MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications

Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan,
Roy Campbell, M. Dennis Mickunas

Department of Computer Science, University of Illinois at Urbana Champaign,
1304 W. Springfield Ave., Urbana, IL 61801.
{ranganat, almuhtad, chetan, rhc, mickunas}@cs.uiuc.edu

Abstract. Location awareness significantly enhances the functionality of ubiquitous computing services and applications, and enriches the way they interact with users and resources in the environment. Many different alternative or complementary location sensing technologies are available. However, these technologies give location information in different formats and with different resolution and confidence. In this paper we introduce "MiddleWhere" a distributed middleware infrastructure for location that separates applications from location detection technologies. MiddleWhere enables the fusion of different location sensing technologies and facilitates the incorporation of additional location technologies on the fly as they become available. MiddleWhere utilizes probabilistic reasoning techniques to resolve conflicts and deduce the location of people given different sensor data. Besides, it allows applications to determine various kinds of spatial relationships between mobile objects and their environment, which is key in enabling a strong coupling between the physical and virtual world, as emphasized by ubiquitous computing. We have integrated MiddleWhere with our ubiquitous computing infrastructure, and have verified its flexibility and usefulness by incorporating various location sensing technologies and building a number of location-sensitive applications on top of it.

1 Introduction

Ubiquitous computing has inspired the construction of active, information-rich physical spaces that encompass large numbers of interconnected computer devices and embedded processors. This dust of computing machinery will be providing new functionality, offering personalized services, and supporting omnipresent applications. Location awareness enables significant functionality to ubiquitous computing applications, users, resources and the ways they interact. It allows ubiquitous computing environments to tailor themselves according to users' preferences and expectations, and reconfigure the available resources in the most efficient way to meet users' demands and provide seamless interaction. For example, applications and data can follow users as they roam around, content can be customized based on users' location, and physical surroundings can be customized according to their inhabitants.

A plethora of different alternative or complementary location technologies and sensors are available. The different technologies have different capabilities and as-

sumptions and provide assorted levels of location accuracy. No single location sensing technology has emerged as a clear winner in all kinds of environments. For example, GPS is the de facto location technology for wide outdoor areas; however it does not work in covered areas or indoors. For indoor environments, many technologies have been proposed based on badges, wireless devices, etc. We expect different location sensing technologies to be deployed in different environments depending on the specific requirements of the environment. Some environments may even have multiple location technologies deployed.

We believe that ubiquitous computing environments must provide middleware support for fusing data from different location technologies to get a more complete picture of the physical environment and its contents, and to determine location with higher accuracy. Further, a middleware-based solution enables the separation of applications from the location detection and sensing technologies. This makes it possible to extend the infrastructure with new location technologies on the fly, as they become available, without any change to existing applications and services.

In this paper, we propose a middleware for location-awareness called "MiddleWhere." MiddleWhere integrates multiple location technologies and presents applications with a consolidated view of the location of mobile objects (which may be persons or the devices they carry). It handles conflicting information from different sensors based on the confidence of their readings. MiddleWhere maintains a model of the physical layout of the environment and allows deriving various spatial relationships between mobile objects and their physical environment. Since no sensor can accurately sense location, MiddleWhere associates a probability value with location information and spatial relationships. Ubiquitous applications or services using this middleware can utilize these probability values, and choose to act upon location information only if it is accurate "enough" for their purposes.

We have integrated MiddleWhere in our prototype ubiquitous computing environment. It currently fuses location information from four different technologies, and has allowed the rapid development of many location aware applications.

1.1 Features of MiddleWhere

MiddleWhere offers a number of advantages in the development of location-aware applications and the deployment of location sensing technologies in ubiquitous computing environments:

1. *Incorporation of multiple location sensing technologies*. MiddleWhere allows the deployment of different kinds of location detection or tracking technologies with different characteristics. Location information can be got from RF-based badges, Ubisense™ tags [1], card swipes, login information on desktops, fingerprint recognizers, Bluetooth, etc. These different technologies give location data with different resolutions and different levels of confidence. MiddleWhere fuses location data from multiple sensors to get a spatial probability distribution of the location of the mobile object. It can also handle conflicting data obtained from different sensors.

2. *Handling the temporal nature of location information.* The quality of location data depends on how fresh it is. As time goes on, the quality of location data reduces. MiddleWhere handles this temporal degradation of the quality of location data by

reducing the confidence of location data with time. For example, people in our building have to swipe their ID cards on a card reader whenever they enter certain rooms. Hence, at the time of swiping their card, their location is known with high confidence. With the passage of time, however, this location data becomes less reliable, since they might have left the room.

3. *Hybrid Location Model*. MiddleWhere employs a hybrid location model that uses both coordinate as well as symbolic models of location. Location-sensitive applications can express locations either in terms of coordinates with respect to a certain axis of reference, or in terms of symbolic names (such as floor or room numbers, etc.) MiddleWhere also allows easy conversion between the two forms of location data. This gives application developers maximum flexibility in expressing locations in a manner suitable to their applications.

4. *Push and Pull Modes of Interaction*. Location-sensitive applications can interact with the Location Service using either a push or a pull mode. They can ask queries about the current location of objects or they can ask to be notified whenever a certain location-based condition becomes true.

5. *Handles region-based and object-based locations*. There are two kinds of location information that most applications are interested in. (a) Object-based location – this relates to the location of objects, e.g. "where is person X?" (b) Region-based location – this relates to the objects found within a region, e.g. "who are the people in room 3105?"

6. *Model of the world*. MiddleWhere uses a spatial database to model the physical world. The physical layout of the environment (such as position of various rooms and corridors) as well as relevant physical objects (like displays and tables) are represented in the spatial database.

7. *Spatial Relationships between objects*. An important feature that distinguishes MiddleWhere from other location middleware is its ability to deduce spatial relationships between mobile objects and their physical environment (which includes other mobile objects, static objects like tables and displays, and physical locations like rooms and corridors). Examples of spatial relationships that are deduced by MiddleWhere include proximity to another object, collocation of two objects in a certain region, containment of an object within a region and so on. MiddleWhere also associates probabilities with these spatial relationships. This allows MiddleWhere to be especially suited to the requirements of ubiquitous computing environments. These environments emphasize a strong coupling between the physical and virtual worlds, and MiddleWhere provides functions to relate the two.

The remainder of the paper is divided as follows. Section 2 gives a brief overview of MiddleWhere architecture. Section 3 describes MiddleWhere's location model. Section 4 talks about the location service and reasoning engine components of MiddleWhere. Section 5 explains the spatial database. Section 6 describes the sensor technologies we used. Section 7 gives a brief explanation of our implementation. Section 8 briefly mentions some applications that use the system. Finally, we conclude the paper with evaluations, related work, and future work.

2 Architecture

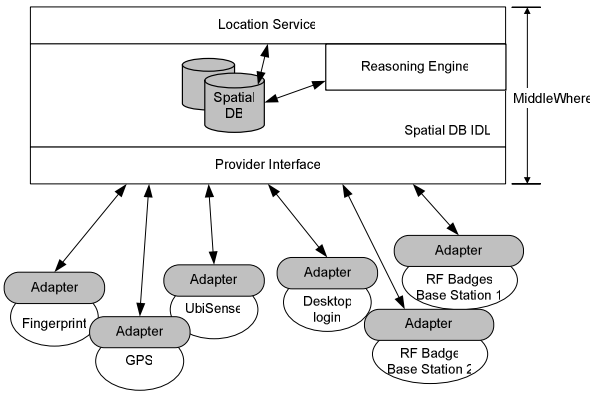

Figure 1. MiddleWhere Architecture

MiddleWhere uses a layered architecture for collecting sensor information, representing it in a spatial database and reasoning about it. Figure 1 shows the architecture of MiddleWhere.

Location sensors send information to the spatial database through the MiddleWhere system. Adapters map raw sensor information into a common representation to be stored in the spatial database. Adapters can be programmed to filter certain events or send information to the MiddleWhere system at varying rates.

The spatial database stores a representation of the physical space as a collection of basic geometric types such as points, lines and polygons. Sensor information is also stored as a separate table in the database. The database provides geometric functions such as distance, containment and intersection that are used for spatial reasoning.

The reasoning engine uses sensor and spatial model information in the database to determine an object's location with a certain probability. It fuses data from different sensors and reasons about relationships between mobile objects and regions. In the next sections we describe the different components of MiddleWhere in more detail.

3 Location Model

MiddleWhere uses a hybrid location model, which includes both coordinate as well as symbolic location information. A coordinate location model expresses location data in the form of an (x,y,z) coordinate with respect to certain reference axes. A symbolic location model gives names to various location regions (such as rooms or floors).

MiddleWhere views location in a hierarchical manner, which makes it suitable for both outdoor and indoor environments. Outdoor environments can be hierarchically divided into countries, states, cities, boroughs, blocks and so on. Indoor locations consist of buildings, floors and rooms. In this paper, we focus on indoor environments, though the middleware can be extended to outdoor environments as well. The coordinate model of location also follows a hierarchical organization. Each building, floor and room has its own coordinate axes and a point of origin. Locations within a room can be expressed with respect to the coordinate system of the room, the floor or the building. MiddleWhere stores the relationships between the different coordinate axes, and hence coordinates can be easily converted from one system to another.

Having multiple coordinate axes for different levels of granularity within a building allows easier specification of location information. For example, if a location-sensitive application is being developed for a specific room, the application developer can specify locations with respect to the room's coordinate system. He does not have to worry about the coordinate system of the floor or the building. Forcing developers to adhere to a single coordinate system would make their applications unwieldy and difficult to change.

MiddleWhere allows defining symbolic locations by giving names to specific regions. Each symbolic location is associated with a coordinate location in a certain coordinate system. For example, names of rooms are associated with the vertices of a polygon representing the room. These vertices are expressed with respect to the coordinate system of the floor.

The location model defines three types of locations: points, lines and polygons. Symbolic locations can be defined for points, lines or polygons. For example, a symbolic point location can be defined for a light-switch by giving its (x, y, z) position with respect to the rooms or floors coordinate system. A symbolic line location can be defined for a door, and a polygon for a table or the floor-space next to a wall mounted screen.

3.1 GLOB

MiddleWhere represents a location in a hierarchical format called a GLOB (Gaia LOcation Byte-string). A GLOB can represent both coordinate as well as symbolic locations. Also, GLOBs can represent point, line or polygon regions. In the case of a coordinate location, the GLOB contains information about the axes with respect to which the coordinates are expressed. A GLOB uses a hierarchical representation of location similar to a directory structure. Some examples of GLOBs are:

- *SC/3/3216/lightswitch1* represents a point location The same location may also be represented in a coordinate format as *SC/3/3216/(12,3,4)* . This means that the *lightswitch1* is located at the coordinate (12,3,4) with respect to the coordinate system of room 3216 in floor 3 of building SC (Siebel Center for Computer Science).
- *SC/3/3216/Door2* or *SC/3/3216/(1,3),(4,5)* represents a door
- *SC/3/3216* or *SC/3/(45,12),(45,40),(65,40),(65,12)* represents the room 3216

3.2 Quality of Location Information

A ubiquitous computing environment provides many different ways of sensing the location of a mobile object (such as a person or a mobile device). Each sensing technology provides location information with different quality. We measure the quality of location information according to three metrics:

1. *Resolution*, which is the region that the sensor says the mobile object is in. Resolution can be expressed either as a distance or as a symbolic location, depending on the kind of sensor. Sensors like RF badges or GPS devices give resolution in terms of distance. For example, some GPS devices have a resolution of 50 feet, which means that the object lies within a circle of 50 feet from the location given. Other sensors

such as card-readers give resolution in terms of a symbolic location, like a room. For example, a card reader says that a person is somewhere inside a room.

2. *Confidence*, which is measured as the probability that the person is actually within a certain area returned by the sensor. This probability is calculated based on which sensors can detect the person in the area of interest.

3. *Freshness*, which is measured based on the time that has elapsed since the sensor reading. All sensor readings have an expiry time, beyond which the reading is no longer valid. Besides, our location model employs a temporal degradation function (tdf) that reduces the confidence of the location information from a particular sensor with time,

$$tdf_{sensor\text{-}type} : conf \times time \rightarrow conf$$

The tdf may degrade the confidence in a continuous or in a discrete manner with time. Each location sensing technology (or sensor-type) in MiddleWhere is associated with a resolution, a confidence level and a temporal degradation function.

4 The Location Service

The Location Service is the source of location information for all location-sensitive applications. It reasons about location information from different sensors and provides a consolidated view to all location-sensitive applications. It performs the following tasks. (1) Fuses data from multiple sensors and resolves conflicts. (2) Answers object-based and region-based queries. (3) Accepts subscriptions for location-based conditions and notifies applications when the conditions become true. (4) Supports the creation of spatial regions and the association of different kinds of properties with these regions. (5) Supports the addition of static objects, along with spatial properties of these objects. (6) Deduces a number of higher-level spatial relationship functions.

4.1 Multi-sensor location information fusion

MiddleWhere allows the use of different kinds of sensors. Different sensors give location information in different formats (either as coordinate or as symbolic locations), and with different resolution, confidence and freshness. Multi-sensor fusion uses data from different sensors to derive a spatial probability distribution of the location of the person, which is the probability that the person is in different regions of space.

4.1.1. Sensor Errors

Before we describe our algorithm, we first characterize the kinds of errors sensors can have. There are in general two kinds of errors that sensor readings can have for a certain region A. (1) The sensor says a person is not in A when he is actually in A. (2) The sensor says a person is in A when he is actually not in A.

All location sensing technologies rely on the person carrying a certain device (like a badge, a laptop or even a finger). Hence the technology only works if the person is carrying the device. Let the probability that he is carrying the device be x. The value

of x can be assumed to be 1 for biometric authentication devices, like fingerprint readers. For devices like badges, the value of x can be determined by observing user behavior.

Most product specifications of location sensing technologies give the conditional probability that the device is correctly detected when and where it is present:

P(sensor says device is in A | device is in A) = y .

In addition, location technologies also have a probability of misidentification,

i.e. P(sensor says device is in A | device is not in A) = z

Such an event occurs if a different person was in the region and the sensor incorrectly identified him. For example, a fingerprint recognizer can wrongly match the fingerprint of a person to someone else.

We shall now derive an expression for the probability, p, of the first kind of error:

p = P(sensor says person is not in A | person is actually in A)

= P(sensor says person is not in A | person is carrying device, person is in A)*P(person is carrying device) + P(sensor says person is not in A | person is not carrying device, person is in A)*P(person is not carrying device)

= (1-y)*x + (1-z)*(1-x)

The probability, q, of the second kind of error is:

q = P(sensor says person is in A | person is actually not in A)

= P(sensor says person is in A | person is carrying device, person is not in A)*P(person is carrying device) + P(sensor says person is in A | person is not carrying device, person is not in A)*P(person is not carrying device)

= z*x + (y+z)*(1-x) = z + y*(1-x)

Thus, each sensor has 2 confidence values associated with it : p and q (which in turn are derived from x, y and z). These values are used when we combine multiple sensor readings. For example, the Ubisense UWB technology deployed in our building can detect the location of a badge within 6 inches 95% of the time. Thus, for Ubisense, the area A is a circle of radius 6" centered at the location returned by Ubisense. The various sensor probabilities are: y =0.95 and z = 0.05 * area(A)/area(U) , where U is the area of coverage of Ubisense. This value comes about because the device wrongly detects the person's location with probability 0.05 and the probability that it says that the person is in A is proportional to the area of A. x is calculated from user studies which measure what percentage of time the user carries his badge with him.

4.1.2. Algorithm for Multi-Sensor Location Fusion

The input to our algorithm is sensor data about the location of people. This data can either be in a coordinate format (i.e. an (x,y,z) coordinate with an error radius) or as a symbolic location (like a room).

The first step in our algorithm is to get all the sensor data in a common format. All locations are converted to a common coordinate format (such as the building's) and are expressed as minimum bounding rectangles. While approximating sensor regions with minimum bounding rectangles decreases the accuracy of location detection, the advantages in terms of performance and simplicity far outweigh the loss in accuracy. Many operations like finding intersection regions, area and containment properties are very easy and fast to perform on rectangles (as opposed to circles or arbitrary polygons).

Various sensor rectangles are then combined with the intuition that different sensors reinforce one another if their rectangles intersect, and are in conflict if their rectangles are disjoint. In order to explain this intuition, we take the case of two different sensor rectangles. There are 3 cases: when one rectangle contains the other, when they intersect and when they are disjoint.

Case 1: One rectangle contains the other:

In Figure 2, sensor s_1 says that the person is in the inner rectangle A, and sensor s_2 says that the person is in the outer rectangle B. The sensors are also associated with probability specs, p_1, q_1, p_2 and q_2 (as obtained from the previous section).

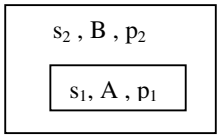

Figure 2.

These two sensor readings partition the world into 3 different regions, rectangle A, rectangle B and the region outside B. Hence, we can calculate the probability that the person is actually within these regions using the data from the two sensors. Let the event that the person is in rectangle j be represented as $person_j$ and let the event that sensor s_i says he is in rectangle j be represented as $s_{i,j}$ (where *i* is 1 or 2 and *j* is A or B). We use Bayes theorem to get the conditional probability that the person is in various regions given the sensor readings.

$$P(person_B \mid s_{1,A}, s_{2,B}) =$$

$$\frac{P(s_{1,A}, s_{2,B} \mid person_B) * P(person_B)}{P(s_{1,A}, s_{2,B} \mid person_B) * P(person_B) + P(s_{1,A}, s_{2,B} \mid \neg person_B) * P(\neg person_B)} =$$

$$\frac{P(s_{1,A} \mid person_B) * P(s_{2,B} \mid person_B) * P(person_B)}{P(s_{1,A}, s_{2,B} \mid person_B) * P(person_B) + P(s_{1,A}, s_{2,B} \mid \neg person_B) * P(\neg person_B)} \quad .(1)$$

(Since sensors s_1 and s_2 are conditionally independent given $person_B$)

Now, $P(person_B)$ is the probability that the person is in the rectangle B. The value of this depends on the movement patterns of B. In order to calculate this, we would need to measure how much time a person spends in different regions. However, in the absence of such data, we assume that the person is equally likely to be in any region. In that case, the probability that the person is in rectangle B is $area_B/area_U$, where U represents the whole universe under consideration and $area_i$ is the area of region i (where i is U or B). In our setting, U is the floor-area of the entire building.

Now, $P(s_{1,A} \mid person_B) =$

$$P(s_{1,A} \mid person_A, person_B) * P(person_A \mid person_B) + P(s_{1,A} \mid \neg person_A, person_B) * P(\neg person_A \mid person_B)$$

$$= p_1 * area_A / area_B + q_1 * (1 - area_A/area_B) \quad \ldots(2)$$

Similarly, $P(s_{1,A} \mid \neg person_B) =$

$$P(s_{1,A} \mid person_A, \neg person_B) * P(person_A \mid \neg person_B) + P(s_{1,A} \mid \neg person_A, \neg person_B) * P(\neg person_A \mid \neg person_B)$$

$$= 0 + q_1 * 1 = q_1 \quad \ldots(3)$$

From Equations (1), (2) and (3),

$$P(\text{person}_B \mid s_{1,A}, s_{2,B}) = \frac{[p_1 * \text{area}_A / \text{area}_B + q_1 * (1 - \text{area}_A/\text{area}_B)] * p_2 * \text{area}_B/\text{area}_U}{[p_1 * \text{area}_A / \text{area}_B + q_1 * (1 - \text{area}_A/\text{area}_B)] * p_2 * \text{area}_B/\text{area}_U + q_1 * q_2 * (1 - \text{area}_B/\text{area}_U)}$$

$$= \frac{[p_1 * \text{area}_A + q_1 * (\text{area}_B - \text{area}_A)] * p_2}{[p_1 * \text{area}_A + q_1 * (\text{area}_B - \text{area}_A)] * p_2 + q_1 * q_2 * (\text{area}_U - \text{area}_B)} \quad \ldots (4)$$

If only a single sensor (say sensor s_2) detected the person, then, using a similar process,

$$P(\text{person}_B \mid s_{2,B}) = \frac{[\text{area}_B] * p_2}{[\text{area}_B] * p_2 + q_2 * (\text{area}_U - \text{area}_B)} \quad \ldots. (5)$$

It can be verified that that $P(\text{person}_B \mid s_{1,A}, s_{2,B}) > P(\text{person}_B \mid s_{2,B})$ if $p_1 > q_1$, which will be true if there is a greater chance of the sensor giving the correct reading than a wrong reading. This implies that the two sensor readings reinforce each other and increase the probability of the person being in the region.

Similarly, we calculate the probability that the person is in area A. Note that all p_i's are net probabilities obtained after applying the temporal degradation function.

Case 2: The rectangles intersect

In Fig 2, the rectangles A and B intersect and a new intersection rectangle is formed - C. In this case, too, we can calculate the probability that the person is in the various areas. We just show one of the areas due to space constraints:

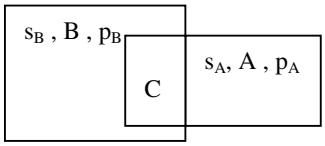

Figure 3.

$$P(\text{person}_C \mid s_{1,A}, s_{2,B}) = \frac{p_1 * p_2 * \text{area}_C}{p_1 * p_2 * \text{area}_C + [p_1 * (\text{area}_A - \text{area}_C) + q_1 * (\text{area}_U - \text{area}_A)] * [p_2 * (\text{area}_B - \text{area}_C) + q_2 * (\text{area}_U - \text{area}_B)]} \quad \ldots(6)$$

Case 3: The rectangles are disjoint

Disjoint rectangles imply that the sensors are giving conflicting information. This means that one of the sensor readings is wrong and should be discarded. We use a set of rules to decide which the wrong reading is. An example set of rules is shown below:

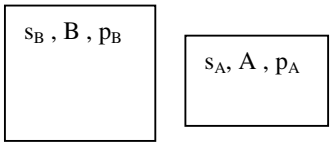

Figure 4

1. If either of the rectangles is moving with time, then take that reading and discard the other one. A moving rectangle implies that the person is carrying a location device (such as a badge) and thus has a greater chance of being valid than a stationary rectangle (which may occur if the person has left his badge in his office).
2. else, if $P(\text{person}_B \mid s_{2,B}) < P(\text{person}_A \mid s_{1,A})$, then discard reading B (or vice-versa)

In order to efficiently combine different sensor readings, we construct a lattice of rectangles, where the lattice relationship is containment. The rectangles in the lattice are both sensor rectangles as well as any new rectangle regions that are formed due to

the intersection of two rectangles. The children of any node in the lattice are all rectangles that are contained by the node.

For example, assume that there were 5 different sensors that detected the location of a person. Their sensor rectangles (S1, S2, S3, S4, S5) are as shown in Fig 5. Besides, the various rectangles create many new intersection regions (D, E, F, G). These regions form a lattice as shown in Fig 6.

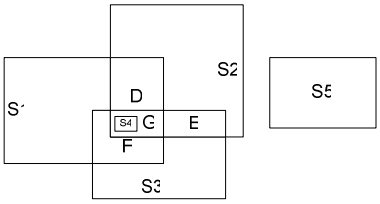

Figure 5. Many sensor rectangles

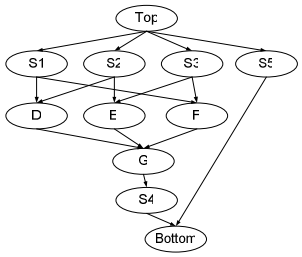

Figure 6. Lattice of rectangles

The probability associated with any node in the lattice is influenced by all sensor rectangles that contain it, intersect it or are contained within it. For example, the probability that the person is actually within the region D (which is the intersection of rectangles S1 and S2) is influenced by sensors s_1, s_2, s_3 and s_4. The general formula for the probability that the person is in any region R given n sensor readings $s_{i,Ai}$ with probabilities p_i ,i=1,…n is

$$P(\text{person}_R \mid s_{i,Ai}) = \frac{\prod_{i=1}^{n}\left[p_i * \text{area}_{\text{int(Ai,R)}} + q_i*(\text{area}_R - \text{area}_{\text{int(Ai,R)}})\right]}{\prod_{i=1}^{n}\left[p_i * \text{area}_{\text{int(Ai,R)}} + q_i*(\text{area}_R - \text{area}_{\text{int(Ai,R)}})\right] + \prod_{i=1}^{n}\left[p_i * (\text{area}_{Ai} - \text{area}_{\text{int(Ai,R)}}) + q_i*(\text{area}_U - \text{area}_R - \text{area}_{Ai} + \text{area}_{\text{int(Ai,R)}})\right]} \quad \ldots(7)$$

The *int()* function in the above equation returns the intersection of two regions. The probabilities of all regions are finally normalized.

4.2 Queries

The Location Service handles both object-based and region-based queries. The lattice obtained before gives a spatial probability distribution of the location of the person. However, most location-sensitive applications just require a single value for the location of a person and do not want to deal with a spatial probability distribution. Hence, we need to infer a single value for the location of the person from the lattice. To do this, we compare all the parents of the "Bottom" node (since these give the smallest areas). If "Bottom" has just one parent, then the rectangle corresponding to that parent is returned to the application. If there are many parents, then it means that the various sensors report two or more disjoint rectangles. Hence, there is a conflict and just one of the parents must be chosen and the rest discarded. To choose the most likely parent, we use the rules for conflict resolution mentioned earlier. For example, if S4 or one of its parent rectangles is moving with time and S5 is stationary, then S4

is chosen as the actual location of the person. S5 is removed from the lattice. The probability that the person is in S4 is the probability associated with the node.

Applications can also make region-based queries, e.g. what is the probability that a person is located within a certain region. To calculate this, we approximate the region with a minimum bounding rectangle and insert this into the lattice. We find the probability of this rectangle (using Eq. 7) and return it to the application.

4.3 Region-Based Notifications

The other common kind of location-based interaction required by applications is a notification when a person enters a certain region of interest. We have an efficient algorithm to determine if a person entered a certain region with a certain probability. All regions of interest required by various applications are organized into a lattice, just as before. All sensor rectangles are inserted into the lattice as well. The probabilities of all rectangles are now calculated (in a manner similar to what was shown before for rectangle D). Conflicting sensors are removed. Finally, if the probability that the person is within a notification rectangle exceeds a certain threshold, the application is notified.

4.4 Classifying the Probability Space

The lattice described above gives the probability that the person is in various regions. Applications can get the probability and handle it, if they choose it. However, most application developers, in our experience, do not want to deal with actual probability values. For example, it is difficult for application developers to specify different behaviors for the application if the probability that the person is in a region is known with a probability of 0.91, as opposed to 0.93, for instance. Hence, to make it more convenient for application developers, we divide the probability space into various regions. Our current implementation divides the probability space into 4 regions based on the accuracy of various sensors:

(0, min(p_i's of all sensors)]: low probability

(min(p_i's of all sensors), median of all p_i's] : medium probability

(median of all p_i's , highest of all p_i's] : high probability

(highest of all pi's, 1] : very high probability

Applications can, thus, choose to be notified if the location of the person is known with low, medium, high or very high probability. Alternatively, an application can explicitly ask for the probability and interpret it as it sees fit.

4.5 Symbolic Regions

Many location-sensitive applications prefer getting location information as a symbolic region rather than a coordinate. For example, when somebody queries for a person's location, he would prefer getting the location as a room number or floor number rather than as a coordinate In order to give location information as a sym-

bolic region, the Location Service maintains a lattice of all symbolic regions. This includes rooms, corridors and other building structures. In addition, other symbolic locations can be defined such as "East wing of the building" or "work region inside a room", etc. The lattice representation also allows incorporating privacy constraints that specify that a user's location can only be revealed upto a certain granularity (like a room or a floor).

4.6 Spatial Relationship Functions

So far we have only talked about single objects and regions. However, the richness of ubiquitous computing interactions arises from the relationships between objects and regions. The Location Service calculates different kinds of commonly used spatial relationships between objects and regions. The availability of these functions in the Location Service simplifies the development of location sensitive applications since application developers do not have to re-write these functions. We also associate probabilities with spatial relations, which are derived from the probabilities of locations of the objects in the relation. There are 3 types of spatial relationships: (a) Relations between two regions. (b) Relations between an object and a region. (c) Relations between two objects.

4.6.1. Relations between two regions

We define several relations between regions based on the Region Connection Calculus (RCC) [2]. RCC is a first order theory of spatial regions. RCC-8 defines various topological relationships: Dis-Connection (DC), External Connection (EC), Partial Overlap (PO), Tangential Proper Part (TPP), Non-Tangential Proper Part (NTPP) and Equality (EQ). Any two regions are related by exactly one of these relations.

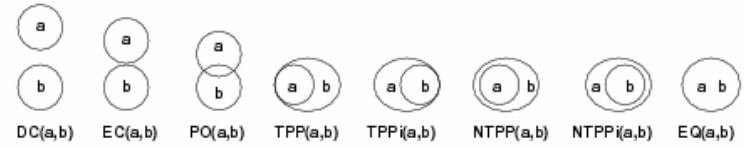

Figure 7 Different relations between regions, as explained in [2]

A key relation is that of external connectedness (EC). If two regions are externally connected, it means that it may be possible to go from one region to another. An example of this is two rooms that are connected by a door. However two adjacent rooms that just have a wall (with no door) in between are also externally connected. To make this distinction, we define three additional relations:

ECFP(a,b) is true if EC(a,b) and there is a free passage to go from a to b.

ECRP(a,b) is true if EC(a,b) and there is a restricted passage to go from a to b.

ECNP(a,b) is true if EC(a,b) and there is a no passage to go from a to b.

An example of a restricted passage is a door that is normally locked and which requires either a card swipe or a key to open. The various relations between regions are useful for a number of applications such as route-finding applications.

Evaluating the relation between 2 regions is just O(1) given the vertices of the two regions. The vertices of all the rooms and corridors in the building are obtained from the blueprints of the building. The vertices of application defined regions are given by the application. Finally, the relations ECFP, ECRP and ECNP are evaluated by checking if there is a door or an obstruction like a wall between the regions. The Location Service reasons further about these relations using XSB Prolog [3].

Another relation between regions is distance. Two kinds of distance measures are used: Euclidean, which is the shortest straight line distance between the centers of the regions, and path-distance, which is the length of a path from the center of one region to the center of the other region.

4.6.2. Relations between an object and a region

MiddleWhere defines various relations between an object and a region. These relations are probabilistic if the location of the object is only known with some probability. Some of the main relations defined are: (a) *Containment*: whether an object is within a certain region. (b) *Usage*: Usage Regions are defined for certain objects (like displays or tables) such that if a person has to use these objects for some purpose, he has to be within the usage region of the object. (c) *Distance*: the distance from an object to a region (Euclidean or path-based).

4.6.3. Relations between two objects

The main relations between two objects are (a) *Proximity*: whether the two objects are closer than a pre-defined distance. (b) *Co-location*: whether the two objects are located in the same symbolic region (of a specified granularity such as room, floor or building). (c) *Distance*: the Euclidean or path-based distance between the two objects.

5 Spatial Database

MiddleWhere uses a spatial database for modeling the physical space and storing location information from various sensors. The spatial database also supports operations on geometric data types such as intersection, union, disjoint and so on. These operations are used by the Location Service for reasoning about spatial relationships between objects and regions.

5.1 Modeling the Physical Space

The physical space consists of objects and regions. Objects are represented as points, lines or polygons while regions are represented using minimum bounding rectangles (MBR). An MBR of a region is a rectangle of minimum area that fully encloses the region. This approximation enables ease of representation and reasoning [4]. The concept of minimum bounding rectangles is used heavily by the spatial data mining community [5]. Minimum bounding rectangles provide approximate boundaries to objects of interest to enable efficient processing of operations such as checking for certain spatial characteristics, verifying proximity of an object to another object and

so on. Once a certain condition is satisfied by a MBR, more accurate processing of the operation is performed taking the actual region boundaries. Figure 8 and Table 1 show the graphical and spatial representation of our floor.

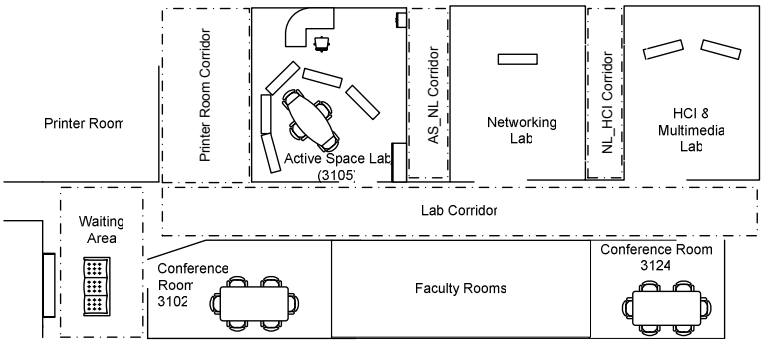

Figure 8. Graphical Layout of floor

Table 1. Database table representing the floor

Object Identifier	Glob Prefix	Object Type	Geometry Type	Points
Floor3	CS	Floor	Polygon	(0, 0), (0,500), (500, 100), (0,100)
3105	CS/Floor3	Room	Polygon	(330, 0), (350, 0), (350,30), (330,30)
NetLab	CS/Floor3	Room	Polygon	(360, 0), (380,0), (380, 30), (360,30)
HCILab	CS/Floor3	Room	Polygon	…
LabCorridor	CS/Floor3	Corridor	Polygon	(310,0), (330,0), (330,30), (310,30)

The *ObjectIdentifier* is a unique name in the name space of *GlobPrefix.* The *GlobPrefix* field specifies the identity of the enclosing space for an object. For example, NetLab is located in Floor3 of the CS department. *GlobPrefix* and *ObjectIdentifier* make up the combined key for the spatial table. The *ObjectType* field assigns semantic information to the object such as *Room*, *Corridor*, *Floor*, *chair, table,* etc. The *GeometryType* field specifies the geometry type used to represent the object. Though we use bounding rectangles to represent objects in our model, certain entities such as non-enclosing walls, light switches, etc are more conveniently represented with other geometry types such as lines and points. Finally, the Points field represents a sequence of points describing the geometry. In addition to the information mentioned above, the database also stores spatial properties of objects, like location, dimension, orientation, etc. Furthermore, modeling the physical space allows SQL queries on objects and regions. An example query is 'Where is the nearest region that has power outlets and high Bluetooth signal?'

5.2 Representing Sensor Information

Sensor information is stored in a separate table in the spatial database. Sensor information from various sensors is converted, by sensor adapters, to a common sensor schema before inserting it in the sensor information table. The table contains temporal

information indicating the time when the sensor reading was obtained. The sensor information table schema and some sample sensor readings are shown in Table 2.

Table 2. Sensor Information Table Schema and sample sensor readings

Sensor Id	Glob Prefix	Sensor Type	MObject Id	Obj Location	Detection Radius	Detection Time
RF-12	SC/Floor3/3105	RF	tom-pda	(5, 22,9)	30	11:52:35
Ubi-18	SC/Floor3/3102	Ubisense	ralph-bat	(41,3,9)	6	11:51:22

We maintain a separate table for storing information about each sensor. This table contains the confidence with which a sensor can detect the location of an object and the time-to-live information of the sensor data. Each sensor is associated with a confidence value that measures the uncertainty that is associated with a sensor's reading. This confidence value is found through empirical means. The time-to-live information indicates the time before a certain sensor reading expires. For example, a card reader has a time-to-live value of 10 seconds. A card reader location value that is older than 10 seconds is considered stale. The sensor table schema and sample data are shown below.

SensorId	**Confidence(%)**	**Time-to-live(s)**
RF-12	72	60
Ubisense-18	93	3

5.3 Location Triggers

Location triggers are events that are generated when a certain spatial condition is satisfied. These conditions include mobile object entering a certain region, mobile object at a certain distance from another object and so on. MiddleWhere uses the spatial database to generate location triggers. Applications can subscribe to receive triggers by specifying spatial conditions. MiddleWhere interprets these conditions into appropriate database triggers and creates these triggers in the database. When a condition is satisfied, the spatial database generates the corresponding trigger. MiddleWhere maintains an internal list of subscribers and trigger identifiers and when it receives a trigger it redirects it to the subscribed application.

6 Location Sensors and Adapters

In order to facilitate plug-and-play support for new location technologies, at the lowest layer of MiddleWhere (Figure 1), we define an object called a *location adapter*. The location adapter is a CORBA client wrapper for the specific location technology at hand. The adapter communicates natively to the interface exposed by the location technology, and acts as a device driver that allows the location sensor to work with MiddleWhere seamlessly.

Upon installing a new location technology, a calibration process needs to be undertaken. This process involves using the characteristics and specifications of the location sensor to convert the location readings to symbolic and/or coordinate location information that matches the location model and coordinate system that MiddleWhere

uses. In addition, the two confidence values p and q (as mentioned in Section 4) are estimated. In essence, the adapter translates the location readings into a GLOB that is fed into MiddleWhere through the provider interface.

Every adapter has an *adapter ID* and an *adapter type.* The adapter ID uniquely identifies a particular adapter. The adapter type classifies adapter objects based on the location technology they wrap. Different instances of the same adapter type can be created to wrap multiple sensors of the same type. For instance, we are running RF badge base stations in three different locations. In each location, an RF badge adapter is instantiated with the correct information.

At this time, we implemented adapters for four different location technologies:

1. *Ubisense*™. Ubisense consists of tags and base stations that utilize Ultra WideBand technology. The base stations are able to pinpoint the location of a tag within 6 inches 95% of the time. As described in Section 4, we estimate the confidence values of Ubiense as follows. Area A is a circle of radius 6" centered at the location returned by Ubisense, where y = 0.95, and z = 0.05 * area(A)/area(U), where U is the area of coverage of Ubisense.
2. *RFID Badges* [6]. These are RF-based active badges that can transmit identification information. This identification information is in the form of a 32 byte string. This string can be written into the badge. The transmitted ID is received by base stations that can be positioned in different locations. The base stations can detect badges within a range of approx. 15 ft. This system cannot give exact coordinates of the badge; instead, it is capable of capturing the IDs of the badges in its vicinity. In our experiments, we found that different obstacles can weaken the signal significantly, thus, the best set up for the RF badges is to define an area of interest, A, and set up a base station in the center of A. No error rates are documented for this device, but we found out that there are good chances that it is not picked up due to the system's inaccuracy. So we set y = 0.75, and z = 0.25 * area(A)/area(U), where U is the area of coverage as documented in RF badges hardware specs.
3. *Fingerprint devices and other biometric logins*. In many scenarios, users of our system are required to authenticate to access some data or perform some tasks. Most biometric authentication technologies require physical presence. We exploit this information to get short-lived but relatively accurate readings of a person's location. Unlike the previous technologies, these devices do not transmit continuous signals when users are in the vicinity. In addition, we assume that these devices are secure, i.e., it is very unlikely that a fingerprint device would detect a positive fingerprint match of a user without that user being there physically! Once a user is identified, there is a good possibility that the user may leave the vicinity. In many cases, users are encouraged to manually logout for security reasons. However, in reality, people often forget to logout before leaving the vicinity. For this reason, a biometric authentication adapter provides two different location readings to MiddleWhere: a short-term reading, and a longer-term reading. For short-term reading, we set the expiration time to 30 seconds, define a small area (in our case, a circle centered at the device position with a radius of 2 feet), set y = 0.99, z=0.01 and x=1 (because of our assumptions). In the second reading, we set the expiration time to *T* minutes, where *T* can be estimated based on user studies for finding how long a person is likely to stay in the room after authenticating. For our purposes we found

that $T = 15$ minutes is reasonable, given the fact that confidence will degrade with time anyway. In this reading, the area is set to the whole room, and z is set to the probability of a user leaving the room before T and without manual logout. If a user elects to logout manually, then this is a clear indication that the user is in the room now, but he is leaving soon. So, the adapter feeds the system with a short-term location reading, where expiration time is 15 seconds, radius is 2 feet, y = 0.99, z=0.01, and x=1. The adapter also forces all location information relating to that user and obtained from the same device to expire immediately.

4. *GPS*. The GPS adapter works as follows. The GPS device tries to achieve a satellite lock. If successful, the adapter should be able to translate longitude, latitude, and altitude information into a coordinate location that matches MiddleWhere's coordinate system. Unlike the above technologies, GPS can give an estimation of its accuracy; therefore, the adapter uses this value for calculating the confidence values. If the GPS receiver estimates an accuracy of 15 feet, we set area A to a sphere with a radius of 15 feet. We can set y=0.99 and z=0.01 (assuming that the accuracy estimate of the GPS is correct), however, x, will still equal the probability of a person not carrying his GPS device.

7 Implementation

In this section, we provide a brief description of MiddleWhere implementation. We use CORBA to enable distributed communication between MiddleWhere components, Applications, and adapters (that wrap the location technologies). To implement the spatial database, we use PostGIS with the PostgreSQL object-relational database. PostGIS adds support for geographic objects and provide basic spatial support.

While MiddleWhere can run as a standalone service in any distributed computing environment, our objective was to develop a general-purpose location middleware that can be integrated into Gaia. Gaia [7] is a generic computational environment that integrates physical spaces and their ubiquitous computing devices into a programmable computing and communication system. Gaia provides the infrastructure and core services necessary for constructing ubiquitous computing environments. We implement MiddleWhere as an extended Gaia service. Gaia applications can discover the location service component of MiddleWhere by querying the Gaia Space Repository service, which provides a list of available services. Gaia applications can then talk directly to the location service. To access location information, we provide push and pull models. An application can choose to query the location service for location information for a particular object or person, or it can define one or more location triggers for regions of interest, where it is notified when an object of interest is detected inside that region. Additionally, applications can choose to query about confidence levels. We integrated four different location technologies in the system (as mentioned in Section 6), at this time, the location sensors cover four different rooms, that includes a lab, a conference room, and two offices.

8 Example Applications

To demonstrate our system, we have developed several location-aware applications. We briefly mention some of them here:

1. *Follow Me Application*. In this application, we define a user session as a set of applications and files that a user interacts with. The session also includes state information and customization options chosen by the user. If a user moves out of the vicinity of the display he is using, the application will automatically suspend the session. When a user is detected in the vicinity of any other display or workstation, the session is automatically migrated and resumed at that machine. In effect, users can resume their work anywhere and anytime without having to remember to save the latest changes or to worry about copying their data to a removable disk. To implement this application, we create a "user proxy," which manages the sessions of a certain user. The proxy then uses MiddleWhere to discover the location of the user. If the location of the user is obtained, the proxy queries the Location Service for nearby displays or workstations that are suitable for resuming the session; if found, the session can be resumed immediately on the new display. The user can customize the behavior of the *Follow Me* application by changing the settings in the user proxy to accommodate privacy preferences.

2. *Anywhere Instant Messaging.* This application allows a user to receive instant messages from a designated list of "buddies" on whichever display is closest to him. A user can customize the application by choosing to block particular users at certain locations, or by configuring the system to display private messages only if the location accuracy is 'high' and other users are not in the immediate vicinity!

3. *Location-Based Notifications.* In this application, notifications are sent to people located in a particular geographical boundary (which could be a region or a sphere, etc.) The notification may be a message like "The store is closing in five minutes," for example. This application is implemented by setting up location triggers in the target area, and maintaining a list of users in the region.

4. *Vocal Personnel Locator:* This application combines voice recognition with location-awareness. A user asks the computer to locate a person or an object using a speech interface. The application then queries the spatial database for the required info, and replies verbally.

9 Evaluation

We have evaluated the performance of MiddleWhere on a 4 CPU 3.06 GHz machine with 3.6 GB RAM. The spatial database used was PostGIS 0.8.1 with PostgreSQL 7.3.4 and the communication middleware was Orbacus. Figure 9 shows the time taken for a trigger to be notified by MiddleWhere. The graph shows the trigger response times for 10 different updates to the location service. The various curves indicate the number of trigger notifications programmed into the location service. We expected the response time to increase with the number of programmed triggers but we found that the response time was almost independent of it. This indicates that MiddleWhere scales well with number of programmed triggers. It can be noticed from

the figure that the first update requires a higher trigger response time than subsequent updates. This is due to the initial setup time taken by MiddleWhere.

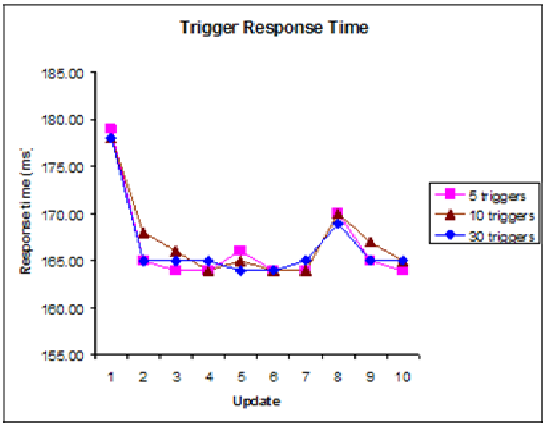

Figure 9: Trigger Response Time

10 Related Work

Location-aware computing has been an active area of research. Most projects on location-tracking focus on accurately reasoning an object's location or sensor fusion. In our work, we also focus on designing a middleware that caters to the requirements of location-aware applications.

The Location Stack [8] defines a layered modeled for fusing location information from multiple sensors and reasoning about an object's location. It, however, does not incorporate a spatial model of the physical world. It does not support representations of immobile objects and so does not support spatial reasoning relative to stationary entities such as rooms, corridors and so on.

The NEXUS project [9] uses a spatial database to model the physical world. It supports toplogical and topographic models similar to the symbolic and coordinate systems supported by our system. The focus of NEXUS is on modeling the physical world – it does not address location determination issues. It does not support multi-sensor probabilistic fusion of location information and reasoning with a spatial model as supported by MiddleWhere.

The Aura Space Service [10] provides spatial models for context-aware applications. It combines coordinate and hierarchical location models into a single hybrid model, which supports spatial queries. MiddleWhere uses the hybrid location model introduced by the Aura Space Service. The focus of the Aura Space Service is only on modeling the physical space and supporting spatial queries. It does not address location inferencing issues and does not provide a framework for spatial reasoning like MiddleWhere.

Sematic Spaces [11] has developed spatial models to represent rooms, buildings and other objects. It uses a topological model for representing relationships among various objects. The topological model is similar to the symbolic model supported by MiddleWhere. The project does not integrate the spatial model with a location system and supports no probabilistic reasoning techniques like MiddleWhere. Further, the spatial model of Semantic Spaces does not seem to support a coordinate system like MiddleWhere.

Location-based Spatial Queries [12] project addresses ways of indexing and caching spatial data for location-based queries. The focus of this project is on developing

database techniques for queries with location constraints. It does not support sensor fusion and reasoning.

11 Conclusion

We presented the design and implementation of MiddleWhere, a distributed middleware system that fuses various location technologies, resolves conflicts, and combines multi-sensor readings to get more accurate location readings for people and objects. The system facilitates the separation between applications and location technologies to enable dynamic add-on of new technologies, without changing existing applications. We demonstrated the potential of the system by integrating four different location technologies, and developing several location-aware applications.

In the future, we plan to incorporate more devices and deploy the middleware widely. We also plan to conduct user studies to get accurate values of various parameters of our system like the probability of carrying location devices and the temporal degradation function. These probability values can then be used by the middleware and location-aware applications to improve their reliability and accuracy.

12 References

[1] UbiSense, "Local position system and sentient computing." http://www.ubisense.net/.

[2] A. G. Cohn, B. Bennett, J. M. Gooday, and N. Gotts, "RCC: a calculus for Region based Qualitative Spatial Reasoning," presented at GeoInformatica, 1997.

[3] "XSB Prolog." http://xsb.sourceforge.net.

[4] A. Guttman, "R-trees: a dynamic index structure for spatial searching," presented at 1984 ACM SIGMOD international conference on Management of data, 1984.

[5] K. Koperski, J. Adhikary, and J. Han, "Spatial Data Mining: Progress and Challenges," presented at SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada, 1996.

[6] RFId, "Radio Frequency Identification, (RFID)," http://www.aimglobal.org/technologies/rfid/.

[7] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing (accepted)*, 2002.

[8] D. Graumann, W. Lara, J. Hightower, and G. Borriello, "Real-world implementation of the Location Stack: The Universal Location Framework," presented at 5th IEEE Workshop on Mobile Computing Systems & Applications, 2003.

[9] O. Lehmann, M. Bauer, C. Becker, and D. Nicklas, "From Home to World - Supporting Context-aware Applications through World Models," presented at the Second IEEE International Conference on Pervasive Computing and Communications, Orlando, FL, 2004.

[10] C. Jiang and P. Steenkiste, "A hybrid location model with a computable location identifier for ubiquitous computing," presented at Lecture Notes in Computer Science, 2498, UbiComp, 2002.

[11] B. Brumitt and S. Shafer, "Topological World Modeling Using Semantic Spaces," presented at Workshop on Location Modeling for Ubiquitous, UbiCom, 2001.

[12] B. Zheng, W.-C. Lee, and D. Lee, "Spatial Index on Air," presented at IEEE International Conference on Pervasive Computing and Communications (PerCom'03), 2003.