A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing

Changhao Jiang[+] and Peter Steenkiste[+*]

[+]Computer Science Department
[*]Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.
{jiangch,prs}@cs.cmu.edu

Abstract. Location modeling and representation are crucial technologies for context-aware applications. In this paper, we present a novel location model combining the virtues of both the hierarchical and coordinate location models, and we introduce a computable location identifier, namely *Aura Location Identifier* (ALI). We then describe how the Aura space service uses this hybrid model to handle spatial queries for context-aware applications. A simple example of such a query is a *range* query, e.g. "*select name from printer where distance(location,'ali://cmu/wean-hall/floor3/3100-corridor#(10,10,0)') <10*", where "location" is an attribute representing the location of printers. Finally, we discuss how we extended the PostgreSQL database system to provide direct support for spatial SQL queries at the database level. These extensions improve performance and increase flexibility for context-aware applications.

1 Introduction

Location information is a critical ingredient for context-aware applications for mobile users, such as diverting phone calls to the receiver nearest to a user, prefetching data to the service portal near the user's path, locating interesting objects/people, navigation, etc. As more context-aware applications are developed, researchers have started to realize that the **modeling** of the physical environment and the **representation** of location are key enabling technologies for ubiquitous computing.

The first issue is modeling of the physical environment. Numerous location models [9, 2, 4] have been defined in different application domains. In general, they can be categorized into two classes: **hierarchical** (or *topological, descriptive, symbolic*) location models and **coordinate** (or *metric, geometric*) location models. These two classes have complementary benefits and drawbacks. From the perspective of context-aware applications, neither model is by itself completely satisfactory.

The second issue is location representation. Most context-aware applications adopt a a distributed collaborative service framework, that stores location modeling data in a centralized data repository. Location-related queries issued by end-users and other services are handled by a dedicated location service. This distributed service paradigm is attractive because of its scalability and modularity. However, we need an effective and efficient location representation method to make this work.

In this paper, we will present a novel hybrid location model that combines the virtues of both the *hierarchical* and *coordinate* location models. It features a computable location identifer, the *Aura Location Identifier*, for the purpose of representing locations and enabling the exchange of location information between distributed contextual services. The hybrid location model was designed to meet the needs of establishing a space service in the Aura Project [6] for use by campus-wide context-aware applications at Carnegie Mellon University. Some example queries that must be supported are: "*find the color printer nearest to Joe*", "*find all conference rooms on the 3rd floor of Wean Hall, with wireless bandwidth above 2mbps*", "*find all the people within Wean Hall*".

The remainder of this paper is organized as follows. In Section 2, we motivate the hybrid location model using the Aura pervasive computing project. In Section 3, we argue that context-aware applications need a hybrid location model. Section 4 proposes a computable location identifier, the *Aura Location Identifer*, and we elaborate on how the hybrid location model supports the ALI representation in Section 5. In Section 6, we present the design and implementation of Aura's space service. Section 7 explains our integration of the hybrid location model into the PostgreSQL database system. In Sections 8 and 9, we review related work and summarize our paper.

2 Aura Project and Space Service

The Aura Project [6] at Carnegie Mellon University is a ubiquitous computing project that focuses on minimizing the distractions to users. The motivation is that human attention is a precious resource that does not benefit from Moore's law so we trade computer resource that do benefit from Moore's law for people's time and attention. Aura is specifically intended for ubiquitous computing environments involving wireless communication, wearable or handheld computers, and smart spaces. Human attention is an especially scarce resource in such environments, since the user is often preoccupied with walking, driving or other real-world interactions. In addition, this environment has many challenges, including the use of hand-held devices with limited network capabilities and battery power, and the presence of diverse and possibly unfamiliar physical spaces.

Automatic adaptation of the computing environment (including applications, networks, operating systems, and middleware services) to the user's context is one of the key techniques used in Aura. For example, if the user experiences poor network performance, Aura should be able to switch to a different network technology if one is available in that space. If the adaptation is successful, this eliminates the need for user intervention. We also want the system to be proactive in identifying ways of helping the user. For example, if the user is trying to view

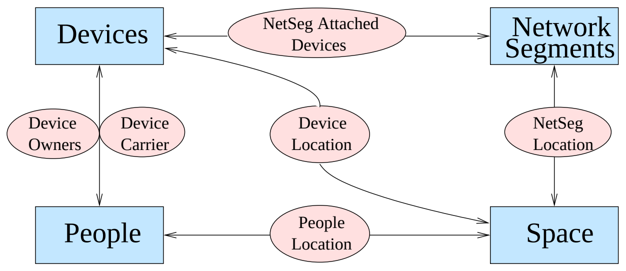

Fig. 1. A set of Aura contextual services for context-aware applications

a high resolution image on a hand-held device, Aura should be able to point out that there is a large wall-mounted display around the corner.

Automatic adaptation requires that the Aura system needs detailed information about the user's context. For this purpose, we developed the Aura contextual information services [8] (Figure 1). The CSI (Contextual Services Interface) infrastructure can provide information about the primary entities in the user's context: people, devices, physical spaces, and networks. It also provides information about relationships between these entities. For example, people-location services [11, 1] provide information on the relation between people and physical space. Note that some information maintained by the CSI is primarily static (e.g. basic information about devices and people) while some information is dynamic (e.g. the people-location relationship).

The API for the contextual services allows users to retrieve information stored or collected by the services using SQL-like queries. For example, a query to the device-space service could request all "printers" that match "`Type=color`" and "`Space=Wean_Hall_8th_Floor`".

The space service and the space relationship services (e.g. people-location) play a critical role in this architecture and they are the focus of this paper. We can identify two requirements. First, different services and applications have to be able to exchange location information. For example, the people-location service must return location information in a format that can then be used to query other services (e.g. to find a nearby printer using the device-location service). Second, we must devise efficient scalable implementations of the space-related services so that these common queries can be handled quickly.

3 Hybrid Location Model

One important issue for space services and more broadly for context-aware applications, is that they must model the physical environment through an appropriate location model. As we mentioned earlier, the prevailing location models fall into two groups: *hierarchical* and *coordinate* [5] models. In this section, we argue that neither model by itself is satisfactory for context-aware applications and a hybrid location model combining the benefits of both models is needed.

Hierarchical location models decompose the physical environment in different levels of precision, and normally feature a self-descriptive location representation. A typical example of a hierarchical location model is the postal address, e.g. *U.S.A., Pennsylvania, Pittsburgh, Carnegie Mellon University, Wean Hall, 3rd floor, Room 3515.* Coordinate location models superimpose a grid on the physical environment thus providing a coordinate system that can be used to represent locations uniquely and accurately using a tuple of numbers. A typical example of a coordinate location model is the GPS coordinate system, in which locations are defined by (*longitude, latitude, altitude*).

Both classes of location models have advantages and disadvantages with respect to the needs of context-aware applications. The hierarchical location model is good for its implicit representation of spatial relationships, such as *containment, closeness*, and it has the virtue of human readability. It also handles some location queries very efficiently, For example, the query "*find the people who live on Forbes Avenue*" is satisfied by people whose address prefix is "*Forbes Avenue*". The biggest disadvantages of the hierarchical location model are its lack of position precision and the fact that it is difficult or inefficient to compute distances. The flaws of hierarchical location model embody the benefits of the coordinate location model. With built-in geometric attributes, coordinate location models are well suited for specifying locations precisely and for computing distance accurately. However, the coordinate location model hides hierarchical relationships. Hence, it needs additional specifications to deduce spatial relationships.

We propose a hybrid location model for context-aware applications that combines the benefits from both sides. The starting point for the hybrid model is the hierarchical location model: we view the world as a hierarchy of spaces and each level further refines and subdivides the spaces of the previous level. We bring in the coordinate model by allowing each space in the hierarchy to define a coordinate system that can be used to define points or areas within that space. Different spaces may use different coordinate systems, as we describe later. The coordinates allow us to define points or areas for which there is no name in the hierarchical name system. For example, we can identify the location of a camera in a room by concatenating hierarchical name of the room with the coordinates of the camera in the room. Similarly, we can describe the coverage area of a cell in a wireless network (which in general does not line up with rooms) using coordinates in the building or floor.

In the remainder of this paper we describe how we can use the hybrid location model as a basis to name locations and areas (Section 4) and to build contextual services that resolve space-related queries efficiently (Section 5 through 7).

4 Aura Location Identifier

The Aura Location Identifier is used to represent locations based on the hybrid location model. We first review the type of information we need to represent, and we then introduce the ALI representation.

4.1 Types of Location

Based on the requirements of context-aware applications, we identify three types of location information that we must be able to represent:

- **Space** location is a physical space entity, e.g. "*room 3115 of 3rd floor of Wean Hall at CMU*".
- **Area** location is a space not physically demarcated, but virtually defined by applications, e.g. "*the area covered by a particular wireless access point*".
- **Point** location is a position of mobile user or object. Usually, we are not interested in the shape and extension of that user or object, but just in its position, e.g. "*the location of printer 'slate'*".

The ALI space type is composed of only a hierarchical name, while the two other ALI types append explicit geometric information to a hierarchical name. We present examples of all three types of ALI in Section 4.3.

4.2 Syntax of Aura Location Identifier

The Aura Location Identifier uses a formatted string representation complying with the generic syntax of a Universal Resource Identifier [3]. Here is the Barcus-Naur Form (BNF) notation for the Aura Location Identifier:

```
<ALI>      ::= [ ali:// ] <Path> ["#" <Position>]
<Position> ::= <Pt-3D> | <Area> ["-" <Height>]
<Path>     ::= <Space> {"/" <Space>}
<Area>     ::= "{" <Pt-2D> "," <Pt-2D> "," <Pt-2D> {"," <Pt-2D>} "}"
<Height>   ::= "(" <Float> "," <Float> ")"
<Pt-3D>    ::= "(" <Float> "," <Float> "," <Float> ")"
<Pt-2D>    ::= "(" <Float> "," <Float> ")"
<Space>    ::= <Char> {<Char>}
<Float>    ::= ["+" | "-"] <Digit> {<Digit>} ["." <Digit> {<Digit>}]
<Char>     ::= <Alphanum> | "-" | "_"
<Alphanum> ::= <Alpha> | <Digit>
<Alpha>    ::= "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|
   "m"|"n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"|"A"|
   "B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"|"O"|"P"|
   "Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"
<Digit>    ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

4.3 ALI Examples

Here are examples of three types of ALI according to the above BNF notation:

- **Space Identifier:** "*ali://cmu/wean-hall/floor3/3100-corridor/3115*"
  This identifier represents *room 3115 on 3rd floor of Wean Hall at CMU*. Its geometric attributes are stored in a centralized location data repository.

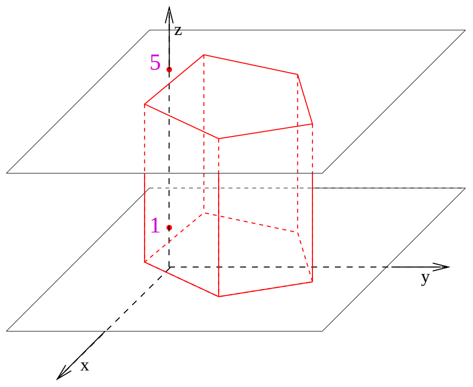

Fig. 2. "*ali://cmu/weanhall/floor3#{(1,0),(-1.5,0.5),(0,3),(2,3.5),(3,1.5)-(1,5)}*" represents an area within the 3rd floor of Wean Hall.

- **Area Identifer:** "*ali://cmu/wean-hall/floor3#{(1,0),(-1.5,0.5),(0,3),(2,3.5) ,(3,1.5)-(1,5)}*"
  This identifier represents an area on the 3rd floor of Wean Hall at CMU. The series of points after the '#' token specify a closed polygon in planes parallel to the X-Y plane of the 3rd floor's *space coordinate system* (Figure 2). "(1,5)" specifies the height range of the area (see Section 5.2 for details).
- **Point Identifier:** "*ali://cmu/wean-hall/floor3/3700-corridor/3718#(10,4,1)*"
  This identifier represents the point (10,4,1) within *room 3718*. The coordinates are relative to *room 3718's space coordinate system*.

4.4 Operators on ALI

A feature of the ALI that distinguishes it from other location representation schemes, is that, combined with a set of operators, the ALI can be viewed as an abstract data type. The benefit is that this provides options for implementing it at different levels of the system, such as the database level, the service level, and the programming language level. For example, we could define an "ALI" class in a object-oriented programming language to realize programming language-level support. As we will see later, we can also define ALI as a user-defined data type in an extended SQL standard to realize database-level support. These options provide flexibility in the context-aware application's design and implementation. It can also provide performance improvements, as we will see in Section 7 for the case of implementing ALI at the database level.

The following are some operators on ALIs that are of interest in ubiquitous computing environments:

- **distance(ali, ali) returns float**
  compute the distance between two locations.

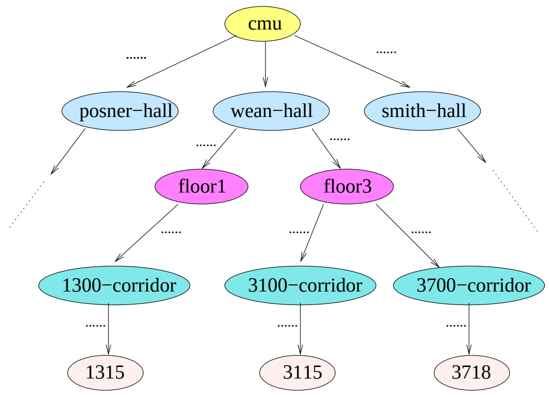

Fig. 3. Hierarchical Space Tree

- **contains(ali, ali) returns boolean**
  tell whether one location contains another.
- **within(ali,ali) returns boolean**
  tell whether one location is within another.
- **super(ali) returns ali**
  get direct super space containing the location
- **sub(ali) returns list of ali**
  get list of spaces which are direct sub space of input parameter

5 Realizing the Hybrid Location Model

The ALI bases its expressive power and computability on the hybrid location model. In this section, we describe the hybrid location model in more detail and we elaborate on how it can be used to realize the above ALI operations.

5.1 Hierarchical Aspect of the Hybrid Location Model

The hierarchical aspect of the hybrid location model means that the physical environment is decomposed into different levels of spaces. For example, the campus of *Carnegie Mellon University* is decomposed into several sub-spaces: *Wean hall, Smith Hall, Posner Hall*, etc. Each of these halls is divided into smaller composing sub-spaces, until we reach enough precision. Such a hierarchy is called a *space tree*. Each node in the space tree corresponds to an actual space in the physical environment, and the parent-child link in the tree implies a super/sub space relationships between two spaces. Based on the *space tree*, it is easy to resolve queries about the containment relationship between two physical spaces.

Figure 3 illustrates part of the *space tree* for Carnegie Mellon University. It is up to the location service (or space service) designer to decide how to

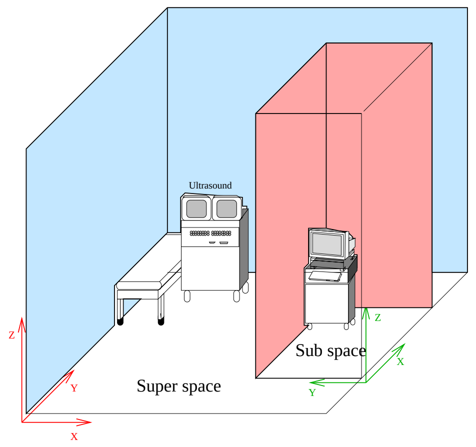

Fig. 4. Each space has its own *space coordinate system*, defined by specifying the *origin* point and three *axes* of “x”, “y” and “z”

decompose the physical environment. The location service needs to maintain a tree style data structure for the *space tree* and must handle queries about spatial relationships based on this data structure.

All types of ALIs described in Section 4.3 corresponds to nodes in *space tree*, e.g. “*ali://cmu/wean-hall/floor3/3100-corridor/3115*” could correspond to a leaf node. The ALI is formatted by joining the names of nodes on the path from that corresponding node to the root.

5.2 Coordinate Aspect of the Hybrid Location Model

In order to compute the geometric relations such as *distance, containment, and intersection* between locations, they need to possess geometric attributes, such as *shapes and point coordinates*, in a well-defined coordinate system. Most coordinate location models use a global reference coordinate system, such as GPS coordinate system. Within a global coordinate system, locations can be uniquely specified, thus making distance computation easier. However, in many cases, it is useful to support coordinates relative to a local reference system rather than to a global system. An important example is indoor spaces where GPS does not work well and a local coordinate system is more convenient. Another example is the coordinate system used by local sensing devices (e.g. badges).

For these reasons, the hybrid location model allows each space in the *space tree* to have its own coordinate system, named the *space coordinate system*.

Figure 4 shows an example of a *space coordinate system* in a physical environment. In the figure, we see two rooms, one containing the other. The bigger room is the super space of the smaller one. Both rooms have their own *space coordinate system*. The position of the computer in the smaller room can be

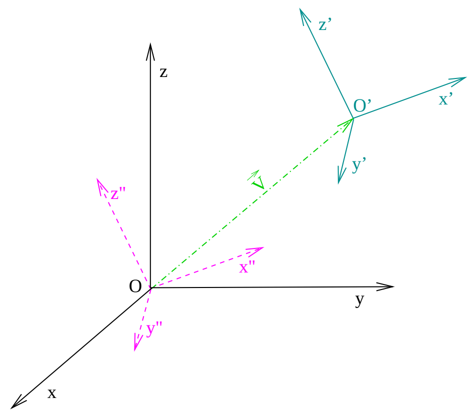

Fig. 5. Super Space Coordinate and Sub Coordinate System

expressed in coordinates of either the super space's coordinate system or the sub space's coordinate system. Coordination translation can be used to have the disparate coordinate system interoperate.

In order to translate coordinates between coordinate systems, the hybrid location model must define the position of the sub space's *space coordinate system* within the super space's *space coordinate system*. To translate coordinates between spaces with no direct super/sub relationship, we must find the path between the two spaces in the *space tree* (see Section 5.1) and translate along the path, since the composing links imply super/sub relations.

The position of a sub space's *space coordinate system* is defined by specifying its **origin point** and **axes** within the super space's coordinate system. The origin point is specified by a displacement vector, from the super space's origin to the sub space's origin (vector $\overrightarrow{OO'}$ in Figure 5). The three axes are specified by three unit vectors, respectively corresponding to the "x", "y", "z" axes of the sub space coordinate system (vector $\overrightarrow{OX''}$, $\overrightarrow{OY''}$,$\overrightarrow{OZ''}$ in Figure 5). The three vectors are expressed in the form of a matrix, called the *rotation matrix* with each row recording an axis vector.

$$\overrightarrow{V} = \overrightarrow{OO'} = [OO'_x, OO'_y, OO'_z]$$

$$R_{matrix} = \begin{bmatrix} \overrightarrow{OX''} \\ \overrightarrow{OY''} \\ \overrightarrow{OZ''} \end{bmatrix} = \begin{bmatrix} OX''_x & OX''_y & OX''_z \\ OY''_x & OY''_y & OY''_z \\ OZ''_x & OZ''_y & OZ''_z \end{bmatrix}$$

Given the position of the sub space's coordination system within the super space's coordinate system, we can use simple linear algebra to translate coordinates between them. Here are the translation formulas:

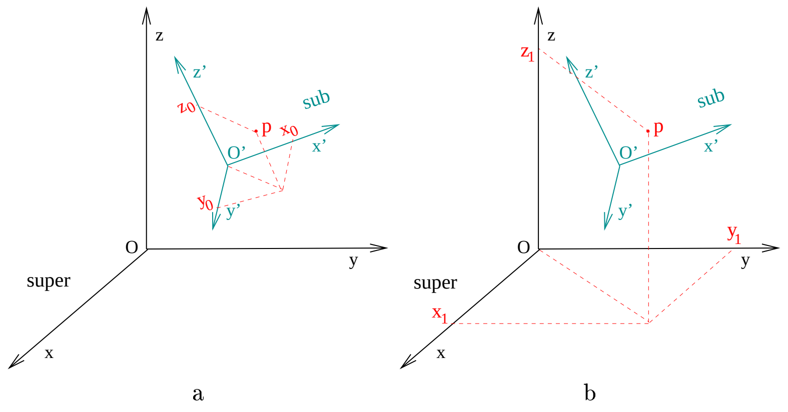

Fig. 6. Coordinate translation between super space and sub space

In Figure 6, if $\overrightarrow{O'P}_{sub} = [x0, y0, z0]$, and $\overrightarrow{OP}_{super} = [x1, y1, z1]$ then:

$$\overrightarrow{OP}_{super} = \overrightarrow{O'P}_{sub} \cdot R_{matrix} + \overrightarrow{OO'}$$

$$\overrightarrow{O'P}_{sub} = (\overrightarrow{OP}_{super} - \overrightarrow{OO'}) \cdot R^{-1}_{matrix}$$

5.3 Integration of the Hybrid Location Model

So far we have discussed the hierarchical and coordinate aspects of the hybrid location model. We need to integrate these two aspects into a seamless system to realize the hybrid location model. This is achieved by adding geometric attributes to the nodes in the *space tree*. The resulting tree is called the *geometric space tree* (Figure 7).

The following geometric attributes are embedded into the space tree nodes:

- **Shape** indicates the geometric shape of the space, such as *cube, cylinder, cube*, etc.
- **Extension**, combined with *Shape* attribute, specifies the volume/area covered by the space.
- **Origin** is the origin point for the *space coordinate system* of the current space relative to the parent's coordinate system.
- **Rotation Matrix** which, as described in the previous section, specifies the directions of three axes of the *space coordinate system* of the current space relative to the parent's coordinate system.

The above four attributes are necessary for computing distance and spatial relationship. However the set of geometric attributes in the *geometric space tree* is extensible. For example, if you want to improve the performance, you can store some redundant information, such as the *centroid*, to accelerate the calculation of distance between spaces. You can also store additional information, such as an *owner* field to store the owner of the space, etc.

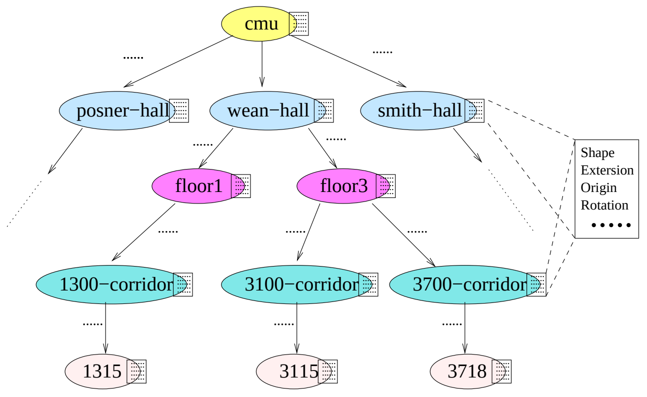

Fig. 7. Geometric space tree

5.4 ALI Operator Implementation

We briefly discuss how to realize the ALI operators of Section 4.4 using a *geometric space tree*. The basic step for the ALI operator implementation is *coordinate translation*, as described in Section 5.2. Theoretically, if we could turn all geometric attributes into a universal reference coordinate system, the implementation of every proposed operator in Section 4.4 becomes a trivial Euclidean Geometry problem. For brevity, we only describe the algorithm for the *distance* operator. Other operator implementations are similar:

distance(ali, ali) returns float *compute the distance between two locations*

1. if the two input locations are in the same *space coordinate system*, return the Euclidean distance between them directly, otherwise go to step (2)
2. find the common super space for the two locations, and translate the coordinates of both locations into new coordinates in the *space coordinate system* of the common super space.
3. return the Euclidean distance between the coordinates of the input locations in the shared *space coordinate system*.

6 A Space Service based on the Hybrid Location Model

In this section, we describe how the Aura space service is implemented using the hybrid location model and ALI representation.

6.1 Space Service for Aura

A *Space Service* provides information about the physical environment and about the spatial relationship between locations. Figure 8 illustrates how a context-aware application may use contextual information services [8] to locate the nearest color printer. It first asks the *People Location Service* for the current location

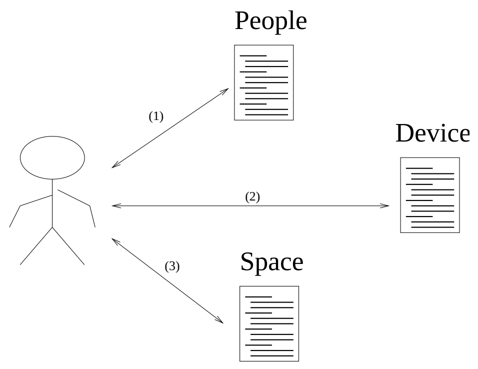

Fig. 8. A context-aware application uses the contextual information services to locate the nearest color printer.

of the user and it retrieves the locations and names of all available color printers from the *Device Service*. It then queries the *Space Service* for the distances between all the printers and the user. Finally the nearest printer is identified as the one with the smallest distance.

Besides **distance**, the *Space Service* also answers queries about **containment**, the **within** relationship between locations, the **super** and **sub** spaces of a location, etc.

6.2 Space Service Implementation

In order for the *Space Service* to handle queries of spatial relations between locations, it needs to (1) model the relevant physical environment, (2) use a common location representing scheme with clients and other services. These two issues are directly addressed by the previously introduced hybrid location model and the Aura Location Identifier.

The *Aura Space Service* adopts the hybrid location model to model the physical environment. The model can either be implemented by building and maintaining a custom data structure representing the *Geometric Space Tree*, or by using a database to store the *Geometric Space Tree*. We use the latter approach for simplicity and easier extensibility. Note that using a custom data structure for the *Geometric Space Tree* may offer some advantages, such as more effient access. Table 1 shows the definition of the relations in the PostgreSQL [13, 10] database for the *geometric space tree*. Most of the attributes were explained in Section 5.3. In our current implementation, we eliminated the *Shape* attribute and we assume all spaces are in the shape of a polygon box, similar to ALIs of type *Area* (e.g. Figure 2. This explains the *Height* geometric attribute in the relation. To support spatial areas, we introduced the new datatype polygon. The *ali* field is a character string representing the *space* ALI for the node. Super-sub spatial relationships are implicitly expressed in it. The *name* field is for the real

Table 1. Relation definition for the *geometric space tree* in PostgreSQL database

Geometric Attributes	Colomn Name	Column Type
NA	ali	varchar
NA	name	varchar
NA	type	enum
Extension	extent	polygon
Height	height	double
Centroid	cx, cy, cz	double
Origin	dx , dy, dz	double
Rotation Matrix	m00, m01, m02 m10, m11, m12 m20, m21, m22	double

name of the space. The *type* attribute describes the use of the space; it is an enumeration of *conference room, teaching room, office, corridor, building*, etc.

As shown in Table 1, an extended *geometric space tree* (with additional attributes as *name, type*) of hybrid location model is implemented by a relation in the database system. Based on this relation, all proposed ALI operators were implemented usnig simple database queries. The operators are rich enough to support the targeted context-applications such as those mentioned in Section 1. At the time of the writing of this paper, we are working on constructing some concrete context-aware applications with this space service and other contextual services (e.g. the Jane and Fred scenarios described in [6]). These applications will be used to evaluate both the choice of operators and performance of the Aura Space Service.

6.3 Populating the Space Service

The information for the Aura Space Service was obtained by extracting hierarchical and coordinate location data from floor plan files of subset of the buildings on the CMU campus. These files use AutoCAD's *".dwg"* format. The first major service (other than the space service) to use the ALI representation is the Aura "people-location" service. It tracks the location of users using the 802.11 wireless network interface on their PDA or laptop. The people-location service is a based on signal triangulation [11]. In the initial version of hybrid location model, we do not support location attributes such as orientation, velocity, and precision. We hope to add these in the next version of the hybrid location model since they are important for some context-aware applications.

Table 2. ALI functions and their operators

Functions	Operators
distance(ali,ali)	<->
within(ali,ali)	<<
contains(ali,ali)	>>
nearest(tab.col,tab.col)	##
isSuper(ali,ali)	=>
isSub(ali,ali)	<=

7 Embedding the Hybrid Location Model into Database System

In the previous section we described how the hybrid location model can be implemented using a database. However, while the service API was based on the the ALI representation, the database used only primitive datatypes. We will call this a *service-level* implementation of the model. In this section we explore an alternative implementation in which the ALI datatype is directly supported by the database itself. We will call this a *database-level* implementation.

7.1 Extension to PostgreSQL

In the previous section, the space service was implemented at the service level and the database did not directly support the ALI representation. As a result, queries about spatial relationships often have to be broken down into a sequence of service queries, thus making it impossible to fully exploit the power of the database system. For example, suppose an application wants to implement the query "*find all printers on the 3rd floor of Wean Hall*". With a service-level implementation of the space service, it would typically use a two-phase execution: (1) query the device service to get a list of all the printers with *name* and *location* attributes through "*select name, location from printer*", and (2) for each printer in the list, test whether it is on the 3rd floor of Wean Hall through a series of "*select containment from space where location1='xxx' and location2='ali://cmu/wean-hall/floor3'*" queries. If the *containment* attribute returns yes, then add the name of printer to the result list. In the database-level service implementation, the ALI representation is directly supported by the database, making it possible to have more complex queries handled by the database system itself. In the case of our example, with a database-level implementation, the query can be done in just one SQL query, "*select name from printer where within(location, 'ali://cmu/wean-hall/floor3')*".

We embed the hybrid location model into the open source object-relational database system PostgreSQL [10] by using its support of user-defined data types and operators in SQL queries [12]. This integration involved three steps:

- Create a table in PostgreSQL's system catalog to store the *geometric space tree* (Table 1).
  The table with environmental location data serves as meta-data to model the physical environment of interest. ALI related operators are implemented by traversing the hierarchical geometric tree in the table.
- Define ALI as a user-defined data type.
  The PostgreSQL database system allows users to create user-defined datatypes that can be used directly in SQL sentences. The creation of the ALI type is done using the SQL command "CREATE TYPE".
- Realize user-defined functions for ALI, i.e. **distance, contains, within, super** and **sub**, and assign user-defined operators for these functions (Table 2).
  The PostgreSQL database system also allows user to define user-defined functions and operators. This is done using the SQL commands "CREATE FUNCTION" and "CREATE OPERATOR".

Based on the above extensions to PostgreSQL, the database system can directly handle certain spatial queries in SQL sentences. Therefore, complicated contextual information queries can often be handled directly by the underlying database system. Here are two examples to show the power of having database support for spatial queries.

- "*SELECT p.name FROM printer p WHERE (p.location* <-> *'ali://cmu/wean-hall/floor3/3700-corridor')* < *10*"
  This is a range query to find all the printers within 10 meters of the 3700 corridor.
- "*SELECT u.name, p.name FROM (SELECT name, office AS ali FROM staff WHERE category='facility') AS u, (SELECT name, location AS ali FROM print WHERE status='error') AS p WHERE (u.ali* <-> *p.ali) = (SELECT (u.ali* ## *p.ali))* "
  This query tries to find the computing facility staff who is nearest to any of broken printers.

7.2 Benefits and Limitations of Contextual Spatial Databases

There are two major benefits to having database-level support for spatial queries: **performance** and **flexibility**.

Database-level support for spatial queries can improve performance in two ways: (1) the tight integration of data storage and service execution logic can reduce unnecessary transport overhead between them, and (2) most database systems have excellent built-in query optimizer, that can dramatically improve the performance of the sophisticated query processing.

In Figure 9, we show a performance comparison between database-level and service-level implementations of ALI. The two graphs plot the query execution time versus the queried table size. These experiments are all done on a PC with configuration shown in Table 3. The service-level ALI is implemented using Java

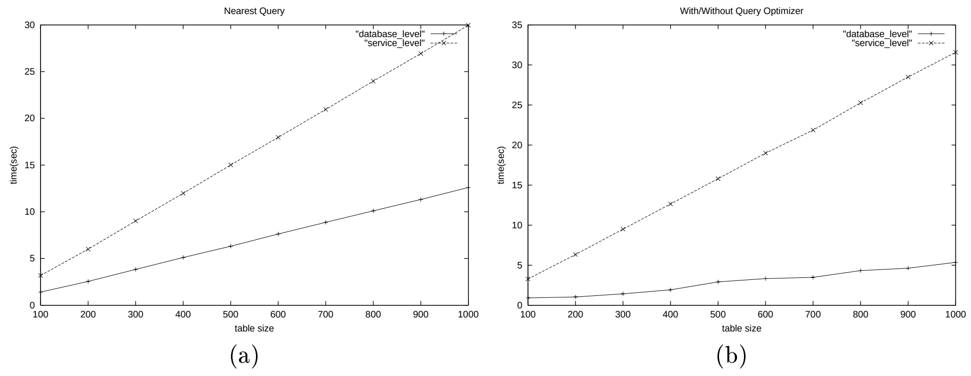

Fig. 9. Performance comparison between (a) *service-level* and (b) *database-level* implementations of ALI.

Table 3. PC Configuration for Performance Test

1 CPU	Pentium 4 1.50GHz
Memory	256M
Cache	256 KB
OS	Linux 2.4.17
PostgreSQL	7.2
JDK	J2SE 1.4.0-Beta3
Perl	v5.6.0

J2SE 1.4.0-Beta3. the database-level ALI was implemented on PostgreSQL 7.2 using Perl v5.6.0 as described above.

Figure 9(a) shows the execution time for the query "*find the nearest printer in a table to location A*" for our service-level and database-level implementations. We see that due to the tight integration of data storage and execution logic, the database-level implementation of ALI consumes proportionally less time than service-level implementation. The query for Figure 9(b) is to "*find the printer within 3rd floor of Wean Hall, whose job queue has less than 3 jobs*". Again, we see that the query optimizer in the database system helps to reduce the consumed time.

The second benefit of having database-level support for ALI is flexibility. The standardized query interface language SQL has been regularly extended from original SQL to SQL3 to accommodate more sophisticated queries. If we can express spatial queries directly on top of the database system, we will be able to take advantage of SQL to express more flexible queries, such as the second example query in Section 7.1.

Though the contextual spatial database offers some intriguing benefits for context-aware applications, there are some constraints limiting its universal applicability. For example, in the Aura Project's contextual information services, there are several services maintaining highly dynamic attributes, which cannot be directly stored in a database system. These dynamic attributes include "people's location", "available bandwidth", etc. This information has to be provided by custom services that cannot be integrated as described in this section (see [8] for more details).

8 Related Work

Context-aware applications or more narrowly location-aware applications depend on efficient location modeling and representation technologies. Many location models have emerged in different application domains. Most of these models are either dealing with very large scale spaces, e.g. geographic location modeling, or with very small scale, e.g. single room indoor location models. Context-aware applications in the ubiquitous and mobile computing field need something in between. Some earlier location models have directly addressed the needs context-aware applications. NEXUS [2, 7] and Semantic Spaces [4] are two good representatives.

NEXUS aims at providing a universal platform for all location-aware applications, which involves two major issues, modeling the world and representing locations. NEXUS introduces two XML-based languages, AWML and AWQL respectively, to address these issues. Distributed spatial model servers collaborate to provide a unified spatial view through some well-devised interfaces to applications. NEXUS is able to describe locations at different levels of precision, which is similar to ALI's hierarchical aspect. This model, to some extent, has a philosophy similar to ALI, since it provides a spatial query service based on a centralized data repository. However, NEXUS's solution is relatively more heavy-weighted than the ALI approach because the ALI representation is only concerned about a particular physical environment, such as a campus, while NEXUS worries about a global scale unified service platform.

Semantic Spaces from Microsoft Research is an example of a hierarchical location model, which decomposes the physical environment into a hierarchy of spaces. The locations of moving users or devices are correlated to actual physical spaces, thus it is capable of answering "containment" queries. However, because of its inherent lack of metric attributes and precision, it is unable to compute distance accurately, and represent location precisely, which are requirements for some ubiquitous computing applications.

9 Conclusion

The modeling of physical environment and representation of locations are key technologies for context-aware applications. This paper presents a hybrid location model using a computable location identifier to meet these needs. The hybrid model integrates the hierarchical and coordinate approaches for representing location. Our experience in building the Aura space service on top of the location model shows its viability for context-aware applications. Leveraging ALI's distinguishing feature of computability, spatial query support at the database level was realized by adding ALI as a user-defined datatype to PostgreSQL. Database support for spatial queries improves the efficiency and flexibility of complex user queries involving fairly static attributes.

10 Acknowledgement

This research was sponsored in part by the Defense Advanced Research Project Agency and monitored by AFRL/IFGA, Rome NY 13441-4505, under contract F30602-99-1-0518. Additional support was provided by Intel. We would like to thank Glenn Judd for providing the underlying Contextual Service Framework and for his many insightful suggestions and comments.

References

1. Bahl, P., and Padmanabhan, V. N. Radar: An in-building rf-based user location and tracking system. In *Proc. IEEE Infocom* (Tel Aviv, Israel, March 2000).
2. bauer, M., Becker, C., and Rothermel, K. Location models from the perspective of context-aware applications and mobile ad hoc networks. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
3. Universal Resource Identifiers (URI): Generic Syntax, August 1998.
4. Brumitt, B., and Shafer, S. Topological world modeling using semantic spaces. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
5. Domnitcheva, S. Location modeling: State of the art and challegnes. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
6. Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing 1*, 2 (April-June 2002), 22–31.
7. Hohl, F., U.Kubach, A.Leonhardi, Rothermel, K., and Schwehm, M. Nexus - an open global infrastructure for spatial-aware applications. In *Proceedings of MobiCom* (Seattle, USA, 1999).
8. Judd, G., and Steenkiste, P. Providing Contextual Information to Ubiquitous Computing Applications. Technical Report CMU-CS-02-154, Department of Computer Science, Carnegie Mellon University, July 2002.
9. O'Connell, T., Jensen, P., Dey, A., and Abowd, G. Location in the aware home. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
10. Postgresql. Software available from http://www.postgresql.org.
11. Smailagic, A., and Kogan, D. Location Sensing in a Context Aware Computing Environment. *IEEE Wireless Communications 9*, 3 (June 2002).
12. Stonebraker, M. Inclusion of new types in relational data base systems. In *Proceedings of the International Conference on Data Engineering,* (Los Angeles, CA, feb 1986), vol. IEEE Computer Society Order Number 655, IEEE Computer Society Press, pp. 262–269.
13. Stonebraker, M., and Kemnits, G. The postgres next-generation database management system. In *Communications of the ACM* (Oct 1991), vol. 34, pp. 78–92.