Education & Training

Editor: Scott F. Midkiff ■ Virginia Tech ■ midkiff@vt.edu

What I Did on My Fall Vacation—A Pervasive Computing Class

Larry Rudolph

EDITOR'S INTRODUCTION

In this issue, Larry Rudolph shares his experience in developing and teaching classes in pervasive computing at MIT and for industry. The course drew on many technologies from MIT's Project Oxygen. He describes goals for the course, the syllabus, problem sets, course components, and final projects. Please let me know your comments and suggestions for future columns. —*Scott Midkiff*

This column describes a project-based, hands-on pervasive computing course offered at MIT during the fall 2001 semester. Later, I helped distill the course into an intensive one-week experience that MIT offered in spring 2002 and once again in winter 2003. Much has been learned from these three instantiations.

The course shares intimate ties with MIT's Project Oxygen, a pervasive, human-centric computing effort (for more information about Project Oxygen and associated technologies, see www.oxygen.lcs.mit.edu and the August 1999 *Scientific American*). Project Oxygen combines research (MIT's Artificial Intelligence Laboratory and Laboratory for Computer Science) and industrial (Acer, Delta, Hewlett-Packard, Nokia, NTT, and Philips) involvement.

There were several somewhat unconventional reasons motivating the course's development. First and foremost, I believe that academia's primary product is ideas and that a course best conveys ideas, especially new ones. When several new ideas are presented together in one course, new unifying themes frequently emerge. But I didn't want the students to have all the fun. I too want to learn about and master the technology behind my colleagues' cool, neat, new toys, and teaching is the best way to learn. Moreover, I wanted to recreate the crossdisciplinary and informal learning of the days before students sat isolated in their dormitory rooms working on their PCs. I wanted to teach a course that involves students working closely together and sharing their experiences with one another. Finally, I wanted a course that cuts across numerous subdisciplines, exposes students to a different view of computer science, and provides an opportunity for true student innovation.

The first offering of the course, referred to as the Boston version, met many of these goals. Some topics were too hard and some technologies too immature. It was a challenge to avoid massive student frustration. Fortunately, by the third offering, referred to as the Taiwan version, many of the shortcomings were addressed and the course ran fairly smoothly.

COURSE LOGISTICS

The course is mostly hands-on and project oriented. We outfitted each student with a Compaq iPAQ handheld computer, an IEEE 802.11b wireless LAN card, a 128-Mbyte memory card, an iPAQ serial cable, and some MIT-exclusive equipment—a BackPAQ (see Figure 1) and a Cricket *listener*, which listens for radio frequency signals and ultrasonic pulses. The BackPaq attaches to an iPAQ, providing two PCMCIA (Personal Computer Memory Card International Association) slots, a video camera, and an accelerometer. Cricket is an indoor location-tracking system.

Despite its hands-on focus, the course still included lectures, tutorials, programming assignments, and a final project. Lectures delved into a particular technology, exposing students to advanced material normally found in specialized graduate courses. In the Boston version, the lectures were the usual hour-long seminar usually given to a general audience. They were generally presented by a faculty member involved in the research. We learned

QUICK FACTS

Course: 6.964—Pervasive Computing: Projects in Project Oxygen
Unit: Laboratory for Computer Science
Institution: Massachusetts Institute of Technology
Instructor: Larry Rudolph
Level: Undergraduate, graduate, and industrial
URL: http://org.lcs.mit.edu/class.html

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/03/$17.00 © 2003 IEEE

that it was better to slightly narrow the focus but delve more deeply. Moreover, there was a lack of continuity and lack of leverage with previous material. The Taiwan version had fewer lecturers and the material became more integrated.

We used tutorials to explain the actual details of programming and related tools. Because each technology had its own unique installation, programming language, and configuration scripts— there was a steep learning curve. The tutorials mitigated the pain by providing sample code and by going through the installation, compilation, and execution of simple examples.

We initially presented the technologies in isolation, with integration happening at a later stage. At some point, the students realized an ad hoc approach's drawbacks. The large number of programming languages—Perl CGI, C, Java, Python, XML, and specialized scripting languages—challenged everyone. We thought we might have gone a bit overboard and have recently started doing everything in Python and Java, but integration is still hard. There still isn't a mature integrative middleware system for pervasive computing that addresses most of the challenges, so we let students appreciate the need for such a system, rather than learn to live with the shortcomings of an existing one.

Topics

When setting the course's content, we asked, Should it be broad, covering the best technologies in the world, or narrow, being restricted to technologies invented at MIT? Given many technologies' immaturity, fragility, continual evolution, and lack of documentation, we decided to look inward, making the most use of local resources. Of the technologies being investigated at MIT, we chose the most promising ones, provided that the developers would respond to desperate calls for help when something went wrong at the last moment. Unfortunately, most universities do not have this luxury, but numerous research groups around the world would welcome the opportunity to share their new technology in the classroom.

Table 1 lists the topics generally covered in the MIT and Taiwan courses. While the students worked on their final projects, we had time to cover pervasive computing research going on at other institutions.

Assignments

Programming assignments (*problem sets* in MIT lingo) and a final project were a major focus of the course. All the assignments revolved around a theme. The problem set theme for the Boston version was mathematical equations—specifically, how to enter and edit equations. We ran Mathematica on a server, and students could interact with it via the handheld, either in formatting or solving equations The theme for the Taiwan version was an instant messenger system, built on Jabber, a Python-based, public-domain IM system. Both versions had pretty much the same set of assignments, but relevant to their theme:

Figure 1. The commercial iPAQ with BackPAQ, jointly developed by Compaq Research Labs and MIT. The BackPAQ contains a video camera, an accelerometer, a microphone and headphone connectors, and two PC card slots.

- Writing a graphical user interface to specify equations or to log in, add a buddy, and initiate a chat session.

TABLE 1
Course topics.

Week	Topic
1	Overview: iPAQ and other handhelds Linux on iPAQ
2	Oxygen Project overview GUI tool: Glade, cross-compilation, Java on handheld
3	Speech recognition (Galaxy, SpeechBuilder) iPAQ audio server and audio streaming
4	Naming, sockets, servers and clients, self-certifying file system Intentional Naming System
5	Cricket location detection (Indoor GPS)
6	Cricket details Grid self-configuring ad hoc networking
7	Security and group keys Scripting details, intelligent appliances
8	Handwriting recognition Object detection from strokes
9	Face recognition, gestures Vision processing (head tracking)
10	Integrative middleware: Metaglue, agents
11	Final project descriptions
12	Other pervasive computing systems (I)
13	Other pervasive computing systems (II)
14	Project presentations

EDUCATION & TRAINING

- Converting the graphical interface to a speech interface.
- Extending the system to include handwritten equations on the iPAQ or transmission of basic objects and shapes drawn on the iPaq screen.
- Using the Cricket listener and beacons to specify equations by walking around the lounge or chatting with anyone that happens to be in a particular physical place, such as near the coffee machine.
- Integrating intentional naming and security into the system.
- Using the Metaglue system to control physical devices via an X-10 interface. Failing to find a solution to a mathematical equation would cause the coffee machine to turn on or you could send an instant message to the coffee machine.

COURSE COMPONENTS

Here, I give some of my thoughts regarding the course's basic components.

The handheld

We chose to run Linux on the iPAQ, thereby providing a full workstation's functionality and the use of standard software tools. Porting Linux itself isn't trivial, although it's been getting much easier, with help from people contributing to the open source software globally. Jamey Hicks of Hewlett-Packard's Cambridge Research Labs (CRL) greatly and generously helped with our port. It was a win-win situation, because the students tickled many bugs that Jamey would quickly fix.

A handheld GUI

We used a simple Python menu template, the GTK+ toolkit, and Glade for touch screen input. The menu template is built into the window manager, and we used files in a default directory to build the main menu. This let students quickly add simple shell commands by placing files into the default directory. Each file specifies the name and position in the pull-down menu hierarchy as well as the command to be executed. The GTK++ toolkit lets you build buttons, menus, lists, and so on. Some advanced students figured out how to install Java and did this via Java's AWT (Abstract Window Toolkit) library.

Speech recognition

My understanding of speech recognition systems comes from commercial recognition systems, such as IBM's ViaVoice and MIT's Spoken Language System's Galaxy System and its simpler SpeechBuilder development tool. Although I'm not familiar with all systems and approaches, I am impressed with Galaxy. With it, you don't need to be an expert to prototype a small system. Some students did really interesting things although they'd never taken a course in speech processing.

Students extended their GUIs to be speech enabled. Although building a speech-enabled interface is generally difficult, it's easy to do using SpeechBuilder. The Galaxy system uses domain-specific recognition and a grammar defines the domain. Writing the domain grammar was surprisingly trivial given a GUI. A true speech recognition system needs to handle all manner of talking. One way to limit this complexity is to focus on a specific individual's speaking patterns. For example, after just a few iterations, I can capture most of the ways that I talk. This is wonderful for fast prototyping but not for real development.

Face recognition

Face recognition consists of two phases, *training* and *recognition*. We used the imager on the BackPAQ to collect images for training and the same imager for recognition. This reduced the amount of training. However, recognition is still not great, so to improve robustness, we combined face recognition with easy-to-use identification mechanisms, such as voice identification. So, a student might pick up a handheld, point it at his face and say, "Hello, I am Larry Rudolph." The face recognition system produced a list of possible candidates along with their confidence scores. The speech recognition system did the same. Combining the two gave very good results. This exercise demonstrates the usefulness of mixed identification approaches and gave the students a flavor of multimodal sensor fusion, a topic that the course should really cover directly.

Access control

Security systems aren't exciting to watch, and it's hard to know if they actually work, but they're too important to overlook. The *s*elf-certifying *f*ile *s*ystem (SFS), part of the iPAQ distribution, works at a low enough level that users simply trust it. It's built using the secure sockets layer with fairly mature cryptographic schemes. This lets you choose the level of authorization, certification, and content integrity. You can use lighter-weight systems when security is less important.

This class focuses on a system that lets you specify access control on the group level. You can define a group and provide access to a group's resources. If you want to revoke access privileges, you simply remove that individual from the group, without having to modify the resource access lists to which that group has access.

Pen input

Personally, I would much rather have a small device that could project onto a flat surface and observe how I interact with that surface than a touch screen. A student in the course took a step in this direction. With the BackPAQ's camera pointed at a notebook, you write using an ordinary pen or pencil. The handheld tracks the differences between frames and extracts strokes, as if they were drawn on a screen. A next step might be to capture notes taken during a lecture and correlate them with slides or other activities in the room.

In an exercise in using the shape recognition toolkit, students must convert a shape to xfig format once it's recognized. A slightly more involved exercise would be to build a low-bandwidth, shared whiteboard application.

You'd map each iPAQ to a part of a large shared space. The shapes that you draw would get placed in the global shared space. If this shared space was a large wall display—perhaps generated by multiple projectors—and if each iPAQ had a location system, your iPAQ would get mapped to part of the shared space in front of where you're physically standing.

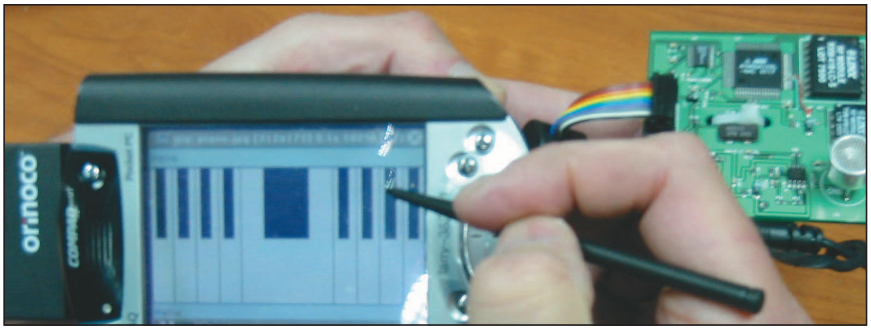

Figure 2. Keyboard from the Music on the Go project.

The network

The course only briefly touched networking, although in the future I would dedicate more time to this topic. With so many wireless iPAQ's and laptops in one small space, everyone saw the limits of 802.11b. Our system administrators thought one base station would suffice, but they were wrong. So, we missed an opportunity to really bring home the need for congestion and other network management techniques.

We explored new types of ad hoc network technology by exercising MIT's Grid network. In the Grid network, you don't need a base station because each handheld can forward packets. As students wandered around the building, the dynamic network connectivity and topology were projected on a wall.

Location management

The underlying hardware technology for location management is rapidly changing. To date, no one seems to understand the practical trade-offs well. In this course, we concentrate on technology that provides both identification and location. In the MIT Cricket location system, beacons are placed in the ceilings and walls. The beacons broadcast their names and other signals using both ultrasound and radio. A cricket listener attached to the serial port on the iPAQ could tell the name and distance from each beacon. The iPAQ used a table of beacon locations to compute its location. Unfortunately, the lack of floating-point support on the iPAQ made this an expensive operation. Nevertheless, this was the most exciting part of the course.

Integration and Metaglue

You can consider a pervasive computing application as a collection of interacting things, or nodes. The MIT Metaglue system is an approach for integrating interacting components. Building an interesting infrastructure to find and connect components is actually fairly easy, but developing all the agents is much harder. Writing applications mostly involves composing agents rather than writing new ones from scratch. Metaglue already comprises more than a hundred agents. You can now develop new agents fairly easily because the base is solid. So far, the students' experiences seem to support this claim, as it was very easy for them to control a wide range of physical devices.

PROJECTS

As in many courses, the real learning comes by doing—in this course, the final project. I encouraged students to work in groups and set simple ground rules. Each project had to use several of the technologies covered in class, be a game that involves several humans, and be fun. Because students could use any technology, you might consider the projects a free-market evaluation of the technologies. Indeed, in the future, I hope to inundate students with technologies and see which ones they choose.

The projects explored the intersection of virtual and physical worlds. Surprisingly, although the students enjoyed the speech-recognition problem set, it played only a marginal role. Everyone used location, but the games were all fragile, matching my experience with demonstrations of pervasive computing technology. I describe some of the more interesting projects here.

Boggle

Students turned this well-known word game into a contact sport. They divided a lounge area into a 4×4 grid, and a projector displayed the letters in each grid location. Players moved around the grid and shook the iPAQ to grab a letter. A tap on the screen ended the word. However, players could only grab a letter if the grid cell was unoccupied.

Competitive Fishing

In this game, the students turned the lounge into the open sea. They projected on the wall a view of fish swimming in the sea. Each student had an iPAQ with a Cricket listener and joined the game by asking, via a speech interface, "Can I play?" That triggered the placement of a ship in the ocean at the location corresponding to the iPAQ's location. The iPAQ showed the fish under the ship, which a player could catch by using the touch screen. Moving around the lounge caused the ship to move around the sea and, hopefully, find better fishing spots.

Music on the Go

The students configured iPAQs to play a variety of instruments, such as a drum, bongo, tambourine, or keyboard (see Figure 2). Shaking the iPAQ struck a drum, or touching the screen played a piano key. The type of instrument depended on the location. So, the musical score specifies not only the note but also the location.

EDUCATION & TRAINING

next issue

Wearable and Ubiquitous Computing at Virginia Tech

Doom

One of the most fun games was Doom, implemented only for a single player. The students projected the game on the lounge wall, but rather than using a mouse or keyboard to move through the Doom world, the player moved around the lounge (see Figure 3). The Cricket listener would send changes in the x, y location to the server. Because the lounge had obstacles (chairs, tables, walls) where the Doom world might have none, the game was particularly challenging. When a player ran out of space, to move forward in the Doom world, the player pressed a button to disconnect from the Doom server, took a few steps backwards, released the button, and walked forward. Navigating both the real and Doom worlds was confusing but fun, and we learned a lot about performance and user interaction.

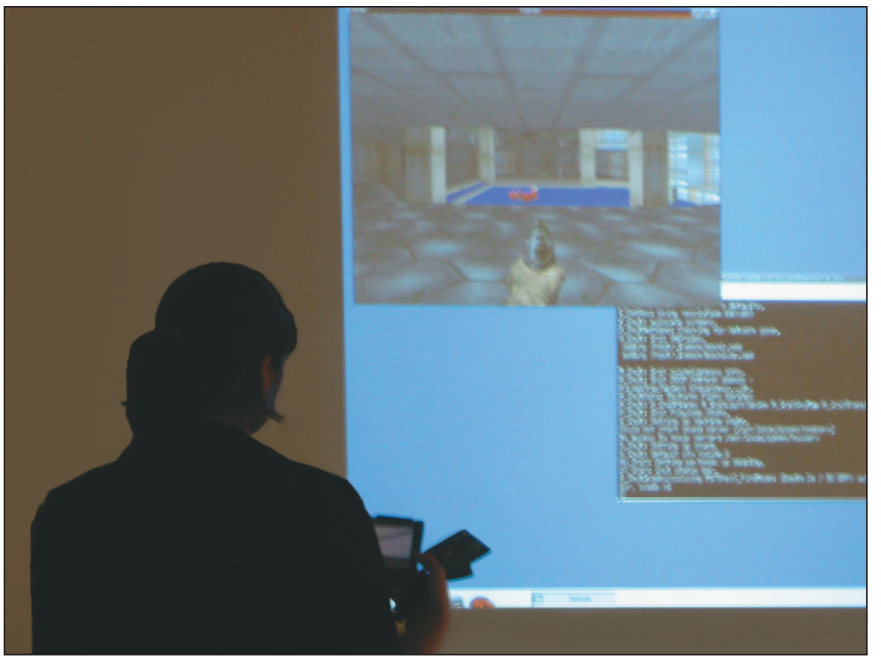

Figure 3. Student playing Doom; as she moves around the physical room, she also moves around the virtual Doom world.

Both students and teachers can learn much from a pervasive computing course. Such a class requires a lot of support, but every research group involved has been thankful for the opportunity to disseminate their technology, especially to a group of friendly users. Clearly, we'll need many more iterations before the course content becomes fully coherent.

We need a much larger emphasis on *defensive programming*. Too many things went wrong (and many more could have). Students quickly developed superstitions, such as to never use beacon 13 or always blow the dust off the listener. When weak batteries or poor network connections became evident, rather than using fresh batteries or switching network cards, we should have spun the symptoms back into the software. For example, a beacon with weak batteries would have its signal detected less frequently than other beacons. You could easily insert monitor code and inform the user that batteries appear weak. Similarly, you could display a message to inform the user when the detector serial cable comes loose. Many students ran into the same problems and had similarly frustrating experiences trying to debug their software when the hardware was flaky. These pioneers can save their successors much frustration.

In hindsight, the course was clearly about *disambiguation technologies*. That is, most subsystems aimed to assign the most reasonable meaning to ambiguous input. Perhaps this will always be the case in human interaction with machines, especially when you desire a level of discourse. Perhaps, paraphrasing Marc Snir of University of Illinois, programs are the contract between humans and machines, and just as we'll always need lawyers, we'll always need programmers—especially for pervasive, human-centric computing. P

ACKNOWLEDGMENTS

MIT's industrial partners (Acer, Delta, Hewlett-Packard, NTT, Nokia, and Philips) and DARPA, through Office of Naval Research contract number N66001-99-2-891702, partly support this work. Numerous others supported the courses, helping with specific course components and assisting with equipment. I specifically thank Jamey Hicks (HP CRL) and Greg Shomo (Techsquare) for iPaq and Linux technical support, and Anant Agrawal, Todd Amicon, Sonia Garg, Ken Steele, Kevin Quigley, Jason Waterman, and Eugene Weinstein for help with the courses.

Larry Rudolph is principal research scientist at the Massachusetts Institute of Technology Laboratory for Computer Science, head of the Oxygen Research Group, a member of the computational structures group, and a cofaculty member of the New England Complex Systems Institute. His research interests include parallel processing and complex systems. He has a PhD in computer science from the Courant Institute at NYU. He is a member of the ACM and IEEE. Contact him at MIT LCS, 200 Technology Sq., Cambridge, MA 02139; rudolph@lcs.mit.edu; http://csg.lcs.mit.edu/~rudolph.