the entire mobile computing community.

7. SUMMARY

This paper began by exposing some of the limitations behind the way mobile computing devices are used today. As the scenario illustrated, today's applications do not enable people to perform many of the tasks they need to do, do not provide satisfying user experiences, and fall far short of the potential for pervasive computing.

For pervasive computing to meet the expectations of mobile users, fundamental changes need to occur in the way people perceive the roles of devices, applications and the environment. Again, devices need to be perceived as portals into the application/data space supported by the environment, rather than repositories of custom software. Applications need to be seen as tasks performed on behalf of a user, not as programs written to exploit the resources of a specific computer. And, the computing environment needs to be recognized as an extension of the user's surroundings, not a virtual space for hosting and running programs.

To realize this vision of devices, applications and environments, we believe a new application model is needed. The model is characterized by a device-independent application development process, which includes abstract specification of the application front-end and the application's resource and service requirements. The model includes a highly dynamic load-time system supporting application discovery, resource and capability negotiation, and application apportioning. The run-time system allows the resources to be dynamically shared among client devices and servers. It also includes monitoring and checkpointing, and enables a running application to migrate from device to device or to simultaneously utilize the interface capabilities of multiple devices.

Several on-going thrusts in pervasive computing, such as the Portolano and Oxygen projects, share our view of the roles of devices, applications and environments. The application model presented here strengthens this common vision, particularly in the area of developing, deploying and managing applications. Moreover, the proposed application model provides common underpinnings that can unify the view of applications across such environments.

In summary, the application model introduces a number of design-time, load-time and run-time challenges. These challenges expose the boundaries of the current state of the art, and must be addressed if the full potential of pervasive computing is to be realized.

8. REFERENCES

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An Appliance-Independent XML User Interface Language. In *Proceedings of the Eighth International World Wide Web Conference*, pages 617–630, May 1999.

[2] N. Adams, , R. Gold, B. Schilit, M. Tso, and R. Want. An Infrared Network for Mobile Computers. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 41–52, August 1993.

[3] K. Arnold and J. Gosling. *The Java Programming Language, Second Edition.* Addison Wesley, 1998.

[4] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification.* Addison Wesley, 1999.

[5] J. Beck, A. Gefflaut, and N. Islam. MOCA: A Service Framework for Mobile Computing Devices. In

Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access, pages 62–68, August 1999.

[6] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R Katz. An architecture for a secure service discovery service. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 24–35, August 1999.

[7] M. Dertouzos. The Oxygen Project. *Scientific American*, 281(2):52–63, August 1999.

[8] R. Eckstein, M. Loy, and D. Wood. *Java Swing.* O'Reilly and Associates, 1998.

[9] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing. The Portolano Project at the University of Washington. In *Proceedings of the Fifth ACM/IEEE International Conference on Mobile Networking and Computing*, pages 256–262, August 1999.

[10] D. Flanagan. *JavaScript: The Definitive Guide.* O'Reilly and Associates, 1998.

[11] E. Guttman, C. Perkins, J. Viezades, and M. Day. *Service Location Protocol, Version 2. RFC 2608.* IETF, November 1998.

[12] K. Hopson, S. Ingram, and P. Chan. *Developing Professional Java Applets.* Sam's Publishing, 1996.

[13] R. Mettala. *Bluetooth Protocol Architecture, Version 1.0.* Bluetooth Special Interest Group, August 1999. `http://www.bluetooth.com/v2/document`.

[14] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaka, E. Siegel, and D. Steere. CODA: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[15] B. Schilit, N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB Mobile Computing System. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 34–39, October 1993.

[16] R. Sessions. *COM and DCOM: Microsoft's Vision for Distributed Objects.* Wiley Computer Publishing, 1997.

[17] J. Siegel. *CORBA Fundamentals and Programming.* Wiley Computer Publishing, 1996.

[18] H. Stone and S. Bokhari. Control of Distributed Processes. *Computer*, 11(7):97–106, July 1978.

[19] SUN Microsystems Inc. *Remote Procedure Call Protocol Specification, Version 2. RFC 1057*, June 1988.

[20] SUN Microsystems Inc. *Remote Method Invocation Specification*, 1999. `http://www.javasoft.com/products/jdk/1.1/-docs/guide/rmi/spec/rmiTOC.doc.html`.

[21] C. Wiecha and S. Boies. Generating User Interfaces: Principles and Use of ITS Style Rules. In *Proceedings of the Third Annual ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 21–30, October 1990.

[22] J. Zukowski. *Java AWT Reference.* O'Reilly and Associates, 1997.