



UBIWISE, A Simulator for Ubiquitous Computing Systems Design

John J. Barton, Vikram Vijayaraghavan
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2003-93
April 29th, 2003*

E-mail: {John_Barton, vikramv@hpl.hp.com}

simulation,
mobile,
ubiquitous,
handheld

We describe UbiWise, a simulator for ubiquitous computing. The simulator concentrates on computation and communications devices situated within their physical environments. It presents two views, each in a separate window on the desktop of the users' PC. One of the views provides a three dimensional world, built on the Quake III Arena graphics engine and serves to simulate a first-person view of the physical environment of a user. The other view, built using Java, shows a close-up view of devices and objects the user may manipulate. These views act as one unified whole by maintaining a client-server model with a central server. Multiple users can attach to the same server to create interactive ubiquitous computing scenarios. We describe how UbiWise looks to researchers and examples of its use as tool for ubiquitous computing research.

UBIWISE, A Simulator for Ubiquitous Computing Systems Design

John J. Barton, and Vikram Vijayaraghavan

Hewlett-Packard Labs,
1501 Page Mill Road
Palo Alto, CA 94304

{[John_Barton](mailto:John_Barton@hpl.hp.com), vikramv@hpl.hp.com}

Abstract. We describe UbiWise, a simulator for ubiquitous computing. The simulator concentrates on computation and communications devices situated within their physical environments. It presents two views, each in a separate window on the desktop of the users' PC. One of the views provides a three dimensional world, built on the Quake III Arena graphics engine and serves to simulate a first-person view of the physical environment of a user. The other view, built using Java, shows a close-up view of devices and objects the user may manipulate. These views act as one unified whole by maintaining a client-server model with a central server. Multiple users can attach to the same server to create interactive ubiquitous computing scenarios. We describe how UbiWise looks to researchers and examples of its use as tool for ubiquitous computing research.

1. Introduction

New hardware drives ubiquitous computing. The concept of ubiquity for computing arose from the potential for inexpensive and tiny computers embedded in or connected to our environment. However, the goals for research in ubiquitous computing lie beyond the hardware itself [1]. In pushing beyond personal computing we must solve many new problems in the interaction of multiple computers with diverse purposes and in the interaction of multiple people with these machines. From our experience [2] and from observing others, we see a limit to progress in understanding these unique aspects, a dependency cycle around hardware construction:

- The design of suitable hardware depends upon understanding how the hardware fits in to ubiquitous computing applications,
- Ubiquitous computing applications cannot be developed without suitable hardware.

This cycle leads researchers to either develop applications for recently introduced hardware--a difficult task given limited software development resources--or to create new hardware and have little time to invest in software.

In this paper we describe UbiWise, a software framework for researchers to simulate hardware and low-level software so that more work can be done on the other dimensions of ubiquitous computing. UbiWise was born out of a desire to make

progress on systems for devices [3] without waiting for the devices themselves. Ultimately we hope this leads to more ideas for hardware and more refined requirements for it. As concrete evidence of the potential we outline three applications of Ubiwise: a digital picture frame with an embedded web server, a conference room appliance, and a sensor-enhanced mobile web browser. Simulation has been an effective tool in other disciplines: Ubiwise is a step towards building such a tool for ubiquitous computing.

1.1. Requirements

To be a tool for ubiquitous computing the simulator must tackle a broad spectrum of opportunities and challenges, including the ability to:

- experiment with new sensor concepts, whether handheld or environmental, without building and deploying them;
- aggregate device functions without connecting real devices;
- develop new service and device discovery protocols without implementing these protocols in multiple mobile devices;
- explore the integration of handheld devices and Internet services without the cost and large teams need to realize this integration;
- aid rapid prototyping of new devices and protocols;
- test robustness and interoperability of subsystems;
- reproduce experiments and results of other researchers;
- mix simulated and prototype devices and service where possible.

While we are aiming at these kinds of challenges we need to insure that the simulator is not more difficult to work with than the hardware prototype it stands in for.

We did not list testing of user-interfaces or examination of the user experience in our requirements. Research in user-interface design and in human-computer interaction already provides good techniques for these aspects of ubiquitous computing. We believe that our tool may have a some role in this area, but we are primarily aiming at the next stage in ubicomp system design, where we know roughly what user will do and generally what user-interface techniques succeed.

1.2 UbiWise Overview

In our simulator, we simulate the physical environment surrounding devices and users with a modified, inexpensive, commercial three-dimensional rendering engine, Quake-Arena III [4]. We simulate prototypes of new devices and protocols with a Java program. These two views are brought up in separate windows on the user's desktop or on separate screens on separate machines. The collection of devices and physical aspects of their state is synchronized through network messages. The physical environment view is maintained by program called UbiSim; the device view is called Wise. We call the combined system "Ubiwise".

We chose to have two views to simplify the use of the simulator for research in ubiquitous computing. As we make a simulator for the physical world more and more accurate, the development time for working with the simulator increases. The Quake-Arena engine achieves high performance and good visual fidelity by using the C programming language, by avoiding complex graphics frameworks, and by using

efficient code in a simple, single-thread, view-control loop to drive the rendering engine. These properties make programming more time-consuming than necessary for our work. We find prototypes can be developed in Java more quickly than C and the Java graphics framework increases the sophistication of the user interface that a research tool can deliver. Efficient code is inappropriate for many research projects and communications programs often need multiple threads. All of these reasons argue for using a Java-base simulator as much as possible.

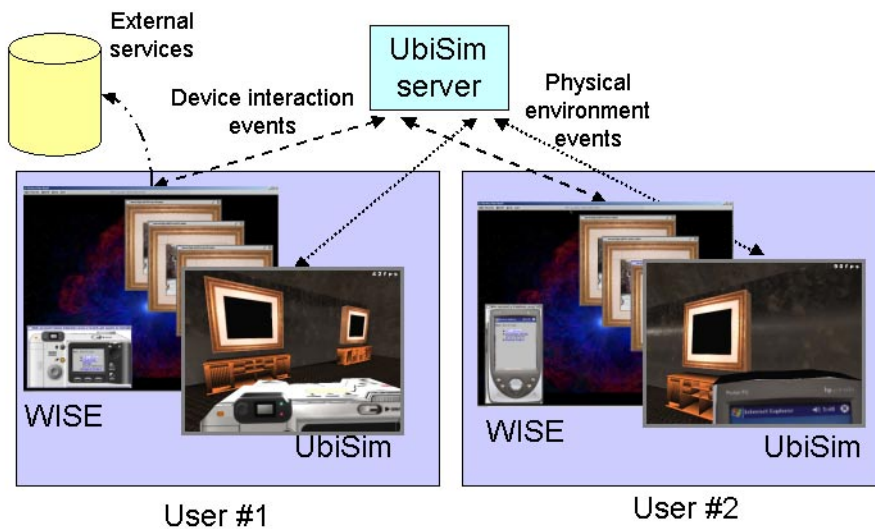


Fig. 1. Ubiwise System Overview

Consequently we split the simulator into a physical environment view and a device view. The set of devices in both views are the same. In the environment view the devices are arrayed in 3D spaces and react to physical events; in the device view they are arrayed in 2D (a desktop-like window) and react to mouse clicks and network events. This way we can get all of the benefits of a 3D environment for the physical aspects, but most of the ubicomp development work can be done in Java and run in a separate process or machine. Moreover this split also factors the development process for scenarios: we can concentrate on physical issues of space and time separate from device, service, and communication issues.

Figure 1 shows an overview diagram of the simulator for two users. Each user has two views, a device view (WISE) and an environment view (UbiSim). The physical world model is synchronized to the UbiSim server; it can run on either user's machine or elsewhere. Real or research prototype external services are shown as well. Note that "user" here means a simulated participant in a ubiquitous computing environment. A researcher may well be at the controls of both sets of views in the course of experiments. Alternatively a researcher may choose to set users into the scenario that are driven by program code, so-called "bots". (We have not explored this option as yet).

2. A Scenario Walkthrough

We walk you through a simple scenario that could be simulated using UbiWise. The scenario combines elements we have implemented for other purposes but it is not an example application. Our intent is to use the scenario to highlight the main concepts behind the simulator and to give a perspective on how it can be used to explore ubiquitous computing problems. That is, we don't provide evidence that this scenario itself is a contribution or useful.

The real world scenario we choose to simulate involves a shopper named Jane looking for a rug and two chairs for her family room that will coordinate with her existing furniture. Jane has two handheld devices to help her, a wireless PDA and a wireless digital camera. She photographs her family room and prints two images, then heads to the furniture store.

There she speaks to a salesperson that offers to superimpose images of the chairs for sale on the images Jane has in her camera. The store has two large digital picture frames for this purpose. Jane uses her digital camera to transfer her photos into one of the frames. After consulting with the salesperson, she downloads the combined image of the chair and her family room. She also loads an order form from the salesperson into her PDA. She repeats this process at the rug store. She emails the final image to her friend for help in making the final decision. At home she decides to go ahead with the purchase. She transfers the order forms to a PC, verifies the information, and submits the order. Here the scenario ends.

The scenario above, though simple, leaves a lot of questions to be explored further. Some of these issues include:

- What kind of interaction should Jane have with the shopping environment?
- What kinds of services are to be made available at the store?
- What should the UI of the handhelds be?
- How does Jane switch between different modes of operation, in this case between image selection and order selection?
- What special protocols, if any, should be used for the data transfer of the images?
- How is the list of shopping services made available to the digital camera?
- Where and how does the digital camera display information?
- How does Jane navigate using the digital camera controls to select images and a web service? How can she know which digital frame to upload her image into?
- What about interacting with other users in the environment. Can Jane share or access their data?

Clearly, these questions apply to many similar scenarios.

This real-world scenario is to be mapped into our simulator. In the simulator, each view deals with providing answers to a different set of questions asked above. The *device interaction, services and protocol* questions are explored in the device simulator. The *environment interaction and multi-user experience* is simulated using the environment simulator.

The core of the simulation is creating the virtual counterparts of devices and the picture frames present in the physical environment. The next section charts the path taken to create these virtual objects, focusing on the camera as our example. It also

introduces and develops the two views and their roles separately. We try to give a sense of the time a researcher will need for the various steps. We wrap up the discussion with a description of the simulator at runtime and we describe how the two views come together to form a complete whole.

2.1. How the Simulation Scenario is Created

The building blocks of UbiWise are the simulated devices, the virtual representations of the physical devices. The construction of the virtual device begins with one or more images of the object. This could be from various sources such as a photo of a real-world device, a digital photo created using one of the many imaging applications available like Photoshop[5], or a hand-drawn image scanned into digital format. This single image is then transformed to create virtual devices in both UbiWise and Wise views. These transformations differ for each view; we start with the environment view then cover the device view. Describing the processes for created elements of the simulation should also help clarify the nature of the simulation.

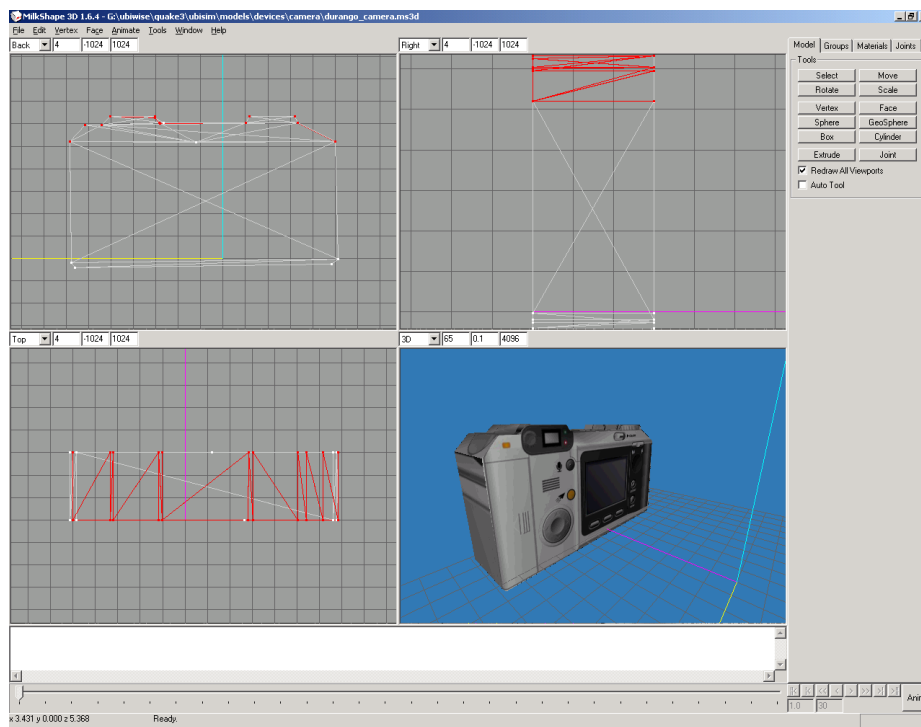


Fig. 2. Designing a 3D model for a wireless, web-browsing digital camera.

2.2 Creating a Virtual Device for the Physical Environment View

For our example scenario we need to simulate Jane's digital camera. This will be a conventional looking camera but it will be able to transfer data wirelessly to nearby

appliances. We start with three digital images used in the design of an HP camera, one for the top, back, and front; they were already in JPEG format. To create a virtual digital camera in the Ubisim view we need to make a wire-frame 3D model out of these images. We used the MilkShape [6] editor, tracing the JPEG image of the camera back with vertices on the corners of line segments around the edge of the image. Then we select vertices in triples to create triangles, forming faces of the object. The back was then “extruded” digitally and the vertices in the extrusion were modified to match the top view image. The process is a bit tedious but takes only a about a day to learn the process and to create an adequate model. Figure 2 shows our second attempt.

We also have to create the environment for the scenario. In this case it is a simple room with two digital picture frames on one wall. For this we use the GtkRadiant[7] editor, modifying an existing 3D model (known as a “map”) for a room (See Fig. 3). The initial position for the user and the devices available to the user can also be specified in the editor. There are many 3D models available to add to the realism of the scene, either from hobbyists on the Web or from commercial products. Operation of this editor is complex and it takes a couple of days to understand it; online resources for models often require format conversions to work in the editor. Thus the effort for the first simulation could be a week’s time; modification of the scenario can be done in a few minutes.

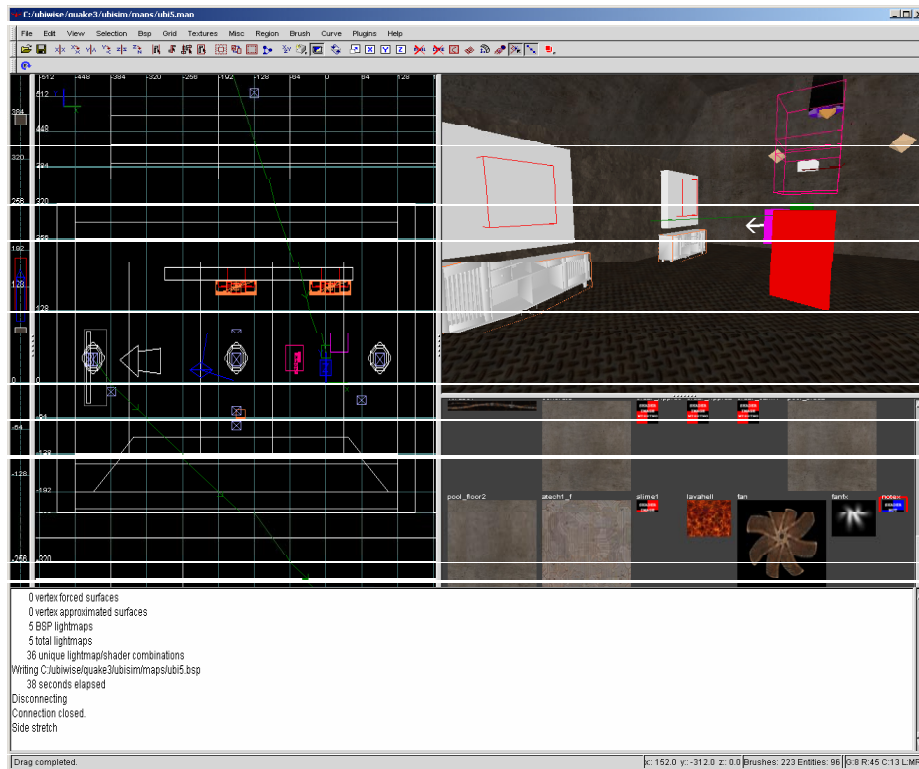


Fig. 3.. Snapshot of a Ubisim environment being built using GtkRadiant.

With a camera model and an interaction room, we then need programming code to simulate the behavior of the camera in the room. The Ubisim environment simulator program consists of modified dynamic link libraries (DLLs) for the Quake Arena III (Q3A) system. The source available for modification roughly approximates the “controller” part of a model-view-controller architecture [8]. Output from above graphical design tools initialize the ‘model’ that is then maintained by the Q3A engine along with the ‘view’. The ‘controller’ code concerns the interaction of entities in the model. This is the part that researchers need to change to simulate new physical interaction or wireless media effects.

In the case of our digital camera we follow the pattern set by the Q3A code for “weapons”: by following the weapon code we can get a handheld device in the hands of a user in the 3D world. The primary interaction the camera needs is to create an image of the 3D environment when the camera user presses the snapshot button. Again the Q3A code for a weapon can be modified, this time we recode the weapon firing trigger to call the rendering engine’s snapshot function. To correct for the field of view of the camera would require a bit more work, but we can delay that until we need that degree of accuracy in our simulation.

Estimating the time for this aspect of a simulation depends greatly on individual experience in C programming and on the kind of interaction one needs to program. Objects without visual representations, say “smartdust” objects which only need interaction-range programming can be added in a day by a programmer familiar with the system. A simulated “heads-up” display would also be straightforward, perhaps a few days. Something like smart jewelry would require some work with graphical code for the object to appear correct from all view points. More difficult examples are devices that modify the environment visually with dynamic content. For example, real-time display of video within the physical environment may not be feasible with the Quake Arena engine.

To summarize, we require three steps for setting up the Ubisim view:

- *creating the objects which will be present in the environment,*
- *creating the surrounding environment,*
- *adding and compiling physical interaction code into a dynamic link library*

Ubiwise includes a number of sample environments along with a library of devices and examples to show how this is done.

Once the simulator is running, the camera user would see a low-resolution image of the camera in the lower part of the UbiSim window (see Fig. 4); Jane can switch between the devices she possesses (this is just the Q3A weapon switching code). Jane can also interact with her environment like “shooting” a photo of one the frames or view scrolling contents of the wall. Multiple users could move in the shared place, see each other, and see the two picture frames on the wall of the store.



Fig. 4. Physical-environment view

2.3 Creating a Virtual Device for the Device View

While Ubisim simulates the environment and user interaction, Wise simulates the user interface, data storage, and network communications of the devices. The devices are specified through a combination of a device-description file in XML and Java code. Each device becomes a window within the Wise outer frame. The frame has menus with tools to aid the developer like device, source, and service browsers, record and playback of macros and help files.

In this scenario, we need to create a virtual digital camera in Wise. For the digital camera we use two views, one for the back and one for the top of the camera. On these views we have the controls like the buttons and switches. We also have browsers for images and for service selection. The first part of an example device-description file for the digital camera is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <device>
    <name>wiseCamera</name>
    <Classname>wise.wiseCamera</Classname>
    <ProtocolClassname>wise.wiseHttp
      </ProtocolClassname>
    <Title>wiseCamera : simulated HP camera</Title>
    <icon>./devices/wiseCamera/wiseCamera_back.jpg</icon>
    <MultiView>
      <name>wiseCameraMultiView</name>
```

```

<Classname>wise.wiseMultiView</Classname>
<View>
  <name>BackView</name>
  <Image>./devices/wiseCamera/wiseCamera_back.jpg</Image>
  <scale>1</scale>
  <Classname>wise.wiseDeviceView</Classname>
  <MultiBrowser>
    <name>wiseBackViewMultiPaneBrowser</name>
    <Classname>wise.wiseMultiPaneBrowser
      </Classname>
  <default>webbrowser</default>
  <Browser>
    <name>webbrowser</name>
    <Classname>wise.wiseWebBrowser</Classname>
    <type>wise.wiseWebBrowser</type>
    <shape>234,98,163,117</shape>
  </Browser>
  <Browser>
    <name>imagebrowser</name>
    <Classname>wise.wiseImageBrowser
      </Classname>
    <type>imagebrowser</type>
    <shape>234,98,163,117</shape>
  </Browser>
  </MultiBrowser>
</MultiControl>
<name>wiseBackViewMultiControl</name>
<Classname>wise.wiseMultiControl</Classname>
<Control>
  <name>wiseBackButton</name>
  <Classname>wise.wiseSimulatedButton
    </Classname>
  <description>Back</description>
  <shape>294,244,41,20</shape>
  <eventNumber>1</eventNumber>
</Control>

```

The description file follows a hierarchical tree with text strings at the leaf ends. The top level of the tree describes the device itself enclosed by the <Device> tags. Then each view of the device is described enclosed by the <View> tags. The XML elements enclosed in a <View> describe “components”. Each the components must have both a <name> tag to identify it and a <classname> to specify the Java class to instantiate. Wise walks down the tree and instantiates each component depth-first using Java’s reflection feature. Any nested XML elements are passed to components as name-value pairs, with the name being a concatenation of the XML element names along a branch and the value being the text at the leaf. This way component programmer is insulated from issues of configuration files and XML parsing. The component instances are stored in a dictionary that the programmer addresses by name to create inter-component connections at run time.

Now that we have given a generic description of the device file, let us return to the virtual digital camera example. The first step is to use the JPEG images to create device views. For this we edit the XML file for the camera to create a view named “BackView” containing an “image” element with a value equal to the JPEG (see <Image>./devices/wiseCamera/wiseCamera_back.jpg</Image> above). This would be enough to create simulated camera view. Two such views are shown in Fig. 5.

The next step is to add controls. Wise provides simulated button classes activated using the mouse and keyboard and derived the JButton classes in Java. The

placement of the controls may be specified as coordinates in the description. By placing the simulated buttons over the camera back image, the simulated buttons have a simple visual representation matching the eventual user interface of the device.

The buttons can trigger actions indirectly through an event mechanism. Wise follows a deliberately trivial event model. Any component can send events to any other component; the events are identified by integers. For example if a user clicks on the button, then the “simulated button” described above sends an event to its containing component with a value given by the `<eventnumber>` tag value. The event handler will typically be a single Java switch statement handling all event types. By avoiding abstractions we sacrifice expressive power and scalability, but we simplify the researcher’s task in creating the scenario.

In addition to input controls like buttons we add browsers for output and user interaction. We have a simple web browser extending the `JEditorPane` found in the Java SDK[9]. We have also used the `XSmiles`[10] browser that has XML rendering capabilities including Xforms[11], and SMIL[12] as well as support for small format devices. Our image browser displays images like a typical digital camera so a user can select an image for service. These browsers can be layered using a multibrowser control component.

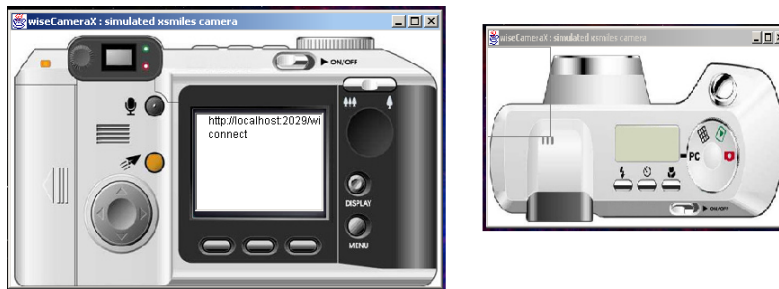


Fig. 5. Two different views of a digital camera

Up to this point we have simulated a camera with a browser but we have no way to get images or other content. Of course a camera should be able to photograph its environment as discussed in the preceding section. Our camera needs to download image and web-pages over a network as well. For this we would need to add a tag called `<ProtocolClassName>` to our device description file. Since we do other projects in nomadic computing based on Web protocols [13], we use an HTTP class here. Of course that makes our simulation especially simple as we can use the underlying desktop PC’s network infrastructure. Other choices would be easy to add as long as the latency and bandwidth requirements can be simulated over a PC network.

This completes the design of our wireless, web-browsing digital camera. Our scenario also requires a PDA and digital picture frames. These would be developed using the same steps above. Our experience with these procedures has been quite positive. As with the physical environment view, the first time one works with the tools to create a simulated device, several days will be needed. However,

modifications and additions can be made quickly. We have been able to completely redesign simulated devices in a day's time.

2.4 The Scenario at Runtime

We have described the environment and device views independently. They do function independently: development for one view can proceed in parallel with the other. This section describes the runtime behavior of the simulator with both views combined.

When the simulator is started, the device view is empty. The physical environment view starts up, loads the map created for the furniture store. This, in turn, loads a model for our user, Jane, her digital camera and PDA, and the digital picture frames on the wall of the store. As each device is loaded, *device creation* messages are sent from the physical environment view to the device view. The device type name sent in these messages is looked up in the device-description files and the corresponding Java classes are instantiated. The instance name in the environment view is also used in the device view. The two views are synchronized through environmental and device messages. If Jane switches from holding her PDA to holding her camera, a *device selection* message from the environmental to device views brings the digital camera into focus on the device view. If Jane downloads data from the digital frame, the results in *data update* events sent from the environment view to the device view, which then displays the image in the digital camera's "image browser".

Given the above setup, on running the simulator, Jane will see as the Wise view window something similar to Fig. 6.

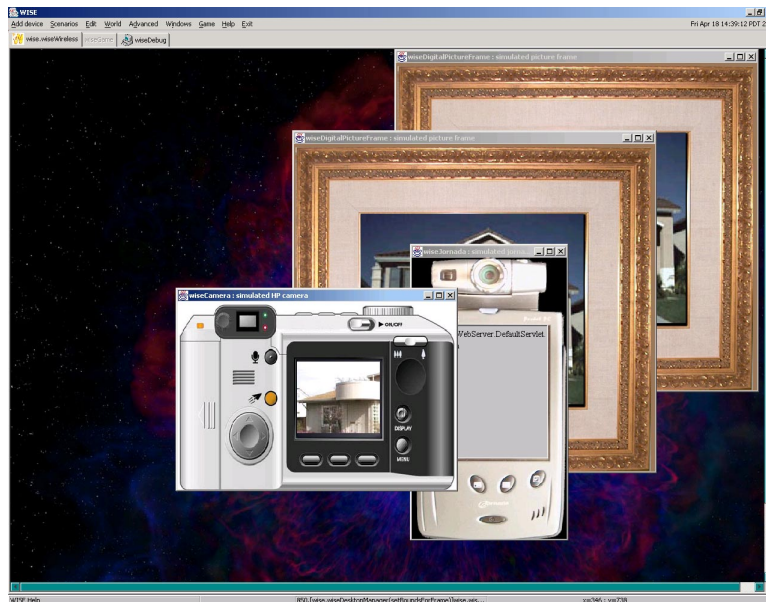


Fig. 6. Device design view for the example scenario.

One final piece to simulate is that of interaction with the web service. We described the “web browser” embedded into our digital camera. This browser can be used to provide list of services available to the camera. What different services are initially available to camera and through what mechanism they are discovered are separate research questions [14] that this simulator can be used to explore. Once the list of services, for example: a web photo-album service [15], publishing service, red-eye reduction web service etc., is available as a set of links on the “web browser” of the camera the user navigates using the virtual controls activated by the mouse and keyboard to a service and selects it. Selecting a link will send an event to the camera object, which has been instantiated. We then write a small piece of code to trigger the underlying protocol, in our case HTTP, to POST the image data to the remote web service. The result data will be displayed appropriately in the “web browser” thus completing our requirements for the scenario.

The Q3A engine has a notion of clients and a server built into it for gameplay. We can extend upon this to provide a simple framework for multiple users. Fig. 1 shows the client-server setup. All of the users and their views share a common physical environment server so they see the same environment and set of devices from different points of view. Different users will have different devices in their hands and consequently they can take different actions.

3.0 Applications

We have applied the simulator to two simple and one more complex project. The two simpler applications—a web-present digital picture frame and a conference room appliance—used the device simulator to explore development of services. The more complex project simulated a futuristic digital camera and services for it. We describe these projects here to illustrate some ways that a simulator can be an effective tool for ubiquitous computing research.

3.1 Web-Present Picture Frame

The digital picture frame in the example scenario in section 6 was prototyped earlier in our lab for another purpose. The idea was to provide a web server or “web presence” in each picture frame that accept image uploads. Since these digital frames now appear to be “on the web”, we can use web protocols to interact with them [2].

Our implementation followed the procedure described in Sec. 5. The only significant difference was the addition of a web server to the device. For this we use the nexus-httpd webserver[16], a small Java HTTP servlet engine. To add a webserver we can simply add a component to the device by adding a `<Server>` element in the device-description file and listing the classname of the servlet or any class which extends it. Between the `<Server>` tags, we can also specify the port number on which it should listen and any other specific information. *Wise* will instantiate the webserver class and pass the specific tags during instantiation.

For the purpose of demonstrating of the digital frame, we used a actual HP Jornada PDA fitted with a small digital camera in its CompactFlash slot (rather than the simulated digital camera in the example scenario of section 6.0). The PDA was then

made to inject its pictures into the simulated frames over HTTP. This illustrates mixing content from the real world with devices in our simulated world.

3.2 A Conference Room Appliance

Our second example created a single simulated device to operate with a set of laptops. The device sits on a conference room table like a telephone but it runs the Stanford IROS system [17] and a version of the sensor enhanced web browser from our lab [18]. Users bring laptops into the room and connect to the room system to integrate their applications with other users. Details of the system and its use model appear elsewhere [19]. The user interface design for the appliance used Ubiwise for prototype development. During early rounds, the images were used to discuss the physical user interface. Then the user interface was coded in the simulator. As in the example in 7.1, we mixed real and simulated devices, in this case using a simulated conference room appliance with real laptops.

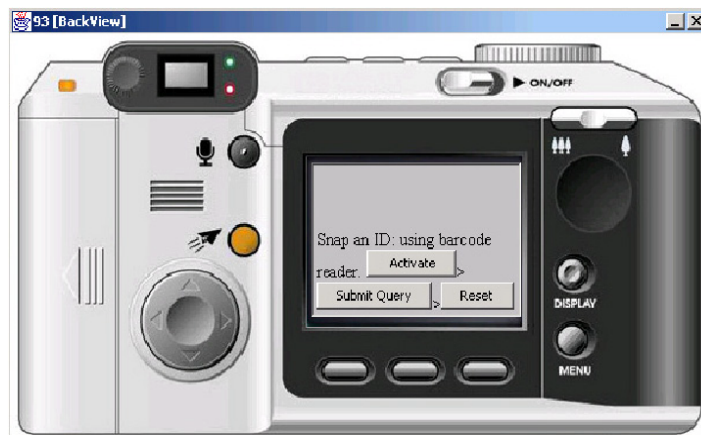


Fig. 7. Screenshot of the back of a simulated bar-code-reading camera equipped with a miniature web browser in the device view.

3.3 Sensor-Enhanced Web-Browser

UbiWise was used as a part of a project on service selection and service access for mobile sensor-enhanced clients using XForms[11]. This application simulated a nomadic computing scenario in which a user arrives in a room she has never visited before with her web-browser enhanced wireless camera equipped with bar-code conversion software. She uses that device:

1. to obtain links to the electronic services in the area,
2. to access services that work with the data (e.g. images) she can produce.

The links here are "physical hyperlinks": bar-codes placed by the entrance to the room (see Fig.7 and 8) are mapped to a web page with the "web/id" service [20]. While we don't know of a digital camera equipped with a tiny wireless web-browser and able to read bar-codes, these capabilities exist separately in other small devices. PDAs with

web-browsers and wireless connections are available. Commercially available PDAs and phones that can read bar-codes via a camera attachment [21]; such software saves the bulk and expense of a specialized scanner. What does not exist is a combined device configured to leverage the physical hyperlinks and the corresponding service configured to work for handheld devices. By using the simulator for this hardware we were able to design and develop a new service model, one that allows devices to upload sensor data to services without device or service specific code.

In the above scenario the sensors, device screens, and controls are simulated along with the XForms processing code. The web client and web service code are not simulated and use existing toolkits. This serves as a good example of how the simulator can be used to test new paradigms in ubiquitous computing.



Fig. 8 Screenshot of the physical environment showing the simulated camera about to sense a bar-code (physical-hyperlink) to find location-specific services.

4.0 Discussion

We have explored the potential of simulation as a tool for ubiquitous computing research. We have been able to meet most of our requirements for some of the problem space of ubiquitous computing. Some aspects, such as using the simulator for system and interoperability testing and for reproduction of experiments will have to wait for the emergence of ubicomp systems and experiments respectively.

The UbiWise simulator attempts to balance fidelity and simplicity in a simulator for ubiquitous computing. As with all simulators we must confront this fundamental design tradeoff. To have adequate “fidelity” implies that the devices created in the

simulator, the user interaction, and the simulated environment must be close enough to reality to validate ideas. Otherwise the simulator will be ineffective. Balancing against fidelity is our desire for “simplicity”: in the simulator it must take dramatically less time and human energy to create devices, protocols, environmental interaction scenarios than it would take with physical devices and environments. Fidelity opposes simplicity in simulations: making the simulator simpler costs fidelity and vice versa. We struck a balance between “fidelity” and “simplicity” in choosing to have two views for the simulator.

We believe that research in ubiquitous computing is only now reaching the point where a simulation tool can be effective. For single devices or for research focused on learning how users might work and live in a ubicomp environment, direct construction may be essential for progress. As we move forward a simulator can provide an alternative to explore other aspects of our problem space. The UbiWise simulator is intended to be a continuing work in progress and all of the new code is released[22] under the LGPL[23].

5.0 Related Work

We don't know of a general-purpose simulator directed at ubiquitous computing. Bylund and Espinoza [24] modified the Quake III Arena (Q3A) code to output the position co-ordinates of a player into the Context ToolKit [25]. They demonstrated the potential of this tool in the development of a location-based “virtual note” project called GeoNotes[26]. GeoNotes had a Java program reading the position co-ordinates and providing a user-interface for placing virtual notes. This work convinced us to extend our initial efforts [27] to a more complete simulator that provided both three and two-dimensional views, each view trying to solve different problems.

Two kinds of existing simulators inspired our work: computer-computer networking simulator like “ns2” [28] and simulators for specific devices often provided with software development kits for handheld devices. The former illustrates the power of a simulation: a great deal of new understanding has come from work with tools like ns2. The latter illustrates the value of even a simple model for the user interface while developing programs for small devices. Our tool is more about the spatial and environmental interaction of devices and users than either of these kinds of simulators.

Along a different axis of comparison we might consider the “wizard-of-oz” or story-board techniques used in HCI experiments as simulations that use paper and humans rather than computer systems for the simulated elements. We view these approaches as complimentary ones in that they bring the design of elements of a system forward to the point where issues of multiple device interaction and of device/service interaction can be examined.

6.0 Acknowledgements

We are grateful for the continued encouragement of Prof. Mary Baker, Stanford University. We thank Fredrik Espinoza and Markus Bylund of the Swedish Institute of Computer Science (SICS) for their encouragement.

References

1. Mark Weiser, "Hot Topics: Ubiquitous Computing" *IEEE Computer*, October 1993.
2. Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Iija Bedner, Debbie Caswell, Phillipe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, Celine Pering, John Schettino, Bill Serra, and M. Spasojevic. "People, Places, Things: Web Presence for the Real World", *MONET* Vol. 7, No. 5 (October 2002).
3. Andrew C. Huang, Benjamin C. Ling, John J. Barton, Armando Fox, "Making Computers Disappear: Appliance Data Services" *ACM SIGMOBILE Seventh Annual International Conference on Mobile Computing and Networking (Mobicom) 2001, Rome Italy*.
4. Id Software, "Quake III Arena", <http://www.idsoftware.com/>
5. See <http://www.adobe.com/products/photoshop/main.html>
6. Mete Ciragan "MilkShape 3D" <http://www.swissquake.ch/chumbalum-soft/>
7. Robert A. Duffy and others, "QERadiant" <http://www.qeradiant.com/>
8. Glenn E. Krasner and Stephen T. Pope, "A cookbook for using the model view controller user interface paradigm in Smalltalk-80". *Journal of Object-Orientated Programming*, 1(3):26-49, August/September 1988
9. Java 2 Platform SE v1.3.1: Class JEditorPane <http://java.sun.com/j2se/1.3/docs/api/javaw/swing/JEditorPane.html>
10. Xsmiles: <http://www.xsmiles.org>
11. Xforms: <http://www.w3.org/Markup/Forms/>
12. SMIL: Synchronized Multimedia, <http://www.w3.org/AudioVideo/>
13. Kindberg, T. and Barton, J., "A Web-Based Nomadic Computing System", *Computer Networks*, Elsevier, vol 35, no. 4, March 2001, and HP Labs Technical Report HPL-2000-110.
14. UpnP: Universal Plug and Play Forum. <http://www.upnp.org>
15. PhotoPoint: <http://www.photopoint.com>
16. Anders Kristensen, Nexus HTTPD http://www.hpl.hp.com/personal/John_Barton/ur/ubiwise/nexus/doc/index.html
17. Portability, Extensibility and Robustness in iROS. Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman and Armando Fox. *Proc. IEEE International Conference on Pervasive Computing and Communications (Percom 2003), Dallas-Fort Worth, TX. March 2003*
18. Sensor-Enhanced Web Clients: An XForms Approach. John Barton, Tim Kindberg, Hui Dai, Bodhi Priyantha, Fahd Al-bin-ali, To be published in the Proc. WWW2003
19. John. J. Barton, Tony Hsieh, Vikram Vijayaraghavan, Tomoto Shimizu, Armando Fox, The Meeting Machine: How Nomads Can Meet In Interactive Spaces.
20. Kindberg, T., "Implementing Physical Hyperlinks Using Ubiquitous Identifier Resolution", *Proceeding of 11th International World Wide Web Conference*, July 2002.
21. International Wireless, <http://www.mitigo.com/>
22. Ubiwise is available as on <http://ubiwise.handhelds.org> as three packages under projects "wise", "ubisim", and "nexus".
23. "LGPL Lesser GNU Public License", <http://www.gnu.org>
24. Markus Bylund and Fredrik Espinoza, "Testing and demonstrating context-aware services with Quake III Arena", *Communications of the ACM*, ACM Press, New York, NY, USA, 2002, Pages: 46 – 48

25. Anind K. Dey and Gregory D. Abowd "The Context Toolkit: Aiding the Development of Context-Aware Applications" In the *Workshop on Software Engineering for Wearable and Pervasive Computing*, Limerick, Ireland, June 6, 2000. See also <http://www.cc.gatech.edu/fce/contexttoolkit/>
26. Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E. and Bylund, M., "GeoNotes: Social and Navigational Aspects of Location-Based Information Systems", in Abowd, Brumitt & Shafer (eds.) *Ubicomp 2001: Ubiquitous Computing*, International Conference Atlanta, Georgia, September 30 - October 2, Berlin: Springer, 2001, p. 2-17.
27. Vikram Vijayraghavan and John J. Barton. "WISE - A Simulator Toolkit for Ubiquitous Computing Scenarios" UbiTools-'01 workshop
28. Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. "Advances in Network Simulation." *IEEE Computer*, 33 (5), pp. 59-67, May, 2000. See also other papers cited on The Network Simulator - ns-2 <http://www.isi.edu/nsnam/ns/>