WISE – A Simulator Toolkit for Ubiquitous Computing Scenarios

Vikram Vijayraghavan, John J. Barton
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-302
October 22nd , 2002*

E-mail: John_Barton@hpl.hp.com

ubiquitous, pervasive, nomadic, simulator, toolkit, Java

This paper presents WISE, a simulator toolkit developed for exploring the new generation of wireless devices, their interactions, and the wireless infrastructure and services designed for these devices. WISE stands for *Wireless Infrastructure Simulation Environment*. Our focus is on the layers of software and that lie above basic device function and networking protocols. Ultimately we hope WISE will develop into a simulator for pervasive or ubiquitous computing.

* Internal Accession Date Only Approved for External Publication
Presented at the UbiTools '01 Workshop on Application Models and Programming Tools for Ubiquitous Computing, UBICOMP 2001, 30 September 2001, Atlanta, GA

WISE – A Simulator Toolkit for Ubiquitous Computing Scenarios

Vikram Vijayraghavan and John J. Barton
Hewlett Packard Laboratories
Palo Alto, CA 94304
John_Barton@hpl.hp.com

This paper presents WISE, a simulator toolkit developed for exploring the new generation of wireless devices, their interactions, and the wireless infrastructure and services designed for these devices. WISE stands for *Wireless Infrastructure Simulation Environment*. Our focus is on the layers of software and that lie above basic device function and networking protocols. Ultimately we hope WISE will develop into a simulator for pervasive or ubiquitous computing.

Motivation

Many toolkits exist for the network layer or physical layer simulation of wireless devices [1][2]. These toolkits provide a means for comparing wireless protocols in a controlled and consistent fashion. This gives a scientific basis for judging the effectiveness of the protocols but equally important the simulators provide a way to demonstrate the protocols and their use models without full-scale development and deployment. WISE is an effort to provide similar benefits to the application layer for wireless systems.

WISE was also motivated by a vision of digital "appliances" interacting directly with Internet services [5]. We believe that more appliances will appear if such services were available, but without the appliances no convincing services can be developed. By developing a simulator for the client side we enable service creation and testing to move forward and further encourage the development of digital appliances.

Consider the problem of developing a wireless digital camera. In addition to issues of industrial design for the camera itself, developers will face issues of the user interface for connectivity and for imaging services as well as usage models for such a device. WISE can be used to develop several aspects in the design of a wireless camera – in deciding the most effective user interface for the camera, in deciding the kind of services provided to the wireless camera, in deciding the protocol(s) to be used by the digital camera, dealing with variable latency issues, dealing with regions of weak/ excellent connectivity, for examples. In addition many aspects of the design can be tested within a simulator and the features of such a device can be demonstrated. All of these benefits can be obtained at a fraction of the cost of a full prototype development.

Goals

WISE is to provide a simulator toolkit with the following properties:

- *Creation and usage of devices*. It must be easy to create a representation of the devices in the simulator. For example this can be an image of the device that looks like the device in question and has the functionality of the device relevant to wireless operations. Ideally, the experimenter with our toolkit can design his own classes using a GUI interface like DENIM/SILK [3] or specific software IDE's like Forte [4] and inject it into the simulator. However the simulator itself should provide minimal functionality in this area.
- *Multiple scenarios*. Devices might exist in different regions of varying connectivity, devices themselves might have different views (for example top, bottom, front etc), and devices might act differently depending on other devices being present (they may be context-sensitive devices). All these real world issues should be supported by the simulator toolkit.
- *Flexible User Interface and Form Factor*. Most of the wireless appliances have unique, small form factor displays and controls. This brings a whole different set of user interface problems. Some of these issues can be studied within a desktop simulator.
- *Data Manipulation* Each device might access data from the infrastructure, for example pulling content from a webserver. Each device might also have its own store of data, like photo in a digital camera.

Both of these require user manipulations and browsing environments that allow users to activate services on the data. For example in a digital camera, a web browser for selecting a storage service needs to work in conjunction with an image browser for choosing an image to send to that service.

- *Communication Protocols*. Wireless appliances have many communication protocols to choose from. Obviously are a must but the link layer access mechanisms are totally different for these wireless appliances. Access points, Bluetooth support, HTTP, and UPNP are examples of the kind of protocols WISE may need to support.
- *Ease of use*. The simulator should be simple enough for end-users of wireless devices to use in testing. Its programming model should allow experimenters to begin quickly and yet not be hindered from developing sophisticated device, scenario, or protocol enhancements.
- *Fidelity of the simulation*. Important issues in wireless device use must be simulated as well as the medium of a desktop computing environment allows. These include connectivity latency, bandwidth, and screen size, but they might also include other issues like battery life.
- *Openness.* The toolkit needs mechanisms to allow new implementations to be added.

The WISE model:

The WISE model presents a layered abstracted view of the environment to the end-user. Each of the layers in WISE can be extended for building new scenarios. The top layer is the World and the next layer includes the Devices. Below devices live layers for data collection and communication protocols.

The **World.** Devices interact in a 2 dimensional pane called a "World". They can interactwith (real, not simulated) Internet services, with other simulated devices using simple overlap collision detection or with characteristics of the physical world model. The simplest example is a World with full connectivity and no device-device interaction. In that world, device-service interaction can be studied. In our "wireless" World has 2D regions of connectivity and regions with no connectivity. As devices are moved in that World, they must contend with the degree of connection.

The **Devices** are represented as pictures of the physical user interface of the device, as it would appear to the end-user. They have basic controls implemented as mouse-clickable buttons. Multiple pictures of a device may be needed if it has controls on multiple sides or if it has lights or dials that change state during use. Typical devices will include a screen that may allow content to be displayed.

The **Data API** is a generic name for methods to access both the data present in the device and the data presented to the user but available on the network (e.g. web pages). The **Web Client** is the underlying layer of the Data API responsible for interaction with the outside world on the Internet via HTTP protocol. This interface sets up a "model" of a model/controller/view design to encourage reuse and design clarity.

The **Communication Protocol API** is responsible for simulating the intra-device and device-infrastructure communication. The **Protocol** layer below that is for supporting various protocols like TCP, UPNP. Currently we simply map the wireless device TCP/IP stack onto the desktop TCP/IP stack to simulate IP connectivity.

Connectivity. The Connectivity API is responsible maintaining the connectivity of the device with respect to its current location. The World stores information dynamically about the Devices in it and the Connectivity API is used to fetch this information and deduce as to what kind of connectivity the Device has currently. This is an example of interaction across layers in the WISE model as the Connectivity API is essentially per Device but interacts with the larger subset – the World**.**

Implementation:

The initial version of WISE has been implemented using Java[6]. It's a package of classes that provide a skeletal framework for simulation. In the current implementation the user select a world as a Java class name and launches WISE with that class name as an argument. Next WISE dynamically creates menus by

reading load Java classes by name. For example, a file called “devices.txt” lists all the devices. Uses pick from the menus to create a scenario.

We have implementations of a simple unstructured world and one with partial connectivity. For devices we have a camera, PDA, handheld scanner, and a mythical wrist-mounted multimodal device. The first two use commercially quality art work for the user interface background. The scanner uses an image that was, well, scanned in on a color scanner. The wrist device was hand-drawn with color pencil and scanned in.
A set of common classes for the communications and web-browsing elements of these devices allow rapid development and these classes are called through interfaces that allow alternative implementations to be added. Thus far we have created two kind of browsers – the WebBrowser and the ImageBrowser. We have one implementation of the protocol API to support HTTP Posts to ADS[5] services.

Below is a snapshot of WISE with a digital camera device and a handheld scanner represented using the toolkit. The picture frame is divided into two regions, one dark region comprised of waveforms representing the connected region and the other one (light background) representing the disconnected region. When the Camera is in the connected region, trying to access the infrastructure will result in a call to inConnectedRegion() which will return false if the range is not in reach. Appropriate warning or action must then be taken by the camera user interface. The camera includes a mini-web client in the view screen capable of displaying links to services for selection by the user.

Future Work and Extensions

WISE is barely past the concept stage. Once the framework of the simulator has been built one could think of adding functionality. Some ideas include:

- Multiple views of the device: Each device might have different views. For example, front and top panels of a digital camera have different functions and controls. In fact such a model can be generalized to a controller – view architecture where there are controls, views and services separately.
- Grouping of Devices – The next step would be to form logical groups of devices and interaction between these groups e.g. some form of multicast device operation.
- Context-sensitive devices: Each device might have to act differently in the presence of another device. Hence there should be some context-sensitivity built in those devices.
- Service models & security: The simulator not only allows looking at the device side but also at the service side. We are currently exploring deploying a directory based service portal and also the implications in security of communication if we take such a path.
- Multiple access points: Scenarios with multiple access points and more involved topologies are to be considered.
- Multi-modal devices and multi-modal scenarios: how device features or devices can be coordinated in their interaction with services.
- Scripted scenarios: fixed scripts would allow testing and comparison of protocols.

Conclusion

We have started a simulator toolkit for pervasive computing in the realm of device and infrastructure design and interaction. It seems to be that there is little work in this direction and we hope to make further contributions here. We intend to use the simulator in a real world scenario to test and provide feedback for the design of next generation wireless digital cameras and other devices connected to the Appliance Data Services project [5]. Our plan is to engage other researchers in the joint development of the toolkit and its application through open-source distribution. The distribution should be in place for the workshop and we will be able to demonstrate the toolkit as well.

Acknowledgements

The digital camera simulator code of Andreas Ziedler aided our initial efforts to develop the simulator. Amy Battles and Andy Goris of Hewlett Packard provided the images of the HP C618 digital camera.

References

1. Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. **A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols**. In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, ACM, Dallas, TX, October 1998. See also http://www.monarch.cs.cmu.edu/ cmu-ns.html
2. BlueHoc simulator - http://oss.software.ibm.com/bluehoc/
3. Forte Visual IDE for Java - http://www.sun.com
4. James A. Landay and Brad A. Myers, "Sketching Interfaces: Toward More Human Interface Design." In IEEE Computer, 34(3), March 2001, pp. 56-64. See especially the CrossWeave project– James Lin and James Landay http://guir.berkeley.edu/projects/denim/research/vl.shtml
5. "Making Computers Disapper: Appliance Data Services" Andrew C. Huang, Benjamin C. Ling, John J. Barton, Armando Fox, ACM SIGMOBILE Seventh Annual International Conference on Mobile Computing and Networking (Mobicom) 2001, Rome Italy. See also http:// swig.stanford.edu
6. Java ™ - Sun MicroSystems Inc. – http://java.sun.com