Physical Prototyping with Smart-Its

Exploring novel ubiquitous computing systems and applications inevitably requires prototyping physical components. Smart-Its are hardware and software components that augment physical objects with embedded processing and interaction to address this need.

Hans Gellersen, Gerd Kortuem, and Albrecht Schmidt
Lancaster University, UK

Michael Beigl
University of Karlsruhe

Most traditional computing systems and applications research treats software as the variable and hardware as the constant. In ubiquitous computing research, it's difficult to maintain this perspective. A defining characteristic of ubiquitous computing systems is embodying physical context and interaction with a given physical locale rather than abstracting from it.[1] Generally, ubiquitous computing is based on a trend that leads away from general-purpose hardware for applications and user interfaces toward more task-specific and varied devices.[2]

So, designing software for ubiquitous computing applications becomes entwined with designing the physical system components. Researchers who build ubiquitous computing systems to explore their properties increasingly must prototype hardware. This creates a quandary, because researchers interested in the challenges and potential of ubiquitous computing typically come from software-centric backgrounds such as distributed systems and user interface engineering. Such researchers can't develop ubiquitous computing applications without suitable hardware, but designing suitable hardware requires understanding how the hardware fits into the application environment.[3]

We aim to lower the hurdle for physically prototyping ubiquitous computing systems by facilitating the study of systems composed of everyday artifacts augmented with physical I/O and wireless networking. Our work, which uses small computing devices called *Smart-Its*, addresses the need to create embedded interactive systems that disappear from the foreground to become secondary to the physical objects with which people interact during everyday activities. Such systems create new design challenges related to prototyping with embedded technologies and require careful consideration of the physical design context.

Embedded ubiquitous computing systems

New research efforts to develop *augmented artifacts* have accentuated the need for physical prototyping. (For more information on prototyping ubiquitous computing systems, see the related sidebar). Augmented artifacts are physical objects commonly associated with purposes other than computing but imbued with wireless communication, physical interaction, and processing. Researchers are exploring the combination of physical objects and digital behavior in both novel user interface devices (such as tangi-

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/04/$20.00 © 2004 IEEE

Prototyping Ubiquitous Computing Systems

Building and using system prototypes to develop an understanding of ubiquitous computing design issues was fundamental to the pioneering work of Mark Weiser and his colleagues at the Palo Alto Research Center.[1] System prototypes have since become a principal research approach in this emerging area. As ubiquitous computing engages with novel device concepts, with new interactions between devices and their physical environment, and with embedding of devices in everyday objects and structures, this commonly involves prototyping physical system components alongside developing communication, system, and user-interaction software. Practically all projects that have made influential contributions to this emerging area have involved some hardware development, although their investigators typically haven't been interested in advancing hardware but in studying the software systems and user-interaction challenges.

Hallmark ubiquitous computing projects such as the ParcTab experiment[1] and the ActiveBadge project[2] have relied on prototyping and strong hardware development capabilities. In contrast, research groups who approach ubiquitous computing from a more software-centric background often resort to off-the-shelf devices or hardware improvisation. For instance, handhelds and PDAs are frequently used to mimic special-purpose information appliances; a classic example for a "hardware hack" is using TV remote controls with taped-down buttons to implement infrared beacons—an ingenious workaround to facilitate positioning and location awareness in the Cyberguide project.[3]

Obviously, such approaches avoid the problem of having to build new hardware and are valuable for trying out application concepts. Yet there's a risk that design decisions are driven by the capabilities of the adopted, rather than the envisioned, devices. Even more critical, software-centric approaches severely limit the investigation of post-development issues such as user experience and runtime behavior (energy use, communication performance, robustness, and so forth). We need to evaluate these issues under realistic conditions, which requires using software and hardware that conform as closely as possible to the final design. So, a strong need exists in ubiquitous computing for design methods and tools for physical prototyping of combined hardware and software artifacts.

REFERENCES

1. M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Comm. ACM*, vol. 36, no. 1993, pp. 74–83.
2. A. Harter and A. Hopper, "A Distributed Location System for the Active Office," *IEEE Network*, vol. 8, no. 1, 1994.
3. S. Long et al., "Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study," *Proc. 2nd ACM Int'l Conf. Mobile Computing and Networking* (MobiCom 96), ACM Press, 1996, pp. 97-107.

ble-interface objects and ambient displays) and in enhanced versions of everyday artifacts (tables, chairs, glasses, and so on).[4,5] Because such artifacts have established meaning and purpose in people's lives, it's compelling to build on them to afford additional functionality while preserving familiarity and ease of use. We can tie augmented artifacts directly into software processes to overcome the media break between the physical flow of activity and the related flow of information. These artifacts offer one way of realizing user interfaces that are embedded in the ambient environment and that facilitate interaction with information as part of everyday activities.

A hindrance to this vision is that we lack a suitable platform for supporting the augmentation of everyday objects. Augmented artifacts create a particular set of development problems: they require embedded computing technology that integrates sensing capabilities, a processing environment, and wireless networking. Furthermore, because the embedded technology should be unobtrusive, we must impose certain constraints related to the physical dimensions and to energy management. Few existing technologies and tools support these kinds of embedded ubiquitous computing systems (see the "Related Work" sidebar).

So, to successfully create embedded ubiquitous computing systems in general and augmented artifacts in particular, we need tools that make it easier to build physical prototypes of artifacts that embody processing, communication, sensing, and physical interaction.

The Smart-Its architecture

The Smart-Its project aims to develop a platform for context-aware embedded systems and augmented artifacts (see www.smart-its.org). Smart-Its are small, embedded context-aware devices that integrate sensing, actuation, processing, and communication (see Figure 1). They're customized for each application and exist in many different configurations. The hardware design allows for a large degree of flexibility in terms of the type and number of sensors and actuators. In addition, developers can tailor Smart-Its to support different means of communication including wired and wireless networks. A flexible hardware design, combined with high-level software abstractions and development tools, lets developers rapidly design new device configurations.

The Smart-Its design's flexibility lets developers use Smart-Its for a variety of purposes. For example, we've developers from various institutions have used them to construct tangible-interface devices, "smart" active environments, and spe-

Related Work

European research initiatives that investigate ubiquitous computing from a human-centered perspective have strongly influenced the Smart-Its project. These initiatives constitute an interdisciplinary approach that brings together technologists, interaction designers, and social scientists. Notable efforts include the Disappearing Computer initiative, an umbrella for 17 pan-European projects (www.disappearing-computer.net), and the Equator project, an interdisciplinary collaboration of eight UK institutions (www.equator.ac.uk). Many research groups engaged in these projects are using Smart-Its as a prototyping platform.

Many ubiquitous computing researchers share the need for physical prototyping. Saul Greenberg has investigated this problem in the context of physical user interfaces, developing the Phidgets hardware and software toolkit (www.cpsc.ucalgary.ca/grouplab/phidgets).[1] The toolkit is based on the architecture of GUI applications but provides developers with powerful abstractions to include physical-interface objects. In contrast to the Smart-Its platform, Phidget objects are tethered and designed for use with a single desktop computer.

Another physical interface toolkit is iStuff[2] developed at Stanford University's Interactivity Lab (see http://istuff.stanford.edu). It's a toolbox of wireless, platform-independent, physical-interface components that can be connected to software services that the iRoom intelligent-room infrastructure can provide.

John Barton has taken an altogether different approach, looking at the prototyping problem in the broader context of how mobile devices interact in ubiquitous computing environments. He built Ubiwise, a simulation environment that lets users take yet-to-be-built devices into a virtual reality environment to explore device interactions.[3] Although the Smart-Its project addresses similar physical-prototyping concerns, it has a unique focus on the augmentation of everyday objects.

As small computing devices that integrate physical interaction and wireless networking, Smart-Its share similarities with the Berkeley Motes,[4] which have come into widespread use for research on wireless ad hoc sensor networks.[5] Motes are becoming more widely considered for ubiquitous computing application but were originally developed to collect sensor data in large-scale networks (www.xbow.com/Products/Wireless_Sensor_Networks.htm). So, they place more emphasis on networking, communication, and data propagation. To support this, Motes provide the TinyOS operating system for very small networked devices and a protocol stack that supports ad hoc routing in large networks—features that aren't available in Smart-Its. Smart-Its, on the other hand, provide distinct support for rapid assembly and design iterations over device concepts.

Other small computing platforms similar to Smart-Its include the EYES sensor node (http://eyes.eu.org), which is being developed by a European project consortium for sensor network research, and the BTNode, developed at The Swiss Federal Institute of Technology, Zurich (www.tik.ee.ethz.ch/~beutel/bt_node.html). The BTNode is a Bluetooth-based sensor node that stresses interoperability.[7] To support Bluetooth networking, BTNodes use a more powerful processor and more memory than Smart-Its.

REFERENCES

1. S. Greenberg and C. Fitchett, "Phidgets: Easy Development of Physical Interfaces through Physical Widgets," *Proc. ACM Symp. User Interface Software and Technology* (UIST 01), ACM Press, 2001, pp. 209–218.
2. R. Ballagas et al., "iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments," *Proc. Conf. Human Factors in Computing Systems* (CHI 03), ACM Press, 2003, pp. 537–544.
3. J. Barton and V. Vijayaraghavan, *Ubiwise: A Ubiquitous Wireless Infrastructure Simulation Environment*, tech. report HPL-2002-303, HP Labs, 2002; www.hpl.hp.com/techreports/2002/HPL-2002-303.pdf.
4. I. F. Akyildiz et al., "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, 2002, pp. 393–422.
5. J. Hill et al., "System Architecture Directions for Networked Sensors," *Proc. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 2000), ACM Press, 2000, pp. 93–104.
6. J. Beutel et al., "Prototyping Wireless Sensor Network Applications with BTnodes," *Proc. 1st European Workshop Wireless Sensor Networks* (EWSN), Springer-Verlag, 2004, pp. 323–338.

cialized wearable devices. Developers have also used them for post hoc augmentation of physical artifacts: when attached to an artifact, Smart-Its can perceive their own state and the environment, communicate with peers in ad hoc networks, and interface with other infrastructures and services. This approach follows the "disappearing computer" philosophy, which places computing in the background of people's interaction with their physical and social environment (see www.disappearing-computer.net).

A Smart-Its system is a collection of communicating Smart-Its devices (see Figure 2). Such a system might be self-contained (that is, consisting exclusively of Smart-Its) or connected through gateways to an external infrastructure (the Internet, servers, or global communication systems). The gateways themselves are realized by dedicated Smart-Its.

Smart-Its communicate with each other over a wireless short-range ad hoc network that uses message broadcasting as a primary communication mode. The network protocol realizes an asynchronous-communication model and is designed to minimize the network stack's

code size. Each Smart-It has a unique hard-coded ID, used as a network identifier. Most communication in a Smart-Its system is local and involves message passing between neighboring nodes.

Smart-Its devices follow a modular design to support flexible configurability. They map the core functionalities (sensing, actuation, computation, and communication) onto separate hardware modules—the *core board* and the *sensor board* (see Figure 3). The core board handles communication; the sensor board handles physical I/O. Computation can be centralized on the core board, in which case a single processor drives both communication and physical interaction. Or, computation can be distributed using an additional processor on the sensor board, which would drive I/O components and process sensor data. Although the typical configuration foresees a core and single sensor board back to back, it's also possible to connect multiple sensor boards to a core board.

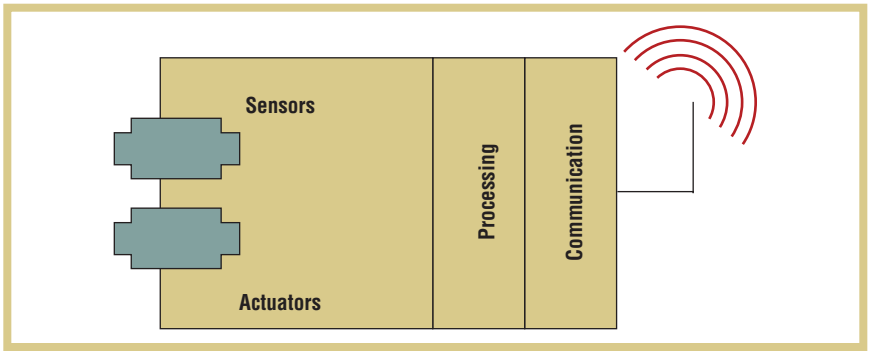

Figure 1. Smart-Its integrate physical I/O devices with a processing environment and wireless communication.

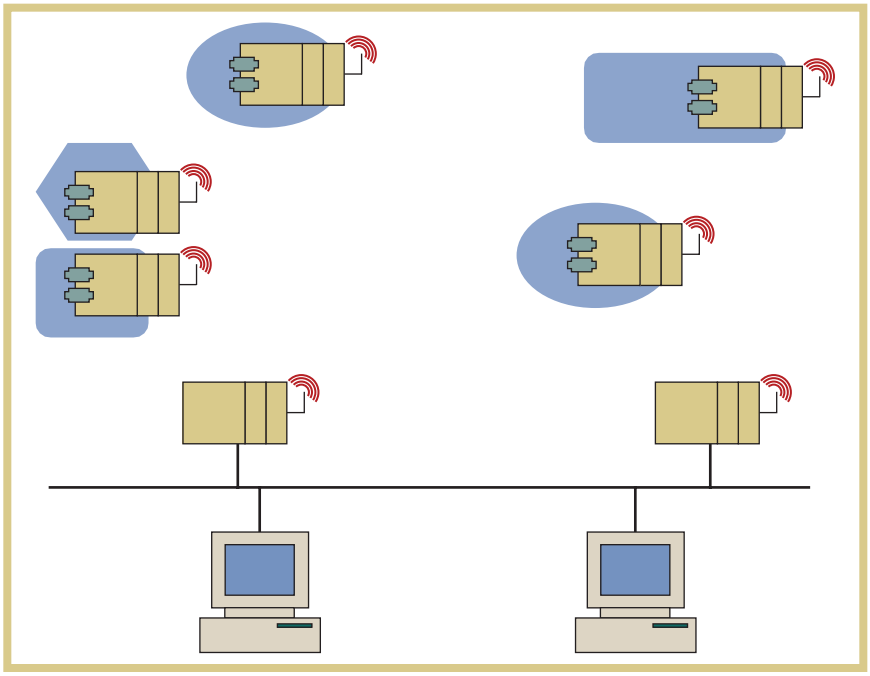

Figure 2. S Smart-Its system composed of multiple individual Smart-Its. The shaded areas indicate different Smart-Its types (they are configured with a different combination of sensor and actuators). The lower two Smart-Its are connected to a back-end infrastructure.

The hardware platform

The Smart-Its platform is available in two different versions, implementing the same underlying archteicture: *Lancaster* and *TecO* Smart-Its. Lancaster Smart-Its are designed for fast prototyping, while TecO Smart-Its are designed for system deployment.

Lancaster Smart-Its are an extensible toolbox of hardware modules designed to prototype augmented artifacts (see www.comp.lancs.ac.uk/eis/projects/smartits). We developed these Smart-Its as an experimental platform to support rapid iterations over the physical and digital design of augmented artifacts. The primary design rationale is to achieve short cycles between design idea and lab prototype and to ease reproduction for outside developers. So, we purposefully adopted a simple implementation of the device architecture for the Lancaster hardware: a core board provides processing and communication, and add-on boards integrate physical-interaction components. The Lancaster core board is based on a PICmicro microcontroller (from Microchip Technology) that offers processing, analog inputs, and digital I/O. Additional components are a ferroelectric RAM chip with 8 KBytes of nonvolatile memory and a transceiver module for wireless communication operating on 433.92 MHz, with raw data rates of up to 160 KBits/s in half duplex mode. The core boards are available in a compact form factor for embedded operation and in a slightly bigger version that provides a serial interface and a larger battery pack.

The Lancaster add-on boards are connected over a standard interface to a core board to provide physical-interaction components that are controlled by the core microprocessing unit (MPU). In the

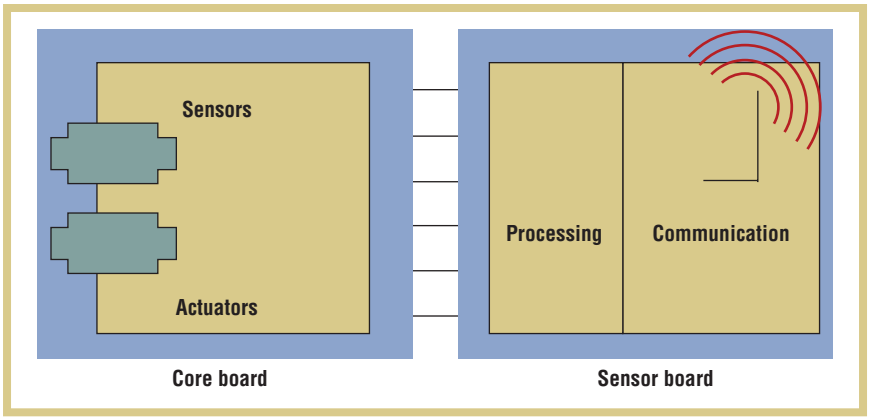

Figure 3. The Smart-Its device architecture is modular and separates physical interaction from wireless communication to support flexible configuration.

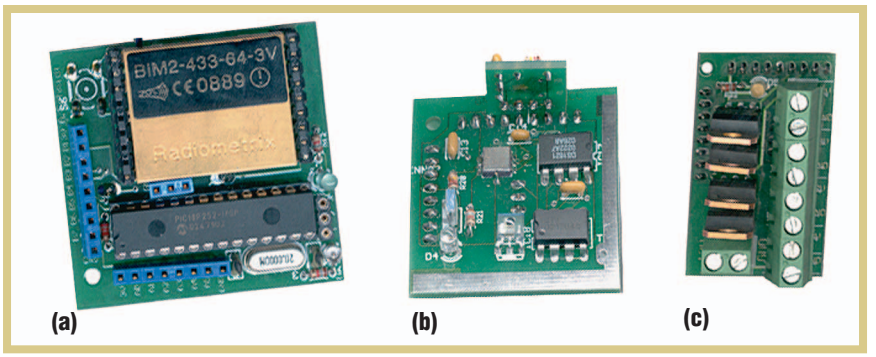

Figure 4. Lancaster Smart-Its are based on hardware modules that are purposefully simple: (a) a core board with microcontroller and radio; examples of (b) an add-on sensor board and (c) an actuator board.

simplest case, an add-on board contains a single sensor, but more commonly it integrates a group of sensors or actuators. A variety of add-on boards are available as part of a growing toolkit, and developers can build additional ones by adapting a generic board design or by designing dedicated boards. Figure 4 shows a Lancaster core board in the center and two examples of Lancaster add-on boards—a sensor board with touch, acceleration, light, and temperature sensors and a power control board for up to four actuators.

TecO Smart-Its are designed for system deployment and are thus at a comparatively higher level of integration than Lancaster Smart-Its (see smart-its.teco.edu). Their hardware constitutes a different implementation of the device architecture, paying more attention to deployment concerns such as size, energy efficiency, and scalable communication. They use smaller components and tighter packaging, making the devices particularly small, light-weight, and unobtrusive (see Figure 5). TecO Smart-Its implement physical interaction in a sensor board that has its own MPU and a core set of sensors, actuators, and interfaces for extension. This lets us use the MPU on the core board for more elaborate and reliable communication protocols.

The core board's main components are the MPU (20 MHz PIC Processor 16F87x), a transceiver for wireless communication operating at 868.35 MHz, a power supply supporting different types of batteries, and a software-controlled field-strength regulation unit. The board further contains basic physical I/O (a movement switch, LED, and buzzer) and various hardware interfaces to connect boards, sensors, and a serial line. The sensor boards contain their own MPU and RAM for autonomously computing context information from sensor observations. The boards integrate sensors for audio, light level, acceleration, pressure, and temperature. Additional components are a piezo loudspeaker, LEDs, and various interfaces and connectors for integrating additional physical-interaction components.

We designed TecO and Lancaster hardware as complementary development platforms. Although we can use both interchangeably to develop Smart-Its applications (requiring only small software modifications), they aim to support different development needs and phases.

The Smart-Its development environment

The Smart-Its development environment assists with the development of Smart-Its systems and applications. It integrates a collection of software and hardware tools to support programming, testing, and maintenance. These tools include a cross-compiler (a standard C compiler for the Microchip PIC platform), a utility for wirelessly transferring code to embedded Smart-Its, and debugging tools based on monitors for network and sensing activity.

The development process

Developing embedded ubiquitous computing applications with Smart-Its involves two major phases: concept and software development (see Figure 6).

Concept development involves iteration over design ideas and early prototypes. It aims to develop an application idea until it's mature enough to shift attention to how a system will realize it. For ubiquitous computing scenarios, this involves device prototyping—assembling

devices to better understand how they fit the application idea. Lancaster Smart-Its, because of their flexibility in building devices, support this part of the process. Software components are prototyped in C on the basis of customized code templates. Developers select an appropriate code template depending on the architecture of their Smart-Its application (stand-alone, serial connection to a PC, or wireless connection to a base station), and insert and parameterize code fragments from libraries (for sensor control, basic data processing, and communication). Hardware abstraction and basic systems functions are embodied in these libraries, and customized code runs directly on the hardware. Each concept development cycle concludes by evaluating a working prototype. The evaluation focuses on the application idea and how well the device concept fits it—investigating system properties is a separate concern.

The ensuing system development phase further develops Smart-Its applications into a deployable system. This phase is underpinned by the TecO Smart-Its platform, which doesn't cater as well for rapid assembly but provides device modules better geared for embedded operation beyond the lab. Moreover, the TecO platform provides a communication subsystem with application interfaces for asynchronous nonblocking and synchronous blocking of calls and control of wireless network behavior such as transmission range (software-controlled regulation of field strength). On this basis, the development focus shifts from hardware (which might still be customized, for instance with different sensors) to communication design and application software development (programming in C). In principle, code is interchangeable between the Lancaster and TecO platforms, but some adaptations are required to account for the different radio communication systems.

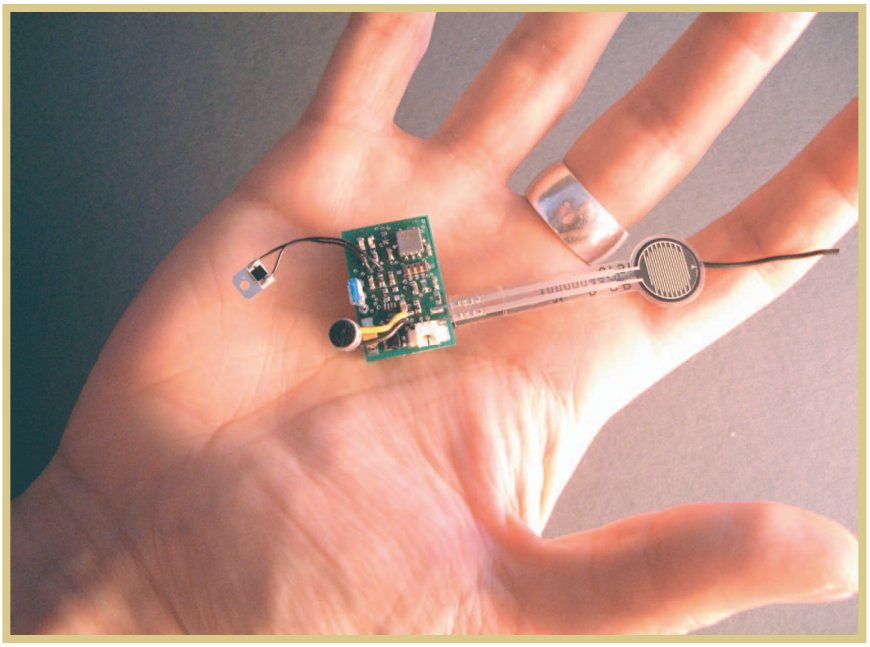

Figure 5. TecO Smart-Its modules are optimized for small scale. This device measures 17¥ 30 mm with sensors for audio; light levels; triple-axis acceleration; humidity, temperature and pressure; and LED and speaker output.

This is an open development framework. Although Lancaster and TecO Smart-Its work hand-in-hand, it's possible to perform either concept or system development with other technologies. For example, system development with dedicated sensor hardware could follow concept prototyping with Smart-Its.

Smart-Its software

Smart-Its are programmed for single applications. Developers can repurpose Smart-Its by replacing the executable code (the TecO Smart-Its support this wirelessly with an over-the-air programming facility). System functionality, such as managing subtasks for physical I/O, communication, and event processing, is compiled into application code before deployment on Smart-Its hardware; however, this remains largely transparent to the developer.

Developers can use C libraries and tools that provide programming abstractions at multiple levels. Drivers shield the developer from hardware detail, both on the level of components (sensors, actuators, memory, power supply, and processor) and of boards (physical connections between components). At the application level, application interfaces and code libraries are provided for the set of tasks a Smart-Its application typically performs. These tasks include acquiring sensor data, converting sensor data into

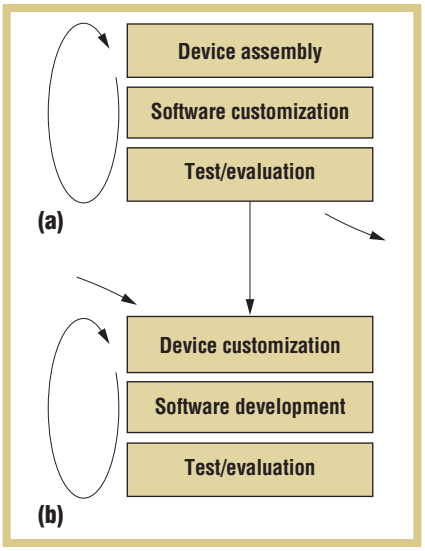

Figure 6. Smart-Its development involves (a) iteration over device and application concepts, followed by (b) iteration over system and application detail.

```
if (get_audiosamples(20))                    //200ms sampling
{
  a=calc_audio_volume();                     // calculate avg. volume from values
  if (a >10) RED_LED_ON;                     // sound detected? ->Red light
  else RED_LED_OFF;
  AUDIO_ADD(a);                              // add this context to outgoing pkt
  ACLsendPacket(50);              // and send it non-blocking
}
```

Figure 7. Smart-Its software: a small program that computes volume level from an audio sample and communicates this to peer devices.

application-specific contexts, communicating context events to other devices, and reacting to context events. The code example in Figure 7 illustrates Smart-Its software with a small program that computes volume level from an audio sample and communicates this to peer devices.

Applications

Researchers have used the Smart-Its platform in a variety of projects to build and investigate embedded ubiquitous computing systems and networked interactive artifacts. The diversity of case studies illustrates the Smart-Its platform's power and flexibility. In addition, we've assessed Smart-Its as a prototyping tool through a series of multiday developer workshops.

Case studies

The Smart-Its platform facilitates experimental investigation of applications that are deployed into physical objects. In the *Cooperative Artefacts* project, we're exploring augmented artifacts that cooperatively reason about their situation in the world. In a concrete case study, we've used Smart-Its to build prototypes of augmented chemical containers that can detect and alert potentially hazardous situations.[6] The embedded Smart-Its are configured with sensors to measure the proximity of containers, and they're programmed to share and evaluate knowledge across containers—for instance, on type and mass of stored chemical. In related work, Smart-Its have been used to augment artifacts with an electronic seal that asserts their authenticity and physical integrity.[7]

In a different application setting, one concerned with personal artifacts, we developed the *Smart-Its Friends* interactive technique.[8] We aimed to find an easy way for users to establish digital links among their personal artifacts. One target scenario included linking credit cards to a personal key chain so that the credit card would function only in the presence of the keys, for added security. The interaction technique we designed exposed artifacts to the same hand gesture as a uniquely shared context for mutual identification. We used Smart-Its to develop this technique, prototyping devices that share acceleration data when held and shaken—obviously, a case in which we needed untethered sensing devices small enough to allow such interaction. In a similar use of Smart-Its, we've explored sensor-based interaction devices in different form factors, including a cube and a wearable pendulum.[9,10]

Another case study is weight surfaces. These are common surfaces such as floors, tabletops, and shelf boards augmented with sensors to measure the applied load and pressure. Weight surfaces let us track activity and acquire context unobtrusively.[5] The underlying sensing infrastructure is based on Smart-Its with customized add-on boards to connect four load cells embedded in each surface. The particular advantage of Smart-Its in this case is wireless data collection and connectivity across surfaces with technology that remains effectively invisible (see Figure 8). In related work, we've demonstrated how Smart-Its can more generally be employed to rapidly build environments with embedded intelligence.[11]

Researchers have also used Smart-Its to investigate how plans and actions might become more closely coupled on the basis of embedded sensing. Researchers performing a study in the context of furniture assembly replaced paper-based instructions with proactive guidance based on multisensor observation of the actual assembly process.[12] They built a demonstrator with Smart-Its devices attached to parts of a wardrobe and to tools (a screw driver and hammer) to collectively sense which stage the assembly was in. Using LEDs, the system highlights problems and suggests the next most appropriate action at any point in time.

Developer workshops

To assess Lancaster Smart-Its as a rapid prototyping platform, we held four developer workshops in which participants could develop simple context-aware embedded devices. Forty participants from more than 10 different institutions attended the two-to-three-day workshops. The participants were involved in ubiquitous computing research initiatives such as *The Disappearing Computer* on the European level, *Equator* in the UK, and *Daphne* in Sweden (see the "Related Work" sidebar). Most participants had a computer science background but weren't skilled in hardware design or electrical engineering.

We organized the workshops around small teams of two to three developers. Each team chose a project idea and then developed the required hardware and software. At a minimum, this involved constructing a sensor add-on board for the Lancaster Smart-Its and implementing software for context acquisition, interpretation, and communication. To ease hardware construction, we provided participants with a visual step-by-step manual. To ease software construction, we supplied the source code of a generic Smart-Its application. For larger projects requiring multiple devices, we gave the teams several ready-made Smart-Its.

Figure 8. We can (a) transform a regular coffee table into an interactive and context-sensitive device by (b) unobtrusively embedding Smart-Its and load sensors.

The workshops resulted in a variety of fully implemented device concepts:

- A wireless radio frequency identification tag reader (with a reader unit built into a Smart-Its add-on board)
- A remote control recognizing a small repertoire of hand-gestures (built with a Smart-It with multiple accelerometers)
- A "smart ball" that generates sound effects when being thrown and caught (with an embedded Smart-Its device customized to detect these actions)

Although simple in many respects, these examples highlight the ease with which inexperienced developers can build embedded context-aware and interactive artifacts using the Smart-Its platform. None of the participants had practical experience in hardware design, yet all succeeded in building their first functional Smart-Its device in one day. In two to three days, they had constructed more complex systems composed of multiple Smart-Its.

Our experiences with the workshops indicate that the Smart-Its platform facilitates rapid prototyping of embedded ubiquitous computing hardware and software. The advantages are

- *Significantly reduced development time*: Instead of several weeks, developers can construct system demonstrators within days.
- *Significantly reduced knowledge and required skills*: Even developers without expertise in hardware design can construct complex sensor-based context-aware devices and systems.
- *Support for reuse*: Developers can easily reuse existing hardware and software components for new systems; similarly, they can exchange hardware and software components between systems.
- *Support for adaptation and modification*: Developers can easily modify existing Smart-Its by adding or removing sensors together with the respective software modules.

We achieve these advantages using

- A modular hardware design that separates the major design concerns (sensing, actuators, context interpretation, and communication) into separate hardware boards
- A template-based software design that lets developers assemble applications from several predefined modules
- Simple, yet powerful APIs for commonly required application functionality

However, we also identified some limitations. One concern is wireless networking's unreliability when the density of communicating Smart-Its within a space is high. Another limitation is the relatively poor support for building data and communication-intensive distributed applications. The Smart-Its platform doesn't yet provide adequate architectural support and high-level abstractions for distributed applications. This shortcoming is a major thrust of current research activities in the Smart-Its project.

Because Smart-Its significantly reduce the time it takes to build initial working prototypes, developers can further explore alternatives and iterative improvements in the physical design. In our own approach to exploring ubiquitous computing systems, physical prototyping support has let us break more easily with common assumptions and consider new architectures for interactive environments.

In its present implementation, Smart-Its constitutes a starting point on which we can base further research into architectural support for system and application developers. One direction we're investigating is rule-based programming of Smart-Its systems. In this approach, application logic is modelled in terms of facts and rules that provide a higher level of abstraction to the programmer. We're working on an embedded inference engine and algorithms for distributed rule processing. P

ACKNOWLEDGMENTS

The Smart-Its project was supported by the Commission of the European Union under key action "Future and Emerging Technologies" as part of "The Disappearing Computer" research initiative (contract IST-2000-25428). We're also receiving support from the UK Engineering and Physical Science Research Council (EPSRC) as part of the Equator interdisciplinary research collaboration.

REFERENCES

1. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 70–81.

2. R. Want et al., "Disappearing Hardware," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 36–47.

3. J. Barton and V. Vijayaraghavan, *Ubiwise: A Ubiquitous Wireless Infrastructure Simulation Environment*, tech. report HPL-2002-303, HP Labs, 2002; www.hpl.hp.com/techreports/2002/HPL-2002-303.pdf.

1/3 fill here

the AUTHORS

Hans Gellersen is a professor for interactive systems at Lancaster University, UK. He's also the coordinator of the European Smart-Its project and a principal investigator in the Equator project in the UK. His research interests include ubiquitous computing, embedded interactive systems, and context-aware systems. He earned his PhD in computer science from the University of Karlsruhe. Contact him at the Computing Dept., Lancaster Univ., Lancaster LA1 4YR, UK; hwg@comp.lancs.ac.uk.

Gerd Kortuem is a lecturer in computer science at Lancaster University, UK. His research interests include user interface technologies, ubiquitous computing, wearable computing, and software engineering. He received his PhD in computer science from the University of Oregon. He's a member of the IEEE Computer Society and ACM. Contact him at the Computing Dept., Lancaster Univ., Lancaster LA1 4YR, UK; kortuem@comp.lancs.ac.uk.

Albrecht Schmidt is the head of the Embedded Interaction research group in the Computer Science Department at the University of Munich (Ludwig-Maximilians-Universität München). His primary research interest is in novel user interfaces and interaction methods for mobile and ubiquitous computing. He received his PhD in computer science from the University of Lancaster, UK. Contact him at LFE Medieninformatik, LMU München, Amalienstrasse 17, D-80333 München, Germany; albrecht.schmidt@ifi.lmu.de.

Michael Beigl is a research associate and the head of the TecO research group at the University of Karlsruhe. His research interests evolve around people at the center of communication and information technology, with a specific interest in novel information appliances and artifacts, mobile and ubiquitous networks, human-computer interaction, and context-aware systems. He received his PhD in computer science from the University of Karslruhe. Contact him at Telecooperation Office (TecO), Vinc.-Prießnitz-Str. 1, D-76131 Karlsruhe, Germany; michael@teco.uni-karlsruhe.de.

4. A. Schmidt et al., "Context Acquisition Based on Load Sensing," *Proc. 4th Int'l Conf. Ubiquitous Computing* (Ubicomp 02), LNCS 2498, Springer-Verlag, 2002, pp. 333–350.

5. A. Schmidt et al., "Ubiquitous Interaction: Using Surfaces in Everyday Environments as Pointing Devices," *Proc. 7th ERCIM Workshop User Interfaces for All* (UI4ALL 02), LNCS 2615 Springer-Verlag, 2003, pp. 263–279.

6. M. Strohbach et al., "Cooperative Artefacts: Assessing Real World Situations with Embedded Technology," *Proc. 6th Int'l Conf. Ubiquitous Computing* (Ubicomp 2004), LNCS 3205, Springer-Verlag, 2004, pp. 249–266.

7. C. Decker et al., "eSeal—A System for Enhanced Electronic Assertion of Authenticity and Integrity of Sealed Items," *Proc. 3rd Int'l Conf. Pervasive Computing* (Pervasive 04), LNCS 3001, Springer-Verlag, 2004, pp. 254–268

8. L.E. Holmquist et al., "Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artifacts," *Proc. 3rd Int'l Conf. Ubiquitous Computing* (UbiComp 01), LNCS 2201, Springer-Verlag, 2001, pp. 116–122.

9. K. Van Laerhoven et al., "Using an Autonomous Cube for Basic Navigation and Input," *Proc. 5th Int'l Conf. Multimodal Interfaces* (ICMI/PUI 2003), ACM Press, 2003, pp. 203–211.

10. N. Villar et al., "Interacting with Proactive Community Displays," *Computers & Graphics*, vol. 27, no. 6, Elsevier, 2003, pp. 849–857.

11. L.E. Holmquist et al., "Building Intelligent Environments with Smart-Its," *IEEE Computer Graphics and Applications*, vol. 24, no. 1, 2004, pp 56–64.

12. S. Antifakos, F. Michahelles, and B. Schiele, "Proactive Instructions for Furniture Assembly," *Proc. 4th Int'l Conf. Ubiquitous Computing* (UbiComp 02), LNCS 2498, Springer-Verlag, 2002, pp. 351–360.