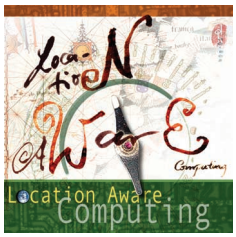
Implementing a Sentient Computing System



Sentient computing systems, which can change their behavior based on a model of the environment they construct using sensor data, may hold the key to managing tomorrow's device-rich mobile networks.

Mike Addlesee
Rupert Curwen
Steve Hodges
Joe Newman
Pete Steggles
Andy Ward
AT&T Laboratories Cambridge

Andy Hopper
AT&T Laboratories Cambridge and University of Cambridge

As computer users become increasingly mobile, and the diversity of devices with which they interact increases, so the overhead of configuring and personalizing these systems increases. A natural solution to this problem would be to create devices and applications that appear to cooperate with users, reacting as though they are aware of the context and manner in which they are being used, and reconfiguring themselves appropriately.

At AT&T Laboratories Cambridge, we have built a system that uses sensors to update a model of the real world. We designed the model's terms—object positions, descriptions and state, and so forth—to be immediately familiar to users. Thus, the model describes the world much as users themselves would. We can use this model to write programs that react to changes in the environment according to the user's preferences. We call this *sentient computing* because the applications appear to share the user's perception of the environment.[1]

Treating the current state of the environment as common ground between computers and users provides new ways of interacting with information systems. Metaphysical concerns aside, a sentient computing system doesn't need to be intelligent or capable of forming new concepts about the world—it only needs to act as though its perceptions duplicate the user's. For example, suppose a user picks up a wireless device that a sentient computing system manages. The system seems to be aware that this event has occurred, and it automatically configures the device to that user.

In earlier work, we described a prototype of this system and stated our intention to deploy it on a large scale.[2] We have now installed an enhanced version throughout an office building. Over the past year, approximately 50 staff members have used the system daily with a set of trial applications.

IMPLEMENTING A SENTIENT SYSTEM

Our project implemented the sentient computing system's model of the world as a set of software objects that correspond to real-world objects. Objects in the model contain up-to-date information about the locations and state of the corresponding real-world objects. Ascertaining object positions with near-human levels of accuracy requires a specially designed sensor system.

Location sensing

The location sensor, shown in Figure 1, determines the 3D positions of objects within our building in real time. Personnel carry wireless devices known as Bats, which can also be attached to equipment. The sensor system measures the time it takes for the ultrasonic pulses that the Bats emit to reach receivers installed in known, fixed positions. It uses these times of flight to calculate the position of each Bat and hence the position of the object it tags by triangulation.

To allow accurate time-of-flight measurements, a wireless, cellular network synchronizes Bats with the ceiling receivers. Base stations simultaneously address a Bat over the wireless link and reset the receivers over a wired network. A wireless back channel supports the Bat's transmission of registration, telemetry, and control-button information.

Current embodiment

We have installed the Bat system throughout our three-floor, 10,000-sq.-ft. office building, and all 50

0018-9162/01/$10.00 © 2001 IEEE

staff members use it continuously. The system uses 750 receiver units and three radio cells, and tracks 200 Bats.

Figure 2 shows the current Bat device. Each Bat measures 8.0 cm × 4.1 cm × 1.8 cm, has a unique 48-bit ID, and draws power from a single AA lithium cell. In addition to two input buttons, a Bat has a buzzer and two LEDs for output. Applications send commands over the wireless network to generate feedback via these devices.

Built around a DSP microprocessor, the location system receiver units use a matched-filter signal-detection approach. Receivers are wired together in a daisy-chain network and are installed unobtrusively above the tiles in our building's suspended ceilings.

To simplify maintenance of the location system, telemetry data can be obtained from the Bats, indicating current battery health and firmware version number. Further, system administrators can send commands over the wireless and wired networks to reprogram Bats and receivers in the field.

About 95 percent of 3D Bat position readings are accurate to within 3 cm. Each base station can address three Bats simultaneously, 50 times each second, giving a maximum location update rate across each radio cell of 150 updates per second. The signals from simultaneously triggered Bats are encoded using a differential-phase modulation scheme, allowing receivers to distinguish among them.

Scheduling and power saving

Because we use Bats to tag many mobile objects in the environment, we must fully exploit the limited number of location-update opportunities. We also take every opportunity to reduce the power consumption of Bats because frequently changing the batteries of several hundred devices would require significant effort.

Base stations use a quality-of-service measure to share location-update opportunities between Bats. The base stations can preferentially allocate location resources to objects that move often or are likely to be moved. A scheduling process that runs on each base station determines not only when the base station will address a particular Bat but also when it will *next* address it. The process passes this information to the Bat across the wireless link, allowing the Bat to enter a low-power sleep state for the intervening time. The scheduling algorithm and Bats incorporate mechanisms that allow rapid changes to the QoS allocations.[3]

Rapid changes to the schedule prove particularly useful when a user presses a Bat button. This action generally indicates that the user wishes to perform some task with the Bat. In these cases, it helps to obtain a low-latency position update for the Bat. The Bat transmits a button-press message over the wireless network, and the base station responds by scheduling an immediate location-update opportunity for that Bat.

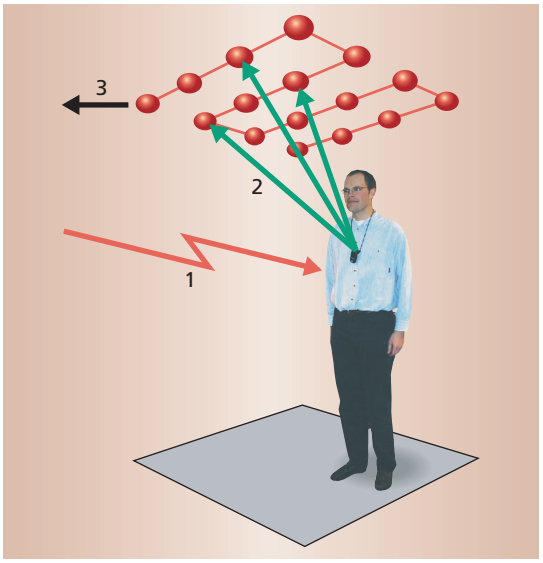


Figure 1. Operation of the Bat location sensor system. A Bat is triggered over a wireless link (1), which causes it to emit an ultrasonic pulse (2). Ceiling-mounted receivers measure the pulse's times of flight, and a controller retrieves the times of flight (3) over a wired network. The controller uses these measurements to calculate Bat-receiver distances and thus the Bat's 3D position.

Figure 2. Bat wireless tag device. Powered by a single AA lithium cell, each Bat has a unique 48-bit ID, two input buttons, and—for output—a buzzer and two LEDs.

Bats register with a base station using their unique 48-bit IDs and receive a temporary 10-bit ID local to the base station. The system subsequently refers to the Bat by its local ID, which it reclaims if the Bat moves out of the base station's range. This procedure limits the Bat population per base station, but results in shorter addressing messages and subsequent power savings.

Each Bat has a sensitive motion detector that lets it tell the base stations whether it is moving or stationary. Because the base station doesn't need to repeatedly determine the position of stationary objects, the system places nonmoving Bats into a low-power sleep state, from which they wake only when they move again. This design saves power and frees up location-update opportunities for allocation to other Bats.

Taken together, the Bat's low-power features result in a battery lifetime of around 12 months.

Indoor sensor alternatives

Alternative indoor-location sensors suitable for wide-scale deployment include other ultrasonic systems such

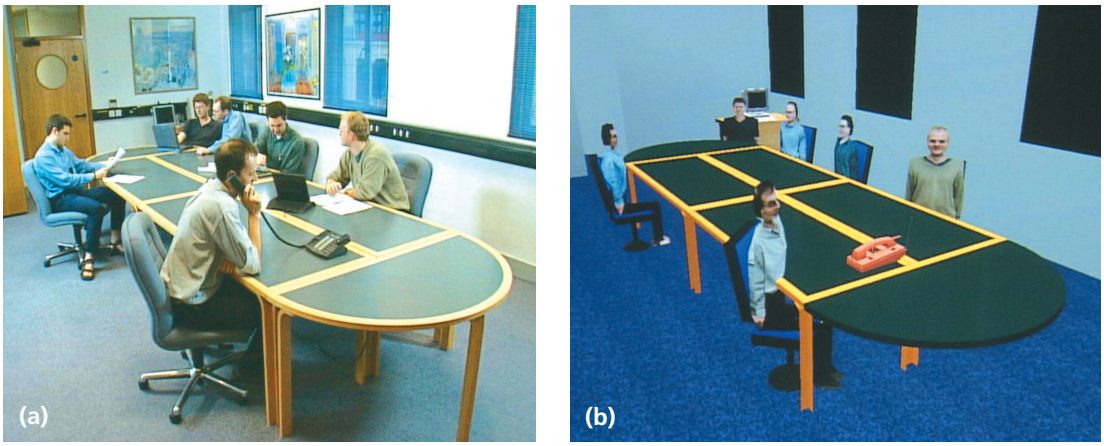


Figure 3. The world as seen by (a) users and (b) the sentient computing system.

as Cricket,[4] indoor radio systems such as PinPoint[5] and RADAR,[6] and infrared systems like Active Badge.[7,8] However, because none of these systems can provide 3D location data with an accuracy of less than one meter, the fidelity of a model constructed using data from them would be poor. Indeed, our experiences in developing and using the Active Badge system suggest that context-aware applications often require location information that has a much finer granularity.

In an outdoor environment, GPS[9] provides accurate position information relative to the large scale of outdoor features, such as roads and fields; thus, it comes close to the ideal of sharing perceptions on this scale. However, many sentient computing applications rely on information accurate to the human scale, which cannot be obtained using such sensors.

MODELING THE ENVIRONMENT

Our system uses data from its sensors and from services such as an Alcatel 4400 CTI telephone switch to update its world model. The model functions essentially as a distributed application programming interface (API) for the environment, allowing applications to control the environment and query its state. The model handles all system-level issues, including sessions, events, persistence, and transactions. We make each software object a Common Object Request Broker Architecture (Corba) object and store its persistent state in an Oracle database.[2] The model schema contains 80 different interface types, while the model of our building contains about 1,900 actual instances of physical objects.

The system uses knowledge of the tracked objects' dynamics to filter the incoming location data, and then uses the filtered data to update the world model. Because the model must mirror the user's perception of the world, the data must be accurate and robust, and must be updated with minimal latency. These requirements constrain our choice of filtering algorithms. We chose a method that follows these rules:

- Sensor errors impose low-amplitude noise on the reported positions of near-stationary objects, so the method heavily damps those streams of sightings that differ by only a few centimeters.
- The system cannot readily distinguish large errors from real object motions, because object accelerations are high relative to the sample rate. Therefore, we simply accept large changes in reported object position. If a large sensor error were to occur, this would result in a large error in the reported object position, but this is acceptable, because larger sensor errors become increasingly unlikely.
- If a reported change in object position implies an unreasonably high velocity, the method makes an exception to the last rule and discards the sighting.

We can further increase the fidelity of the model by determining if someone appears to be seated. The system looks for periods of constant-velocity motion above a certain speed. Seeing such motion, it assumes that the person is walking forward and is upright, and records the floor-to-Bat height. If the Bat ever drops below this level significantly, we infer that the user has taken a seat. Because the user's body shadows the Bat's ultrasonic signal, only receivers in front of the person will detect signals from the Bat (assuming that the person wears the Bat somewhere on the front of the body). Therefore, we can estimate the person's orientation from the Bat's position and the pattern of receivers that detected it.

Figure 3 shows a 3D visualization of the model, which indicates how the sentient computing system's view of the world accurately matches the environment's real state.

SOFTWARE SUPPORT FOR SENTIENT SYSTEMS

We have found that we require several common features in many of our sentient computing applications. It is sensible to integrate such services with the API provided by the world model, so that they become available to all applications.

Spatial monitor

A *spatial monitor* formalizes imprecise spatial relationships in terms of containment and overlapping relationships between suitable 2D spaces, as the "Spatial Monitoring—Programming with Space" sidebar describes. This allows developers to use an event-driven style of programming that treats spaces on the floor like buttons on a traditional GUI dis-

Spatial Monitoring—Programming with Space

Consider an application that moves a user's desktop between systems. This application would register with the spatial monitor, expressing interest in a circular space around a person, shaded green in Figure A, and an area in front of each display where the user could see the screen, shown as a blue-colored oval.

When in front of the screen, the area around the user is contained within the area in front of the screen. The spatial monitor detects this relationship and sends a positive containment event to the application. The application responds by displaying the user's desktop on that screen.

When the user moves away from the screen, the spatial monitor sends a negative containment event to the application, causing it to remove the user's desktop from the screen.

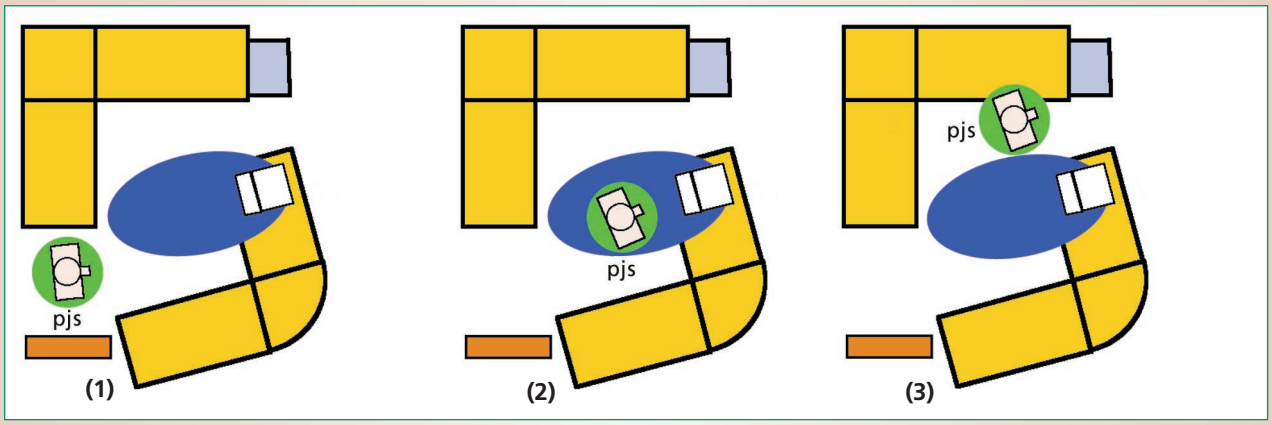


Figure A. Spatial monitoring application that moves users' desktops around with them. The application registers with the spatial monitor (1); as the user (pjs) approaches the display (2) or moves away from it (3), the spatial monitor sends a positive or negative containment event to the application that transfers or removes the desktop to or from the screen.

play, while people and other mobile objects become somewhat like mouse pointers.

In our building, the spatial monitor handles approximately 3,700 individual spaces. The monitor, world model, and underlying database all run on one Sun Ultra 250 workstation, which has 500 Mbytes of memory and two 300-MHz processors.

Timeline-based data storage

Sentient computing lets users personalize their network appliances. Many of these appliances generate data, which the system must store in a way that does not require the user to specify its destination. For example, if a user takes a photograph with a wireless camera, the computing system needs to automatically store the picture in such a way that the user can easily retrieve it.

Based on earlier work,[10] we developed a method that creates a timeline that contains the data each individual generates and also incorporates information from the model. Users can query this timeline with a temporal query language to retrieve data based on context.

APPLICATIONS

Sentient computing systems provide a wide range of location-aware applications for users.

Browsing

An important application class, model browsers simply display the environment's current state. In addition to the 3D visualization shown in Figure 3, we have also implemented the continuously updated map shown in Figure 4.

Figure 4. A 2D-map visualization of the world model. In Room 210, a user (djc) is sitting in front of his workstation while using his telephone, which is colored red to indicate that it is off the hook.

The map can display personnel, furniture, telephones, workstations, and other relevant information so that users can

- find a colleague by name;
- immediately find the phone nearest to a person, determine whether it is in use or not, then place a call by clicking on the phone displayed on the map;
- see, by the disposition of their neighbors on the map, whether to contact a person or not—for example, a person standing next to a visitor may not wish to be disturbed; and

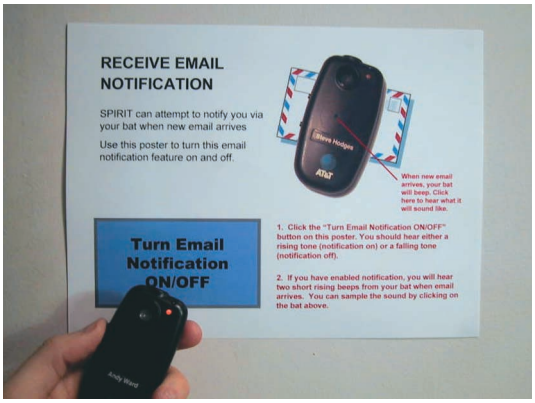


Figure 5. A smart poster. In this example, a user controls an e-mail notification service with a smart poster.

- request to be informed when a person leaves a room or arrives in the building, or to receive notification when the phone located nearest that person is not in use.

Follow-me systems

Application developers can make services ubiquitously available to "users" by moving their interfaces to the nearest appropriate input or output device, a technique we call *follow-me*.

A follow-me desktop application can display a user's Virtual Network Computing[11] desktop on the nearest computer screen, triggered by the buttons on the user's Bat.

When an external call targets the extension number of an absent user, the user's Bat makes a ringing sound. By pressing a button on the user's Bat, the user can have the call routed to the nearest phone. If the user chooses instead to ignore the call, the system will forward it to a receptionist.

We have installed several cameras in one room in our building. A user can request to be tracked by these cameras, and the system will create a video stream in which the nearest camera is selected to keep the user in-shot as he or she walks between different desks, displays, and whiteboards.

Each of these applications uses a different interpretation of the word "nearest." Spatial containment formalizes these notions. To support follow-me desktops, each workstation has a space extending from it around the area in which the keyboard can be used. Follow-me phone calls are supported by spaces that take into account obstacles posed by other objects, like desks. Camera selection is supported by spaces that delineate the floor area in which a given camera will get a good view of the user.

Simply measuring the distance between two objects does not fully capture the concept of *nearness* as people use it. For example, a telephone that appears to be physically close to a user may be inaccessible because of intervening desks and walls. Therefore, interpreting nearness in terms of proximity would not give good results for any of the applications we've described. A proximity-based approach would too often return the workstation behind the user, the phone on the other side of the room divider, or the camera that is next to the user but happens to be pointing above his or her head.

Novel user interfaces

If we consider a Bat to be a pointer in a 3D user interface that extends throughout our building, we can create new types of user interfaces using Bats and model information. Thus, we can use lower-dimensional spaces to create arbitrarily shaped mouse panels in 2D, slider controls in 1D, and buttons in 0D, then project the 3D Bat position into these spaces to control the user interface components.

Mice. We have placed several 50-inch displays around our building. When a Bat enters the mouse panel space that is placed in the model around each screen, the system projects the 3D Bat position into a 2D mouse pointer reading, then transmits it to the workstation controlling the screen. Any person's Bat can then be used as a mouse for any of these screens.

Virtual buttons. A button in a traditional GUI simply indicates an area of the display with some special application-level significance, accompanied by a label that has some meaning to the user. We can take the same approach with our 3D interface by registering a point in the model as having an application-level significance and labeling the real world with a physical token at the corresponding point. To activate these *virtual buttons,* a user places a Bat on the physical label and presses one of the Bat's real buttons. The button-press message from the Bat interrupts the scheduler, thereby ensuring that the Bat requests a location update within 50 ms and allowing it to process the virtual button press with low latency. Because each person's Bat bears a unique identifier, the system knows who originated each virtual-button event, thus virtual buttons can control personalized applications.

One application of this technique, the *smart poster*, contains labels for one or more virtual buttons, as Figure 5 shows. We use smart posters to control follow-me applications. For example, a user can enable or disable his follow-me desktop or telephone service with a smart poster. Applications that smart posters control will normally send some audible feedback directly via the user's Bat.

We have also used a smart poster to control a networked scanner. Users press buttons on the poster to choose resolution, color level, data compression format, and destination—which can be either their e-mail in-box or information timeline.

Augmented reality. If, in the future, users have a permanently available view of the model, it would not be

necessary to create a label for a virtual button in the physical world. We have experimented with a nonstereoscopic head-mounted display, tracked using data combined from three Bats and an inertial sensor, to create an *augmented reality* display that superimposes information from the model onto the user's view of the real world. This system could display labels for otherwise invisible button spaces. Systems administrators could also use the augmented reality display to maintain those parts of the world model that do not update automatically, such as the positions of furniture. Augmenting the user's view of the environment with the model's current state can make any omissions or inconsistencies immediately obvious.

Data creation, storage, and retrieval

If we use Bats to locate mobile networked devices, we can use their positions to infer who holds them. We can then increase the productivity of each device by

- personalizing the device spontaneously when a given user picks it up, thereby making it easy for users to share devices;
- filing any data the device generates according to user preferences by, for example, placing the data in the user's timeline or e-mailing it to the user; and
- placing additional contextual information in the timeline to aid data retrieval.

Currently, the user must explicitly perform configuration, data storage, and indexing—a fixed overhead that presents a disincentive to use mobile devices for small tasks. Applying sentient computing to this challenge may result in large productivity improvements.

Given the current lack of small devices with wireless connectivity, we have had to use devices that store time-stamped data in nonvolatile memory. We also store time-stamped ownership events that the model generates, and, when the device synchronizes with the network, we correlate the information streams to determine which data corresponds to which user. This approach precludes any kind of device personalization at the time of use, but it does allow us to experiment with personalized filing, postprocessing, and context-based retrieval.

We have used several types of shared digital cameras, located by Bats, to automatically store photographs in a user's timeline. We have used a shared digital voice recorder to automatically store audio memos in the user's timeline, and we have implemented a service that transcribes the voice data using that user's voice model. We also file the transcription in the timeline along with the voice data used to generate it. One advantage of a context-aware storage mechanism is that it can associate arbitrary data items simply by putting them close to one another without maintaining any kind of referential integrity. We can use this capability to compose information sources in powerful ways. Once the system completes the automatic transcription, we can use a text search on the transcription to retrieve the voice data.

We have implemented a photo annotation program as another example of this information-retrieval style. When a fixed camera takes a photo, the application queries the model to determine who is in the camera's field of view, then stores a suitable caption in the timeline at the same time it stores the photograph. Thus, a text search for "+photo +Hopper" returns sections of a timeline containing photographs of Andy Hopper. We cannot provide this service for presently available handheld cameras because we cannot establish the zoom state of the lens for a given photo.

Figure 6 shows a sample timeline that contains data that several users generated contemporaneously: a photograph taken using a handheld camera, an annotated photograph taken using a fixed camera, and a scanned document.

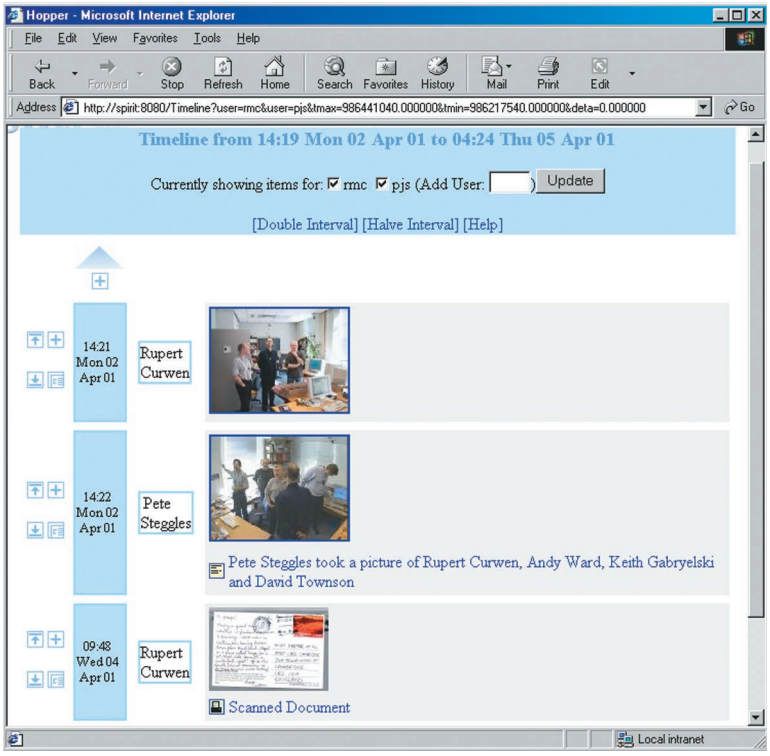


Figure 6. A sample timeline containing data generated by several users: a photo taken with a handheld camera, an annotated photo taken by a fixed camera, and a scanned document.

We do not yet know how much value any of our applications will create, but our initial experiments show promise. We believe that the only way to fully determine the utility of sentient computing systems is to build and use them on a real-world scale.

We can imagine a future in which inexpensive wireless technologies such as IEEE 802.11 and Bluetooth[12] provide ready access to numerous networked and mobile devices. However, maximizing the utility of this richly equipped environment requires removing the obstacles to sharing and configuring devices. Sentient computing systems create applications that appear to perceive the world, making it easy to con-

figure and personalize networked devices in ways that users can easily understand.

But sentient computing is more than a solution to the problems of configuration and personalization. When people interact with computer systems in this way, the environment itself becomes the user interface. We think this goal a natural one for human-computer interaction. ✷

References

1. L. Brown, ed., *The New Shorter Oxford English Dictionary*, Oxford University Press, Oxford, UK, 1993.
2. A. Harter et al., "The Anatomy of a Context-Aware Application," *Proc. 5th Ann. Int'l Conf. Mobile Computing and Networking* (Mobicom 99), ACM Press, New York, 1999, pp. 59-68.
3. A. Ward, *Sensor-Driven Computing*, doctoral dissertation, University of Cambridge, UK, 1998.
4. N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System, *Proc. 6th Ann. Int'l Conf. Mobile Computing and Networking* (Mobicom 00), ACM Press, New York, 2000, pp. 32-43.
5. J. Werb and C. Lanzl, "Designing a Positioning System for Finding Things and People Indoors," *IEEE Spectrum*, Sept. 1998, pp. 71-78.
6. P. Bahl, V. Padmanabhan, and A. Balachandran, "Enhancements to the Radar User Location and Tracking System," tech. report MSR-TR-2000-12, Microsoft Research, Redmond, Wash., Feb. 2000.
7. A. Harter and A. Hopper, "A Distributed Location System for the Active Office," *IEEE Network*, Jan. 1994, pp. 62-70.
8. R. Want et al., "The Active Badge Location System," *ACM Trans. Information Systems*, Jan. 1992, pp. 91-102.
9. I. Getting, "The Global Positioning System," *IEEE Spectrum*, Dec. 1993, pp. 36-47.
10. S. Fertig, E. Freeman, and D. Gelernter, "Lifestreams: An Alternative to the Desktop Metaphor," *ACM SIGCHI Conf. Human Factors in Computing Systems Conf. Companion* (CHI 96), ACM Press, New York, 1996, pp. 410-411.
11. T. Richardson et al., "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, 1998, pp. 33-38.
12. J. Haartsen et al., "Bluetooth: Vision, Goals, and Architecture," *Mobile Computing and Comm. Rev.*, vol. 2, no. 4, 1998, pp. 38-45.

Mike Addlesee *received a PhD in electrical engineering from the University of Cambridge. His research interests include digital signal processing, image compression, genetic algorithms, portable multimedia computer networks, and sentient computing.*

Rupert Curwen *received a DPhil from the University of Oxford, and continued his computer vision research at Yale University and General Electric Corporate Research and Development in Schenectady, New York. Since joining AT&T Laboratories Cambridge, he has been studying the software design and applications of sentient computing systems.*

Steve Hodges *received a PhD from Cambridge University. His research interests include embedded sensor systems, low-power wireless communication, RF tagging technologies, and mobile robotics. In addition to his work at AT&T Laboratories Cambridge, he is also involved with research at the University of Cambridge Department of Engineering, where he is a Visiting Industrial Fellow.*

Andy Hopper *received a PhD from the University of Cambridge. He is a member of the ACM, a fellow of the IEE, and a fellow of the Royal Academy of Engineering. He is the professor of communications engineering at Cambridge University Engineering Department and managing director of AT&T Laboratories Cambridge (formerly the Olivetti & Oracle Research Laboratory). His research interests include many aspects of computer and communications systems engineering with particular emphasis on sentient computing.*

Joe Newman *received an MEng in engineering from Cambridge University. At AT&T Laboratories Cambridge, he has been working on augmented reality within a sentient environment. His research interests include tracking technologies and mobile computing.*

Pete Steggles *received a diploma in computer science from Cambridge University. At AT&T Laboratories Cambridge, he is involved in designing the software architecture for AT&T's sentient computing system. His research interests include distributed system design, domain-specific languages, and the applications of sentient computing.*

Andy Ward *received a PhD from Cambridge University. At AT&T Laboratories Cambridge, he has continued his PhD work on sensor-driven computing as part of AT&T's sentient computing project. His research interests include location-aware technologies, low-power wireless networking, and mobile systems.*

Contact the authors at {maddlesee, rcurwen, shodges, ahopper, jnewman, psteggles, award}@uk.research.att.com.