Ambient Interaction Framework – Software Infrastructure for the Rapid Development of Pervasive Computing Environments

Peter Sutton, Margot Brereton, Clint Heyer and Ian MacColl

School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane QLD 4072 Australia

{p.sutton,margot,ianm}@itee.uq.edu.au, clint@thestaticvoid.net

Abstract

The Ambience project aims to develop and implement ambient computing – pervasive devices that blend naturally into the normal interactions and physical space of human work practice. Towards this end, the Ambient Interaction Framework (AIF) has been developed. The AIF is a software toolkit to facilitate the rapid development of pervasive computing environments through the integration of physical devices. The AIF supports various input and output devices, allowing them to work together in a cohesive manner. The extensible framework is supported by a Java API and relies on the decoupled nature of the communication provided by the Elvin content-based messaging service. The AIF eases the task of constructing new pervasive computing environments by encapsulating the data input and output capabilities of physical devices and automating the production and receipt of Elvin notifications. The encapsulated devices can then be integrated into new environments by writing a minimal amount of application-specific Java code. The paper also describes a proof-of-concept implementation - an Ambient Café.

1 Introduction

Humans discover and understand their world through visual, gestural, tactile and conversational interactions. In contrast, the conduit between the physical and virtual worlds is typically a Graphical User Interface (GUI) using keyboard, mouse and windows-based displays. Recent palm-based devices replace the keyboard and mouse with a stylus but maintain the interactional paradigm. The user of such an interface must typically stand or sit still, use fine motor skills and focus on a narrow range of pixels in order to drive the interface, shutting out the context in which she is working, which usually means suspending conversation, group interaction, natural movement, attention to the state of the surrounding world, and productive thought. Such interfaces do not take advantage of most of the sophisticated motor skills people have developed over thousands of years to understand and manipulate physical devices in physical space.

Miniaturisation, fast processors and wireless technology allow us to postulate seamless ways of blending the information infrastructure with social interaction in physical space to create information environments. This insight underpinned Weiser's (1993) vision of Ubiquitous Computing in which computing devices would be widely distributed but invisible, operating in the background to support natural human interaction, rather than dominating and demanding human attention. In other words, one would talk, gesture and interact naturally, and the information environment would respond appropriately, rather than the present situation in which one has to attend to the information environment explicitly to have it respond. For example, in Weiser's vision, when you walk into a room an ad-hoc network could be formed connecting devices on your person to a variety of sensors, actuators, input and output devices so that you can accomplish the task at hand with local relevant resources. Through multi-modal interface systems you will be able to interact with the information environment in natural ways using common skills of speech, gesture, glance, movement and manipulation.

This grand vision of ambient computing has largely remained unrealized beyond the research laboratory. It is our belief that the underlying difficulty, in designing new interaction paradigms, is not due to technical challenges alone. Ambient computing involves a "very difficult integration of human factors, computer science, engineering, and social sciences" (Weiser, 1993). Ambient computing embraces "wicked" problems (Rittel and Webber, 1973) which are only understood by attempting to design solutions. These solutions and their effectiveness in turn help us to understand the nature of the problem.

Because of the wicked nature of the problem, work is proceeding by populating the design space with examples. Several researchers have embraced the design of innovative interfaces and ambient computing (Allport, Rennison and Strausfeld 1995, Ishii et al 1998, Streitz 1999, Weiser 1993, Wexelblat 1995).

The Ambience project[1] approaches the problem by drawing upon methodologies used to understand human activity that have been developed in the social sciences, anthropology and various fields of design. We take an iterative approach of observation, analysis, envisioning, design, deployment and evaluation. Through this approach, researchers from different disciplines develop a dialogue and move closer to understanding a complex problem space. This paper concentrates on the design and deployment aspects of ambient environments - other aspects of the Ambience project and associated methodologies are beyond the scope of this paper.

The Ambient Interaction Framework (AIF) has been developed to simplify the design and deployment of prototype ambient computing environments by providing a common infrastructure on which these environments

[1] The Ambience Project is supported by the Cooperative Research Centre for Enterprise Distributed Systems Technology (DSTC).

can be constructed. The AIF supports various input and output devices, allowing them to work together in a cohesive manner. The extensible framework is supported by a Java API and is based on the Elvin content-based messaging service.

The remainder of this paper is organised as follows. Section 2 provides more information about the Ambience project. Section 3 describes some related work in the development of infrastructure for ambient or pervasive computing. Section 4 describes the Elvin content-based messaging service on which the AIF is based. The AIF is described in detail in Section 5, followed in Section 6 by a description of a proof-of-concept implementation. Some conclusions are drawn in Section 7.

2 The Ambience Project

As noted above, the Ambience Project takes an iterative approach to the problem of creating ambient interaction environments. This iterative approach is depicted in Figure 1.

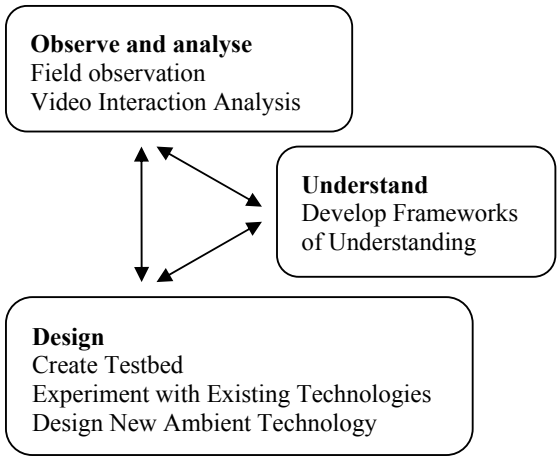

Figure 1: Ambience Project Methodology

By observing people doing tasks in their normal environment, it is possible to describe the actions they perform, the kinds of information they utilise and the potential for technology to assist with the task. By inserting technology into the activity and observing, it is possible to understand how the technology is used and how it changes the activity.

The Ambience project aims to:

1. *create a community of people working on the design of interaction technologies* that transgresses the borders between 'digital', 'physical' and 'social'. This community draws upon specialists from software, anthropology, cognitive science, social science, interaction design, industrial design, mechanical design, electronic design and architecture;
2. *develop a theoretical framework that characterises possible human interactions in physical space, virtual space and information space*; and
3. *create an Ambient Environment testbed* to explore integration of the physical environment with the information infrastructure in order to provide integrated ambient information support.

The Ambience Environment testbed is described further below.

2.1 The Ambience Testbed

The Ambience testbed is intended to allow us to easily couple sensors, actuators, displays and other devices in human-physical space with a wired and wireless networked hardware and software infrastructure.

In general, independent ambient devices and interaction technologies are ineffective. Value is derived by networking these devices, enabling information to be shared among the devices and the existing computing infrastructure such as servers and desktop PCs.

The testbed is intended to experiment with:

- integration of the traditional computer desktop with emerging human-computer interaction technologies e.g. speech recognition, eye movement, and gesture analysis, location tracking, 3D and virtual reality, personal digital assistants and visualisation;
- low-cost provision of presence and awareness information through audio, video, event and communications services (to support user presence and awareness in distributed workgroups through a variety of telecommunications and computing services);
- readily available high-cost, high-quality audio and video environments for short-term, high-intensity direct collaboration, with provision of quality-of-service over heterogeneous networks;
- integration of devices ranging from personal, low-bandwidth devices such as Personal Digital Assistants, mobile phones and highly portable low-profile or specialised computing and communications devices through to group-oriented, high-bandwidth devices such as smart whiteboards, sensor-equipped rooms and furniture, and Virtual Reality immersive environments;
- integration of domain specific devices designed during the project to support domain specific needs; and
- software technologies that support device-appropriate interaction abstractions while also providing interactive access to networked systems independent of the interaction modality.

As a step towards the development of this testbed, we have developed the Ambient Interaction Framework by building on the Elvin content-based messaging service. Before describing the AIF and Elvin in detail, we look at other work in developing ubiquitous computing infrastructure.

3 Infrastructure for Pervasive Computing

Much of the development work in the field of ubiquitous computing has been of specific devices and applications. There has been limited work on *general purpose* architectures, infrastructure and frameworks. Most of this limited work has been at the network level, e.g., the development of communication protocols and technologies such as Jini (Sun Microsystems 2001). In

the area of ubiquitous awareness, Kantor and Redmiles (2001) have recently recognised the need for a software engineering framework for providing user awareness. Their solution, named CASS (Cross Application Subscription Service), enables users to better control the number, type and manner of awareness notifications received from disparate information sources.

Our work, is aimed at providing a *lower-level ubiquitous computing infrastructure*. Consequent with our belief that interaction must move away from the keyboard and screen to be truly ubiquitous, our Ambient Interaction Framework provides interoperable device-level infrastructure. This enables prototype ubiquitous computing environments to be rapidly constructed from collections of physical devices which have been encapsulated into the AIF. The most closely related work are the "Phidgets" developed by Greenberg and Fitchett (2001) – though the emphasis in that work is on the hardware components themselves. The emphasis here is on the interface software – both the interface to the hardware and the interface to other components within the pervasive computing environment.

An important feature of the AIF is its reliance on decoupled communication when communicating with other components of the system, that is, the sender of information is unaware of how many and who the recipients are, and the recipients are unaware of the identity of the senders. Decoupled communication is described further in section 3.1 below and the Elvin content-based messaging system on which the AIF is based is described in section 4.

3.1 Decoupled Communication

Decoupled communication can take many forms and names, including publish-subscribe, store-and-forward, event notification, message queuing, subject-based routing and content-based messaging. Sometimes these terms are used interchangeably and there is indeed overlap between many of the concepts. In this work we will classify decoupled communication as one of two types: message queuing or subject/content based messaging.

Message Queuing. Message queuing is essentially a form of directed communication that doesn't require a connection. Messages are usually directed to a particular destination and are queued on both the disconnected device and a server, so that when the device is reconnected to the network, messages are transferred in both directions.

Subject/Content Based Messaging. Subject or content based messaging[2] decouples the producers from the consumers. Producers publish messages but are unaware of the destination. Consumers subscribe to particular groups of messages and one-to-many messaging is possible (i.e. a message produced by a source may be passed to many destinations).

There are significant differences between subject and content based messaging. Subject-based addressing allows consumers to subscribe only on the basis of message header information (or meta-data), usually the subject or channel or some combination of meta-data items[3]. This form of communication lends itself to multicast propagation with multicast groups corresponding to particular channels. A limited amount of coupling remains between producers and consumers - they are coupled on the basis of the channel name (Segall 2000).

Content-based addressing, on the other hand, allows the consumer to subscribe based on any aspect of the content of a message - not just the subject or other header. This provides greater flexibility to users and application developers as there is less coupling between producers and consumers, for example, consumers aren't forced to take a complete channel feed, they can selectively filter appropriate messages at the router (or server).

Content-based schemes have neither restrictions on the visibility of messages nor restrictions on what elements of a message can be used for selection. The major distinction of content-based routers is that message routing is determined by the consumers of the information rather than the producers (Segall 2000).

There are many systems available which provide decoupled communication. A representative selection of these is examined in this section and Elvin is described in Section 4.

TIB/Rendezvous. TIBCO's TIB/Rendezvous product (http://www.rv.tibco.com) is an established messaging middleware with many customer installations worldwide. TIB/Rendezvous decouples producers and consumers by using publish/subscribe; however, it uses subject-based addressing and does not provide full addressability of content.

Siena. Siena (Scalable Internet Event Notification Architecture) (Carzaniga, Rosenblum and Wolf 2001) is an example of wide-area event notification content-based routing. Siena allows subscriptions to address all fields of notifications. The emphasis in this work is on scalability - supporting content-based messaging over a wide area network.

OpenQueue. OpenQueue (http://openqueue.sourceforge.net) is an open source protocol for publish/subscribe message queuing. While connected to a server, a subscribing client receives published messages in real time. When a client reconnects after an absence, the server sends all messages queued for that clients while it was off-line. Messages are published in "topics" and for each subscriber to a topic the server maintains a queue of messages in that topic. OpenQueue can therefore be

[2] This is sometimes called subject or content-based *routing*. This work will consider content-based routing to be at a lower level than content-based messaging, e.g. routing algorithms or network techniques (e.g. mapping to multicast groups) to support such messaging. Content-based messaging would normally be based on content-based routing, but need not be.

[3] The subject or channel is also sometimes known as the *topic* or *group*.

considered a cross between subject-based addressing and message queuing.

Gryphon. Gryphon (Aguilera et al. 1999) is a distributed message brokering system - it maps a subscription database to a network of underlying brokers that distribute the messages. Gryphon supports content-based subscription and indeed, supplies content-based routing. Like Siena, the emphasis in this project is on scalability

3.2 Advantages of Content-Based Messaging

Content-based messaging has many advantages over other forms of communication. These include greater decoupling (no channel names); reduced data flow (more filtering can occur at the "server"); and greater flexibility. The Elvin content-based messaging service provides these features and more. Before describing Elvin in more detail, we examine Jini Network technology and the reasons it was not used as a basis for the AIF.

3.3 Jini

Jini (Sun Microsystems 2001) is a Java-based network technology that enables the spontaneous assembly and interaction of services and devices on a network. Jini clients and servers are decoupled in that they do not require prior (i.e. compiled-in) knowledge of each other, however, after service discovery has occurred, the actual communication is directed.

The AIF was based upon a content-based messaging service (Elvin) rather than Jini for several reasons. Jini was viewed as too much overhead, in that a complex set of interactions would have been required between the various processes. Jini is particularly useful for when objects need to be accessible in a code sense (i.e., method calls and manipulation), and involve many different types of interactions between a wide variety of devices. The AIF, however, is more specialised, and is not intended to provide interprocess code-level accessibility. The AIF is intended to support communication to/from/between basic devices and processes. Commonly, this would be one-way communication, for example light sensors continually emitting light level readings, irrespective of who's listening and what they listening for. Elvin's quenching capabilities (discussed below) prevent this from being a waste of network bandwidth.

A further disadvantage of Jini is the Java specificity. Whilst the AIF is currently Java-based, the underlying Elvin communication is platform and language independent and this would permit versions of the AIF to be developed in many languages.

4 Elvin Content Based Messaging

Elvin (Arnold et al. 2000) began as a publish-subscribe notification service, but has since evolved into a content-based messaging service. Elvin consists of:

- An easy to use API, allowing application developers to generate and consume information simply. There are language bindings for C/C++, Java, Python, Smalltalk, and Emacs Lisp, which are supported by a number of development tools.
- Dynamic definition of both information formats (messages) and subscriptions. This is a key feature required to allow scalability across organisational boundaries.
- Flexible and dynamic message content delivery defined as the application developer requires. Information is distributed only to the points where it is needed, allowing greater system throughput to be achieved. The importance of bandwidth efficiency over individual throughput is a fundamental design criterion of the Elvin service.
- A simple but powerful subscription language able to express complex constraints on the information routed to applications. Elvin allows all of the information to be used for routing choices - everything behaves like an addressable subject in a more traditional publish/subscribe system.
- Quenching is an unique feature of the Elvin service. It allows producers to receive information about what consumers are expecting of them so that they need only generate the events that are in demand. This is important for some classes of producers where the act of producing the event is expensive.
- A decoupled security model designed to maintain the flexibility of publish/subscribe messaging. Traditional security mechanisms are point-to-point, allowing for authenticated communication between two parties. Elvin provides a flexible security mechanism where producers and consumers can have overlapping key sets that combine to allow multiple-party authorisation. This is used to control the delivery of notifications whilst maintaining the flexibility of loosely-coupled components.

In its basic form, Elvin operates by having a server acting as a notification router between multiple connected clients. Clients can act as producers and/or consumers of events, and the server is responsible for routing notifications of interest to consumers. This has been extended to include "federations" of multiple servers but the concept of routing notifications based on content to interested clients remains the same.

5 Ambient Interaction Framework

The Ambient Interaction Framework builds upon the content-based messaging provided by Elvin to provide a toolkit from which the software infrastructure for physical ubiquitous computing environments can be rapidly created. This section describes the design goals of the AIF, provides details of the physical devices which have been encapsulated into the framework, and shows some example AIF code.

5.1 Design Goals

The design goals of the Ambient Interaction Framework were as follows:

- rapid development - it must support the quick development of systems of devices;
- device independence - it must be able to support a wide variety of devices;

- portability - it must be able to be run on many platforms; and
- decoupled communication - devices should not require knowledge of the network and/or other devices and services.

It is worth reinforcing, that the goal has not been to solve a particular awareness or other problem, but to develop a *toolkit* which allows the rapid creation of and experimentation with pervasive computing environments. Fig. 2 shows where the AIF fits within a pervasive computing environment.

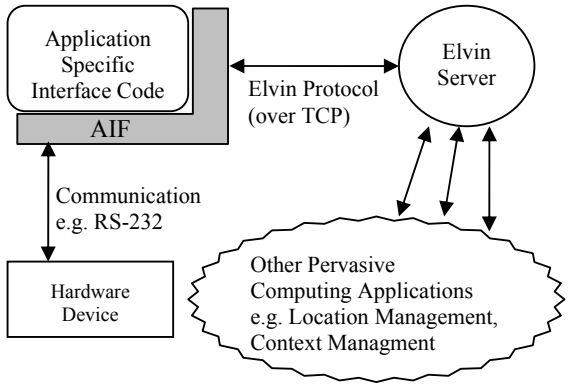

Figure 2: Role of the AIF in a Pervasive Computing Environment

The goal of rapid development has been met by incorporating the device and communication specific code into the AIF. This means that only a minimal amount of application specific code needs to be implemented. The goal of portability has been met by the use of Elvin and by implementing the AIF in Java. The goal of decoupled communication has been met by basing the AIF on the Elvin content-based messaging service. The goal of device independence has been satisfied by the use of Elvin, a common Elvin notification format, and the concept of encapsulations. Details of the common notification format and AIF *encapsulations* are provided below.

5.2 Common Notification Format

Elvin notifications consist of a set of attribute-value pairs, where the attributes are strings and the values can be of type string, 32-bit integer, 64-bit integer, double-precision floating point number or opaque byte array (Arnold et al. 2000).

The AIF imposes minimal restrictions on the format of Elvin notifications. The required notification fields are as follows:

```
edu.dstc.ambient:     1
type:   "String describing the type of sensor"
id:     "String containing the sensor id"
```

The first attribute-value pair marks this notification as belonging to the AIF. The type and id fields contain arbitrary, programmer-defined strings which describe the physical device (often a sensor). All other fields in the notification can be chosen by the programmer - it may be as little information as an integer sensor value, or as much as a detailed description of the location of the device, it's capabilities, the measurement units of any data provided and so on.

5.3 Ambient Interaction Framework Encapsulations

Ambient Interaction Framework encapsulations have been performed for several physical devices. These include NIQ's EZIO boards (http://www.ezio.com), microphones and Lego Mindstorms (http://mindstorms.lego.com) RCX-controlled motors. An *encapsulation* refers to writing wrapper code in order to make the device functionality available via Elvin notifications. The encapsulation of input devices (e.g. sensors) will provide methods by which the data-gathering capabilities of the device can be controlled. The encapsulation of output devices (e.g. actuators) will provide methods by which the actions can be controlled by Elvin notifications.

Details of the encapsulated devices are provided below.

EZIO Boards. EZIO boards provide a simple means to connect digital and analog inputs and outputs to various computing devices via an RS232 serial connection. EZIO boards support 10 digital (5V) inputs, 10 digital outputs, 8 analog (0 to 5V) inputs and 2 pulse-width modulated (PWM) outputs.

The Ambient Interaction Framework encapsulation of EZIO boards allows the programmer to easily specify the conditions under which notifications are to be emitted. For example, for analog sensors, it is possible to configure the AIF sensor interface to emit notifications of the value of the sensor under the following circumstances:

- regular sampling - emit notifications of the sensor's value periodically;
- regular sampling with delta - sample the sensor periodically but only emit a notification if the value has changed by a certain amount; and
- threshold - emit notifications if the sensor value passes under (or above) some defined threshold value.

Using Elvin, it is possible to automatically quench (i.e. not transmit) notifications to which no consumers are subscribed. This can save considerably on network bandwidth.

Microphone. An AIF encapsulation of audio-input devices has been performed. This allows for the notification of events based on audio-input levels. As with the EZIO board analog inputs, audio noise-level events can be generated periodically, periodically provided some threshold has been passed or the noise level has changed by a certain delta amount.

LEGO Mindstorms. An AIF encapsulation of a LEGO Mindstorms RCX control "brick" and Mindstorms motors has been performed. The AIF interface currently supports the control of the RCX unit (e.g. playing sounds, checking the battery level, turning the unit off) and the control of any attached motors.

```
Class Hierarchy
     class java.lang.Object
          class edu.dstc.ambient.AbstractSensor (implements edu.dstc.ambient.Sensor)
               class edu.dstc.ambient.EZIOSensor (implements edu.dstc.ambient.Sensor)
                    class edu.dstc.ambient.AnalogSensor
                         class edu.dstc.ambient.CloseSensor
                    class edu.dstc.ambient.DigitalSensor (implements edu.dstc.ambient.Sensor)
          class edu.dstc.ambient.DataValue (implements java.lang.Comparable)
               class edu.dstc.ambient.IntDataValue
          class edu.dstc.ambient.EZIO
          class edu.dstc.ambient.EZIOOutput (implements edu.dstc.ambient.Effector)
               class edu.dstc.ambient.DigitalOutput
          class edu.dstc.ambient.MotorOutput (implements edu.dstc.ambient.Effector)
          class edu.dstc.ambient.PWMOutput (implements edu.dstc.ambient.Effector)

Interface Hierarchy
     interface edu.dstc.ambient.Effector
     interface edu.dstc.ambient.Sensor
```

Figure 3: AIF Class Hierarchy

5.4 AIF Class Hierarchy

A subset of the AIF class hierarchy is shown in Fig. 3. These classes support interactions with EZIO boards and LEGO Mindstorms motors. The base classes are designed to be easily extensible and provide base level functionality such as the Elvin communication.

5.5 Example AIF Code

An example Java code fragment from an AIF based application is shown below (Fig. 4). This code assumes:

- there is an EZIO board attached to serial port COM1;
- there are touch and light sensors connected to A2D ports 8 and 4 respectively; and
- there is a digital output device (e.g. relay control) connected to digital port 2.

This code is all that is needed to setup the board, Elvin connection, sensor sampling rate, notification generation conditions and the conditions under which the digital output is to be switched on and off (phrased as Elvin subscriptions). Comments within the code explain the operation of the program further.

```
// We have an EZIO board attached to COM port 1, and an Elvin
// server running on host elvin.dstc.edu.au
EZIO ezio = new EZIO("COM1");
ElvinURL elvinURL = new ElvinURL("elvin://elvin.dstc.edu.au");
Producer prod = new Producer(elvinURL);
Consumer cons = new Consumer(elvinURL);
// Add a touch sensor on port 8 with id "#1" and sample it every 200ms
// Add a light sensor on port 4 with id "#2" and sample it every 200ms
AnalogSensor touch = new AnalogSensor(ezio, 8, 200, "#1", "Touch");
touch.setElvinConnection(prod);
AnalogSensor light = new AnalogSensor(ezio, 4, 200, "#2", "Light");
light.setElvinConnection(prod);
// Turn on automatic notification for the touch and light sensors,
// but only send notifications if the value changes by 10 or more
touch.setDelta(10);
touch.setAutoNotify(true);
light.setDelta(10);
light.setAutoNotify(true);
// Add a digital output on port 2 and set the conditions (Elvin
// subscription) under which it will be turned on and off. The output
// will be turned on if touch sensor is pressed (value > 128) and will
// be turned off if the touch sensor is released OR an "Off" message
// is received via Tickertape.
DigitalOutput dout = new DigitalOutput(ezio,2,cons);
dout.setOnNotification("edu.dstc.ambient == 1 &&
     type == \"Touch\" && value > 128");
dout.setOffNotification("(edu.dstc.ambient == 1 &&
     type == \"Touch\" && value <=128) ||
     (require(TICKERTAPE) && TICKERTEXT == \"Off\")");
```

Figure 4: Example Application Using AIF

6 Proof-of-Concept Implementation

To test the appropriateness of the AIF design, several proof-of-concept implementations have occurred. One of these, an Ambient Café, is described below.

6.1 Ambient Café

An early prototype version of the AIF was used in the construction of an Ambient Café - a student project within the Bachelor of Information Environments degree (Docherty et al. 2001) at The University of Queensland. The goal of the Ambient Café project was to create a technologically enabled café environment (entrance, dining area, entertainment area and back office) in which the computers were hidden from the users. The users (customers and staff) were to interact with the "system" in an ambient manner without knowing that they were using computers, and indeed, without knowing that there was a "system".

The resulting environment featured many "invisible" computers, supporting various sensors, actuators and display devices, such as bend, pressure, touch, and light sensors, microphones, lights, projection displays, speakers, and tickertape displays. Sensors and some actuators were connected to interface boards (often the EZIO board) which were connected to standard PCs[4]. The software for each sensor was implemented using the Ambient Interaction Framework. This allowed quick development of the software for the Café. Additional software components were able to use standard Elvin interfaces to receive notification of events of interest. These software components could then carry out other actions, such as changing the information on particular displays.

Some of the behaviours exhibited in the environment included:

- a back office display of occupied seats and tables based on pressure sensors on chairs;
- overhead lighting that led customers to their table as they walked into the café (see Fig. 5.);
- a front-window display of how busy the café was based on occupancy and ambient noise level; and
- an interactive graffiti wall that warped images based on activities that occurred within the café (e.g. flowers being moved within a vase). This formed part of the "entertainment" aspect of the café. The graffiti wall images were also available over the Internet.

The underlying infrastructure was the prototype version of the Ambient Interaction Framework based on Elvin. The decoupled nature of content-based communication enabled the individual components of the Café to be developed independently by small groups and to work seamlessly when brought together. This is the key advantage of the AIF and the use of Elvin. Much of the communication and hardware interface functionality is provided by the AIF - the developer worked only on the application specific behaviour aspects of the system.

[4] Standard PCs were used for cost reasons. A network-enabled embedded computer could easily have been used instead.

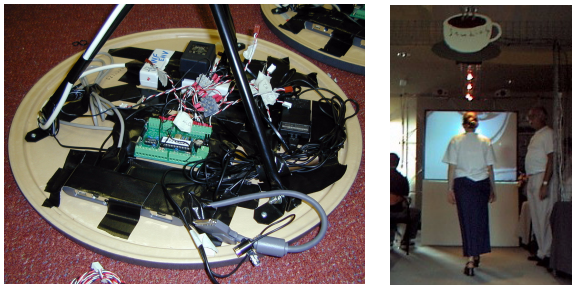

Figure 5: Scenes from the Ambient Café. *Left*: the underside of one of the café tables. *Right*: Entrance to the Ambience Café demonstrator. The overhead lights are controlled by events generated by pressure mats under the carpet.

7 Conclusions

The Ambience project of the DSTC and The University of Queensland aims to develop and implement ambient computing via a three pronged approach: observation and analysis of user behaviour, development of a theoretical framework, and, the design and development of an Ambient testbed in which experimental pervasive computing environments can be developed. Towards this latter goal, we have developed the Ambient Interaction Framework (AIF).

The AIF is based on the Elvin content-based messaging service. Elvin offers features crucial to the AIF including: *decoupled communication*, meaning senders and receivers and be developed independently, and; *quenching* - a reduction in bandwidth requirements by not transmitting messages to which no-one is listening. As the number of devices in pervasive computing environments grow these features will become more important.

The Ambient Interaction Framework eases the task of constructing new pervasive computing environments by encapsulating the data input and output capabilities of physical devices and automating the production and receipt of Elvin notifications. The encapsulated devices can then be integrated into new environments by writing a minimal amount of application-specific Java code.

The development of several ambient environments has proved the usefulness of the AIF. Future work will see the extension of the AIF to other physical devices and the development of other experimental environments (in progress at the time of writing) in order to further test the approach and extend the AIF.

8 Acknowledgements

The work reported in this paper has been funded in part by the Cooperative Research Centre Program through the Department of Industry, Science and Resources of the Commonwealth Government of Australia.

9 References

AGUILERA, M., STROM, R., STURMAN, D., ASTLEY, M., and CHANDRA, T. (1999): Matching Events in a Content-based Subscription System. *Principles of Distributed Computing.*

ALLPORT, D., RENNISON, E., and STRAUSFELD, L. (1995): Issues of gestural navigation in abstract information spaces. *Proceedings of CHI '95,* 206-207, ACM.

ARNOLD, D., BOOT, J., HENDERSON, M., PHELPS, T., and SEGALL, B. (2000): *Elvin - Content-Addressed Messaging Client Protocol*, Proposed Internet Draft. http://elvin.dstc.edu.au/download/internet-draft.txt.

CARZANIGA A., ROSENBLUM, D., and WOLF, A. (2001): Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems* **19**(3): 332-383, ACM.

DOCHERTY, M., SUTTON, P., BRERETON, M., and KAPLAN, S. (2001): An Innovative Design and Studio-based CS Degree. *Proceedings of 32nd SIGCSE*. ACM Press.

GREENBERG, S. and FITCHETT, C. (2001): Phidgets: easy development of physical interfaces through physical widgets. *Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, 209-218.

ISHII, I., WISNESKI, C., BRAVE, S., DAHLEY, A., GORBET, M., ULLMER, P., and YARIN, P. (1998): Ambient Room: Integrating Ambient Media with Architectural Space. *Proceedings of CHI '98*, ACM.

KANTOR, M., REDMILES, D. (2001): Creating an Infrastructure for Ubiquitous Awareness. *Proceedings of Interact'01*, 431-438, IOS Press.

RITTEL, H., and WEBBER, M. (1973): Dilemmas in a General Theory of Planning. *Policy Studies* **4**(1): 155-169.

SEGALL, B., ARNOLD, D., BOOT, J., HENDERSON, M., and PHELPS, T. (2000): Content Based Routing with Elvin4. *Proceedings AAUG2K*, Canberra, Australia.

STREITZ, N. (1999)
http://www.darmstadt.gmd.de/ambiente/.

SUN MICROSYSTEMS (2001): *JINI Network Technology Datasheet*. From http://www.sun.com/jini.

WEISER, M. (1993): Hot Topics: Ubiquitous Computing. *IEEE Computer*, October. See also
http://www.ubiq.com/hypertext/weiser/UbiHome.html

WEXELBLAT, A. (1995): An approach to natural gesture in virtual environments. *ACM Transactions on Computer-Human Interaction* **2**(3): 179-200.