CHAPITRE 2. INTRODUCTION AUX SYSTEMES UBIQUITAIRES - UBIQUITOUS COMPUTING SYSTEMS

This chapter presents the fundamental challenges and requirements for building Ubiquitous Computing systems.
The main reference for this chapter is the following:
Jakob Bardram and Adrian Friday, Ubiquitous Computing Systems, Book Chapter 2 in John Krumm's "Ubiquitous Computing Fundamentals "CRC Press (2010), p. 394 http://www.crcpress.com/product/isbn/9781420093605.

Contents

2.1 INTRODUCTION

- The prevalent computing paradigm is designed
  - for personal information management, including personal computers (PCs) such as desktops and laptops with fixed configurations of mouse, keyboard, and monitor; wired local area network;
  - dedicated network services with fixed network addresses and locations, such as printers and file servers; and
  - a user interface consisting of on-screen representation and manipulation of files documents, and applications through established metaphors such as the mouse pointer, icons, menus, and windows.
- Ubiquitous computing (ubicomp) strives at creating a completely **new paradigm of computing environment** in almost all of these respects.
- Ubicomp systems aim for a ***heterogeneous set of devices***, including
  - invisible computers embedded in everyday objects such as cars and furniture,
  - mobile devices such as personal digital assistants (PDAs) and smart phones,
  - personal devices such as laptops, and
  - very large devices such as wall-sized displays and tabletop computers situated in the environments and buildings we inhabit.
  - All these devices have different operating systems, networking interfaces, input capabilities, and displays.
  - Some are designed for end user interaction—such as a public display in a cafeteria area—whereas other devices, such as sensors, are not used directly by end users.
  - The interaction mode goes beyond the one to- one model prevalent for PCs, to a many-to-many model where the same person uses multiple devices, and several persons may use the same device.
- Interaction may be *implicit*, *invisible*, or *through sensing natural interactions* such as speech, gesture, or presence;
  - a wide range of sensors is required, both sensors built into the devices as well as sensors embedded in the environment.
  - Location tracking devices, cameras, and three-dimensional (3-D) accelerometers can be used to detect who is in a place and deduce what they are doing.
  - This information may be used to provide the user with information relevant in a specific location or help them adapt their device to a local environment or the local environment to them.
  - Networking is often wireless and ad hoc in the sense that many devices come in contact with each other spontaneously and communicate to establish services, when they depart, the network setup changes for both the device and the environment.
- Ubicomp environments involving technologies such as the ones described above have been created for a number of application domains, including
  - meeting rooms (also known as ***smart rooms***), classrooms, cars, hospitals, the home, traveling, and museums.
- In order to get a feeling of what ubicomp systems would look like, let us consider some examples from a future hospital (Bardram et al., 2006):
  - Doctors and nurses seamlessly move around inside the hospital using both personal portable displays (e.g., a super lightweight tablet PC) as well the large multitouch displays available on many walls inside the wards, conference rooms, operating rooms, and emergency departments.

    - Indoor location tracking helps in keeping track of clinicians, patients, and equipment, as well as assisting the clinicians and patient with location- and context-dependent information.
    - For example, the patient is constantly guided to the right examination room, and
    - on the doctor's portable devices, relevant information on the nearby patient is fetched from the central servers and presented according to the doctor's preference on this specific type of devices.
    - If he needs more display space, he simply drops the portable display in a recharge station, and moves to a wall display where the information is transferred.
    - In the conference room, the large conference table is one large display surface that allows for colocated collaboration among the participating physicians.
    - The location tracking system as well as biometric sensors keep track of who is accessing medical data, and prevents nonauthorized access.
    - Unique identification tags and medical body sensor networks attached to patients as well as to the patient's bed and other equipment inside, for example, the operating rooms, constantly monitor the patient and provide a high degree of patient safety.
    - Not only are critical medical data such as pulse, electrocardiogram(ECG), and heart rate monitored, but also more mundane safety hazards such as wrong side surgery and lack of relevant instruments are constantly monitored and warnings issued if the "system" detects potential problems.

- Ubicomp systems research is concerned with the underlying technologies and infrastructures that enable the creation and deployment of these types of ubicomp applications.
- Ubicomp systems research addresses a wide range of questions such as:
    - how to design hardware for sensor platforms,
    - how to design operating systems for such sensor platforms;
    - how to allow devices to find each other;
    - how to allow devices to use the services on each other;
    - how to design systems support for resource impoverished devices that run on batteries and need to save energy;
    - how to run large distributed infrastructures for seamless mobility and collaboration in creating applications for such settings as smart rooms and hospitals; and a wide range of other systems aspects.
- To some degree, ubicomp systems questions and challenges overlap and coexist with other systems research questions, but as outlined in this chapter, the ubicomp vision and the nature of ubicomp applications, present a unique set of challenges to ubicomp systems research.

The chapter commences with a discussion of

- the key topics and challenges facing ubicomp systems, highlighting assumptions that are often made in traditional systems thinking that are unreliable in this problem domain.
- Then, design rationale and process for creating "good" ubicomp systems is explored, leading to advice on how to choose hardware and software components well and consolidated tips on what to look for when deploying ubicomp systems "in the wild."
- discuss the process of evaluating and documenting ubicomp systems—essential if the system is to be of any importance in moving the field forward.
- available software and hardware components and datasets that can help you bootstrap your experimental systems development.
- Building ubicomp systems is essential to the progress of the field as a whole.
- Experimentally prototyping ubicomp systems enables us to

    - experience them, discover what they are like to use, and reason about core precepts such as the boundaries of the system, its invisibility, the role of its users, and the degree of artificial intelligence endemic to it.
- By implementing systems, we discover what comprises ubicomp systems, what is and is not computationally tractable, form hypotheses to be tested, and uncover the research challenges that underpin and inform the evolving vision of ubicomp itself.

The aim of the chapter is to offer advice to those planning to create ubicomp systems to sensitize them to the issues that may face them in the design, implementation, deployment, and evaluation stages of their projects.

2.2 UBICOMP SYSTEMS TOPICS AND CHALLENGES

- Creating ubicomp systems entails a ***wide range of technical research topics and challenges***.
    - Compared to existing systems research, some of these topics and challenges are ***new*** and ***arise*** because of the intention to build ubicomp applications.
    - For example, ubicomp applications often involve <u>**scenarios where devices, network, and software components change frequently**</u>.
- The challenges associated with such extremely volatile executing environments are new to systems research.
- *These kinds of challenges are introduced because we intend to build new computing technology that is deployed and runs in completely new types of physical and computational environments.*
- Other topics and challenges existed before ubicomp but are significantly aggravated in a ubicomp setting.
- For example, new challenges to security arise because trust is lowered in volatile systems; spontaneous interaction between devices often imply have a trusted third party in common.

More significant topics and challenges to ubicomp systems research.

2.2.1 Resource-Constrained Devices

- Most ubicomp applications and systems involve devices with limited resources.
- Moore's law: ever-increasing CPU speed, memory, and network bandwidth in servers and desktop computers.
- With ubicomp, a wide range of new devices are built and introduced, which are much more resource-constrained.
    - Devices such as PDAs, mobile phones, and music players have limited CPU, memory, and network connectivity compared to a standard PC,
    - Embedded platforms such as sensor networks and smart cards are very limited compared to a PC or even a smart phone.
- When creating systems support in a ubicomp setting, it is important to recognize
    - the constraints of the target devices,
    - that hardware platforms are highly heterogeneous and incompatible with respect to hardware specifications, operating system, input/output capabilities, network, etc.

- Resource-aware computing is an approach to develop technologies where the application is constantly notified about the consumption of vital resources, and can help the application (or the user) to take a decision based on available resources now and in the future.
    - For example, video streaming will be adjusted to available bandwidth and battery level or the user may be asked to go to an area with better wireless local area network coverage.

- Generally speaking, the most limiting factor of most ubicomp devices is **energy**:
    - A device that is portable or embedded into the physical world typically runs on batteries,
    - The smaller and lighter the device needs to be, the lower its battery capacity.
    - For this reason, one of the main hardware constraints to consider when building ubicomp systems and applications is **power consumption** and/or opportunities for **energy harvesting**— including recharging.
    - A central research theme within ubicomp is **power foraging**, that is, technologies for harvesting power in the environment based on, for example, kinetic energy from a walking person.
    - **Cyber foraging** is a similar research theme where devices look for places to offload resource-intensive tasks, for example, a portable device may offload computations to server-based services, or if the user tries to print document located on a file server from a PDA, the document is not first send to the PDA and then to the printer, but instead sent directly from the file server to the printer.

Computation, accessing memory, and input/output all consume energy.

- The major drain on the battery is, however, wireless communication, which is also typical for mobile or embedded ubicomp devices.
- Power consumption in wireless communication is hence another major topic in ubicomp systems research, investigating resource-efficient networking protocols that limit power consumption due to transmitting data, while maintaining a high degree of throughput and reliability,
    - For example, since processing consumes much less power than communication, mobile ad hoc sensor networks (MANETs) seek to do as much in-network processing as possible, that is, ensuring that nodes in a sensor network perform tasks such as aggregating or averaging values from nearby nodes, and filtering before transmitting values.

2.2.2 Volatile Execution Environments

- A core research topic in ubicomp systems research is **service discovery**,
    - Technologies and standards that enable devices to
        - discover each other,
        - set up communication links,
        - start using each others' services.
    - For example, when a portable device enters a smart room, it may want to discover and use nearby resources such as public displays and printers.
- Several **service discovery technologies** have now matured and are in daily use in thousands of devices. These include: ***Jini***, ***UPnP***, ***Bonjour/multicast DNS*** (mDNS), and the ***Bluetooth*** discovery protocol.
- Nevertheless, several challenges to existing approaches still exist, including:
    - the lack of support for multicast discovery beyond local area networks,
    - the lack of support beyond one-to-one device/service pairing, and rather cumbersome methods for pairing devices, often involving typing in personal identification numbers or passwords.
- Research is ongoing to improve upon service discovery technologies.

- Ubicomp systems and applications are often ***distributed***;
- They entail interaction between different devices—mobile, embedded, or server-based—and use different networking capabilities.
- Looking at ubicomp from this distributed computing perspective, a fundamental challenge to ubicomp systems is their ***volatile nature***:

    - The set of users, devices, hardware, software, and operating systems in ubicomp systems is highly dynamic and changes frequently.

- One type of volatility arises because of the ***spontaneous nature of many ubicomp systems***: devices continuously connect and disconnect, and create and destroy communications links. But because from a communication perspective—these devices may leave the room (or run out of battery) at any time, communication between the mobile devices and the services in the smart room needs to gracefully handle such disconnection.
- Another type of volatility arises due to ***changes in the underlying communication structure***, such as <u>**topology**</u>, <u>**bandwidth**</u>, <u>**routing**</u>, and <u>**host naming**</u>.
    - For example, in an <u>ad hoc sensor network</u>, the network topology and routing scheme is often determined by nodes available at a given time, the physical proximity of the nodes in the network, their current workload, and battery status; in addition, this network routing scheme should be able to handle nodes entering and leaving the network.
    - A simpler example arises in smart room applications where devices entering the room do not know the network name or addresses of the local services, and in this case services discovery would entail obtaining some network route to the service.
- Volatility arguably also exists in more traditional distributed systems; client software running on laptops are being disconnected from their servers in a client-sever setup, and PDAs and cell phones whose battery is flat are able to reconnect once recharged.
    - The main difference, however, is that unlike most traditional distributed systems, the connectivity changes are common rather than exceptional, and often of a more basic nature.
    - For example, in a client-server setup the server remains stable, and both the client and server maintain their network name and address.
- For these reasons, existing distributed computing mechanisms such as the Web (HTTP), remote procedure calls, and remote method invocation (Java RMI, .NET Remoting, or CORBA) all rely on stable network connections (sockets) and fixed network naming schemes.
- In a ubicomp environment, these assumptions break down.

2.2.3 Heterogeneous Execution Environments

- Most ubicomp applications and systems inherently live in a ***heterogeneous environment***.
- Ubicomp applications often involve a wide range of hardware, network technology, operating systems, input/output capabilities, resources, sensors, etc.,
- In contrast to the traditional use of the term <u>application</u>, <u>which typically refers to software that resides on one—at most, two—physical nodes</u>, *a ubicomp application typically spans several devices, which need to interact closely and in concert in order to make up the application*.
    - For instance, the Smart Room is an application that relies on several devices, services, communication links, software components, and end user application, which needs to work in complete concert to fulfill the overall functionality of a smart room.
    - Hence, handling heterogeneity is not only a matter of being able to compile, build, and deploy an application on different target platforms—such as building a desktop application to run on different versions of Windows, Mac OS, and Linux.
    - It is to a much larger degree a matter of continuously— that is, at runtime—being able to handle heterogeneous execution environments, and that different parts of the ubicomp application run on devices with highly varying specifications.

For example,

    - when a user enters the smart room and wants to access the public display and print a document, this may involve a wide range of heterogeneous devices, each with their specific hardware, operating systems, networks interfaces, etc.;
    - the user may be carrying a smart phone running Symbian;
    - he may be detected by a location tracking system based on infrared sensors on a Berkley Mote running the TinyOS;
    - his laptop may use mDNS for device discovery,
    - whereas the public display may be running Linux using the X protocol for sharing its display with nearby devices.

- The challenge of heterogeneity partly arises because ubicomp is a new research field and a new standard technology stack including hardware, operating system, etc., has yet to mature.
- Ubicomp applications will always need to use different kinds of technologies ranging from small, embedded sensors, to large public display and mobile handheld devices.
- As such, heterogeneous hardware devices are a fundamental part of ubicomp applications, and the corresponding operating systems and software stacks need to be specifically optimized to this hardware; the small sensor nodes need a software stack optimized for their limited resources and the large display similarly needs a software stack suited for sophisticated graphics and advanced input technologies.

Therefore, a core systems topic to ubicomp is to create <u>base technologies</u> that are able to handle such heterogeneity by balancing the need for optimizing for special-purpose hardware while trying to encapsulate some of the complexities in common standards and technologies.

2.2.4 Fluctuating Usage Environments

- The challenges discussed above are all concerned with issues relating to the *execution environment of ubicomp applications*.
- However, there is also a set of challenges that are associated with the *nature of ubicomp applications and how they are designed to be used*.

- Traditional computing usage model
    - Users use PCs for information management locally or on servers;
    - They engage in a one-to-one relationship with the PC;
    - the physical use context is fairly stable and is often tied to a horizontal surface such as an office desk or the dining table at home;
    - The number and complexity of peripherals are limited and include well-known devices such as printers, external hard drives, cameras, and servers.

- Compared to this usage model, ubicomp applications and hence systems live in a far more complicated and ***fluctuating usage environment***.
    - Users have not one but several personal devices, such as laptops, mobile phones, watches, etc.
    - The same device may be used by several users, such as the public display in the smart room or a smart blood pressure monitor in the patient ward of a hospital.
- Ubicomp systems need to support this many-to-many configuration between users and devices.
- Furthermore, compared to the desktop, the physical work setting in ubicomp exhibits a larger degree of alternation between many different places and physical surroundings.
- Mobile devices mean that work can be carried around and done in different places, and computers embedded into, for example, furniture that is constantly used by different people.

– Finally, doing a task is no longer tied to one device such as the PC, but is now distributed across several heterogeneous devices as explained above.
– This means that users need technology that helps them stay focused on a task without having to deal with the complexity of setting up devices, pairing them, moving data, ensuring connectivity, etc.

A core research challenge to ubicomp is to create systems, technologies, and platforms allowing the creation of applications that are able to handle such fluctuating usage environments.
– Special focus has, so far, been targeted at handling three types of fluctuation in usage environment: (1) ***changing location of users***, (2) ***changing context of the computer***, and (3) ***multiple activities (or tasks) of the users***.

1. Fluctuations related to different location of the users arise once mobile devices are introduced.
   1.1. ***Location-based computing*** aims to create systems and applications that provide relevant information and services based on knowledge about the location of the user.
      - For example, in the **GUIDE** (Cheverst et al., 2000) project, tourists were guided around historic sites by a location-aware tour guide, which automatically would present relevant descriptions based on the tourist's location.
      - Central to location-based systems research is the ***challenge of sensing the location of the user or device***.
      - A wide range of technologies already exist, such as global positioning systems (GPS). But because they all have their advantages and disadvantages, new location technologies are still emerging. Hightower and Borriello (2001) provide an older, but still relevant, overview of available location technologies and their underlying sensing techniques. More details in the chapter on Location technologies.

   1.2. ***Context-aware computing*** aims at adapting the application or the computer in accordance with its changing context.
      - "Context of the application" includes **information about who is using the computer; who else is nearby; ambient information about the room, including light, sound, and temperature; physical materials and tools used; and other devices in a room.**

For example,
- *a context-aware hospital bed having embedded computers, displays, and sensors can be built to react and adapt to what is happening in its proximity;* ***it may recognize the patient in the bed; it may recognize clinical personnel approaching; it may bring up relevant medical information on the display for the clinicians; and it may issue a warning if the nurse is mistakenly trying to give the patient another patient's medication*** (Bardram, 2004).
- The context of a computer/device may change for two reasons: ***either the device moved to a new context (mobile device) or the physical context of an embedded computer changed*** because, for example, new people and devices entered a room.

A core research challenge to ubicomp systems research is to
– investigate proper technical architectures, designs, and mechanisms for context sensing, modeling, aggregation, filtering, inferring, and reasoning; for context adaptation; and for distribution and notification of context events.
(More detail in the chapter on context aware computing)

1.3. **Activity-based computing (ABC)** (Bardram and Christensen, 2007) aims at handling fluctuations based on users' need for handling many concurrent and collaborative activities or tasks.

For example,
In a hospital each clinician (doctor or nurse) is engaged in the treatment and care of several patients, each of whom may have a significant amount of clinical data associated.
For the clinician, it is associated with a substantial mental and practical overhead to switch between different patients, because it involves using several devices, displays, and medical software applications.

- The ABC approach helps users manage the complexity of performing multiple activities in a complex, volatile, heterogeneous ubicomp systems setup involving numerous devices in different locations.
- Hence, focus is on systems support for aggregating resources and services that are relevant to an activity; supporting those activities, and their associated resources, moving seamlessly between multiple devices; supporting multiple users working together on the same activity—potentially using different devices; supporting intelligent and semiautomatic generation and adaptation of activities according to changes in the work environment; and supporting the orchestration of multiple services, devices, and network setup to work optimally according to the users' changing activities.

2.2.5 Invisible Computing

- Invisible computing is **central to the vision and usage scenarios of ubicomp**, but handling and/or **achieving invisibility is also a core challenge**—

for example, having embedded sensor technology that monitors human behavior at home and provides intelligent control of heating, ventilation, air conditioning, and cooling (HVAC), or pervasive computing systems in hospitals that automatically ensure that patient monitoring equipment is matched with correct patient ID, and that sensor data are routed to the correct medical record. In many of these cases, the computers are invisible to the users in a double sense. First, the computers are embedded into buildings, furniture, medical devices, etc., and are as such physically invisible to the human eye. Second, the computers operate in the periphery of the users' attention and are hence mentally invisible (imperceptible).
From a systems perspective, obtaining and handling invisible computing is a fundamental change from traditional computing, because traditional systems rely heavily on having the users' attention; users either use a computer (e.g., a PC or a server through a terminal or browser) or they do not use a computer. This means, for example, that the system software can rely on sending notifications and error messages to users, and expect them to react; it can ask for input in the contingency where the system needs feedback in order to decide on further actions; it can ask the user to install hardware and/or software components; and it can ask the user to restart the device. Moving toward invisible computing, these assumptions completely break down.
A wide range of systems research is addressing the challenges associated with building and running invisible computers. Autonomic computing (Kephart and Chess, 2003), for example, aims to develop computer systems capable of self-management, in order to overcome the rapidly growing complexity of computing systems management. Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity for the users. An autonomic system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions. Whereas autonomic computing is, to a large degree, conceived with centralized or cluster-based server architectures in mind, multiagent systems research (Zambonelli et al., 2003)

seeks to create software agents that work on behalf of users while migrating around in the network onto different devices.
Agents are designed to ensure that lower-level systems issues are shielded from the user, thereby maintaining invisibility of the technology. Research on contingency management (Bardram and Schultz, 2004) seeks to prevent users from being involved in attending to errors and failures.
In contrast to traditional exception handling, which assumes that failures are exceptional, contingency management views failures as a natural contingent part of running a ubicomp system. Hence, techniques for proactive management of failures and resource limitations need to be put into place. For example, off-loading an agent before a mobile device runs out of battery, and ensure proactive download of resources before leaving network coverage.
A similar research topic is graceful degradation, which addresses how the system responds to changes and, in particular, failures in the environment (Friday et al., 2005). Most existing technology assume the availability of certain resources such as Internet connectivity and specific servers to be present permanently. However, in situations where these resources are not available, the entire system may stop working. Real life demands systems that can cope with the lack of resources, or better still, systems should be able to adapt gracefully to these changes, preserving as much functionality as possible using the resources that are available.

2.2.6 Security and Privacy

Security and privacy is challenging to all computing. With ubicomp, however, the security and privacy challenges are increased due to the volatile, spontaneous, heterogeneous, and invisible nature of ubicomp systems (particularly imperceptible monitoring).
First, trust—the basis for all security—is often lowered in volatile systems because the principals whose components interact spontaneously may have no a priori knowledge of each other and may not have a trusted third party. For example, a new device that enters a hospital cannot be trusted to be used for displaying or storing sensitive medical data, and making the necessary configuration may be an administrative overhead that would prevent any sort of spontaneous use. Hence, using the patient's mobile phone may be difficult to set up.
Second, conventional security protocols tend to make assumptions about devices and connectivity that often do not hold in ubicomp systems.
For example, portable devices can be more easily stolen and tampered with, and resource-constrained mobile or embedded devices do not have sufficient computing resources for asymmetric public key cryptography.
Moreover, security protocols cannot rely on continuous online access to a server, which makes it hard to issue and revoke certificates.
Third, the nature of ubicomp systems creates the need for a new type of security based on location and context; service authentication and authorization may be based on location and not the user. For example, people entering a cafe may be allowed to use the cafe's printer. In this case, if a device wants to use the cafe's printer, it needs to verify that this device indeed is inside the cafe. In other words, it does not matter who uses the printer, the cafe cares only about where the user is (Kindberg and Fox, 2002).
Fourth, new privacy challenges emerge in ubicomp systems (Langheinrich, 2001). By introducing sensor technology, ubicomp systems may gather extensive data on users including information on location, activity, who people are with, speech, video, and biological data. And if these systems are invisible in the environment, people may not even notice that data are being collected about them. Hence, designing appropriate privacy protection mechanisms is central to ubicomp research. A key challenge is to manage that users—wittingly or unwittingly—provide numerous identifiers to the environment while moving around and using services. These identifiers include networking IDs such

as MAC, Bluetooth, and IP addresses; usernames; IDs of tags such as RFID tags; and payment IDs such as credit card numbers. Chapter 3 examines some of these challenges in more detail.

Fifth, the usage scenarios of ubicomp also set up new challenges for security. The fluctuating usage environment means that numerous devices and users continuously create new associations, and if all or some of these associations need to be secured, this means that device and user authentication happens very often. Existing user authentication mechanisms are, to a large degree, designed for few (1–2) and long-lived (hours) associations between a user and a device or service. For example, a user typically logs into a PC and uses it for the whole workday. In a ubicomp scenario, where a user may enter a smart room and use tens of devices and services in a relatively short period (minutes), traditional user authentication using, for example, usernames and passwords is simply not feasible. Moreover, if the devices are embedded or invisible, it may be difficult and awkward to authenticate yourself—should we, for example, log into our refrigerator, a shared public display, and the HVAC controller in our homes? All in all, a wide range of fundamental challenges exists in creating security and privacy mechanisms that adequately takes into concern the technical as well as the usage challenges of ubicomp systems.

2.2.7 Summary

This section discusses many of the core challenges that are endemic to ubicomp and hence, in a supporting role, ubicomp systems research. These include coping with impoverished and resource-constrained devices, energy harvesting and usage optimization, environmental and situational volatility, heterogeneity and asymmetry of device capabilities, adapting to dynamic and fluctuating execution environments, invisibility and its implications for ubicomp systems, and privacy/security challenges. The next section will look at the process of designing systems to meet some of these challenges.

2.3 CREATING UBICOMP SYSTEMS

Building, deploying, and maintaining ubicomp systems require considerable and sustained effort. You should think carefully before you start building, about why you are building it, what you hope to learn, and what is going to happen to it in the future. Making good design decisions and being pragmatic about your objectives early on can save you enormous amounts of potentially unrewarding effort later on. Understanding why you are building a system can help you think more strategically about how to achieve the impact you desire or answer your research question more expediently. This section highlights key reasons to build ubicomp systems, best practices for developing systems, and common issues and pitfalls facing ubicomp systems developers.

2.3.1 Why Build Ubicomp Systems?

There are many reasons to build ubicomp systems. What you hope to do with the system, who the intended users are (both technically as developers and in terms of user experience), and the planned longevity should shape the design and implementation decisions for the project. Possible targets for ubicomp systems research include Prototyping future systems to explore u • biquity in practice

- Empirical exploration of user reactions to ubicomp
- Gathering datasets to tackle computational problems relating to ubicomp
- Creating ubicomp experiences for public engagement or performance
- Creating research test beds to agglomerate activity and stimulate further research
- To explore a hypothesis concerning ubicomp more naturalistically
- To test the limits of computational technologies in a ubicomp setting
- Addressing the perceived needs of a problem domain or pressing societal issue.

2.3.2 Setting Your Objectives

How one goes about achieving these objectives effectively should naturally impact how you undertake the research. It is important to consider where you will place your engineering effort and, importantly, whether parts of the problem need to be fully implemented and indeed are reasonable computationally to achieve the outcome you desire. For example, a small-scale study to test how users react to context-aware systems would require considerable effort to achieve sufficiently reliable context determination automatically, whereas emulating the context determination using "Wizard of Oz" techniques may be adequate to gain the results and far easier to engineer. Conversely, a system that is intended to run for a long time unattended (research test bed, smart room infrastructure) will need a much stronger focus on robust design and defensive programming if it is to survive without a high degree of attention and support. It is important to realize that some parts of your system will require considerable effort to achieve, but may not in themselves lead directly to novel results. The trick is to keep one eye on your objective to ensure that your focus never entirely shifts from the goals of the project, ensuring that your efforts are rewarded. Naturally, some flexibility is required because the goals of your system may shift over time. An initial prototypical exploration may uncover an interesting ubicomp problem to solve or hypothesis to test. A project whose initial focus is public engagement may uncover a rationale for wider ubicomp systems design that may lead to further exploration and more focused empirical studies. This is a natural and intended consequence of the scientific exploration of the ubicomp problem space (Figure 2.1), which has two key consequences: (1) that you remain sufficiently aware and agile to recognize such changes and plan for them consciously and (2) that the important lessons from your exploration are communicated effectively to the community.

How to achieve this last point is discussed further in Section 2.5.

2.3.2.1 Testing Your Ideas

Having established the context of your system and its objectives, it is common sense not to rush into the design phase without first testing and refining your ideas. There are many possible approaches with various time and effort implications, for example, one might create the following:

- Low-fidelity prototypes, which can be simple scenarios that can be discussed, paper prototypes, or even models of devices or graphical storyboards of proposed interactions—anything that can add richness to the discussion of the system with potential users.
- Video prototypes, although considerably requiring more effort to create, can communicate the concepts in the system quite effectively and act as a useful reference for explaining the system later on.
- Rapid prototypes of user interfaces using prototyping toolkits (see Section 2.6) can afford a more realistic synthesis of the intended user experience.
- "Wizard of Oz" prototypes of parts of the system may allow the final behavior of the system to be emulated and thus experienced by others.

In general, the more labor-intensive options are only really worth investing in if the project itself is a significantly larger undertaking or there is additional value to having the prototypes or associated media. One of the cheapest and lightest weight mechanisms is simply to present the proposed system to someone else to gain informal feedback. If they find the idea entirely preposterous or can see obvious significant flaws, it is certainly worth revisiting your scenario. This works best if the person is not a member of the project team!

2.3.3 Designing "Good" Systems

Once you have decided on your system's objectives and are happy with your ideas, the next challenge is to design a system that is fit for its purpose. There are many important concerns unique to ubicomp that you should consider in your design.

2.3.3.1 Computational Knowledge of the Physical World

From a systems design perspective, it is far from clear what the interfaces and internals of a ubicomp system should be, necessitating an experimental approach. In his much-venerated article in Scientific American (Weiser, 1991), Weiser espoused embedded virtuality and calm computing: the notion that computational devices were effectively invisible to its users and that interfaces to such systems were through entirely natural, sensor driven, and tactile interactions. Arguably, such systems almost empathically support the user in their daily tasks, requiring a high degree of knowledge about the user's desires and intents—in some cases, Artificial Intelligence. Only recently have we begun to see researchers challenge some of these precepts and explore other possible visions of ubicomp (e.g., Rogers, 2006). As a discipline, it is important that we continue to explore the boundary between "the system" and "the user" to find the balance points for computational tractability and effective user support.

A key challenge for ubicomp systems designers then, is to consider the "barrier" between the physical world and virtual (computational) world. Unlike conventional software applications, interfaces to ubicomp systems are often distributed, may have many forms of input and output involving several devices, and often incorporate subtle, oblique forms of interaction involving hidden or ambient sensors and displays. In their influential article, Fox and Kindberg (2002) encapsulate the divide between the responsibilities of the system and the user as the "semantic rubicon": "[that] demarcates responsibility for decision making between the system and the user." More specifically, in terms of system design, crossing the semantic rubicon implies defining the knowledge the system can have of the physical world and of user(s) behavior, that is, through sensing and user interaction; the counterpart, that is, the knowledge the user has of the system and how they might influence it; and the mechanisms and permissible interactions for one to influence the other.

As a system designer, you must decide what knowledge your system will need about the real world to function, how it will get into the system, how to represent it, how this state will be maintained, and what to do if it is incorrect. Unless this knowledge is easy to sense, or trivial to reason with, then you must also decide what the implications are if the knowledge is imperfect or conclusions are erroneously reached. There is clearly a significant difference in implication if the outcome of misconstruing the user's situation while laying still is to call the emergency services rather than dim the lighting! Designing when to involve the user with decisions, or in the context of the semantic rubicon, when the decision of the system becomes the decision of the user, may well be crucial to the acceptability of the system or its fit to its task, especially in sensitive or deployed settings. Key questions you should ask yourself are:

1. What can be reliably sensed?
2. What can be reliably known?
3. What can be reliably inferred?

The degree to which you can answer these questions for the intended function of your system will help determine the feasible scope, or set some of the research challenges.

2.3.3.2 Seamfulness, Sensibility, and Tolerant Ignorance

There are clearly limits to what your system can know about the physical world and the people who inhabit it. Sensors have innate properties due to their construction and the underlying physics that governs how well they sense. They may not be optimally placed or sufficiently densely deployed to

cover the area or activity that you wish to detect. High-level sensors, such as location systems, have complex behaviors governed by properties of the built environment and its associated "radio visibility"—these are time varying properties that also significantly depend on where they are used. The activity you wish to observe may simply be challenging to detect due to its subtlety, or difficult to isolate from other activities, noise, or the concurrent activities of other people. There is also the question of the reliability of what is sensed in the presence of partial or total sensor failure (e.g., erroneous sensor readings may be misinterpreted as activity).
This was once articulated as the challenge of designing systems that exhibit "tolerance for ignorance" (Friday et al., 2005). Although it is certainly challenging to consider how to build systems that continue to function well in the face of ongoing indeterminacy and uncertainty, a first step is to consider the scope and boundaries to your system. In their paper, Chalmers et al. (2003) present "seamful design," the notion that the seams or the boundaries and inaccuracies of the system can be exploited as a resource for system designers. They consider the example of a locationbased mixed reality game called "Can you see me now," where runners physically on the streets attempt to catch online players virtually overlaid on the same space. The runners (Figure 2.2) were tracked using GPS, and it quickly became apparent that the seams of the system—in this case, the ability to track the runners—was having an impact on the game play: "analysis of system logs shows estimated GPS errors ranged from 4 meters to 106 meters, with a mean of 12.4 meters. Error varied according to position in the game area, with some of the more open spaces exhibiting typically only a few meters error while the more narrow built–up streets suffered considerably more" (Chalmers et al., 2003).
Over the 2 days, the runners had time to talk with each other and develop tactics, as exemplified in this quote:
Crew: What defines a good place to catch them? Runner: A big open space, with good GPS coverage, where you can get quick update because then every move you make is updated when you're heading toward them; because one of the problems is, if you're running toward them and you're in a place where it slowly updates, you jump past them, and that's really frustrating. So you've got to worry about the GPS as much as catching them.

In this case, awareness of the limitations and characteristics of the system allowed the longer-term players (the runners) to improve their ability to play (use the system). This raises the design issue of how far to go toward exposing the seams of the system. As reported by Chalmers et al. (2003), Benford is quoted as proposing four strategies for presenting information to the user:

- Pessimistic: Only show information that is known to be correct
- Optimistic: Show everything as if it were correct
- Cautious: Explicitly present uncertainty
- Opportunistic: Exploit uncertainty (cf. Gaver et al., 2003)

One might regard the pessimistic and optimistic approaches as being a "more traditional," perhaps engineering-led approaches. It is very common to present a location of a user on a map as a dot, for example, although this does not typically communicate any underlying uncertainty or imprecision in the location estimate or may not even reflect whether the system believes this to be the true location of the user (e.g., if no GPS satellites are in view, this may simply be a historic artifact). Adjusting the size or representation of such a dot to reflect the confidence in location would enable the user to develop a greater trust and understanding of the system. The cautious approach is widely adopted on a typical mobile phone: the "bars of signal strength" indicator provides an intuitive iconographic representation of underlying features of the system architecture, which is a resource for the phone user both to reason about the success of making a phone call but also a plausible social device for claiming they got cut off due to "low signal strength." Cautious or even opportunistic ubicomp designs may offer systems that are amenable to user comprehension or even appropriation. What can and cannot be sensed or its underlying seams may even be an opportunity for design. Benford et al. (2004) present some examples of physical ubicomp interfaces in the context of what is sensible, sensable, and desirable (Boucher et al., 2003). This taxonomy helped the authors categorize uses of their devices, but also spot opportunities for other types of interaction with their devices that they had not originally foreseen.

2.3.3.3 User Mental Model and Responsibility

The corollary of considering the semantic rubicon and seamfulness of your system is to carefully plan the role the user will play in the system's operation. Ubicomp systems often differ significantly in the degree of understanding and "intelligence" they are intended to show toward the users' goals and desires. There are a spectrum of design choices as to when to involve the user in sensing or understanding the physical world and in decision making or instigating actions. For example, at one end of the spectrum, we might consider a scenario where the system fully "understands" the user's wants, and takes actions preemptively in anticipation of these (one can argue the degree and scope over which this is achievable).

At the other extreme, perhaps more cautiously, we might design assuming no action is taken by the system without user assent, or where the user provides sensory input (e.g., confirming the activity they are currently engaged in, although clearly this could quickly become tiresome). A compromise position might involve partially automating to support the user's perceived needs, but offering the ability for the user to intervene to cancel or override the actions proposed by the system. A further approach might be adaptive: for example, using machine learning that starts by involving the user in decisions but learns from this, moving toward automation of common or consistently detected tasks (but, crucially, can move back to learning mode again if unreliable or undesirable!). To help consider where on this scale parts of your system might lie, consider:

- The frequency or inconvenience of potential user involvement
- The severity or undesirability of the consequences if the system gets it wrong
- The reliability of detecting the appropriate moment and appropriate action
- The acceptability to the user of automating the behavior

As an example of how people can cooperate with ubicomp systems and supplement sensing capabilities, consider these two examples. In the GUIDE (Cheverst et al., 2000) context-aware tour guide system, city visitors could enter a dialogue with the system (involving selecting a series of photos of landmarks they could see from their position) to reorient the system when it was outside the scope of the wireless beacons used to determine location. This extended the effective range of the system without requiring the logistical and financial expense of adding additional microcells. Self-reported position was used very effectively in "Uncle Roy All Around You" (Benford et al., 2004), where players with mobile devices built up trust in unknown online players by choosing when and where to declare their position to the system to test and reaffirm the advice being offered to them. It would have clearly been possible to use GPS or cell fingerprinting to automate locating the players, but instead this became a key feature of the cooperation between online and mobile players. It is worth thinking about how the seams and possible limitations of your system can be used as a resource for design.

A key question is, "What do you intend for the user to understand or perceive of the system in operation?" To grow comfortable with it, adopt it, and potentially appropriate it, the user must be able to form a mental model of cause and effect or a plausible rationale for its behavior. In more playful or artistic ubicomp systems this question may be deliberately provocative or challenging, but this should still benefit from being a conscious and designed behavior.

2.3.3.4 It Is Always Runtime

Ubicomp systems are composed of distributed, potentially disjoint, and partially connected elements (sensors, mobile devices, people, etc.). The term “partially connected” here reflects that these elements will often not be reliably or continuously connected to each other; instead, the system is the product of spontaneous exchanges of information when elements come together. Clearly, interaction patterns and duration will vary with the design and ambition of any given system, but it is important to consider a key precept: once deployed, all changes happen at runtime. In a system of any scale, you will typically not have simultaneous access to all the elements to (for example) upgrade them or restart them. This has a number of implications:

1. Systems requiring a carefully contrived startup order are likely to fail.
2. If the availability of elements may be sporadic, your system should be able to gracefully handle disconnection and reconnection or rebinding to alternate services.
3. Assume that individual components may fail or be temporarily isolated (which is especially true of software elements on mobile devices) and design your system accordingly so that state can be recovered.
4. Decide proactively how to handle data when an element is disconnected: are the data kept (e.g., buffered) until reconnection, and if so, how much will you buffer before discarding. What strategy will you choose to decide which data to keep or discard (oldest, freshest, resample, etc.)?
5. Consider including version information in protocols used in systems designed to run longitudinally to at least identify version mismatches.

2.3.3.5 Handling Transient Connections

Network connections (or the lack or failure thereof) can have profound effects on the performance of ubicomp systems and, crucially, the end user experience. The effects on unsuspecting software throughout a device’s software stack can be serious: network names stop being resolved, closed connections can lead to software exceptions that stop portions of the code from executing, input/output system calls can block leading to stuck or frozen user interfaces. Considering what will happen if elements in the system that you are assuming to be always available—especially if they are on the critical path in terms of system responsiveness—fail, will help you identify and ideally mitigate for these potential problems.

When networks fail, it is common for data to be buffered and dropped at many levels in the protocol stack. In ubicomp, where data are often sensor traces informing the system of important events relating to interactions in the world, this buffering can introduce an array of associated problems. For example, old (buffered) data can be misleading if not timestamped and handled accordingly. Consider buffered GPS traces logged on a mobile device while the connection to the backend system is down: old locations can appear like current inaccuracy; the fast replay of buffered locations that normally occur 1 per second might look like you stopped then started moving very quickly and like lag if the replay is rate-paced or there is a perceptible latency in the system. Finally, your fresh data over a multiplexed interface will be behind the buffered data—a potentially serious delay can arise if the connection speed is low and the buffer large. If fresh data are due to user interaction, then the system will appear very unresponsive until the buffer is drained. It is very common for a frustrated user to try to interact multiple times or in many ways in the face of inexplicable delays in unresponsive interfaces, thereby exacerbating the problem. This is another good reason for revealing the connection status to the user using an appropriate metaphor (Cheverst et al., 2000; Satyanarayanan, 2001).

2.3.3.6 The State of the World

Transient connections and component failures have an impact on how consistent the state of your overall system will be. It is important to design in strategies for recovering from both of these cases.

Parts of your system may be replicable or sufficiently available to use well-known techniques to mask such failures and achieve some degree of fault tolerance. However, in many ubicomp systems, software components are often intimately linked to specialist or personal hardware, or may be placed in unique locations, which makes traditional techniques involving redundant replication or fail-over inappropriate. In such systems, we need an alternative. Techniques that have been reported in the literature include

- Optimistic replication of state, which allows partitioned elements to continue to function while disconnected and then reconcile the journal of changes made offline upon reconnection (e.g., the CODA mobile file system [Kistler and Satyanarayanan, 1992] allowed optimistic writes to cached files while disconnected, which were then replayed upon reconnection).
- Converging on eventually consistent state. Bayou (Terry et al., 1995) used gossip-style "anti-entropy" sessions during user encounters to propagate updated state via social networking, converging on a final state (e.g., scheduling group meetings by exchanging possible times and availabilities and iterating toward an agreed option—tentative and committed state, in this case appointments, was reflected to users in the user interface).
- Use of persistent stores or journals to allow recovery of state (locally or remotely). A central database or state repository is often used.
- Externalizing state (e.g., to a middleware platform, such as a Tuple Space), so that most components are lightweight and can recover state from the middleware (Borchers et al., 2002; Friday et al., 1999).
- Use of peer caches to replicate state for later repair. An on-demand state "repair" scheme (Floyd et al., 1997) was used in L2imbo (Friday et al., 1999), where peer replicas detect missing state by snooping for sequence numbers and asking neighbors to repair any missing data. Recursively, the system converges.
- Epidemic propagation of state using "gossip"-style protocols (Demers et al., 1987).

Where data do not have to be communicated in real-time (i.e., noninteractive gathering or logging of data), they can, of course, be batched and exchanged according to some schedule or when the opportunity arises. It is often surprising how quickly persistent data or event debug logs and application output can grow to fill a (particularly embedded) device. For long-running systems, ensuring capacity by estimating growth based on the running system and considering housekeeping will help avoid unexpected problems when the device is full later on. Full disks lead to numerous problems with database integrity, virtual memory management, and consequent and typically unexpected system call failures.

2.3.3.7 Is It Working?

Debugging ubicomp systems is extremely challenging. Elements are often distributed and may not be available or remotely accessible for debugging. In many cases, embedded elements may not have much, if any, user interface. A common requirement is to monitor the system's output to check status messages or to be able to perform tests by injecting commands to emulate interactions and test components. Common strategies include

- Use of conventional mechanisms such as log files • and network packet tracing to passively monitor running components
- Including status protocol messages that can be intercepted (often as periodic heartbeat messages)
- Adding status displays including use of hardware such as LED blink sequences, audible and visual feedback
- Including diagnostic interfaces such as embedded web servers that can be interrogated
- Enabling remote access to components such as remote shells, etc.

- Externalizing of state or communications by using a middleware such as a publish-subscribe event channel, Tuple Space, Message Oriented Middleware, etc.

For example, iROS (Ponnekanti et al., 2003) used the “EventHeap” a derivation of the Tuple Space to pass all communications between elements of the Stanford Interactive Workspaces project (Figure 2.3). All communication is observable, so liveness of a component is easy to establish. By injecting events manually, components could be tested. New applications and devices can be introduced that work with existing components by generating or using compatible events. Later, the behavior of the workspace could be changed at runtime by dynamically rewriting events to “replumb” the smart space. The design based around a central EventHeap contributed to the longevity and adaptability of the project, enabling a number of interesting extensions and projects to be built upon the system over time. Naturally, the need to communicate via the central entity meant that the performance of the system was bounded by the performance over the network to the EventHeap and load on this component, and dependent on its availability and robustness.

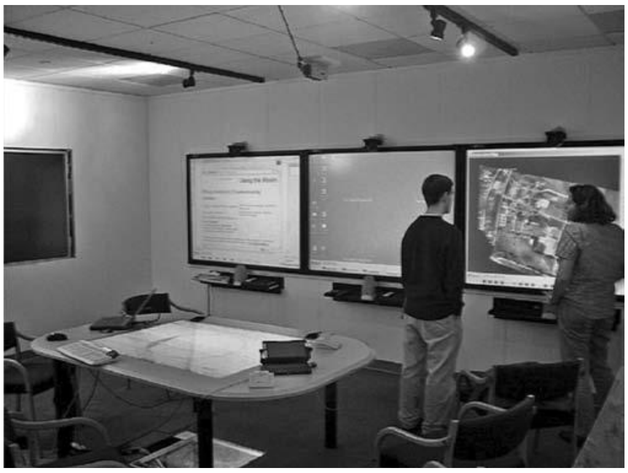
FIGURE 2.3 Stanford Interactive Workspaces Project (iWork).

2.3.4 Summary
Designing good systems, by some metric such as elegance, robustness, extensibility, usability, or fitness for purpose, is extremely challenging and requires thoughtful design. This section stressed the importance of first setting and being cognizant of your objectives, but also early testing of your ideas. Our second, but not secondary, focus is on important boundaries and thresholds between the system, its environment, and its users; encouraging purposeful and intentional designs with respect to system knowledge of the world, accuracy and dependability of sensing, tolerance to ambiguity, and the role of the users and their interplay with the system. Finally, the section discusses important technical differences between ubicomp systems and many conventional system designs: volatility, transience of connectivity, handling of state and techniques for evolving, and debugging live ubicomp systems. The next section turns to the important business of implementation.

2.4 IMPLEMENTING UBICOMP SYSTEMS
2.4.1 Choosing “Off-the-Shelf” Components
As with any computer-based system, the design of your ubicomp system is just the first step in realizing it. The design is often refined as implementation choices are made and their limits tested. Given the richness and ambition of typical ubicomp systems and the typical development resources and timescales, pragmatic choices have to be made as to define what you will build and what you will appropriate to construct your system.

It is natural to seek third party components from hardware and software vendors or, increasingly, from the public domain.
A key challenge is balancing this expedient use of off-the-shelf hardware and software against more bespoke solutions. Although the latter may offer a better fit to the problem domain or intended deployment environment than an off-the-shelf solution, it requires enormous effort to develop new technologies that meet the functional, aesthetic, reliability, or time constraints of the project. Again, this is a choice best made in the context of your objectives (which may aim to explore the creation of novel devices).
Building using proven components or implementations of standards may increase robustness or extend the range of functionality available to you more quickly, but there are also limitations to this approach that you should keep in mind when evaluating your choice:

1. You should not underestimate how much time can be spent in attempting to integrate disparate pieces of hardware and software.
2. Software perhaps successful or designed for one domain will not necessarily confer similar benefits to your domain.
3. The chosen software or hardware may place constraints on what you can build, or offer far more functionality than you require (with implications on software complexity and footprint).
4. Using proprietary hardware or software may imply the need to work around features and limitations that are outside of your control.
5. Versatile toolkits, for all their tempting power and flexibility, may introduce unneeded functionality and unwanted software bloat (particularly problematic when working with embedded and mobile devices).

Ask yourself critically whether the flexibility is really needed and compare to other strategies, such as just taking parts of the toolkit in question or simply coding the portion you need.
A hidden side effect of using third party libraries is that they may introduce dependencies that are not easy to understand or are too tightly integrated into the tool you have chosen to be removed or replaced. In longer-lived systems, the interdependencies between different versions of libraries and the level of skill and tacit knowledge required to update the system can become a particular burden. Do not forget that unknown systems may contain bugs or security vulnerabilities, or exhibit unwanted behavior. Because these are components that you do not necessarily fully understand, they may be difficult to detect and may take time to fix—a good justification for building with components you can get the source code for! It is important not to “let the tail wag the dog,” that is, to consider carefully whether the limitations or implications of accepting a constraint or technology are worth the compromise to your overall design. Recall the seamful design and role of the user design considerations discussed previously, and review carefully whether the perceived limitations can be embraced or taken advantage of in some way.

GAIA: Building on a Solid Foundation

In the GAIA, a meta-operating system for smart rooms (Figure 2.4) (Roman and Campbell, 2000), an industry quality CORBA middleware implementation was chosen as the core for the system. System components were implemented as distributed objects with CORBA IDL interfaces. This implementation choice enabled the project to build on a reliable core and focus on developing the higher-level GAIA OS services. As new services for CORBA matured (event channels, Lua scripting), these features could be exploited to enrich the GAIA OS. This implementation choice enabled the project team to focus their development effort on higher-level services such as security and configuration management without worrying about object distribution and lifecycle management. An undesirable side effect of the choice was the proprietary dependency it introduced, potentially limiting uptake by other sites that might otherwise have wished to adopt GAIA but did not want to accept the licensing implications. There was also limited scope for optimizing the interconnection and performance of the many objects and communication channels underpinning each GAIA smart room application.

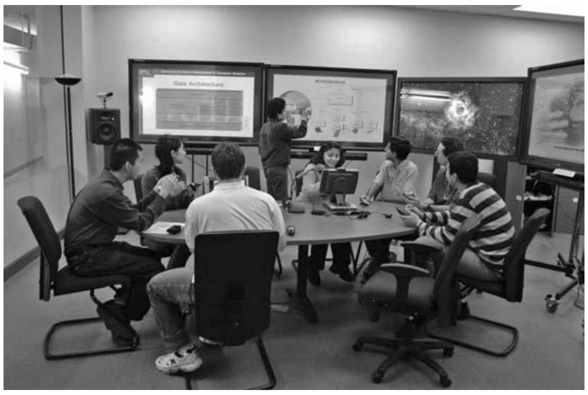

FIGURE 2.4 GAIA meta-operating system integrated a wide range of situated and mobile devices to offer an interactive smart room.

Cooltown: The Power of a Well-Chosen Paradigm HP's Cooltown was a system to support nomadic computing by associating digital information and functionality with "people, places and things" (Kindberg et al., 2002). An extremely versatile system, allowing both access to information and access to services, Cooltown was based on straightforward and elegant technical choice: that people and artifacts could be tagged, and the tag resolved to a uniform resource identifier (URI) that linked to a web point of presence for the person or artifact in question. The flexible use of tags, decentralized resolvers, and the innate flexibility in the design of URIs made the entire system extremely lightweight and extensible. Infrared beacons (Figure 2.5) broadcast URIs to mobile devices to link physical artifacts to digital information.

In summary, paraphrasing Ockham's razor: the implementation choice that makes the fewest assumptions and introduces the fewest dependencies without making a difference to the observable behavior of the system is usually the best.

2.4.2 Deploying Ubicomp Systems

One of the most valuable lessons to take from looking at successful ubicomp systems is the need to mature the system through actual use. Colloquially, by "eating your own dog food," or rather, deploying and using the system initially yourself (but ideally also with other users, who are not necessarily the developers) can gain early feedback and highlight usability and interaction issues that may otherwise get missed until such decisions are too well entrenched to be easily reversed. This lends itself to an agile development process where simple prototypes are put out early and refined during the development cycle.

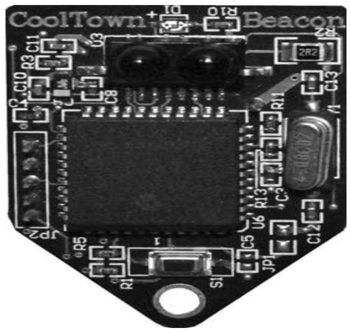

FIGURE 2.5 Cooltown beacon that enabled the physical environment to be augmented with digital information (top edge, just 3.3 centimeters wide).

With many systems, developers are also just another class of user; running training sessions with developers and having to explain the system and its application programming interfaces (APIs) to others can be a valuable source of insights. If your aim is to encourage adoption by others, and you plan to put the software in the public domain, then doing a “clean room install” helps “quantify the magic” and tacit knowledge that the systems’ own developers are able to apply when using and installing the system. Documenting this type of information (e.g., as installation and maintenance guides) in a wiki associated with the software can help smooth adoption and also provide a resource for continuity if there are changes of personnel in the project team over the longer term. Deploying systems for people to use is always a costly process. Designing a system that meets peoples’ expectations, and indeed, helping set those expectations requires great care and expertise. The key is, of course, identifying the stakeholders and involving them in discussions from an early stage. How to design with users, known as participatory design, is a major topic for discussion in its own right; so we direct the interested reader to such texts as that published by Schuler and Namioka (1993). See also Chapter 6 for further discussion on participatory design.
Ubiquitously deploying technologies inevitably implies that, at some point, technologies must move out of the research laboratory and into the “real world.” Experience has shown that with this comes a number of real world constraints and practical concerns that may be unexpected and are certainly worth being highlighted (Fox et al., 2006; see Table 2.1).
There are many issues due to the real world and organizational settings that can catch the unwary developer by surprise (Hansen et al., 2006; Storz et al., 2006). For example,

1. The need to comply with health and safety or disabilities legislation, which can constrain the citing of equipment and place certain usability requirements for disabled users (for guidance on how to design inclusively for all users and design assistive technologies for those with disabilities in particular, the reader is referred to Clarkson et al., 2003).
2. To be sensitive to data protection legislation, which may impact what data you can store, whether users have the right to opt-in, opt-out, or declare (e.g., with notices) that the system is in operation. Public deployments are by their very nature public, so you should prepare to be accountable for your system and prepare yourself, your team, and your work for public scrutiny.
3. Environmental factors (including weather, pollution, etc.) can have a devastating effect on equipment that is not adequately protected. It is worth doing test installs before your main deployment to uncover unexpected issues due to environmental factors (particularly important for external and outdoor deployments).
4. Privacy and organizational sensitivity. The nature of putting technologies into real-world situations can potentially open vulnerabilities (perceived or actual) to expose private information or interfere with existing systems or processes. This is particularly true for organizations managing sensitive data or in high-pressure situations, such as healthcare and emergency services. It is always worth approaching such situations responsibly and involving and addressing the concerns of local experts.

With any system, there is an ongoing cost in supporting the system that is proportional to the length of the deployment. Robust engineering and clever design can help mitigate this cost, but it is a research challenge in itself to drive this to zero and make the system self-maintaining. To keep down the impact of remote maintenance and support, you should ensure that it is possible to remotely monitor it, ideally as the user perceive sit (remote cameras and microphones can be extremely valuable, but are unpopular in many deployment settings). Remote access via the network is also important for resolving problems, especially if the system is inaccessible or far removed from the project team: it i easy to assume that the system will need less ongoing maintenance than perhaps it does, in fact, require, especially in the early phases of the project. If the system is physically inaccessible, then this is likely to cause problems going forward.

Particularly in unsupervised deployments, there is always a chance of unexpected or accidental intervention. Equipment that is installed and left in working order can sometimes find itself unplugged unexpectedly (e.g., by cleaners looking for a power socket or due to a power outage). You should realize that you cannot mitigate against all eventualities, but if your system requires complex manual setup or cannot be diagnosed and maintained remotely, then you are asking for trouble!

Runtime Orchestration of the Ambient Wood

Ambient wood was an augmented "ubicomp woodland" (Figure 2.6) designed to promote learning about woodland environments (Rogers et al., 2004). Mobile sensor devices allowed children to collect geo-tagged light and moisture readings; installed information appliances allowed information concerning tagged objects to be explored. The system also used a mesh of wirelessly connected devices installed in the wood to generate ambient sounds based on sensed contextual triggers. Technically, all devices synchronized their data to a shared dataspace ("Equip" middleware), allowing interactions in the woodland to later be visualized during supervised teaching sessions. This system was not designed for unsupervised operation or for longitudinal deployment. To maintain the quality of the experience for end users, a degree of orchestration was required to address any problems that arose during each teaching session in the wood. To help make orchestration easier (e.g., to introduce a new object representing a sound or piece of information, inspect readings being sent from devices, etc.), the developers integrated a multiuser dungeon (MUD) into the system to provide another interface onto the Equip data. Each area of the augmented woodland was represented in the MUD as a "virtual room." The team could easily use this interface and MUD metaphors to interactively "walk around" the representation of the experience and to remotely control it by inspecting, picking up, and dropping virtual objects. Orchestration of the configuration of hardware and software beyond the game was still largely a laborious manual process.

TABLE 2.1 Self-Check Questions to Consider before Undertaking a Real-World Deployment (Source: Hansen et al. (2006), IEEE Pervasive Computing, 5(3): 24–31.)

Categorie	Issues	Questions
Hardware	Cost, Security, Environment, Power, Network, Space, Safety issues	What will implementation cost? Will scaling up the system affect the price? Is special equipment needed? Is the equipment secure? Is there a risk of theft? Does the environment pose special requirements on the equipment? Is the system going to be used outdoors? Can it handle vandalism? Can it withstand being dropped or cleaned? Does the system require a power plug? How long can it run without being recharged? Do the batteries run flat if radio communication is used excessively? How do you recharge the system? How will the device communicate? Does it require an Ethernet connection? Is the wireless infrastructure in place? Do you need to transmit data in an external network? How much physical space does the system use? Is there space on the wall for large wall displays? Is there table space for another computer? Do the doctors have enough room in their pockets for another device? Is there space on the dashboard for another display? Will a system malfunction affect safety issues? What is the contingency plan in case of a full system crash? Will the system interfere with other systems? Can the system pose a threat to the user?
Software	Deployment and updates, Debugging, Security, Integration, Performance and scalability, Fault tolerance, Heterogeneity	How is the software transferred to the device? Does the deployment mechanism scale to a large number of devices? Can you update the system? How do you update the different devices? Are the devices accessible after deployment? If the system malfunctions, how do you find the error? Does the system store debugging information? How do you detect serious errors in the system? How is logging done? Does the system need to be secure? How does it keep information confidential and secure? Is there a concrete security risk? Is the

		deployed system stand-alone? Does it need to communicate with other deployed systems or integrate with third party systems? Is there a public API and converters for communicating between systems? How does the system perform? Is system performance acceptable in the real-world setting? How many devices are needed for deployment? Does the system scale? What happens when an error occurs? Can the system recover automatically? Can the daily system users bring the system back up to a running state? Is the developer team notified about errors? Is the system configured for remote support? Does the system run in a heterogeneous environment? Do heterogeneous elements need to communicate?
Using setting	Usability, Learning, Politics, Privacy, Adaptation, Trust, Support	Will end users use the system? If so, how many? Can the average user use the system? Does the interface pose problems? Does the system's overall usability match the average user? How do the users learn to use the system? Is it individual instruction or group lessons? Does the system need superusers? Is a manual or help function needed? How does the user get support? Who controls the system? Does the system change the power balance in the user setting? Who benefits from the system? Is the person that benefits from the system the same as the person that provides data to the system? Does the system require extra work from users? Does the system reveal private information? What kind of personal information does the system distribute and to whom? Is the organization ready for the system? Is there organizational resistance? Will the system change formal or informal structures in the organization? Does the user trust the system? Is the information given to users reliable? Who sends the information? Will the developers support the system? Does the support organization have remote access to the system?

2.4.2.1 Expect the Unexpected

In all deployments, the unexpected is the hardest thing to prepare for. Volatility is unfortunately endemic to the real world and hence to your ubicomp system (Coulouris et al., 2005). As a thought experiment and ideally during predeployment testing, consider how your system will react to

- Presence or use by unknown users
- Unrecognized devices (e.g., new phones and laptops)
- Changes to the wireless environment (new wireless networks)
- Devices being power cycled
- Batteries failing (particularly no moving parts or LEDs!)
- You not being there (it is easy to forget that the developer is present during development and testing, potentially impacting the sensing, wireless connectivity, etc.)
- Improper use (developers can quickly learn which interactions "break" the system and almost subconsciously adapt to avoid exercising these paths)

Anticipating these types of conditions, testing for them, and ideally having a strategy to deal with them will serve you well, particularly for longer-term deployments. Even something as simple as logging unexpected conditions to a persistent store can help with the posthoc diagnosis of "mysterious" system misbehavior.

2.4.3 Summary

Constructing ubicomp systems that not only meet the objectives and ambition of your project, but that are also sufficiently robust to be deployed for evaluation purposes, and in extremis, long-term use is very challenging.

Off-the-shelf hardware and software can be an expedient means of building ambitious systems more quickly, but do not come without strings attached: careful, qualified, and dispassionate evaluation of the choices and implications of those choices is called for. The end game as we strive for ubicomp “for real,” has to be moving toward daily and widespread use of ubicomp systems. Deployments, however, are not to be undertaken lightly; diligent preparation based on, for example, the above anticipated issues presented from past deployment-led projects will help you avoid many of the more common pitfalls. The next section concentrates on how to evaluate and learn from your built system.

2.5 EVALUATING AND DOCUMENTING UBICOMP SYSTEMS

In this section, we shall look more closely into how ubicomp systems can be evaluated and how the insight from your research can be documented and communicated.

2.5.1 Evaluating Ubicomp Systems

Evaluation of ubicomp systems and/or their smaller subcomponents needs to be carefully designed from the outset of a research project; different types of research contributions often need to be evaluated differently.

For example, routing protocols and their applicability under given circumstances can often be evaluated using network simulation tools, whereas systems support for smart room technologies would often involve real-world testing with end users. It is particularly important to ensure a tight coupling between the claims you make about your system and the evaluation methods that you use to demonstrate that these claims hold. If, for example, you claim that your network protocol scales to many nodes, a simulation is a reasonable evaluation strategy; but if you claim that the protocol supports biologists to easily pair devices in a deployment situation, this claim needs to be evaluated with biologists using the nodes (and their protocol) in a deployment field study.

Now, the observations above may seem trivial and obvious. Unfortunately, however, our experience is that it is exactly the discrepancy between claims made by researchers and their evaluation strategy, approach, and methods that often leads to criticisms of the designed systems.

Generally speaking, there are a number of approaches to evaluating ubicomp systems with varying degrees of ambition and required effort. A few important ones are introduced below.

2.5.1.1 Simulation

The design of a system can be modeled and subsequently simulated. For example, simulation is the research tool of choice for a majority of the MANET community. A survey of MANET research published in the premiere conference for the MANET community shows that 75% of all papers published used simulation to evaluate their research, and that the most widely used simulator is the Network Simulator (NS-2) (Kurkowski et al., 2005).

Once a system or a systems feature has been implemented, simulations can also be used to evaluate properties of the implementation. Simulations are typically used to evaluate nonfunctional systems qualities such as scalability, performance, and resource consumption. For example, the systems qualities of a ubicomp infrastructure for a smart room may be simulated by deploying it in a test setup where a number of test scripts are simulating the use of the infrastructure according to a set of evaluation scenarios.

While running the test scripts, the technical behavior of the infrastructure is gauged with respect to responsiveness, load balancing, resource utility, and fault tolerance. Such a technical simulation of a complicated piece of system infrastructure is extremely valuable in systems research, and helps discover and analyze various technical issues to be mitigated in further research. It is, however, important to recognize that this type of technical simulation says absolutely nothing about the infrastructure’s functional ability to, for example, support the creation of smart rooms, or about the usefulness and usability of the application built on top of it. To verify claims about the usefulness and usability of a system, you would need to make user-oriented evaluations.

2.5.1.2 Proof-of-Concept

Just as Marc Weiser coined the concept ubiquitous computing, he also described how these technologies were designed at Xerox Palo Alto Research Center (PARC) by building and experimenting with so-called proof-of-concepts. A proof-of-concept (PoC) was defined as The construction of working prototypes of the necessary infrastructure in sufficient quality to debug the viability of the system in daily use; ourselves and a few colleagues serving as guinea pigs. (Weiser , 1993) A PoC is a rudimentary and/or incomplete realization of a certain technical concept or design to prove that it can actually be realized and built, while also to some degree demonstrating its feasibility in a real implementation.

A PoC is not a theoretical (mathematical) proof of anything; it is merely a proof that the technical idea can actually be designed, implemented, and run. In analogy, even though Jules Verne introduced the concept of traveling to the moon in his famous 1865 novel, From the Earth to the Moon, the actual PoC was not designed, built, and run until a century later.

Creating PoCs is the most prevalent evaluation strategy in ubicomp systems research. The original work on the pad, tab, and wall sized ubicomp devices and their infrastructure at Xerox PARC is the classic example of this. But a wide range of other PoC examples exists, including ABC infrastructures for hospitals (Bardram and Christensen, 2007), different PoC for tour guiding systems such as GUIDE (Cheverst et al., 2000) and the San Francisco Museum Guide system (Fleck et al., 2002), home-based ubicomp systems such as EasyLiving (Brumitt et al., 2000), and ubicomp systems for smart rooms such as Gaia (Roman and Campbell, 2000), iRoom (Borchers et al., 2002), and iLand (Streitz et al., 1999).

However, looking at it from a scientific point of view, a PoC is a somewhat weak evaluation strategy. A PoC basically shows only that the technical concept or idea can be implemented and realized. Actually, however, a PoC tells us very little about how well this technical solution meets the overall goals and motivation of the research. For example, even if several PoCs of an ABC infrastructure have been implemented, this actually only tells us that it is possible to build and run a technical implementation of the underlying concepts and ideas. A PoC, however, does not tell us anything about whether it actually meets any of the functional and/or technical goals. For example, does the ABC framework support the highly mobile and collaborative work inside hospitals? Moreover, the PoC does not tell us anything about the nonfunctional aspects of the infrastructure: Does it scale to a whole hospital? Is the response time adequate for the life- and time-critical work in a hospital? Is it extensible in a manner that would allow clinical applications to be built and deployed on top of it? All of these questions can only be answered if the PoC is put under more rigorous evaluation.

2.5.1.3 Implementing and Evaluating Applications

A stronger evaluation approach is to build end user applications using ubicomp systems component and infrastructures, and then put these applications into subsequent evaluation. For example, the Context Toolkit was used to build several applications such as the In/Out Board and the DUMMBO Meeting Board. These applications can then be evaluated by end users in either a simulated environment or in a real-world deployment. For example, the ABC framework was used to implement a series of clinical applications, which was subsequently evaluated in a test setup where a hospital was simulated (Bardram and Christensen, 2007).

This evaluation approach is strong in several respects. First, using underlying systems technologies such as components, toolkits, or middleware infrastructures to build real applications, demonstrates that the systems components are indeed useful for building systems. Second, the act of building these applications helps the systems researcher to judge whether their building blocks actually help

the application developer meet his or her application goals. For example, how easy is it to model, capture, and distribute context information using the Context Toolkit?
Third, once the application is built and put into use, this provides a test bed for the underlying systems components and helps you answer more nonfunctional questions, such as: How well does the system scale, perform, and handle errors? For example, the implementation of clinical applications on top of the Java Context-Awareness Framework (JCAF) framework and the subsequent evaluation sessions helped the creators inspect how well the context-aware technology scaled to multiple concurrent users, and what happened when clients lost network connectivity.
The infrastructure and the application were put into pilot use in a hospital and evaluated over a 6-month period (Hansen et al., 2006). Figure 2.7 shows the use of context-aware public displays and smart phone in use.
However, as pointed out by Edwards et al. (2003), this evaluation strategy has its pitfalls and drawbacks. Essentially, if not carefully designed, the application and the subsequent evaluation may tell us little, if anything, about the systems aspect of the whole application. It is important to be absolutely clear about what your test application will tell you about your systems components, infrastructure, or toolkit. It is easy to get distracted by the demands of building useful and usable application in themselves and lose sight of the real purpose of the exercise, which may purely be to understand the pros and cons of the systems part. Moreover, whether or not the evaluation of an application turns out to be extremely successful may have very little to do with the systems properties of the application.
For example, an application may fail simply because of poor usability, or because it is so novel that users have a hard time actually using it. This failure, on the other hand, may say very little about the usefulness of the underlying systems support.

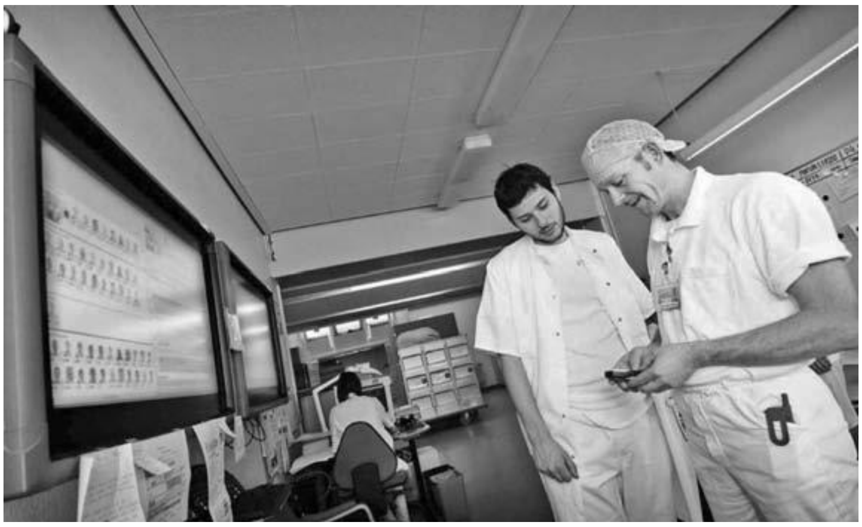
FIGURE 2.7 Context-aware technology deployed inside a hospital. (From Hansen et al., IEEE Pervasive Computing, 5(3), 24–31, 2006. With permission.)

2.5.1.4 Releasing and Maintaining Ubicomp Systems
The strongest evaluation of ubicomp systems components is to release them for third party use, for example, as open source. In this manner, the system research is used and evaluated by other than its original designers, and the degree to which the systems components helps the application programmers to achieve their goals directly reflects the qualities of the system components. One may even argue that there is a direct correlation between the number of application developers and researchers using the system in their work, and the value and merits of the work.
Releasing and maintaining systems software does, however, require a substantial and continuing effort. Releasing systems building blocks such as hardware platforms, operating systems, toolkits,

infrastructures, middleware, and programming APIs entail a number of things such as a stable and well-tested code base, technical, and API documentation; tutorials helping programmer to get started; example code and applications; and setting up licensing policies. And once the system has been released for third party use, issues of bug reporting and fixing, support, general maintenance, and new system releases need to be considered.
This is a real dilemma in systems research and evaluation. On the one hand, the best way to evaluate your systems research is to implement the idea in sufficient quality for the rest of the world to use it, and then continuously document, support, maintain, and evolve the technology. The degree to which the world adopts your technology is a direct indicator of its usefulness. On the other hand, this limits the amount of systems research that you can do within your career to a few contributions, and there seems to be an internal and external pressure for continuously moving on to new systems research challenges. But—without a doubt— designing, implementing, documenting, releasing, and evolving systems contributions for third party use is the golden bar in systems research and is a goal pursued and reached by many researchers. Section 2.6 provides pointers on released ubicomp systems research, which can be used in further research.

2.5.2 Learning from What You Build

All ubicomp systems are complex and time consuming to design, implement, and deploy. It is easy to expend all efforts of the project on creating, deploying, and evaluating the system, while neglecting to dedicate sufficient resources to communicating your findings and experiences (both positive and negative) to others. As attributed to Plutarch between AD 46 and 120, "Research is the act of going up alleys to see if they are blind." If we do not communicate, then others will be doomed to repeat our mistakes and not learn from our innovations. It is important not to waste your efforts by sharing software, datasets, and knowledge for others to build on (or even contribute to).

2.5.2.1 Communicating Your Findings

There are many ways to communicate with the community at large, and comparatively recent innovations such as open-source software projects and contributory resources such as "wikis" make it easier to put work online and marshal interested parties around initiatives. Still, it takes work to engage with a community and provide the resources they will need to be able to work with and/or contribute to your system. For this purpose, there are several approaches to use:

1. Making your system available enables others to try, critique, compare, adopt, and potentially contribute to your project (e.g., the iRos interactive workspaces software (Borchers et al., 2006) and equip rapid prototyping toolkit (Greenhalgh and Egglestone, 2005) are both available in source form). If you do not just put materials online, but try to make the initiative open source, then it's important that you remain responsive and keep the information up-to-date, at least while the project is in active development.
2. Publishing datasets is another effective means of providing resources for the community to build on and also invites the scientific practice of experimental validation through repeatability and comparative analysis of approaches (the CRAWDAD wireless traces (Kotz and Henderson, 2009) and Massachusetts Institute of Technology (MIT) PlaceLab Datasets (Intille et al., 2006) are good exemplars of this approach).
3. Publishing (e.g., online) schematics, instructions, and documentation also provides critical insight into how to reconstruct experiments and follow on from your work [e.g., Multitouch table (Schmidt, 2009), Smart-Its (Smart-ITs, 2001)].
4. Traditional academic routes of dissemination (papers, magazine articles, demonstrations, workshop participation, etc.), which provide a means to obtain valuable peer feedback on your work.

As with any packaging of the work, be it open source or commercialization activities, any such effort should be undertaken advisedly. It takes effort to seed these initiatives, for example, creating documentation, putting up example code, instructions and tutorials, making public versions, choosing appropriate licenses, etc. However, one has to question whether it is valid to undertake the research without considering and budgeting for evaluating it and communicating your findings.

2.5.2.2 Rigor and Scientific Communication

Ubicomp systems are always difficult to describe due to their complexity and wide-ranging lessons that one accrues during a typical project. Not everything that becomes a time sink is worth communicating; conversely, it is easy to forget the many problems and compromises that have been overcome or bypassed that may hint at important research questions worth detailing. Keeping a laboratory notebook as you progress can be a valuable resource when writing papers and dissertations.

Academic forms of dissemination (e.g., papers) do not always value experience reports or negative results as much as they should. This, in turn, has a tendency to encourage some researchers to focus on positive contributions of their work and why it is new or different from existing approaches at the expense of objectivity; it is far more common in other disciplines to repeat experiments and validate the work of other scientists, rather than focus on novelty and differentiation. Your work should be grounded in the literature; it is definitely acceptable to learn and build on the work of others, and it is acceptable to stress the commonalities as well as the differences. If there is genuinely nothing new to learn from your proposed project, then you have to question whether your objectives are correct; an early search of the literature is particularly important for this reason. For a thought-provoking discussion of experimental methods in Computer Science, the interested reader is directed to Feitelson (2005).

2.5.3 Documenting Ubicomp Systems

This last section shall present what, in our experience, is the best way to document ubicomp systems research and what needs to be addressed. This may work as an outline of your technical documentation as well as some basic directions for writing good ubicomp systems papers for the research community.

- Explain the specific (systems) question and challenge that you are addressing.
- Enumerate and explain the assumptions you make—both technical as well as any assumptions on the developers' and users' behalf.
- Carefully relate your work to others, paying special attention to where your work extends the work of others, and where it differs. Your work may differ in several areas, but it is important to highlight a few significant differences that constitute your main contributions.
- Divide documentation into
    - Technical documentation—contains all the technical details on the system, its implementation, and evaluation.
    - Research paper describing the overall research approach, questions, contribution, concepts, and technical innovation—always be careful not to include trivia or irrelevant implementation details; refer to the technical document if necessary.
- Describe your evaluation—especially why the system was evaluated in the manner with which it was conducted—addressing the following issues:
    - Evaluation strategy and overall approach.
    - The aim of the evaluation, including a description of how to measure it. Outline evaluation criteria and how to measure success.
    - Evaluation setup, including technical setup, configuration, runtime environment, simulation parameters, users, their background, the physical setup, etc.

    - Results of the evaluation, including measurable results such as time measurements of performance, throughput, and resource contribution, as well as qualitative results such as user feedback based on interviews, observations, and questionnaires.
- Discuss the contribution of the system as related both to the results of the evaluation as well as to the results from others.

In technical and scientific documentation, it is important to maintain objectivity and honesty when reporting results and findings. Try to avoid unnecessary adjectives and provide a prosaic description. Carefully present and discuss what can be learned from your research and the results you have obtained. Documenting and reporting on apparently negative results may entail a contribution in itself; it may be an inspiration for others to try to address this particular challenge or it may be associated with a flaw in the evaluation setup, which can be fixed once discovered.

2.5.4 Corollary

On a final note, it is worth keeping the scientific mindset to the fore in order to cultivate a scientific and balanced approach in describing your work. Scientists should be uncertain, open-minded, skeptical, cautious, and ethical—readers will question your work and will be cautious to accept your claims without appropriate evidence and grounding with respect to other approaches. Balanced and objective self-reflective analysis and evaluation of your work is crucial to its acceptability by others and particularly by the best quality conferences and journals. Evaluations must be methodologically sound and include adequate explanation of how they were conducted, because this is important for confidence in the quality of the results and trustworthiness of the inferences drawn from them.
Results and lessons should also be clearly and concisely presented. Try, if possible, to “quantify the magic” (Barton and Pierce, 2006) that made your system work for the setting and users you chose; understanding the scope and limitations of your system and how seriously these might impact the generalization of your work is important for setting the boundaries and research questions for further work in that area.
A simple guiding principle is: “What does the reader learn from reading my paper?” If a paper lacks useful insight, lessons, or results, then it is highly likely to be rejected.

2.6 GETTING STARTED

If I have seen further than others, it is by standing upon the shoulders of giants Many ubicomp systems research projects have put tools, toolkits, and datasets into the public domain. Here are a few examples of tools that we, as experimental scientists and designers of next-generation ubicomp systems, can download and evaluate. These can provide a quick route to getting your ideas up and running and allowing low-cost experimentation with ubicomp systems. You should feel positively encouraged to offer feedback to the creators, contribute to projects and dataset archives, and objectively compare your work with others in the domain; as a matter of principle, we can only benefit as a community from trying out each other’s systems and paradigms, and working together to address the many challenges ubicomp poses. In general, there are different types of technology that can help you realize your system and prototype your ideas:

1. Rapid prototyping tools for creating situated or mobile ubicomp systems
2. Libraries that can form components of your system, for example, handling computer vision, gesture recognition, processing sensor data, handling context
3. Hardware components including wireless sensors for augmenting artifacts or forming sensor networks

Given the typical lifetime of the average research project or these types of technology, this section merely aims to serve as an indicator of the types of systems available to you. More up-to-date resources should be kept in the public domain where they can be added to by active researchers, such as yourself (e.g., see http://ubisys.org).

2.6.1 Prototyping Your Ideas

There are many hardware and software platforms available to assist with deploying test ubicomp infrastructures to test your ideas and novel forms of interaction. Tools such as ActivityStudio (Li and Landay, 2008), exemplar (Hartmann et al., 2007), and iStuff Mobile (Ballagas et al., 2007) support the creation of low-fidelity functional prototypes that can be used to experiment with different ubicomp application designs. Each has a different focus: ActivityStudio provides an environment for moving from field notes, through a storyboarding and visual programming step through to simulation and in situ deployment of a Web-based prototype; exemplar encourages demonstration of sensor-based interactions (e.g., gestures) that are then filtered and transformed using a visual development environment to trigger other applications; and iStuff Mobile provides a visual programming interface for novel mobile phone–based interfaces (an otherwise notoriously difficult platform to develop for).

Systems such as the EQUATOR Component Toolkit (ECT) (Greenhalgh and Egglestone, 2005) and Wiring (Barragán, 2006) provide programmatic glue for constructing ubicomp systems that integrate sensing (input from sensors such as phidgets, motes, and d.tools boards), actuation (of physical actuators including X10 modules, output to Internet applications, etc.). ECT uses a visual graph-based editor to allow runtime interconnection of modules. The underlying EQUIP instances can support multimachine and distributed configurations. Wiring offers a high-level language based on the popular open-source visualization language Processing (Fry and Reas, 2001). Similar dataflow-like graphing metaphors are also exploited in Max/MSP (Zicarelli, 1997), a commercial system used by artists and designers to create interactive installations. Max uses a powerful graphical wiring metaphor (an interesting and flexible design in its own right) for connecting input and output components with channels that communicate messages. PureData (Puckette, 1996) and jMax (Cecco et al., 2008) are open-source derivatives of Max. Extensions to these (e.g., Digital Image Processing with Sound for jMax) allow real-time processing and transformation of video suitable for use in video-based installations and art pieces.

The Context Toolkit (Salber et al., 1999), Java Context-Awareness Framework (Bardram, 2005), and Context Aware Toolkit (CAT) (Prideau, 2002) allow sensors of context to be decoupled from higher level context reasoning in applications. Whereas the Context Toolkit and JCAF support ubicomp applications based on the integration of distributed context sensors, CAT does a similar job for embedded wearable devices.

Topiary (Li et al., 2007) and MyExperience (Froehlich, 2009) use context in a mobile environment to trigger interactions with the user. However, the aim of the two systems is quite different: Topiary's focus is low-fidelity contextual presentation of interactive design sketches to the user, whereas MyExperience is designed to ask the user a contextually relevant set of questions to survey them in situ (a methodology known as the experience sampling method).

2.6.2 Smart Room in a Box

If your aim is to create a smart environment populated with multiple displays and interaction devices, then Stanford's interactive workspaces spin off iROS (Borchers et al., 2006) is available as open source. A meta-operating system for creating interactive rooms, iROS includes a set of core middleware (Event Heap, DataHeap, iCrafter) for unifying machines and displays together to form larger interactive surfaces. The MeetingMachine (Barton et al., 2003; Barton, 2003) repackages iROS to create a shared networked appliance supporting the exchange, discussion, and collation of electronic documents in a meeting setting. Radically different approaches are taken by Plan B (Ballesteros et al., 2008), where smart environments are built on the Bell Labs Plan 9 operating system (Pike et al., 2003) and PCOM (Rothermel et al., 2006), a peer-to-peer component middleware for constructing pervasive applications.

2.6.3 Public Domain Toolkits

There are many useful libraries that can provide solutions to well-known algorithmic or integration problems, for example, integrating computer vision or detecting human activities.

2.6.3.1 Vision and Augmented Reality

A very common requirement is to integrate computer vision systems. OpenCV (OpenCV, 2009) provides over 500 algorithms for real-time vision processing. CANTag (DTG Research Group, 2005) supports the tracking of fiducial tags including their orientation and rotation, enabling a range of possible interaction gestures. Using a similar technique, the popular augmented and mixed-reality ARToolkit (Lamb et al., 2007) has been used to great effect for overlaying 3-D graphics onto similar fiducials. Building on this, the Designers Augmented Reality Toolkit (Macintyre et al., 2005) integrates this into an experience design environment based on Macromedia Director, simplifying augmented reality experience design.

2.6.3.2 Sensing

Ubicomp systems are often required to interpret sensor data to identify user interactions or human activities. Weka (Frank et al., 2008) is a collection of machine learning algorithms for preprocessing, classification, processing, and visualizing of data. Sensor networks increasingly underlie many ubicomp installations, particularly in the healthcare and emergency services domains. For example, DexterNet (Kuryloski et al., 2008) is an open framework for integrating wearable sensors for medical applications. It provides support for communicating with medical sensors (e.g., ECG), network support for communicating readings from the device, and a higher layer toolkit called SPINE (Giannantonio et al., 2008; Gravina et al., 2008), which helps simplify code development for embedded wearable sensors and deployment on arrays of sensor nodes.

2.6.3.3 Hardware

Many of the toolkits described above have been adapted to allow the integration of commodity hardware and tangible prototyping tools. Common ones include

- Wireless sensor nodes such as Motes (Crossbow, 2008), SunSpots (Sun Microsystems Laboratories, 2004), jStamps (Systronix, 2009), and µParts (Beigl et al., 2005)
- Interface prototyping boards such as the popular Phidgets (Phidgets, 2009), Arduino (Arduino, 2009), and d.tools (Hartmann et al., 2006) kits
- Wearable sensor boards for medical applications, for example, Harvard's CodeBlue (Welsh et al., 2008) and University of Alabama in Huntsville's Wearable Health Monitoring Systems (WHMS) (Otto et al., 2008)

Most of these have active communities developing tutorial materials and examples.

2.6.4 Datasets

A positive side effect of the standardization of some of these components (particularly sensor platforms) is that it becomes possible to repeat experiments and validate other people's findings. Datasets from such platforms are being increasingly collected online and are often open to contributions from other researchers. Examples of useful datasets already in the public domain include:

1. MIT's House_n PlaceLab (Intille et al., 2006) includes traces of human activity that have been used by the community to develop activity detection algorithms.

2. Intel’s Place Lab (Hightower et al., 2006; Lamarca et al., 2005) provides a freely available system for mobile localization, together with contributed location traces, which have been used to look at destination prediction, context-aware assistive technologies, and privacy. 3. Dartmouth’s CRAWDAD (Kotz and Henderson, 2009) is a community archive of wireless network traces that have been used for a wide range of uses including developing improved MAC layer protocols and location prediction.
3. Berkeley’s Wearable Action Recognition Database (Yang et al., 2008) and WHMS (Otto et al., 2008) activity traces for developing human action recognition systems based on wearable motion sensors.

The appearance and growth of initiatives such as these can help us collectively identify and solve common systems problems in ubicomp, enabling the field to move forward more rapidly.

2.6.5 Summary

Prototyping your ideas using available prototyping and smart room tools is a laudable approach for exploring the ubicomp design space and soliciting feedback. These types of tools help us commodify ubicomp systems, simplifying rapid creation of prototypes and broadening ubicomp experience. Ubicomp systems researchers are to be encouraged to use, refine, contribute to these initiatives, and start new ones as needed, to continue the technological dialogue that helps support our community. Exploiting and contributing new tools and datasets in the public domain can only serve to stimulate further research activity and promote increased adoption of scientific practices such as repeatability and comparison.

2.7 CONCLUSION

Systems research is central to ubicomp research and provides the fundamental building blocks for moving the field forward in terms of new applications and user experiences. As a research field, ubicomp must continue to build and evaluate systems components that ease the design, implementation, deployment, and maintenance of real-world ubicomp applications.
This chapter has outlined the special challenges pertaining to ubicomp systems and applications, including issues of designing systems that have to run in resource-constrained, volatile, and heterogeneous execution environments. But, in addition to these technical challenges to ubicomp systems design, the chapter has also tried to highlight that the special characteristics of ubicomp applications force systems researchers to address a whole new set of systems challenges, including the need to design for fluctuating environments and circumstances, and invisible computing. To a large degree, the assumptions that contemporary personal and clientserver computing relies on, breaks down in a ubicomp environment.
The chapter then moved on to discuss how to create ubicomp systems, putting emphasis on the experimental nature of systems design and implementation. Advice on how to implement and deploy ubicomp systems was given with reference to concrete ubicomp technologies and projects. Special emphasis was placed on evaluating and documenting ubicomp systems research; it is essential for the research community that ubicomp systems research is properly evaluated and documented in order to move the field forward. Evaluation of ubicomp systems is far from easy, and the chapter offers advice on how to conduct evaluation under specific conditions, including the use of simulation, proof-of-concepts, end user application building and evaluation, and technology releases to the research community. Similar advice is offered on how to document ubicomp systems research, both technically and scientifically.
With this chapter, we hope that researchers are motivated to engage in creating systems support for the ubicomp application area and, with the chapter in hand, have some specific pointers and tools for

engaging in this research. After all, the ubicomp systems research field is still in its infancy, and there is ample space for new exciting systems innovations.

REFERENCES

1. Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H., 2002, The case for cyber foraging, in Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, ACM, New York, NY, pp. 87–92.
2. Bardram, J. E., and Christensen, H. B., 2007, Pervasive computing support for hospitals: An overview of the activity-based computing project. IEEE Pervasive Computing 6(1): 44–51.
3. Bardram, J. E., Baldus, H., and Favela, J., 2006, Pervasive computing in hospitals, in Pervasive Healthcare: Research and Applications of Pervasive Computing in Healthcare, CRC Press, Boca Raton, FL, pp. 49 –78.
4. Bardram, J. E., and Schultz, U. P., 2004, Contingency management, Palcom Working Note #30, Technical report, Palcom Project IST-002057.
5. Bardram, J. E., 2004, Applications of context-aware computing in hospital work— examples and design principles, in Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 1574–1579.
6. Barton, J., and Pierce, J., 2006, Quantifying magic in ubicomp system scenarios, in Ubisys 2006: System Support for Ubiquitous Computing Workshop, 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006), Orange County, CA, USA, September 17–21.
7. Benford, S., Seagar, W., Flintham, M., Anastasi, R., Rowland, D., Humble, J., Stanton, D., Bowers, J., Tandavanitj, N., Adams, M., Farr, R. J., Oldroyd, A., and Sutton, J., 2004, The error of our ways: The experience of self-reported position in a location-based game, in Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, September, pp. 70–87.
8. Borchers, J., Ringel, M., Tyler, J., and Fox, A., 2002, Stanford interactive workspaces: A framework for physical and graphical user interface prototyping, Wireless Communications, IEEE (see also IEEE Personal Communications) 9(6): 64–69.
9. Boucher, A., Steed, A., Anastasi, R., Greenhalgh, C., Rodden, T., and Gellersen, H., 2003, Sensible, sensable and desirable: A framework for designing physical interfaces, Technical report, Technical Report Equator-03-003.
10. Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S., 2000, EasyLiving: Technologies for intelligent environments, in Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing, Bristol, UK, 25–27 September, pp. 12–29.
11. Chalmers, M., MacColl, I., and Bell, M., 2003, Seamful design: Showing the seams in wearable computing, IEEE Seminar Digests (10350), 11–16.
12. Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C., 2000, Experiences of developing and deploying a context-aware tourist guide: The GUIDE project, 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, MA, August, ACM Press, New York, NY, pp. 20–31.
13. Clarkson, P. J., Coleman, R., Keates, S., and Lebbon, C., 2003, Inclusive Design: Design for the Whole Population, Springer, London.
14. Coulouris, G., Dollimore, J., and Kindberg, T., 2005, Distributed Systems: Concepts and Design, 4th ed., Addison-Wesley, Reading, MA.
15. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D., 1987, Epidemic algorithms for replicated database maintenance, in PODC ’87: Proceedings of the 6th annual ACM Symposium on Principles of Distributed Computing, ACM Press, New York, NY, pp. 1–12.
16. Edwards, W. K., Bellotti, V., Dey, A. K., and Newman, M. W., 2003, Stuck in the middle—The challenges of user-centered design and evaluation for infrastructure, in CHI ’03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, NY, pp. 297–304.
17. Feitelson, D. G., 2005, Experimental computer science: The need for a cultural change, Technical report, School of Computer Science and Engineering, Hebrew University, Jerusalem.
18. Fleck, M., Frid, M., Kindberg, T., O’Brien-Strain, E., Rajani, R., and Spasojevic, M., 2002, Rememberer: A tool for capturing museum visits, in Proceedings of UbiComp 2002: Ubiquitous Computing, pp. 379–385.
19. Flintham, M., Anastasi, R., Hemmings, T., Crabtree, A., Greenhalgh, C., and Rodden, T., 2003, Where On-Line Meets on-the-Streets: Experiences with Mobile Mixed Reality Games, ACM Press, New York, NY, pp. 569–576.
20. Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., and Zhang, L., 1997, A reliable multicast framework for light-weight sessions and application level framing, IEEE/ACM Transactions on Networking 5(6): 784–803.
21. Fox, A., Davies, N., de Lara, E., Spasojevic, M., and Griswold, W., 2006, Real-world ubicomp deployments: Lessons learned, IEEE Pervasive Computing 5(3): 21–23.
22. Friday, A., Roman, M., Becker, C., and Al-Muhtadi, J., 2005, Guidelines and open issues in systems support for ubicomp: Reflections on ubisys 2003 and 2004, Personal and Ubiquitous Computing 10: 1–3.
23. Friday, A., Davies, N., Seitz, J., and Wade, S., 1999, Experiences of using generative communications to support adaptive mobile applications, Kluwer Distributed and Parallel Databases Special Issue on Mobile Data Management and Applications 7(3): 319–342.
24. Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P., 2002, Project Aura: Toward distraction-free pervasive computing, IEEE Pervasive Computing 2(1): 22–31.

25. Gaver, W. W., Beaver, J., and Benford, S., 2003, Ambiguity as a resource for design, in CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, NY, pp. 233–240.
26. Hansen, T. R., Bardram, J. E., and Soegaard, M., 2006, Moving out of the lab: Deploying pervasive technologies in a hospital, IEEE Pervasive Computing 5(3): 24–31.
27. Hightower, J., and Borriello, G., 2001, Location systems for ubiquitous computing, Computer 34(8): 51–66.
28. Kephart, J. O., and Chess, D. M., 2003, The vision of autonomic computing, IEEE Computer 36(1): 41–50.
29. Kindberg, T., and Fox, A., 2002, System software for ubiquitous computing, Pervasive Computing IEEE 1(1): 70–81.
30. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., and Spasojevic, M., 2002, People, places, things: Web presence for the real world, Mobile Networks and Applications 7: 365–376.
31. Kistler, J. J., and Satyanarayanan, M., 1992, Disconnected operation in the coda file system, ACM Transactions on Computer Systems 10(1): 3–25.
32. Kurkowski, S., Camp, T., and Colagrosso, M., 2005, MANET simulation scenarios: The incredibles, ACM Mobile Computing and Communications Review (MC2R) 9(4): 50–61.
33. Lamarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., and Schilit, B., 2005, Place Lab: Device positioning using radio beacons in the wild, in Proceedings of the 3rd International Conference on Pervasive Computing, May. Langheinrich, M., 2001, Privacy by design— principles of privacy-aware ubiquitous systems, in Ubicomp 2001: Ubiquitous Computing, vol. 2201, Lecture Notes in Computer Science, pp. 273–291, Springer Verlag, Berlin.
34. Ponnekanti, S., Johanson, B., Kiciman, E., and Fox, A., 2003, Portability, extensibility and robustness in iROS, in Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), March, pp. 11–19.
35. Rogers, Y., 2006, Moving on from Weiser's vision of calm computing: Engaging ubicomp experiences, UbiComp 2006: Ubiquitous Computing 4206: 404–421.
36. Rogers, Y., Price, S., Fitzpatrick, G., Fleck, R., Harris, E., Smith, H., Randell, C., Muller, H., O'Malley, C., Stanton, D., Thompson, M., and Weal, M. J., 2004, Ambient wood: Designing new forms of digital augmentation for learning outdoors, Third International Conference for Interaction Design and Children (IDC 2004), ACM Press, New York, NY, pp. 1–9.
37. Roman, M., and Campbell, R. H., 2000, Gaia: Enabling active spaces, in EW 9: Proceedings of the 9th Workshop on ACM SIGOPS European Workshop, ACM Press, New York, NY, pp. 229–234.
38. Roman, R., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K., 2002, A middleware infrastructure for active spaces, IEEE Pervasive Computing 1(4): 74–83.
39. Satyanarayanan, M., 2001, Pervasive computing: Vision and challenges, Personal Communications IEEE 8(4): 10–17.
40. Salber, D., Dey. A. K., and Abowd, G. D., 1999, The context toolkit: Aiding the development of context-enabled applications, CHI'99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, NY, pp. 434–441.
41. Schuler, D., and Namioka, A. (Eds.), 1993, Participatory Design: Principles and Practices, Erlbaum Associates, Hillsdale, NJ.
42. Storz, O., Friday, A., Davies, N., Finney, J., Sas, C., and Sheridan, J., 2006, Public ubiquitous computing systems: Lessons from the e-campus display deployments, IEEE Pervasive Computing 5: 40–47.
43. Streitz, N. A., Geissler, J., Holmer, T., and Konomi, S., 1999, ILand: An interactive landscape for creativity and innovation, in Proceedings of the ACM Conference on Human Factors in Computing Systems: CHI 1999, ACM Press, New York, NY, pp. 120–127.
44. Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H., 1995, Managing update conflicts in bayou, a weakly connected replicated storage system, in SOSP '95: Proceedings of the 15th ACM Symposium on Operating Systems Principles, ACM Press, New York, NY, pp. 172–182.
45. Weiser, M., 1991, The computer for the 21st century, Scientiﬁc American 265(3): 66–75.
46. Weiser, M., 1993, Some computer science issues in ubiquitous computing, Communications of the ACM 36(7): 74–84.
47. Zambonelli, F., Jennings, N. R., and Wooldridge, M., 2003, Developing multiagent systems: The Gaia methodology, ACM Transactions on Software Engineering and Methodology 12(3): 317–370.

URLs

1. Arduino Project, 2009, Arduino: An Open-Source Electronics Prototyping Platform, http://arduino.cc/.
2. Ballagas, R., Memon, F., Reiners, R., and Borchers, J., 2007, iStuff mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous Computing, http:// research.nokia.com/people/tico_ballagas/istuff_mobile.html.
3. Ballesteros, F. J., Soriano, E., Guardiola, G., de las Heras, P., de Mingo Gil, S., Higuera, O., Lalis, S., and Garcia, R., 2008, Plan B: An Operating System for Distributed Environments, http://lsub.org/ls/planb.html.
4. Rogers, Y., Price, S., Fitzpatrick, G., Fleck, R., Harris, E., Smith, H., Randell, C., Muller, H., O'Malley, C., Stanton, D., Thompson, M., and Weal, M. J., 2004, Ambient wood: Designing new forms of digital augmentation for learning

outdoors, Third International Conference for Interaction Design and Children (IDC 2004), ACM Press, New York, NY, pp. 1–9.

5. Roman, M., and Campbell, R. H., 2000, Gaia: Enabling active spaces, in EW 9: Proceedings of the 9th Workshop on ACM SIGOPS European Workshop, ACM Press, New York, NY, pp. 229–234.
6. Roman, R., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K., 2002, A middleware infrastructure for active spaces, IEEE Pervasive Computing 1(4): 74–83.
7. Satyanarayanan, M., 2001, Pervasive computing: Vision and challenges, Personal Communications IEEE 8(4): 10–17.
8. Salber, D., Dey. A. K., and Abowd, G. D., 1999, The context toolkit: Aiding the development of context-enabled applications, CHI'99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, NY, pp. 434–441.
9. Schuler, D., and Namioka, A. (Eds.), 1993, Participatory Design: Principles and Practices, Erlbaum Associates, Hillsdale, NJ.
10. Storz, O., Friday, A., Davies, N., Finney, J., Sas, C., and Sheridan, J., 2006, Public ubiquitous computing systems: Lessons from the e-campus display deployments, IEEE Pervasive Computing 5: 40–47. Streitz, N. A., Geissler, J., Holmer, T., and Konomi, S., 1999, ILand: An interactive landscape for creativity and innovation, in Proceedings of the ACM Conference on Human Factors in Computing Systems: CHI 1999, ACM Press, New York, NY, pp. 120–127.
11. Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H., 1995, Managing update conflicts in bayou, a weakly connected replicated storage system, in SOSP '95: Proceedings of the 15th ACM Symposium on Operating Systems Principles, ACM Press, New York, NY, pp. 172–182.
12. Weiser, M., 1991, The computer for the 21st century, Scientiﬁc American 265(3): 66–75.
13. Weiser, M., 1993, Some computer science issues in ubiquitous computing, Communications of the ACM 36(7): 74–84.
14. Zambonelli, F., Jennings, N. R., and Wooldridge, M., 2003, Developing multiagent systems: The Gaia methodology, ACM Transactions on Software Engineering and Methodology 12(3): 317–370.
15. Arduino Project, 2009, Arduino: An Open-Source Electronics Prototyping Platform, http://arduino.cc/. Ballagas, R., Memon, F., Reiners, R., and Borchers, J., 2007, iStuff mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous Computing, http:// research.nokia.com/people/tico_ballagas/istuff_mobile.html.
16. Ballesteros, F. J., Soriano, E., Guardiola, G., de las Heras, P., de Mingo Gil, S., Higuera, O., Lalis, S., and Garcia, R., 2008, Plan B: An Operating System for Distributed Environments, http://lsub.org/ls/planb.html.
17. Kotz, D., and Henderson, T., 2009, CRAWDAD: Community Resource for Archiving Wireless Data at Dartmouth, http://crawdad.cs.dartmouth.edu/.
18. Kuryloski, P., Giani, A., Giannantonio, R., Gilani, K., Gravina, R., Seppa, V.-P., Seto, E., Shia, V., Wang, C., Yan, P., Yang, A., Hyttinen, J., Sastry, S. S., Wicker, S., and Bajcsy, R., 2008, Dexternet: An Open Platform for Heterogeneous Body Sensor Networks and Its Applications, Tech. Rep. UCB/EECS-2008 174, EECS Department, University of California, Berkeley, December, http:// www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-174.html.
19. Lamb, P., Looser, J., Grasset, R., Pintaric, T., Woessner, U., Piekarski, W., and Seichter, H., ARToolKit: A Software Library for Building Augmented Reality (AR) Applications, http://www.hitl.washington.edu/artoolkit/.
20. Li, Y., and Landay, J., 2008, ActivityStudio: Design and Testing Tools for Ubicomp Applications, http://activitystudio.sourceforge.net/.
21. Li, Y., Hong, J., and Landay, J., 2007, Topiary: A Tool for Prototyping Location- Enhanced Applications, http://dub.washington.edu:2007/topiary/.
22. Macintyre, B., Bolter, J. D., Gandy, M., and Dow, S., 2005, DART: The Designers Augmented Reality Toolkit, http://www.cc.gatech.edu/dart/.
23. OpenCV Project, 2009, OpenCV (Open Source Computer Vision), http://opencv. willowgarage.com/wiki/.
24. Otto, C., Milenkovic, A., Sanders, C., and Jovanov, E., 2008, WHMS: Wearable Health Monitoring Systems, http://www.ece.uah.edu/~jovanov/whrms/.
25. Phidgets Inc., 2009, Phidgets: Products for USB Sensing and Control, http://www. phidgets.com/.
26. Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H., and Winterbottom, P., 2003, Plan 9 from Bell Labs., http://plan9.bell-labs.com/ plan9/.
27. Prideau, J., 2002, CAT: Context Aware Toolkit, http://www.cs.uoregon.edu/research/wearables/CAT/.
28. Puckette, M. S., 1996, Puredata (PD), http://puredata.info/.
29. Rothermel, K., Becker, C., Schiele, G., Handte, M., Wacker, A., and Urbanski, S., 2006, 3PC: Peer 2 Peer Pervasive Computing, http://3pc.info.
30. Schmidt, D., 2009, Multi-Touch Table, http://eis.comp.lancs.ac.uk/~dominik/ cms/, accessed Feb. 27, 2009.
31. Smart-ITs, 2001, http://www.smart-its.org/, accessed Feb. 27, 2009.
32. Sun Microsystems Laboratories, 2004, Sun SPOT: Sun Small Programmable Object Technology, http://www.sunspotworld.com/.
33. Systronix Inc., 2009, JStamp: Java Embedded Processor, http://jstamp.systronix.com/.
34. Welsh, M., Wei, P. G.-Y., Moulton, S., Bonato, P., and Anderson, P., 2008, CodeBlue:
35. Wireless Sensors for Medical Care, http://fiji.eecs.harvard.edu/CodeBlue.

36. Yang, A., Kuryloski, P., and Bajcsy, R., 2008, WARD: A Wearable Action Recognition Database, http://www.eecs.berkeley.edu/~yang/software/WAR/.
37. Zicarelli, D., 1997, Max/msp/jitter, http://www.cycling74.com/.