Wireless Networks 1 (2001) 1–16

The Anatomy of a Context-Aware Application

Andy Harter [a], Andy Hopper [a,b], Pete Steggles [a], Andy Ward [a], Paul Webster [a]

[a] *AT&T Laboratories Cambridge, 24a Trumpington Street, Cambridge CB2 1QA, United Kingdom*

E-mail: {ach,ah,pjs,amrw,pmw}@uk.research.att.com

[b] *Laboratory for Communications Engineering, Cambridge University Department of Engineering, Trumpington Street, Cambridge CB2 1PZ, United Kingdom*

E-mail: hopper@eng.cam.ac.uk

We describe a sensor-driven, or sentient, platform for context-aware computing that enables applications to follow mobile users as they move around a building. The platform is particularly suitable for richly equipped, networked environments. The only item a user is required to carry is a small sensor tag, which identifies them to the system and locates them accurately in three dimensions. The platform builds a dynamic model of the environment using these location sensors and resource information gathered by telemetry software, and presents it in a form suitable for application programmers. Use of the platform is illustrated through a practical example, which allows a user's current working desktop to follow them as they move around the environment.

Keywords: Mobile computing, sentient computing, context-aware computing, location sensors, resource monitoring, middleware, spatial indexing, CORBA, visualisation, HCI

1. Introduction

The essence of mobile computing is that a user's applications are available, in a suitably adapted form, wherever that user goes. Within a richly equipped networked environment such as a modern office the user need not carry any equipment around; the user-interfaces of the applications themselves can follow the user as they move, using the equipment and networking resources available. We call these applications *Follow-me* applications.

Follow-me applications are a special case of context-aware applications [9]. A context-aware application is one which adapts its behaviour to a changing environment. Other examples of context-aware applications are 'construction-kit computers' which automatically build themselves by organising a set of proximate components to act as a more complex device, and 'walk-through videophones' which automatically select streams from a range of cameras to maintain an image of a nomadic user.

Typically, a context-aware application needs to know the location of users and equipment, and the capabilities of the equipment and networking infrastructure. In this paper we describe a sensor-driven, or *sentient*, computing platform that collects environmental data, and presents that data in a form suitable for context-aware applications.

The platform we describe has five main components:

- A fine-grained location system, which is used to locate and identify objects.
- A detailed data model, which describes the essential real-world entities that are involved in mobile applications.

- A persistent distributed object system, which presents the data model in a form accessible to applications.
- Resource monitors, which run on networked equipment and communicate status information to a centralised repository.
- A spatial monitoring service, which enables event-based location-aware applications.

Finally, we describe an example application to show how this platform may be used.

2. Indoor Location Sensing

An ideal location sensor for use in *indoor* environments would possess several important properties. Not only would it provide fine-grain spatial information at a high update rate, but would it also be unobtrusive, cheap, scalable and robust. Unfortunately, the indoor environment is a challenging one in which to implement such a system. Radio-based location techniques (e.g. GPS [1]), which are successful in the wide area, are afflicted by severe multipath effects within buildings. Electromagnetic methods (e.g. [3]) suffer interference from monitors and metal structures, whilst optical systems (e.g. [4] and [13]) require expensive imaging detectors, and are affected by line-of-sight problems in environments containing opaque objects. However, location systems that use ultrasonic techniques appear to have many desirable properties, and one such system that has been developed at our laboratory [14][15] is described below.

2.1. Principles

Small units called *Bats*, shown in Figure 1, are attached to equipment and are carried by personnel. Bats consist of a radio transceiver, controlling logic and an ultrasonic transducer. The current version measures 5cm $\times$ 3cm $\times$ 2cm and weighs 35g. Each Bat has a globally unique identifier. *Ultrasound receiver units*, shown in Figure 2, are placed at known points on the ceiling of the rooms to be instrumented. Receivers are connected by a wired daisy-chain network.

Figure 1. A Bat unit

A *base station* periodically transmits a radio message containing a single identifier, causing the corresponding Bat to emit a short unencoded pulse of ultrasound. Simultaneously, the ultrasound receivers in the rooms covered by the base station are reset via the wired network. Receivers monitor the incoming ultrasound and record the time of arrival of any signal from the Bat. Using the speed of sound in air (which can be estimated from the ambient temperature), the times-of-flight of the ultrasound pulse from the Bat to receivers can be converted into corresponding Bat-receiver distances.

If distances from the Bat to three or more non-collinear receivers can be found, its position in 3D space may be determined using the process of *multilateration* (an extension of trilateration utilising three or more measurements). The position of the object to which the Bat is attached can then be deduced. Reflections of the ultrasonic signal from objects in the environment are commonplace, and can cause incorrect measurement of Bat-receiver distances. These er-

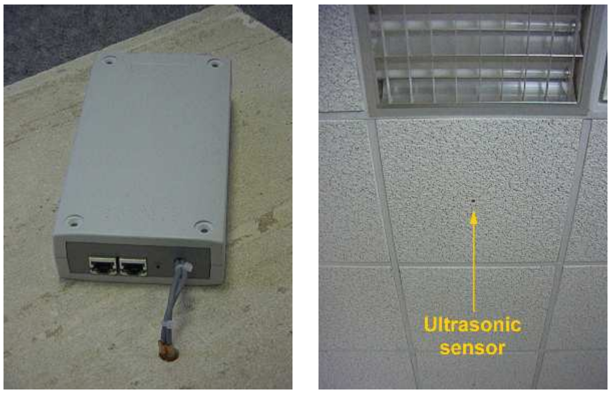

Figure 2. Top and bottom views of an installed receiver

roneous distance measurements are eliminated by the use of a statistical outlier rejection algorithm, so as to improve the accuracy of the calculated Bat position.

After a distance-measuring pulse has been emitted, the base station waits for reverberations of the pulse to die out before triggering another Bat, ensuring that receivers can ascribe incoming ultrasonic signals to the correct Bat. This process divides time into *timeslots*, each of which can be used to locate a single Bat. In typical office spaces, reverberations may take up to 20ms to die away, implying that there may be up to 50 timeslots (and hence location updates) per second per base station.

Information from the location system can also be used to determine the orientations of objects. By placing several Bats at known points on a rigid object, and finding their positions in 3D space, both the position and orientation of the object may be deduced. In situations where tagging of an object with multiple Bats might be cumbersome, an alternative approach may be taken. Most objects are opaque to ultrasound, and will cast a shadow in any signal emitted by a Bat. The transmission pattern of a Bat attached to such an object will therefore be directional. By determining the position of the single Bat, and the positions at which its signal was detected, the known transmission pattern can be used to estimate the orientation of the Bat, and hence that of the object to which it is attached.

2.2. Sensor scalability

The location sensing system described above has several features that make it suitable for wide-scale deployment in environments of interest to this work. It can provide different location update rates for different types of object, handle changing sets of objects to be located, and is scalable to both large numbers of objects and large areas of operation.

The limited number of timeslots must be efficiently distributed between the set of Bats to be tracked. Each timeslot can be allocated to any Bat by the base station. A value called the *Location Quality of Service* (LQoS), associated with each object, indicates the desired interval between location updates for that object. The base station schedules timeslots to Bats based on the currently requested LQoS values.

The scheduling environment is dynamic, and LQoS values associated with objects may change throughout the day. For example, the base station might normally monitor Bats carried by people (who move often) a few times a second, and it might monitor those attached to workstations only once every few minutes. If, however, a person were to walk up to a workstation, the workstation might be monitored more frequently, because it would then be more likely to be moved. Scheduling information can also be used to assist power saving in Bats. For example, if the base station knows that an object will not be located for some time, it can command the Bats associated with that object to temporarily enter a low-power sleep state in which they do not check all incoming radio messages.

The set of Bats to be tracked by the location system may change over time, as objects enter and leave its operating space. Mechanisms therefore exist for introducing new Bats into the set to be polled by the base station, and for deleting Bats from that set so that location resources are not wasted on uncontactable Bats. Bats outside the operating space of any base station enter a low-power *searching* state. When a Bat in the searching state locks on to the transmissions from a base station, it uses a slotted-ALOHA contention resolution protocol [7] to send its unique identifier to the base station via its radio transceiver, thus registering its presence with the base station. If, on the other hand, a base station allocates several timeslots to a registered Bat, but sees no indication from receivers that the Bat has responded by transmitting ultrasound, it can conclude that either the Bat is obscured or the Bat has left the operating space of the location system. The base station can resolve these possibilities by requesting that the Bat transmit its unique identifier via its radio transceiver—if repeated attempts to elicit a response from the Bat fail, the base station reclaims resources allocated to the Bat.

The number of Bats that can be monitored by the system is determined by the size of their address space, which can be made as large as required. The area covered by the location system may be increased by the use of multiple base stations. A time-division multiplexing (TDM) strategy is used to ensure that transmissions from neighbouring base stations do not interfere with each other. All base stations use common timeslots derived from a global clock, and only one TDM channel is active in each timeslot. Base stations whose radio cells overlap are allocated different TDM channels, and do not transmit in the same timeslots. The choice of a TDM strategy therefore limits the location rate within individual radio cells, but permits a simple, low-cost implementation of the Bat radio transceiver.

When a Bat moves between radio cells, it must perform a handover of control between one base station and another. Systems which make handover decisions based on standard criteria such as received radio signal strength, link error rates or base station load could be developed. However, handover decisions can also be made by base stations using the known Bat positions and coarse estimates of the extents of radio cells, which are made when the base stations are placed.

2.3. Location system evaluation

The current location system is installed within two rooms and a corridor at our laboratory. It uses two base stations and 100 receivers to cover approximately 280m^3. The 50 available timeslots per second are shared equally between the two base stations, and location information is used to guide handover of Bats between radio cells. In the near future, we intend to install the system throughout our building, which has a floor area of approximately 1000m^2 and a volume of 2500m^3. Preliminary measurements indicate that the building can be covered using six radio cells operating with four TDM channels, implying an aggregate location rate of 75 updates per second. The receiver density used in the extended installation will be similar to that chosen for the current system.

The current location system can address up to 65535 Bats using a 16-bit address space. The battery lifetime of a Bat is strongly dependent upon the manner in which it is used—Bats attached to objects that are located frequently have little opportunity to enter the low-power sleep mode, and hence have shorter battery lifetimes than those attached to more static objects. In practice, however, Bats located several times a second have battery lifetimes of several months. Bats also have input and output facilities that take advantage of the bidirectional radio link with the base stations—a data channel allows command information to be sent to peripherals connected to the Bat, and two buttons on the Bat allow it

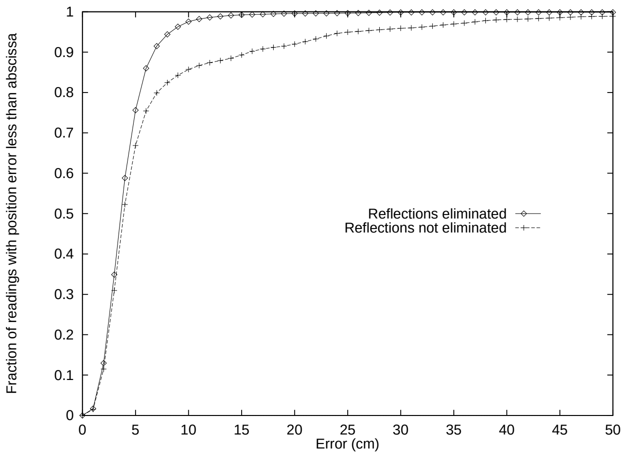

Figure 3. Position accuracy of ultrasonic tracker

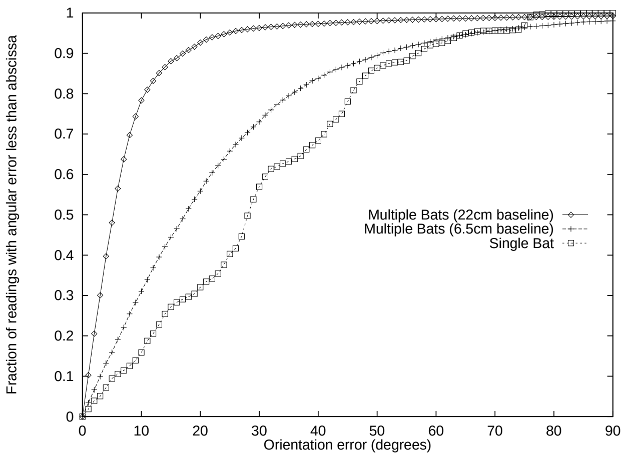

Figure 4. Orientation accuracy of ultrasonic tracker

to act as a ubiquitous control device.

Figure 3 shows the position accuracy of the location system, determined using 100,000 measurements within a 10m^3 volume. It can be seen that 95% of readings are within 9cm of their true positions. The importance of the reflection elimination algorithms is evident—the position accuracy of the location system would be significantly worse if such methods were not employed. Figure 4 shows the accuracy of the methods for determining object orientations described above. The method involving multiple Bats is seen to be more accurate than that relying on shadowing of a single Bat, especially when the distance between multiple Bats on an object (the *baseline*) is large.

3. Modelling the Environment

Ideally, context-aware systems should know as much as the user about those aspects of the environment which are relevant to their application. A key element of a general platform for such systems is a detailed model describing entities in the real world and their possible interactions. This model sets out the types, names, capabilities and properties of all entities which are relevant to context-aware applications, and acts as a bridge, allowing computer systems to share the users' perceptions of the real world. Furthermore, the extent of the data model defines the limits of the context-aware system's view of the world, and consequently its possible domain of application.

Given that the environment is composed of a collection of real objects, it is appropriate to model it using an object-oriented modelling technique. We have developed a modelling language to represent the data model as an entity relationship diagram augmented with multiple inheritance. In the richly equipped networked environment which is the context for this work, models have been devised to represent people, computers, keyboards, displays, networks, telephones, and furniture.

3.1. Systems infrastructure

The software counterparts of real-world entities are implemented as persistent distributed objects using CORBA and an Oracle 7 database. A package called *Ouija* [11] provides an object-oriented data modelling language which is used to generate an object layer on top of the relational model used by the Oracle database.

Objects are stored in the database as rows of data and associated operations written in PL/SQL, Oracle's proprietary procedural extension to SQL. They are accessed via a proxy server, which receives CORBA calls from client ap-

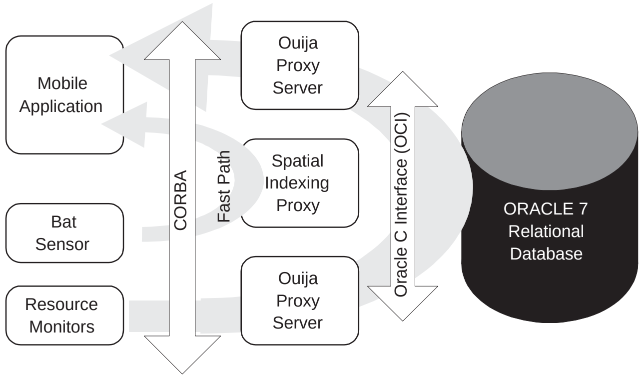

Figure 5. Three-tier architecture

plications and forwards them to the appropriate PL/SQL operation via OCI, Oracle's proprietary call interface. This effectively provides a CORBA mapping for PL/SQL, because all queries on persistent state are performed by the database. All C++ code to implement the proxy is automatically generated by the Ouija utility.

The resulting three-tier model is shown in Figure 5. It is recognised that some information, particularly that from the Bat sensors, is updated too frequently to be stored in the database. This information therefore travels along a 'fast path' which goes only via the proxy.

It is possible that thousands of objects might be stored in the system. Many of these may only be updated or queried infrequently, so their proxy implementation will be inactive much of the time. On-demand loading allows the proxies to operate as a cache of persistent objects. A proxy server creates an instance of an object in the database when it is first accessed. The proxy server can manage the number of active objects by unloading the implementations on a least-recently-used basis. Subsequent calls to unloaded objects will cause them to be reloaded. This approach reduces the total level of resources required by the objects and provides system scalability.

The on-demand mechanism also provides system robustness. Clients can assume that objects are always accessible. If a proxy server crashes, a new one can be started and the clients will re-bind to the new server transparently to the application. This makes writing applications easier, and allows proxy servers to be upgraded and extended without needing to restart clients.

A complete description of the on-demand loading mechanism and its implementation is beyond the scope of this paper. Full details can be found in [11].

3.2. Updating the model

In order to populate the database described above, information about real world objects and their properties must be gathered. In many cases, this data is static and need only be asserted once. However, for those elements which are dynamic, for example keyboard activity or machine load, automatic update methods are necessary to maintain a current view of the environment.

Processes called *resource monitors* are installed on all networked machines. The monitors use operating system calls to discover information about the current status of machines, and periodically report changes in status to objects in the database via CORBA interfaces. The monitors have been structured for portability and run on a wide range of operating systems. Furthermore, monitors are lightweight and have been designed so as not to impinge upon the normal use of the machine or the communications infrastructure.

Three classes of resource monitor have been implemented:

1. Machine activity, e.g. keyboard activity.
2. Machine resources, e.g. CPU usage, memory usage.
3. Network point-to-point bandwidth and latency.

The machine activity monitor, for example, wakes up once every five seconds. It checks the level of mouse and keyboard activity, and the identity of the logged on user. If there has been any change, the entry for the machine in the database is updated. Other monitors behave similarly.

A centralised approach was chosen in order to reduce query latency to a minimum. If the information were not concentrated in one place, applications would need to collect data before acting upon it. This would require a number of network calls and increase the query latency. Storing the data in a centralised repository ensures that complex queries can be easily expressed using standard RDBMS technology.

Much effort has been put into the design of the monitoring clients and in the communications infrastructure to ensure the database does not prove to be a bottleneck. We can apply filtering and caching techniques at the client level and in the middle tier to achieve this:

Update Frequency: The frequency at which items are monitored is based on how quickly the item tends to change.

Relevancy: If a value has not changed significantly since the last time it was updated, it is not sent to the database. 'Significance' depends on the data being monitored. For example, free space on a disk changing by 1% is not significant; a disk becoming 99% full is.

Caching: Data caching is used to improve client retrieval times. This can dramatically reduce query times. One application uses the database to provide a 'distributed ps', which lists all of a user's processes regardless of the machine they run on. Caching improves performance by a factor of five.

Use of filtering reduces the rate of updates to the database by over 90%. Evidently, there is a trade off between the absolute accuracy of the information, the response time and the resource overhead imposed by the monitors.

In the current deployed system, monitors run on 50 machines. Update frequencies range from five seconds for keyboard/mouse activity, up to 45 seconds for disk free space. The monitors also update information about all long running processes, of which there are around 1500. The model is maintained to an acceptable level of accuracy with a transaction rate of around 70 transactions per minute.

4. Programming with Space

This section considers how Bat readings are interpreted to generate accurate object location information, and how applications are provided with a suitable abstraction to support location-aware programming.

4.1. Converting Bat locations to object locations

Using the Ouija modelling system, each physical object is represented by an object in the database and by a CORBA object in a proxy server. CORBA interfaces can be added to Ouija proxy objects using multiple inheritance, enabling communication with the Bat system via a simple CORBA callback mechanism. This callback function is implemented in a different way for each Ouija type to provide a type-specific method of calculating object positions. Personnel wear a single Bat clipped to their clothing, and the location of the Bat is translated directly into a location for the person. Computer workstations are tracked using two Bats placed on either side of the display. The location of the two Bats is used to calculate a more accurate value for the orientation component of the location of the workstation by treating the line between the Bats as a dipole which is constrained to rotate only about the vertical.

4.2. Filtering and error detection

To smooth out small random errors in location of computers a simple low-pass filter is used, trading off a loss in speed

Figure 6. 3D Visualisation of Location Data

of response for a more stable sequence of location readings. This filtering reflects the empirical observation that locations of computer displays rarely if ever oscillate at high frequencies. When locating people, less filtering is used because the same observation cannot be made.

In common with all sensors, the location system is prone to occasional large errors caused, for example, by environmental ultrasound. A method for detecting these errors is implemented, based on thresholding against the maximum velocity expected for objects. Again, maximum expected speeds are allocated on a per-type basis.

4.3. LQoS registration

Each CORBA object which represents a physical object has an interface allowing applications to make quality of service requests to the location system. In the absence of any specific requests, the objects make suitable default requests on a per-type basis: the default rate for people is about two readings per second, that for computers is about one reading every five seconds.

4.4. A location-aware API

One application of this system is to provide human users with a browsable model of the world which they can explore. We have implemented a 3D model of our building which is updated in real time with location data. This is shown in Figure 6. The data is distributed using an event service which ensures that many users can view the information at the same time. This, however, is a raw form of the location data and many applications will benefit from a further layer of interpretation.

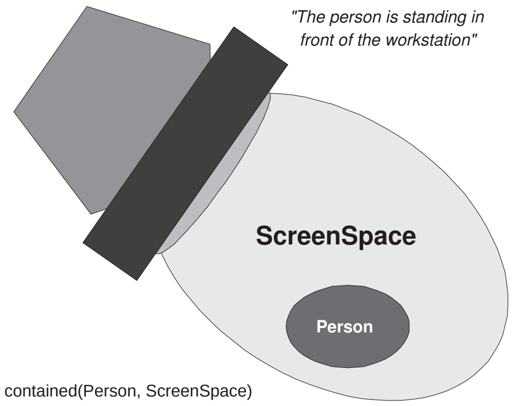

Figure 7. Representation of a spatial fact using geometric containment.

4.5. Absolute and relative spatial facts

The techniques described above show how a location system can provide absolute spatial facts about objects, such as 'the person is at (x, y, z), facing in direction θ'. In general, location-aware applications are interested in relative spatial facts, such as 'the person is standing in front of the workstation'. It should not have to be the task of the application to translate absolute spatial facts into relative spatial facts. To generalise and automate this process of translation, a simple method of formalising relative spatial facts is required.

The approach used expresses relative spatial facts about objects in terms of geometric containment relationships between spaces associated with those objects. An example of this technique is illustrated in Figure 7. Here, a light-coloured space represents the area around the front of a workstation screen, and a dark-coloured space represents the spot where a person is standing. Then the statement 'the person is standing in front of the workstation' is considered to be true if and only if the space associated with the person is contained by the space associated with the workstation.

Note that Figure 7 represents a 2D environment. This simplification can be made because people and objects tend not to move in an unrestricted manner in three dimensions. Thus, the containment relationship can be evaluated on a number of 2D planes corresponding to individual floors of the building.

4.6. Spatial monitoring

If application writers express relative spatial statements in terms of geometric containment statements, a 'spatial monitor' can be implemented to translate absolute location events for objects into relative location events. Applications can then receive a stream of events expressing facts directly relevant to them, such as 'the person has walked up to the workstation', or 'the person has walked away from the workstation'.

The interaction of an application with the spatial monitoring system is illustrated in Figure 8. Applications associate a space with a type of object (for example, the light and dark-coloured spaces around workstations and people in Figure 7), and register callbacks for given spatial facts (for example, a request to be notified when a dark person-space becomes contained by a light workstation-space).

As shown in Figure 8, the location events generated by the Bat system are translated into object location events by the appropriate Ouija proxy object. Then, spaces which are associated with the given object (by virtue of application-level associations of spaces with object types) are looked up, generating a stream of space location events. This is used as the input to an indexing system, which generates the stream of all containment events. The callbacks associated with containment events are then looked up and called.

4.7. Scalability issues

The design of a spatial monitor has certain scalability implications. A realistic system capable of supporting a

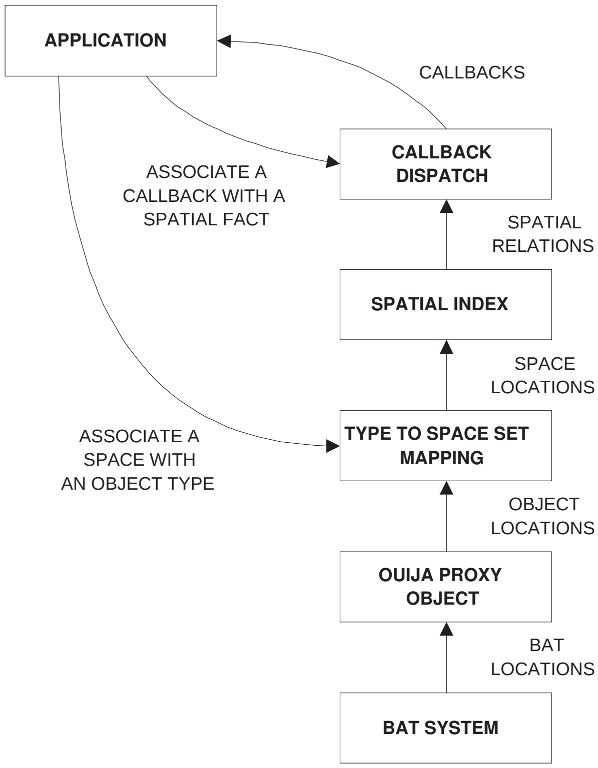

Figure 8. Transformation of absolute location events to application-centric callbacks

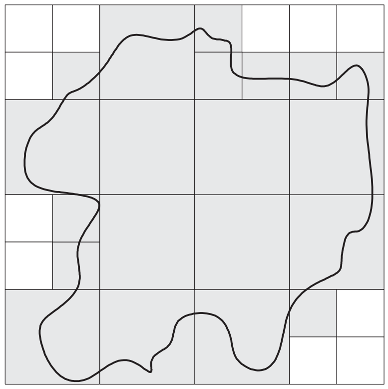

Figure 9. Maximal cover of a space

reasonably-sized building might achieve a data rate of a few hundred object movements per second, in a population of tens or hundreds of thousands of objects. Every time an object moves, some calculation must be done to evaluate geometric statements and send the appropriate events. Clearly some form of spatial indexing is required to minimize the computational complexity involved. The indexing system must support an arbitrary mixture of sizes and geometries and provide fast and predictable insertion, updating and query.

4.8. Containment tree indexing system

The chosen indexing system uses a quadtree called the *containment index*. A quadtree is a standard indexing structure, generated by breaking down the plane containing the spaces into sub-quadrants, where each quadrant cell is represented by a node in a tree [8]. The root node represents the entire indexed region of the plane, and the four sub-quadrant nodes are children of the root node. In turn, each of those four cells and nodes has four sub-quadrants and children respectively, and so on, to form the hierarchical quadtree.

When a space is placed into the containment index, the *maximal cover* of the space is calculated. The maximal cover is the smallest set of quadtree cells (i.e. the largest cells) required to cover the space at a particular *resolution*. The resolution dictates the minimum size of quadtree cell that is considered during the indexing operation—higher indexing resolutions consider smaller quadtree cells, and so result in a better approximation to the space in the index. An example maximal cover is shown in Figure 9. The cells in the containment tree corresponding to the cells in the maximal cover of the space are labelled with that space's identifier.

Suppose two spaces s and t are covered exactly by their maximal covers at some resolution (i.e. the maximal covers do not extend beyond the spaces' boundaries). Then it is possible to express the spatial containment relation between s and t in terms of the relation between their maximal covers:

- **The containment indexing theorem:** Space s is contained by space t if and only if, for each cell x in the maximal cover of s, there exists exactly one cell in the maximal cover of t that contains x or is equal to x.

Suppose the maximal covers do extend beyond the boundaries of s and t, actually covering spaces S and T. Clearly S and T are approximations to s and t respectively, so, by using the containment indexing theorem, the approximate containment relations between spaces can be evaluated by considering relations between the cells in their maximal covers.

Approximate evaluations of containment relations are adequate for mobile applications because the Bat location system only provides approximate information; moreover, the properties of the spaces themselves are to some extent arbitrary, since they are merely formalisations of imprecise, application-level relations such as 'next to'.

A space can be inserted into or deleted from the tree by a process of tree walking. If, while walking the tree, a count is maintained of other spaces which are encountered, the resulting totals can be used to identify all the containment relationships involving that space. For example, while inserting a space s, whose maximal cover has cardinality n, if a space t is encountered on each of the n paths taken to insert s, then by the containment indexing theorem, t contains s. Similarly, if the containment tree below each of the cells annotated with s is searched, and on n' occasions a space t' (whose maximal cover has cardinality n') is encountered, then s contains t'. When inserting a space into the index, therefore, all corresponding positive containment events can be generated. Similarly, all negative containment events can be generated when deleting from the index. When moving a space in the index, it is possible to generate all new positive and negative containment events by subtracting the intersection from each of the positive and negative event sets.

The cost of placing a space in the index is proportional to the product of its perimeter and the indexing resolution. The cost of searching the tree underneath the space is proportional to the product of area of the space and the density of population of the index. When moving a space s in the index, all spaces t which contain s can be found in time proportional to the perimeter of s. Also, all spaces t' which are contained within s can be found in time proportional to the area of s. A space precompilation step is used to ensure that the indexing cost is independent of the type of space indexed: concave polygons are as easy to index as circles.

Using a related algorithm it is possible to simultaneously generate all positive and negative overlapping events when a space is moved in the index. This approach can be used to incorporate a degree of hysteresis into applications. By registering for positive containment and negative overlapping between one large stationary space and a small mobile one it is possible to avoid an oscillating event stream when the small space borders the large one.

4.9. Implementation and performance

An implementation of the containment indexing system has been developed and optimised for performance on symmetric multiprocessor systems. The implementation details are outside the scope of this paper. More information is available in [10].

Performance tests were made in order to assess whether the spatial monitoring system would adequately support a large building containing several hundred people, and thousands of pieces of equipment. The tests involved indexing a large population of different sized squares, and then measuring how quickly a given space could be moved within the index. Within an index of 10000 by 10000 units, populated with grids of squares of size 15, 31, 63, 127 and 511 units (a total of about 15000 spaces all indexed with a minimum cell size of 1 unit), several squares of size 15 units (indexed at a resolution of 1 unit) were moved. Throughput was about 1700 updates per second on a Sun SMP machine with seven

167Mhz Sparc processors. We believe that this level of performance is well in excess of our requirements.

5. Case Study

Previous papers have described the Active Badge System [12] [2], which is capable of locating objects to a room-scale granularity. One application of the Active Badge System allows a user to redirect ('teleport') their X Window System environment to different computer displays [5]. The application can find personnel and determine the set of displays near them using Active Badge information. Personnel can redirect their working environment to one of the displays by using buttons on their Active Badge. However, the current implementation has some limitations:

- Since the Active Badge is only able to locate a user to a room-scale granularity, repeated Badge button presses are needed to 'cycle' through the candidate machines. In a room containing many machines, this process is time consuming.
- In order to provide feedback to the user regarding the current machine selection, that machine's screen is made to flash. Any user of the machine will be distracted by this action, even though the fact it is occupied means that it is unlikely to be selected.
- While cycling through candidate displays, if a machine is not operating or is not running the X display software, there will be a delay while the connection times out and moves to the next machine. This problem occurs because no resource information is available, so all machines located in a room must be assumed to be working and able to display X sessions.

This case study brings together the elements discussed in this paper and presents a Follow-me application known as 'Bat Teleporting' which builds upon this existing work and addresses the above limitations.

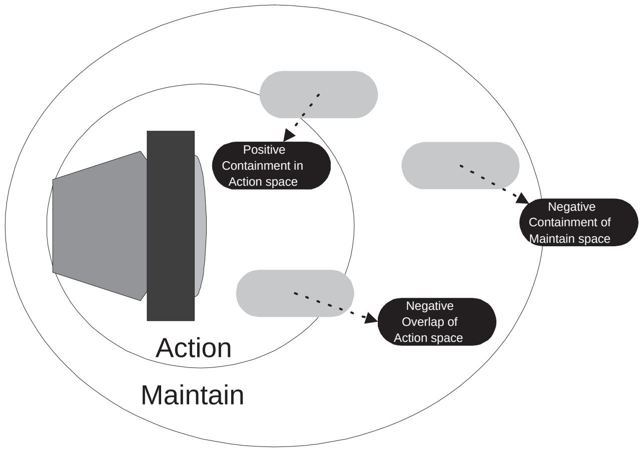

Figure 10. Zones and buttons used in Bat Teleporting

5.1. Bat Teleporting

The *Virtual Network Computing* (VNC) system [6] provides a windowing-system-independent means for a user to access his desktop environment from any networked machine. A server manages the desktop and renders it to a hidden frame-buffer. Changes to the frame-buffer are sent to viewing clients using a simple protocol. Keyboard and mouse events are sent back to the server from the clients.

In one mode of operation, clients can be set to listen for incoming connections from servers. The servers have a CORBA interface, which can be used to initiate connections with particular clients, or close existing connections down. By using events generated from Bat sightings to make appropriate calls to the CORBA interface, a desktop can be teleported onto a display just by the action of the user moving up to it. Similarly, it can be sent away when the user moves away.

5.2. Implementation

The Bat Teleporting application is entirely event driven. Consider Figure 10. For each computer, two spatial zones

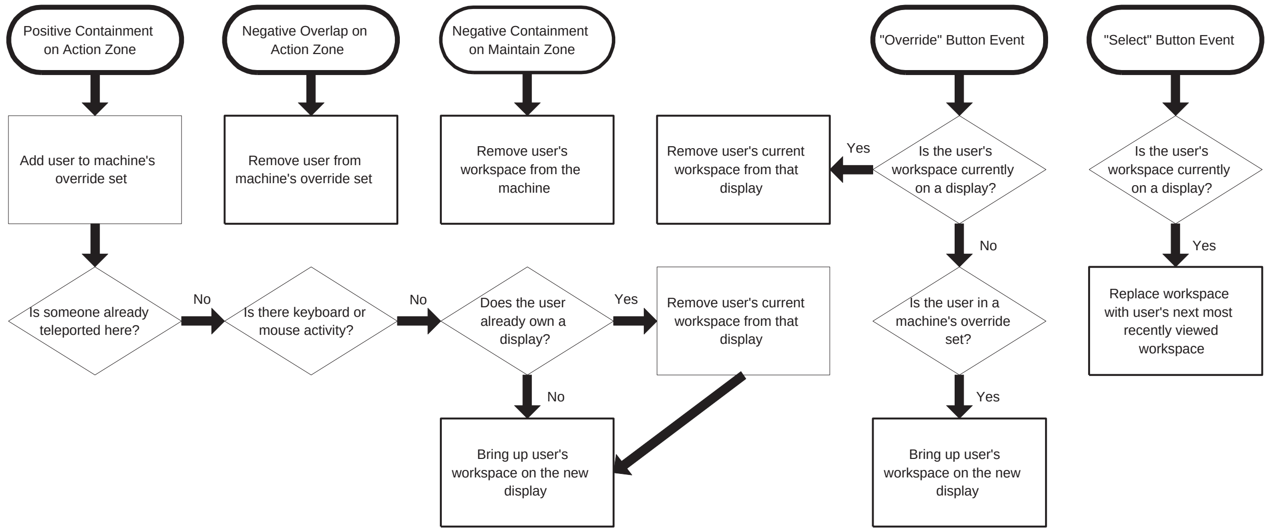

Figure 11. Event-condition-action diagram describing Bat Teleporting

are defined. The inner 'Action' zone is close to the machine and triggers the teleport operation. It is arranged to be small so that the user has to actively approach the machine in order to bring their desktop up on the display. The outer 'Maintain' zone is larger. Within this zone, the user will keep their desktop on the display. When the user leaves this zone, the desktop is removed.

A user may have a number of desktops on different machines, and there needs to be a way to select between them. One of the buttons on the Bat is used to allow selection of an alternative desktop once the user has teleported to a display.

Information generated by activity monitors can be used to determine whether there has been any mouse or keyboard activity on a particular machine within the last 30 seconds. The Bat Teleporting application can also check to see whether there is already a user teleported to the machine. In either of these cases, a user's desktop should not be brought up on the display as it would interfere with someone else's work. However, there are situations where a user may wish to override this feature and teleport their own desktop to the machine. The second button on the Bat is used for this purpose. When a desktop is already displayed, the 'override' button will send the desktop away. The required button press events are distributed by the same event service used to send out raw location events.

Three relative geometry conditions are registered with the spatial monitor: positive containment of a person in the Action zone, negative containment of a person in the Maintain zone, and negative overlap of a person with the Action zone. The actions triggered by these events are summarised by the event-condition-action diagram shown in Figure 11. Note that the override set referred to in this diagram is defined to be the set of all users who are within the inner 'Action' zone, but who were unable to teleport due to another user owning the display.

5.3. Commentary

The Bat Teleporting control application was written in around a week. Response to users' actions is excellent, and the application feels very natural to use. For example, in the time taken for a user to sit down, their desktop is called up on that machine ready for them to continue work. The use of activity information prevents the appearance of another desktop over that of a currently active user. Early experiences with Bat Teleporting are encouraging and it is anticipated that the system will become as indispensable a part of our working environment as the Active Badge-enabled applications which preceded it.

6. Conclusions

We have presented a sentient computing infrastructure for implementing context-aware systems. Key features of the infrastructure include:

- A fine-grained sensor system which provides accurate, up-to-date location information. The finer granularity of this system in comparison to the Active Badge enables us to provide more context information to applications, with consequent benefits to the user interface.
- A rich data model reflecting the resource information required to support context-aware applications.
- A distributed system of persistent objects which can be queried by context-aware applications. This presents a highly available world model which applications can easily utilise.
- A resource monitoring system for collecting information about the computing environment. This enables applications to adjust their behaviour to accommodate system capabilities and usage patterns.
- A spatial monitoring system which allows event-based applications to be written. This enables application writers to write location-aware applications in the same event-driven style as traditional GUI applications.

These components have been integrated to support a Follow-me application, Bat Teleporting, which improves upon a previously-developed system in several ways. The supporting infrastructure is robust, scalable and simultaneously usable by a large number of different applications, several of which have been prototyped. Work is underway to develop these prototypes into large-scale, fully-deployed services. Latest results from the project may be found via the World-Wide Web, at: http://www.uk.research.att.com/spirit

Acknowledgements

The authors would like to thank their colleagues for innumerable contributions over many years to the discussion about context-aware systems and mobile applications. Specifically we would like to thank Jamie Walch for his 3D model visualisation and browsing application, Steve Hodges for his work on Bat units, and Tristan Richardson and Quentin Stafford-Fraser for their work on the VNC system. Rupert Curwen provided many useful comments on the text.

References

[1] Getting, I. *The Global Positioning System*. IEEE Spectrum, Vol. 30, No. 12, December 1993. pp. 36–47

[2] Harter, A., Hopper, A. *A Distributed Location System for the Active Office*. IEEE Network, Special Issue on Distributed Systems for Telecommunications, Vol. 8, No. 1, January 1994. pp. 62–70

[3] Raab, F., Blood, E., Steiner, T., Jones, H. *Magnetic Position and Orientation Tracking System*. IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-15, No. 5, September 1979. pp. 709–718

[4] Rekimoto, J. *Matrix: A Realtime Object Identification and Registration Method for Augmented Reality*. In Proceedings of the Asia Pacific Computer Human Interaction Conference (APCHI'98), Kanagawa, July 1998.

[5] Richardson, T., Mapp, G., Bennett, F., Hopper, A. *Teleporting in an X Window System Environment*. IEEE Personal Communications Magazine, Vol. 1, No. 3, Third Quarter 1994. pp. 6–12

[6] Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A. *Virtual Network Computing*. IEEE Internet Computing, Vol. 2, No. 1, 1998. pp. 33–38

[7] Roberts, L. *ALOHA Packet System With and Without Slots and Capture*. Computer Communications Review, April 1975.

[8] Samet, H. *The Quadtree and Related Hierarchical Data Structures*. ACM Computing Surveys, Vol. 16, No. 2, June 1984. pp. 187–260

[9] Schilit, B., Adams, N., Want, R. *Context-Aware Computing Applications*. Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, December 1994. pp. 85–90

[10] Steggles, P. *Location System*. U.K. Patent Specification, Application No. GB 9812635.2, June 1998.

[11] Steggles, P., Webster, P., Harter, A. *The Implementation of a Distributed Framework to support 'Follow Me' Applications*. Proceedings of the 1998 International Conference on Parallel and Distributed Processing Technique and Applications (PDPTA'98), Vol. 3, Las Vegas, NV, July 1998. pp. 1381–1388

[12] Want, R., Hopper, A., Falcão, V., Gibbons, J. *The Active Badge Location System*. ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992. pp. 91–102

[13] Ward, M., Azuma, R., Bennett, R., Gottscahlk, S., Fuchs, H. *A Demonstrated Optical Tracker with Scalable Work Area for Head-Mounted Display Systems*. Proceedings of the 1992 ACM SIGGRAPH Symposium on Interactive 3D Graphics, Cambridge, MA, March 1992, pp. 43–52

[14] Ward, A., Jones, A., Hopper, A. *A New Location Technique for the Active Office*. IEEE Personal Communications Magazine, Vol. 4, No. 5, October 1997. pp. 42–47

[15] Ward, A. *Sensor-driven Computing*. PhD thesis, University of Cambridge, August 1998.

Biographies

ANDY HARTER received the BA and PhD degrees from Cambridge University in 1982 and 1990 respectively. Since then he has been with ORL, now AT&T Laboratories Cambridge, where he has pioneered systems which support mobile personalised working, including work with the Active Badge.

More recently, the research he has guided into distributed systems and thin-client architectures have come together in the sentient computing framework. Other activities and interests include internet devices, communications philosophy and VLSI. (ACM 4432373)

ANDY HOPPER received the B.Sc. degree from the University of Wales in 1974 and the Ph.D. degree from the University of Cambridge in 1978. 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.

PETE STEGGLES received a BA in Philosophy and a Diploma in Computer Science, both from Cambridge University. He joined AT&T Laboratories Cambridge in 1996, where he has designed and developed the spatial monitoring system and the software architecture for AT&T's sentient computing system.

ANDY WARD received the BA and PhD degrees from Cambridge University in 1994 and 1999 respectively. 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.

PAUL WEBSTER is a researcher at AT&T Laboratories Cambridge. He received his PhD in Computer Science from the University of Manchester in 1996. His research interests include sentient computing, thin-client systems, simulation and low power systems design.