A Hybrid Location Model with Computable Location Identifier for Ubiquitous Computing

Changhao Jiang and Peter Steenkiste

Carnegie Mellon University, Computer Science Department
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 will present a novel location model combining the virtues of both the hierarchical and coordinate location models, and introduce a computable location identifier, namely *Aura Location Identifier* (ALI), for it. Then we will give *Aura Project*'s solution to constructing a space service on top of this model to handle spatial queries for context-aware applications. A simple example of such queries is a *range* query, "*select name from printer where distance(location,'ali://cmu/wean-hall/floor3/3100-corridor#(10,10,0)') <10*", where "location" is an attribute recording locations of interested printers. In the final part, the paper describes the extension of hybrid location model into the PostgreSQL database system for direct spatial SQL query support at the database level to improve performance and to achieve more flexibility for context-aware applications.

1 Introduction

Location information is critical contextual information for ubiquitous computing and mobile computing applications. It is a prerequisite ingredient to construct context-aware applications, such as diverting phone calls to the receiver nearest to a user, prefetching data to the service portal near the user's path, locating the closest resource, locating interested objects/people, navigating around a place, etc.

As more context-aware applications are initiated and explored, researchers realize that the **modeling** of the physical environment and the **representation** of locations are key enabling technologies.

The first issue is physical environment modeling. Numerous location models [10, 5, 6] 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 directly suitable.

The second issue is location representation. Most context-aware applications adopt a paradigm consisting of a distributed collaborating service framework, which stores location modeling data in a centralized data repository and sets up a dedicated location service (or space service) to handle location related queries issued by end-users or other services. This paradigm is preferable for its scalability and interoperability. However, we need an effective and efficient location representation scheme to make this work.

In this paper, we will present a novel hybrid location model that combines the virtues of both *hierarchical* and *coordinate* location models. It features a computable location identifer, namely *Aura Location Identifier*, for the purpose of representing locations and enabling the exchange of location information between distributed contextual services. The hybrid location model with computable location identifiers is intended to meet the needs of establishing a space service in Aura Project [1] for campus-wide context-aware applications at Carnegie Mellon University. Some concrete examples for targeted context-aware applications are: "*find the nearest color printer 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 paper is organized as follows. In section 2, we motive the hybrid location model by giving some background on the Aura Project and its space service. In section 3, we argue that context-aware applications need a hybrid location model. Section 4 proposes a computable location identifier, *Aura Location Identifer* (ALI), to represent locations. Section 5 elaborates on the underlying hybrid location model for ALI. 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 section 8 and 9, we give a brief overview of related work and conclude our paper.

2 Aura Project and Space Service

The Aura Project 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.

One of the main techniques used in Aura is to have the computing environment (including applications, networks, operating systems, and middleware services) automatically adapt to the user's context. For example, if the user experience 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 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.

In order to be able to adapt, the Aura system needs a lot of information about the user's context. For this purpose, we developed the Aura contextual information services [9] (see 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 [4] which provide critical information on where people are in physical space corresponds to the people-location relationship. 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. people-location relationship).

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

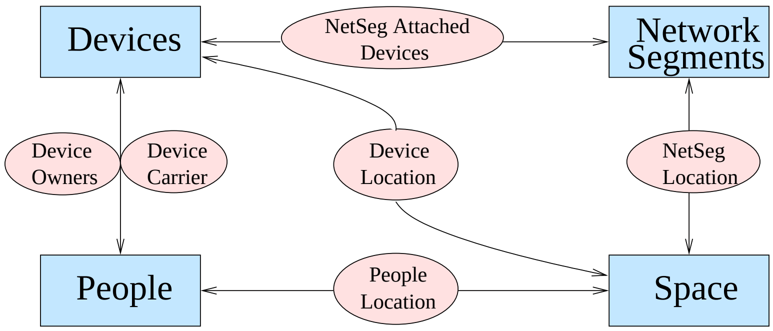


Fig. 1. A set of Aura contextual services for context-aware applications

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 the space service and more broadly for context-aware applications, is that they must model the physical environment through an appropriate location model. As introduced before, the prevailing location models fall into two groups: *hierarchical* and *coordinate* [7] models. In this section, we argue that neither of them is satisfactory for context-aware applications and a hybrid location model combining the benefits from both sides 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 virtually grid the physical environment with a superimposed reference coordinate system, thus locations can be uniquely and accurately represented by 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 supports algorithms for handling some location queries, such as "*find the people live in Forbes Ave.*". People whose address prefix is "*Forbes Ave*" satisfy this query. However, the biggest disadvantage of the hierarchical location model is the lack of position precision. Also it is often incapable or inefficient to compute distance. 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 considerable extra specification to enable deduction of 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 coordinate allows 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 intro the ALI representation.

4.1 Types of Location

Based on the interest 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 some space not physically demarcated, but virtually defined by applications, e.g. "*the area covered by a particular wireless access point*".
- **Point** location is 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 space name, while the two other ALI types append explicit geometric information to a hierarchical name. We present examples of the 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 [2]. 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"|
```

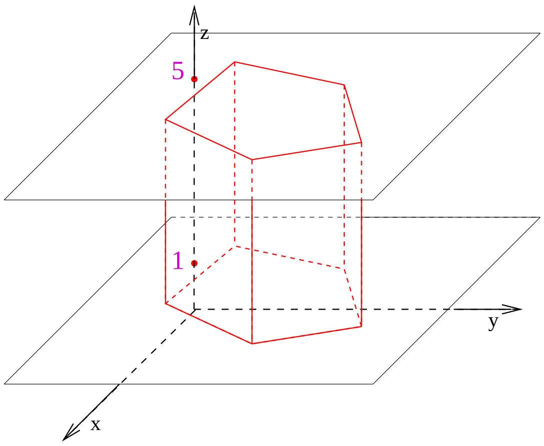


Fig. 2. "*ali://cmu/weanhall/floor3#{(1,0),(-1.5,0.5),(0,3),(2,3.5),(3,1.5)}-(1,5)}*" represents an area within the 3rd floor of Wean Hall.

```
   "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 the location of *room 3115 on 3rd floor of Wean Hall at CMU*. Its geometric attributes are stored in a centralized location data repository.
- **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 '#' token specify a closed polygon in planes parallel to X-Y plane of 3rd floor's *space coordinate system* (see section 5.2 for details). "(1,5)" specifies the height range of the area. All coordinates are given in the 3rd floor's *space coordinate system*. (See figure 2)
- **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 coordinate is in *room 3718's space coordinate system*.

4.4 Operators on ALI

An important characteristic 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 implement it at different levels of the system, such as the database level, service level, and 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 also may 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.
- **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 will describe the hybrid location model in more detail and we elaborate on how it can be used to realize the ALI operations.

5.1 Hierarchical Aspect of 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 interested 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*, in which each node corresponds to an actual space in the physical environment. The parent-child link in the tree implies super/sub space relationships between two spaces. With a constructed *space tree*, it is easy to tell whether the containment relationship exists 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 decompose the physical environment. The location service needs to maintain a tree style data structure for the *space tree* and handles queries of spatial relationship based on this data structure.

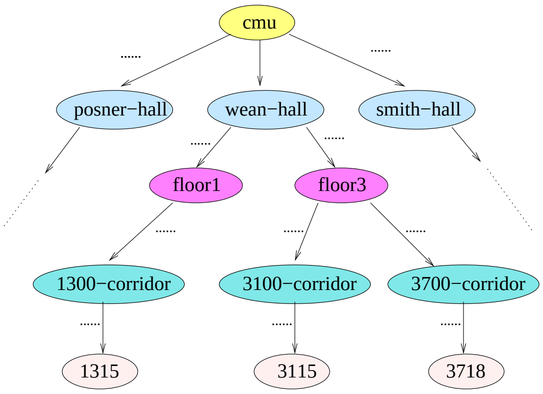


Fig. 3. Hierarchical Space Tree

Every *Space* type ALI described in section 4.3 (e.g. "*ali://cmu/wean-hall/floor3/3100-corridor/3115*") corresponds to a node in *space tree*. 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 Hybrid Location Model

In order to compute the geometric relations such as *distance, containment, and intersection* between locations, they need to possess geometric attributes in a well-defined coordinate system, such as *shapes, extensions, point coordinates*, etc. Most coordinate location models provide 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 give coordinate 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 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 expressed in coordinates of either the super space's coordinate system or the sub space's coordinate system. With graceful processing of coordination translation, the disparate coordinate system can interoperate.

In order to translate coordinate between coordinate systems, the hybrid location model defines the sub space's *space coordinate system* within the super space's *space coordinate system*. Thus coordinates can be translated between the

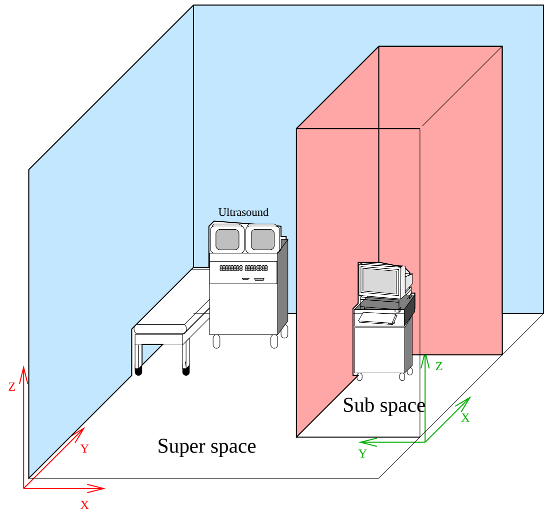


Fig. 4. Each space has its own *space coordinate system*, defined by specifying the *origin* point and three *axes* of "x", "y" and "z"

super space and sub space. To translate coordinates between spaces with no direct super/sub relation, we could find a unique path between them in *space tree* (see section 5.1) and translate along the path, since the composing links imply super/sub relations.

A sub space's *space coordinate system* is defined by specifying the **origin point** and **axes** within its 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 "x", "y", "z" axes of 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.

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

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

Once we define 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:

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

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

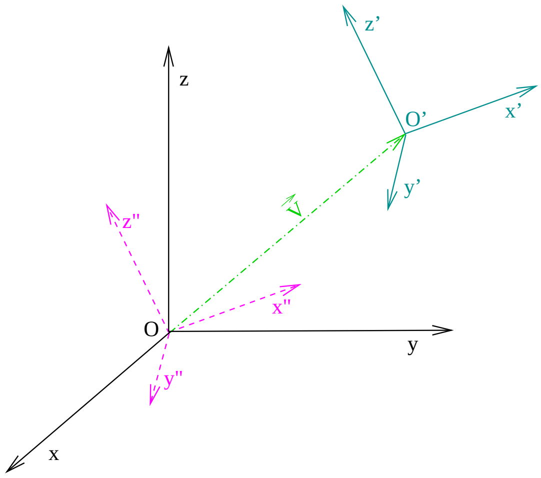


Fig. 5. Super Space Coordinate and Sub Coordinate System

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

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 through bundling nodes in the *space tree* with geometric attributes. The resulting tree is called *geometric space tree* (see Figure 7).

The geometric attributes embedded into the nodes include following:

- **Shape** indicates the geometric shape of the space, such as *cylinder, cube, sphere*, 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.
- **Rotation Matrix** which, as described in the previous section, specifies the directions of three axes of the *space coordinate system* of the current space.

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 calculation of distance between spaces. You can also store additional information, such as an *owner* field to store the owner of the space, etc.

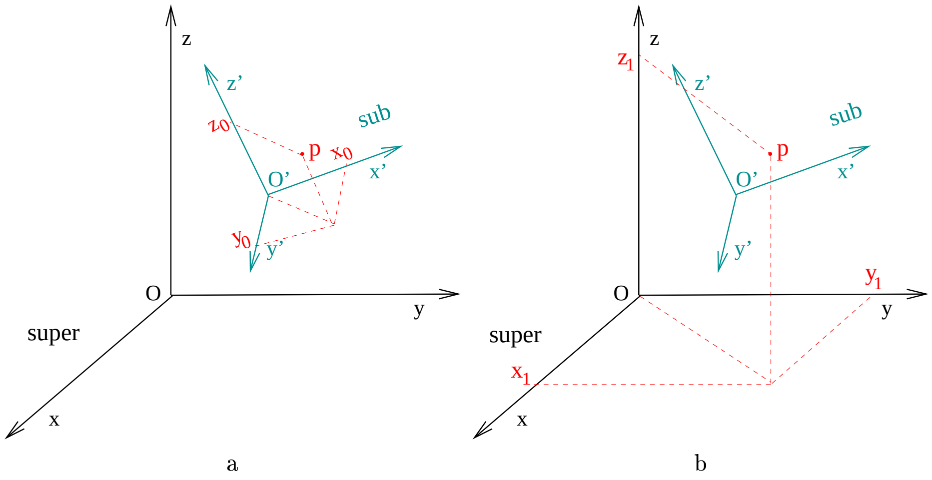


Fig. 6. Coordinate translation between super space and sub space

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*, (see Section 5.2 for details). Theoretically, if we could turn all geometric attributes into a universal reference coordinate system, implementation of every proposed operator in Section 4.4 become a trivial Euclidean Geometry problem. For brevity, we only give an algorithm description of the *distance* operator's implementation. Other operator implementations are similar.

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

1. if 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 these two locations, and translate the coordinates of both locations into new coordinates under the same *space coordinate system* of the common super space.
3. return the Euclidean distance between these two new coordinates in the shared *space coordinate system*

6 Space Service on Top of 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 of physical environment, and spatial relationships between locations. Figure 8 illustrates a typical scenario of services'

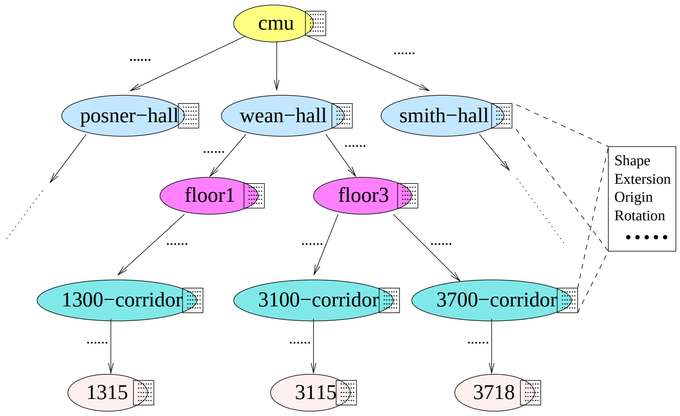


Fig. 7. Geometric space tree

interaction with user client, in which some context aware application wants to find the nearest color printer to a particular mobile user. It first asks the *People Location Service* for current location of that user, and retrieves all the locations and names of available color printers from the *Device Service*, then it queries the *Space Service* the distances between all the printers to that user. Finally the nearest printer is identified as the one with the smallest distance.

Besides **distance**, the *Space Service* also provides support to tell **containment**, **within** relationship between locations, to get the **super**, **sub** spaces of 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 commonly-agreed 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 *Space Service* adopts the hybrid location model to model the physical environment. It could either maintain a data structure representing a *Geometric Space Tree*, or just use a database to store the *Geometric Space Tree*. We used the latter approach for simplicity and easier extensibility. Note that realizing a custom data structure for the *Geometric Space Tree* may offer some advantages, such as better efficiency. Table 1 shows the definition of the relations in PostgreSQL for the *geometric space tree*. Most of the attributes were explained in Section 5.3 (In the current version of *Space Service*, we eliminates the *Shape* attribute, and assume all spaces are in the shape of a polygon box, just as locations represented by the *Area* ALI, so there is an additional geometric attribute of *Height* in the relation). The *ali* field is the character string of the *space* ALI

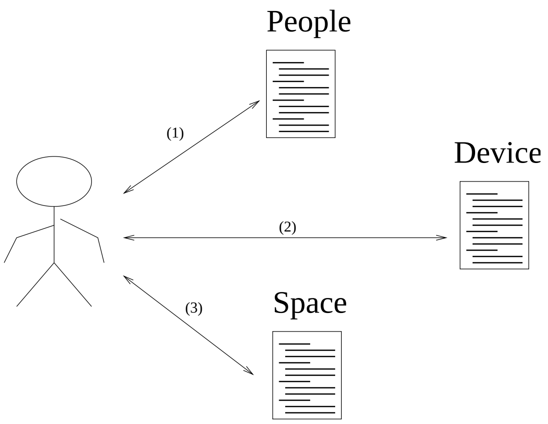


Fig. 8. A typical senario for contextual information services' interaction with application clients. User clients and services communicate through Aura Contextual Services Interface. [9]

representing the space. Super-sub spatial relationship is implicitly expressed in it. The *name* field is for the real name of the space. The *type* attribute is an enumeration of *conferenc room, teaching room, office, corridor, building*, etc.

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

Geometric Attributes	Colomn Name	Column Type
NA	ali	varchar
NA	name	varchar
NA	type	enum
Extension	extent	polygon[1]
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

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 as simple database queries. They are rich enough to support the targeted context-applications such as those listed in section 1.

7 Embedding the Hybrid Location Model into Database System

In the previous section we described how the hybrid location model can be supported using a database. Note however that will the service API was based on the the ALI representation, the database itself used only more 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

We embed the hybrid location model into the open source object-relational database system PostgreSQL[3] by using its support of user-defined data types and operators in SQL queries.

In the previous section, the space service was implemented at the service level (above database level). As a result, complicated spatial related queries often have to be broken down into a sequence of sub-queries which then have to be combined. In a database level service implementation, we try to do the opposite, i.e. multiple simple queries can be integrated into one complex query that is handled directly by the database system. For example, supposed we want to implement "*find all printers on the 3rd floor of Wean Hall*". With a service level implementation, we 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) test each printer in the list, to see 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. 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 integrated the hybrid location model into PostgreSQL in three steps:

- Create a table in PostgreSQL's system catalog, store *geometric space tree* (see table 1) in it.
  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.
  PostgreSQL database system allows user to create user defined data type to be used directly in SQL sentences. The creation of ALI is done by a 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.(see Table 2)

PostgreSQL database system also allows user to define user defined functions and operators. These are done by SQL commands, "CREATE FUNCTION" and "CREATE OPERATOR".

Table 2. Realized ALI functions and its appointed operator

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

Based on the above extension to PostgreSQL, the database system can directly handle some spatial query 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 database support of spatial queries.

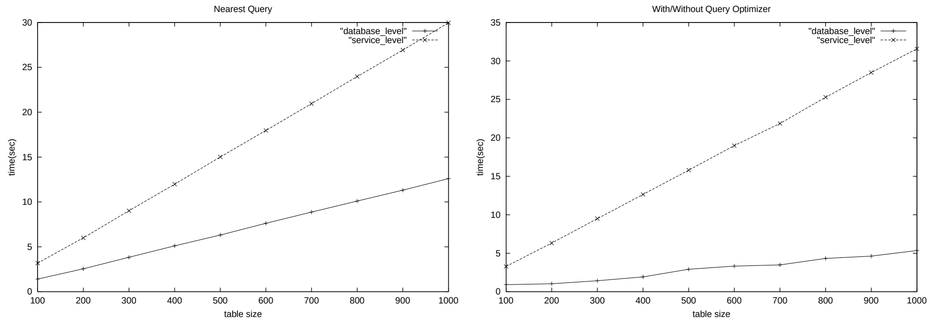
– "*SELECT p.name FROM printer p WHERE (p.location* `<->` *'ali://cmu/wean-hall/floor3/3700-corridor')* `<` *10*"
  This is a range query to find all the printers within 10 meters to 3700 corridor.
– "*SELECT u.name, p.name FROM (SELECT name, office AS ali FROM staff WHERE category='facility') AS u,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

7.2 Benefits and Limitations of Contextual Spatial Databases

There are two major gains from the database system support for spatial queries: **performance** and **flexibility**.

The first benefit of database level support is performance improvement. We identify two reasons for the performance gain from database support: (1) tight integration of data storage and service execution logic can reduce unnecessary transport overhead between them, and (2) most database systems have very excellent built-in query optimizer, which could dramatically improve the performance of the sophisticated query processing.

In Figure 9, we show performance comparison between database level and service level implementation of ALI. The two graphs plot query execution time



(a) (b)

Fig. 9. Performance comparison between *service level* and *database level* implementation of ALI. In above figures, two different queries are executed separately on two ALI implementations. Query for fig a) is "*find the nearest printer to location A*", query for fig b) is "*find the printer within 3rd floor of Wean Hall, whose job queue is less than 3 jobs*"

versus queried table size. These experiments are all done on one PC with configuration shown in table 3. Service level ALI is implemented using Java J2SE 1.4.0-Beta3. Database level ALI was implemented on PostgreSQL 7.2 using Perl v5.6.0. Figure 9 - (a) shows the performances of query, "*find the nearest printer in a table to location A*" for our service-level and database-level implementations. In this plot, we can see that due to the tight integration of data storage and execution logic, database level implementation of ALI consumes proportionally less time than service level implementation of ALI. The query for fig 9 - (b) is to "*find the printer within 3rd floor of Wean Hall, whose job queue has less than 3 jobs*". In this plot, we can see that the query optimizer in the database system helps to reduce the consumed time.

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

The second benefit of database level support of ALI is more 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 database system, we would also be able to take advantage of the SQL to express more flexible queries, such as the second example query in section 7.1.

Though contextual spatial database offers some intriguing benefits for context-aware applications, there are some constraints limiting its universal viability. 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 be provided by custom service that cannot be integrated as described in this section (see [9] for more details).

8 Related Work

Context-aware applications or more narrowly location-aware applications depend on efficient location modeling and representation technologies. There have been many location models emerging from different application domains. However, many of them are dealing with either too large scale as geographic location modeling or too small scale as one room indoors location model. What in need by many context-aware applications from ubiquitous computing or mobile computing field is somewhat between them. There are some location models presented from the perspective of context-aware applications. NEXUS [5][8] and Semantic Spaces [6] are good representatives.

NEXUS aims at providing a universal platform for all location-aware applications. The two major issues of modeling world and representing locations are solved by two XML based languages, AWML and AWQL respectively. 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 similar philosophy as ALI, since it provides a spatial query service based on a centralized data repository. However, NEXUS's solution is comparatively 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 global scale unified service platform.

Semantic Spaces from Microsoft Research belongs to hierarchical location model, which decomposes the interested 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 in many cases are demanded by 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 experience of building Aura Project's space service on top of the location model shows its viability for context-aware applications. On the basis of ALI's distinguishing feature of computability, spatial query support from database system level has been realized for improved efficiency and flexibility. However, these benefits are limited to some applications with fairly static attributes.

References

1. Carnegie Mellon University, Project Aura, http://www.cs.cmu.edu/~aura.
2. Berners Lee, et al, Universal Resource Identifiers (URI): Generic Syntax, RFC 2396, Auguest 1994.
3. PostgreSQL, http://www.postgresql.org.
4. 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).
5. 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).
6. Brumitt, B., and Shafer, S. Topological world modeling using semantic spaces. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
7. Domnitcheva, S. Location modeling: State of the art and challegnes. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).
8. 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).
9. Judd, G., and Steenkiste, P. Providing contextual information to ubiquitous computing applications, 2002. Submitted to Ubicomp 2002.
10. O'Connell, T., Jensen, P., Dey, A., and Abowd, G. Location in the aware home. In *Workshop on Location Modeling for Ubiquitous Computing* (2001).