ACCESSING LOCATION AND PROXIMITY INFORMATION IN A DECENTRALIZED ENVIRONMENT

Location Resolution Operations

Thomas Hadig
Stanford University, Stanford CA 94305, USA
Email: hadig@stanford.edu

Jörg Roth
University of Hagen, 58084 Hagen, Germany
Email: joerg.roth@fernuni-hagen.de

Keywords: Location-based services, service infrastructures, positioning

Abstract: Location-aware applications take into account a mobile user's current location and provide location-dependent output. Often, such applications still have to deal with raw location data and specific positioning systems such as GPS, which lead to inflexible designs. To support developers of location-aware applications, we designed the Nimbus framework, which hides specific details of positioning systems and provides uniform output containing physical as well as semantic information. In this paper, we focus on two important operations provided by the framework, described by two questions "Where am I?" and "What is in my proximity?" Our solution takes into account the requirements of clients in mobile environments. Our algorithms are based on a decentralized and self-organizing runtime infrastructure and are, thus, highly scalable and accessible for mobile users. We demonstrate the effectiveness of our approach by a number of simulations.

1 INTRODUCTION

Accessing information about the current location will be an important service in future mobile and ubiquitous application environments. Applications, which take into account the current location, are called *location-aware applications*; if we want to focus on networked services, often the term *location-based services* is used. Typical examples for such applications are:

- Find the nearest hotel, hospital, or gas station? How can I get there by bus or by car?
- My personal device (e.g. PDA, cell phone) should remind me to check my tires when I enter a gas station the next time.
- When I take a picture with my digital camera, the location should be stored in the picture's meta data.

To support developers of such applications we created the *Nimbus* framework. Nimbus provides a common interface to location data and abstracts the position capturing mechanisms. To achieve an optimal flexibility, it provides physical coordinates as well as semantic information about the current location. With Nimbus, mobile users can switch between satellite navigation systems such as GPS, positioning systems based on cell-phone infrastructures, or indoor positioning systems without affecting the location-based service. A developer can, thus, concentrate on the actual service function and does not have to deal with positioning sensors or capturing protocols.

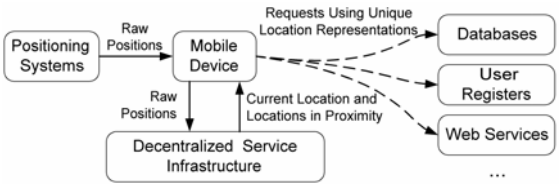

Figure 1: Data flow in the Nimbus framework

Fig. 1 shows the overall data flow in our framework. A mobile device gets raw location data from one or more positioning systems. Our framework transforms these data and produces unique location data using two basic operations, the *location resolu-*

tion and the *proximity resolution*. As these data have a unique format, it can easily be used as a search key to access databases, user registers, web services, etc. In this paper, we describe the Nimbus framework and focus on the resolution operations.

2 RELATED WORK

Many location-based applications and services have been developed over the last years. Tourist information systems are ideal examples for such applications. The systems CYBERGUIDE (Abowd et al., 1997) and GUIDE (Cheverst et al., 2000) offer information to tourists, taking into account their current location. Usually, such systems are bundled with a general development framework, which allows a developer to create other location-aware applications. A second example for location-based applications is context-aware messaging. Such systems trigger actions according to a specific location. ComMotion (Marmasse et al., 2000) is a system which links personal information to locations and generates events (e.g. sound or message boxes), when a user moves to a certain location. CybreMinder (Dey and Abowd, 2000) allows the user to define conditions under which a reminder will be generated (e.g. time is "9:00" and location is "office"). Conditions are stored in a database and linked to users. Whenever a condition is fulfilled, the system generates a message box.

Several frameworks deal with location data and provide a platform for location-based applications. Leonhardt (1998) describes a conceptual approach to handle multi-sensor input from different positioning systems. Cooltown (Kindberg et al., 2000) is a collection of location-aware applications, tools and development environments. As a sample application, the Cooltown museum offers a web page about a certain exhibit when a visitor is in front of it. The corresponding URLs are transported via infrared beacons. Nexus (Hohl et al., 1999) introduces so-called augmented areas to formalize location information. Augmented areas represent spatially limited areas, which may contain real as well as virtual objects, where the latter can only be modified through the Nexus system. OpenLS (Open GIS) is an upcoming project and provides a high-level framework to build location-based services.

Location-based services will become increasingly popular in the future. Especially mobile phone providers expect a huge market for such services (UMTS Forum, 2000). Typical applications respond to questions like "Where is the nearest hotel?" or "Which of my friends is in proximity?" Further examples are city guides or navigation systems. The first marketable service platforms come from the mobile phone providers. Services such as Nightguide or Loco Guide (Vodafone, 2003) serve as location-based information portals based on WAP technology. Such services reach a huge number of users, but they have very coarse-grained tracking capabilities still based on the GSM cell information.

Geographic information systems (GIS) and spatial databases provide powerful mechanisms to store and retrieve location data (Tomlin, 1990). Such systems primarily concentrate on accessing large amounts of spatial data. In our intended scenarios, however, we have to address issues such as connectivity across a network and mobility of clients, thus we have to use data distribution concepts, which are only rarely incorporated into existing GIS approaches.

3 THE NIMBUS FRAMEWORK

We designed the Nimbus framework to simplify the development of location-aware applications. Using this framework, developers can concentrate on the actual application function and can use location-dependent services of our platform. We distinguish three layers (fig. 2):

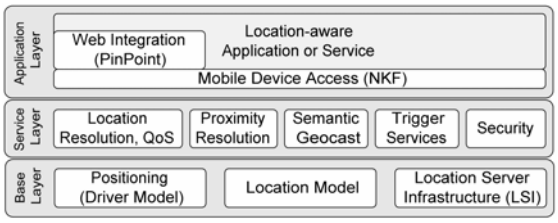

Figure 2: The Nimbus framework

The *base layer* provides basic services related to positioning systems. The framework can use arbitrary positioning systems, ranging from satellite positioning systems, positioning with cell phone networks to indoor positioning systems, based on, for example, infrared or ultrasound. To achieve the required flexibility, we attach the positioning system via a driver interface. This interface allows the framework to switch between positioning systems at runtime. The *location model* contains a formalism to describe locations and a set of rules to model the world. Finally, the *Location Server Infrastructure* (Roth, 2003a) stores the location data and provides services to access these data. It mainly consists of a federation of so-called *location servers*, each storing a piece of the entire location model.

The *service layer* provides higher-level location services such as the *location resolution* and the *proximity resolution* described in this paper. The application can specify requirements concerning

precision and costs using quality of service parameters (*QoS*). If more than one positioning system is accessible at a certain location, the framework selects an appropriate system according to the specified parameters. A further service of this layer is the *semantic geocast* (Roth, 2003b) which extends the original idea of physical geocasting. Trigger services inform the application when a certain location was reached. A set of security functions protect the users and the framework against attacks.

The *application layer* contains the actual location-aware application or service. A communication middleware called *Network Kernel Framework* (Roth, 2002a) was designed for small mobile devices such as PDAs or cell phones and offers communication primitives to access the servers. To develop location-aware Web applications we offer a high-level component called *PinPoint* (Roth, 2002b). As an example application, we developed a Web-based tourist guide with PinPoint.

3.1 The Nimbus Location Model

The Nimbus location model contains a formal specification of sets which describe locations, a set of rules that define the relations between these sets, and a set of operations that process location data. Even though we express the model independently of the later implementation, we strongly considered a decentralized storage. Especially the operations should be executed efficiently in a distributed federation of individual servers.

Semantic Locations: The concept of semantic locations heavily influenced our model, thus we start with a brief introduction of this concept. The notion of semantic locations is not new (Pradhan, 2000): besides *physical* locations such as GPS coordinates we can consider *semantic* locations such as "John's office at the university". Physical locations usually can be expressed by numbers, semantic locations by names.

Semantic locations are an ideal tool for a number of applications, sometimes in combination with physical locations. They have important advantages: first, they have a meaning to the user; in contrast, physical locations usually have no meaning at all to most people. Second, they can easily be used as a search key for traditional databases, tables or lists without the need of spatial databases.

In this section, we want to relate semantic locations to physical locations. Let P denote the set of all physical locations. We call each coherent area $S \subseteq P$ a *semantic location* of P. We further call each set $C \subseteq 2^P$ of semantic locations, a *semantic coordinate system* of P. (2^P denotes the power set of P.) Note that we do not assume two semantic locations to be generally disjoint. A reasonable semantic coordinate system C contains semantic locations S with certain meanings, e.g. countries, states, cities, districts, streets, places, mountains, rivers, lakes, and forests.

We further introduce a *name* for a semantic location. Let N be the set of all possible names. We define a function $NAME$: $C \rightarrow N$, which maps a semantic location to a string. We require names to be unique, i.e. $NAME(c_1) \neq NAME(c_2)$ for $c_1 \neq c_2$. We call a semantic location with its corresponding name a *domain*. For a domain d, $d.name$ denotes the domain name, $d.c$ the semantic location.

In principle, a semantic coordinate system C could be an arbitrary subset of 2^P that contains coherent areas. Looking at real-world scenarios, however, we usually find hierarchical structures, e.g., a room is inside a building, a building is in a city, a city is in a country, etc. Thus, we divide C into *hierarchies*. A hierarchy contains domains with a similar meaning, e.g., domains of cities or domains of geographical items. Each hierarchy has a *root domain* and a number of *subdomains*; each of them can in turn be divided into subdomains. We call a top node of a subhierarchy a *master* of the corresponding subdomains. We denote $m \triangleright s$ for master m of subdomain s. Further $\succ$ denotes the reflexive and transitive closure of $\triangleright$, i.e. $d_1 \succ d_2$ if either $d_1 = d_2$ or d_1 is a top node of a subtree which contains d_2.

We call a link between a subdomain and its master a *relation*. Relations carry information about containment of domains. Hierarchies are built according to three rules:

- The area of a subdomain has to be completely inside the area of its master, i.e. if $d_1 \triangleright d_2$ then $d_2.c \subset d_1.c$.
- The name of a subdomain d_2 extends the name of its master d_1 according to the rule $d_2.name$=<extension> + '.' + $d_1.name$, where <extension> can be an arbitrary string containing letters, digits and some special characters. With the help of this rule, we can effectively check if $d_1 \succ d_2$ or $d_1 \triangleright d_2$ with the help of the names.
- Root domain names of two hierarchies must be different.

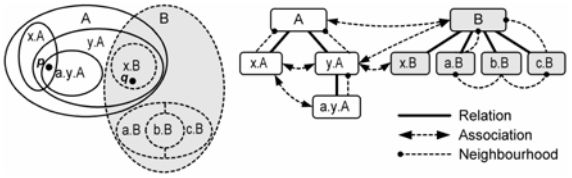

Figure 3: Sample hierarchies

Fig. 3 shows an example with two hierarchies. In addition to relations, we use two more links: *association* and *neighbourhood*.

Associations: In principle, the model is now sufficiently expressive to specify realistic sets of se-

mantic locations and their relationship among each other. One important question could be: "Given a physical location *p*, which semantic locations contain *p*?" E.g., in fig. 3 point *p* resides in the domains A, x.A, y.A, and a.y.A. As a master fully encloses a subdomain, the results A and y.A do not carry useful information. A useful answer would be x.A and a.y.A.

This so-called *location resolution* could be performed by browsing through all hierarchies from the root down to the smallest domains covering *p*. This, however, would cause a large number of requests and in a real infrastructure a considerable amount of network traffic. Therefore, we introduce a second relationship between domains, the *association*:

Two domains d_1, d_2 are associated, denoted $d_1 \sim d_2$, if they share an area, i.e. $d_1.c \cap d_2.c \neq \{\}$ (*condition 1*) and neither $d_1 \succ d_2$ nor $d_2 \succ d_1$ (*condition 2*). Condition 2 prevents from superfluously linking masters to their subdomains as they always share an area. Associated domains can be in different hierarchies or in the same hierarchy (see fig. 3). Using associations, we only need one domain d_0 that contains the position *p*. All domains $d \sim d_0$ are candidates to additionally contain *p*. In turn, no more domains have to be checked, and thus we can avoid the time-consuming search through all hierarchies.

We can reduce the number of candidates even more because we are only interested in the most specific domains. If, in the example above, we want to know which domains contain the point *q*, we are only interested in the domains y.A and x.B, and not in A or B. Taking this into account, we can modify condition 1 as follows: associations only link two domains, if the shared area is not fully covered by their respective subdomains. This leads to the short definition $d_1 \sim d_2$ iff $\Delta(d_1) \cap \Delta(d_2) \neq \{\}$, where $\Delta(d)$ denotes the area of *d* without its subdomains' area.

In fig. 3, the shared area of A and x.B is fully covered by the domain y.A, thus A and x.B are not associated as this link would not carry additional information. Starting at x.B we only have to check y.A. Note we cannot always reduce the number of queries. E.g. starting at y.A we have to check x.B *and* B as there is an area of $y.A \cap B$ outside of x.B.

Neighbourhood: In order to find locations in the proximity of a mobile node, an additional type of relationship is needed. If a mobile node leaves a domain d_1, its semantic location changes to a different domain d_2. Then, d_2 is called a *neighbour* of d_1. This means that either the areas of the two domains overlap but d_1 does not contain d_2 or the areas have a section of the boundary in common. Neighbourhood links connect a domain and a subset of its neighbours. The link is unidirectional. In the figures, we indicate the direction of links by a dot: if d_1 refers to a neighbour d_2, the dot is shown at the line termination of d_2.

Two domains are either related or associated, if they overlap. In case of an association, no neighbourhood link is necessary as they would carry the same information. If the domains are related, the subdomain is located inside the area of the master domain; then, the subdomain can have a neighbourhood link to the master domain as the master covers the entire border of the subdomain.

If several neighbours cover an overlapping part of the domain border, it is sufficient to link only one neighbour as other neighbours can be found by following the association and relation links of the neighbour. For the example in fig. 3, the domain a.B has the neighbours B, b.B and c.B. As B covers the complete boundary, only this neighbourhood link is necessary. In the case of b.B, a neighbourhood link to a.B and c.B is necessary.

In the Nimbus framework, we implemented an algorithm that searches the neighbours automatically. During the start-up phase of a location server, it searches for associated domains. After that, the server checks for uncovered sections of the border and actively searches domains that cover these sections. In addition, servers becoming unavailable due to, e.g., network problems are automatically recognized and replaced by other neighbours.

Of course, there are more links conceivable between domains. We could, e.g. link two domains, if they are connected by a street or a subway line. We can store such links as meta data in a domain record, but they do not have any influence on the infrastructure. For the operations described later, relations, associations and neighbourhood links are sufficient.

3.2 The Runtime Infrastructure

Fig. 4 shows the distributed infrastructure which consists of three *segments*:

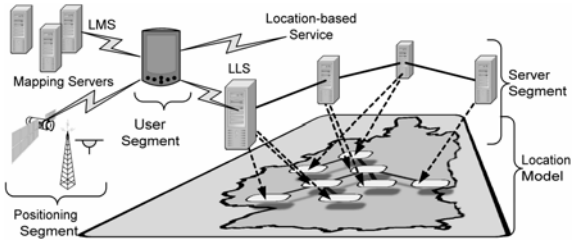

Figure 4: The infrastructure

The *positioning segment* contains the positioning systems, e.g., indoor positioning systems, satellite navigation systems or systems based on cell phone networks. The runtime system accesses the positioning systems through *position drivers* which al-

low the change of positioning systems even at runtime. As many positioning systems provide local positioning data, we may need the help of *mapping servers* to transform local locations to global ones. Each mapping server is responsible for a specific positioning system, e.g., a mapping server inside a building may be responsible for the indoor positioning inside this building. A lookup procedure allows the mobile client to find the appropriate mapping server for a specific location, called the *local mapping server* (*LMS*).

The *user segment* contains the mobile nodes with a runtime system and the mobile part of the location-based service. We developed a lightweight runtime system for the mobile nodes. We shift heavy duty tasks to the servers, thus the computational power of PDAs or mobile phones is sufficient.

The *server segment* contains the location servers that store the domain data. Each location server is responsible for a specific domain and all subdomains, for which no other location server exists. When a mobile node moves to a specific location, it automatically looks up an appropriate location server for the new domain, called the *local location server* (*LLS*). The LLS is the representative of the infrastructure for a mobile node. As mobile users are distributed among different location servers, this infrastructure is highly scalable. It can be observed that our system does not overload top-level servers.

We use a lightweight toolkit to process polygonal data (Vivid Solutions, 2003). The toolkit handles all geometric operations in the runtime memory and can quickly check, if a point is inside or outside a polygon. We store domain information using XML files in which the most important entry is the polygon specifying the area *d.c*. We can conveniently edit these XML files with the help of a graphical domain editor.

The entire system is self-organizing. A discovery procedure presented in (Roth, 2003a) connects the server to its domain master and looks up associated servers and neighbours.

3.3 Resolution Operations

One goal of our approach is to provide uniform location information, which is independent from the actual positioning system. For each position, we want to provide both physical as well as semantic locations, even though typical positioning systems only offer one type. GPS e.g. offers physical locations, whereas some indoor positioning systems directly produce semantic location output. Having both types, the application can choose the appropriate type (or even both types) for the specific operating condition. We distinguish two resolution operations:

- *Location resolution*: Given a physical location *p*. What domains d_i contain *p*? (*semantic resolution*). And in turn: Given a semantic location by its name *n*. What is the physical extension *d.c* of the domain *d* with this name? (*physical resolution*)
- *Proximity resolution*: Given a physical location *p*. What domains d_i are inside a certain circle around *p*.

The first kind of resolution is an operation with two directions, both concerning the mobile user's current location. We could either ask for the physical or semantic location, depending on the location data provided by the positioning system.

The physical resolution is simple, as we only have to look up the appropriate domain and return *d.c*. The more complex operation is the semantic resolution, as multiple hierarchies and domains may be involved. The algorithm can be outlined as follows:

Look up an arbitrary domain d_0 *with* $p \in \Delta(d_0)$
names $\leftarrow \{d_0.name\}$
for all $d \sim d_0$ *do*
  if $p \in \Delta(d)$
    names $\leftarrow$ *names* $\cup \{d.name\}$
return names

If we have an arbitrary domain which fulfils the first condition, we efficiently can loop through the associated domains.

3.4 Proximity

The algorithm described above allows the user to find information about his/her current physical and semantic location. However, those do not provide information about the proximity of the current location. Questions like "Where are the closest restaurants?" cannot be answered. Therefore, an additional proximity algorithm has been defined.

The basic idea of this algorithm is to search all local domains and their neighbours in order to find semantic domains that answer the question. The algorithm is required to perform this search automatically and efficiently. The following definitions are used:

A *match* is a semantic domain that answers the question. It is assumed that the test can be performed by a filter using the semantic name of the domain. This requires that all questions are given in form of a regular expression that can be compared to the domain name. Assuming a hierarchical structure of

domains describing restaurants with the master domain restaurant.com, the question mentioned before can be written as "Where are the closest domains matching *.restaurant.com ?".

The *distance* of a semantic domain to the current position of the mobile node is defined as the smallest physical distance between the current position and any point inside the area of the domain. We are aware of other notions of distance. E.g. the distance to a domain using a train may significantly differ from our distance. At this point however, we assume that distances can be computed by simply looking at the domain area *d.c*. Other distances may use meta data stored inside domain records.

One assumption is that the complete area is covered with domains, i.e., there is no location that would not have a local location server. In addition, it is also assumed that the questions include a search limitation on the number of matches and/or on the maximal search distance in order to prevent denial-of-service attacks and erroneous requests from causing a high load on the servers.

The algorithm uses the following strategy:

1. Set the list of domains to search (*D*) to contain only the local semantic domain.
2. Beginning of a loop over all domains that have not yet been searched. Iterate the following points using the domain with the smallest distance as current domain.
3. Query the location server of the current domain for all its neighbours, subdomains, and associated domains.
4. Remove those domains that have been searched already from the returned list.
5. Add all remaining domains to the list *D* of domains to search.
6. Apply the filter to the remaining domains. Those domains that fulfil the requirement of the filter are matches. In case of a match, the master domains of the domain are also tested by the filter.
7. Determine the distance *d* of the closest domain in *D*.
8. If the required number of matches with a distance smaller than *d* has been found, the algorithm can terminate successfully.
9. If the distance *d* is larger than the search distance limit, the algorithm can terminate with an error message.
10. Start the next iteration of the loop in step 2.

Step 3 requires access to other location servers across a network. Thus, this step needs a considerable amount of time compared to other steps. The algorithm ensures that network queries are reduced to a minimum (see section 3.6).

3.5 Proof of Correctness

In the following, a proof of correctness is summarized. The complete formalism and proof cannot be shown due to space considerations but is available from the authors.

A proof has to show that the algorithm terminates and produces the correct result. As the matches are found in an iterative procedure, it has to be shown that all domains are checked during the iterations and that the matches are found such that the condition in step 8 returns the correct result. These three items will be demonstrated below.

Termination: The algorithm will iterate over the local domains until either the condition in step 8 or step 9 is fulfilled. Especially, it is not possible that the list of domains to search runs empty as each point is covered by a local location server and, therefore, each domain can be reached from every other domain by a path of neighbours, associated domains, or subdomains.

In order to show the termination of the algorithm, it is sufficient to demonstrate that one of the conditions is reached in every case. This is true for step 9. As the number of domains per unit of area is finite, at some point, all domains within the search radius have been reached and the next domain to search is outside this limit.

Finding all domains: As described in the previous paragraph, there is a path between every pair of domains. It can be demonstrated by induction on the distance from the local domain that the algorithm follows the possible paths in a breadth first search. Thus, all domains are searched in the order of their distance from the local domain.

Correctness of the results: As shown above, all domains are searched. Therefore, all matches are found. As the order of searching is done by distance from the current domain, all matches with a distance smaller than the distance of the current domain to the local domain have been found. Therefore, the condition in step 8 ensures that no other matches closer than those returned exist.

3.6 Scalability and Simulation

It is important to understand the behaviour of the algorithm in the case of a large number of domains or a large number of requests.

In order to estimate the latter, the number of queries to remote location servers for each request is studied. It is assumed that the number of semantic domains covering each point is approximately constant as is the average size of those domains, i.e., the number of domains per unit area f is a constant. Then, for a search which ends at a distance r, the

algorithm has to search $\pi \cdot f \cdot r^2$ domains. This number of queries is minimal for a decentralized environment. The size of the queries has been optimized such that all necessary information is transported with as little overhead as possible. Assuming an average length of the semantic domain name of 50 characters, the size of one query will be approximately 58 bytes for the request and for the answer 10 bytes plus 86 bytes for each neighbour returned. Please note that this does not take into account the overhead due to the TCP/IP protocol.

Similarly, the memory usage in the mobile node has been optimized. An entry in the list of matches or domains to search is approximately 66 bytes. The number of entries in the list can be estimated using the same arguments that were used for the bandwidth. At a specific search distance *r*, all domains with a distance smaller than *r* have been searched. Therefore, those are no longer in the list of domains to search. Searching those domains has added new domains to the list that are directly neighbouring the searched domains, i.e., they have a distance that is larger than the current search distance by up to the size of the domains searched. Assuming an average domain size of Δr, the list contains approximately $\pi \cdot f \cdot ((r+\Delta r)^2 - r^2) = \pi \cdot f \cdot (2\Delta r \cdot r + \Delta r^2)$ domains.

A simulation of the behaviour in a realistic environment with several differently sized domains in several hierarchies has been performed as a large scale test. The location server definitions have been extracted from the TIGER/Line (U.S. Census Bureau, 2000) dataset from the U.S. Census Bureau. This dataset provides area definitions for semantic locations, such as political borders (state, county, etc.), school districts, shopping centre, parks, rivers, etc. From this data set, a subset of about 1300 locations in the San Francisco Bay Area has been used to create configuration files for the location servers.

Table 1 gives an overview of the variety and amount of configuration files created. As the creation has been automated, the configuration files for all of the United States can be created which would include 3232 counties and over 50,000 subdivisions.

In several simulation runs, a selection of the location servers has been used to validate the algorithm and determine the number of queries and the size of the lists in the mobile node. For our simulations we used five Linux computers inside a LAN with up to 1.2 GHz CPUs and up to 512 MB RAM, each of which stored a huge number of domains. This however conflicts with our original idea of decentralized data storage. On the one hand, we could not measure long distance network transactions, and on the other hand we overloaded single servers. Especially the high memory load caused by the number of server processes significantly reduced the performance. Due to the limited size of the computing power available for the simulation, no final conclusion can be drawn on the real performance in a completely decentralized environment. With the simulations, however, we verified the algorithms and its implementations. The results further confirm the estimates given above for the bandwidth and memory requirements.

3.7 Extensions

Several extensions of the algorithm are planed. On the side of the user interaction, the algorithm can be setup such that matches are returned to the user immediately. This would allow the users to stop a search once a successful match is found or to continue beyond the original search limit if no matching domain is reached.

The use of call-back filters will allow the use of information other than the semantic domain name for describing the filter.

Bauer et al. (2002) proposed additional symbolic links to express topological aspects or to express proximity, which may be different from geometric distance. Based on this idea, adding neighbourhood links according to, for example, bus routes connecting different domains could lead to a distance measures such as travel time. However, those links can no longer be established by the algorithm but have to be added to the location server configuration files.

Table 1: Overview of the type and number of the domain definitions extracted from the TIGER/Line files

Semantic Name	Number & Descript.
county.ca.us	4 Counties in California
name.*county*.ca.us	14 Subdivisions of counties
name.place.ca.us	67 Place (e.g. city)
name.amusement-center.com	6 Amusement centres
name.apartment.com	28 Apartment complexes
name.shopping-center.com	277 Shopping centres
name.airport.transport.com	7 Airports
name.train-station.transport.com	1 Train station
name.winery.com	4 Wineries
name.library.com	90 Libraries
name.elementary.district.school.edu	46 Elementary school district
name.secondary.district.school.edu	9 Secondary school district
name.unified.district.school.edu	15 Unified school district
name.individual.school.edu	174 Schools
name.golf-course.geo	22 Golf courses
name.island.geo	8 Islands
name.lake.geo	178 Lakes
name.river.geo	28 Rivers
name.park.geo	271 Parks
name.center.gov	4 Government service centres
name.medical.org	6 Hospitals
name.institutions.religious.org	6 Churches
name.cemetery.religious.org	33 Cemeteries

4 CONCLUSION AND FUTURE WORK

In this article we introduced resolution operations which provide unique location data independent of the underlying positioning systems. We took into account the distributed storage of location data in a decentralized federation of location servers.

Developers of location-based services and applications can use the Nimbus framework as a platform and do not have to deal with position capturing and resolution. As the corresponding infrastructure is self-organizing and decentralized, it is highly accessible and scalable.

The framework provides methods to find all semantic domains at the current position. In addition, proximity searches of the type "Where is the closest restaurant?" are also supported. The data for those searches is inferred from the semantic domain name. This allows the Nimbus framework to answer those queries without a central database or complicated configurations on the server side. The algorithm is open and easily extendable to more complicated queries.

Finally, the whole system has been implemented and tested using several hundred location server configurations. A tool has been developed that allows the automated creation of the XML configuration files using the TIGER/Line dataset published by the U.S. Census Board for the United States. Tests established the scalability and correctness of the algorithms.

REFERENCES

Abowd, G. D.; Atkeson, C. G.; Hong, J.; Long, S.; Kooper, R.; Pinkerton, M., 1997. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3: 421-433

Bauer, M.; Becker, C.; Rothermel, K., 2002. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks, *Personal and Ubiquitous Computing*, Vol. 6, No. 5, Dec. 2002, 322-328

Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.; Efstratiou, C., 2000. Developing a Context-aware Electronic Tourist Guide, *CHI'00*, ACM Press

Dey, A., K.; Abowd, G., D., 2000. CybreMinder: A Context-aware System for Supporting Reminders, *Second International Symposion on Handheld and Ubiquitous Computing 2000*, Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer-Verlag, 187-199

Hohl, F; Kubach, U.; Leonhardi, A.; Schwehm, M.; Rothermel, K., 1999. Nexus - an open global infrastructure for spatial-aware applications. *5th Intern. Conference on Mobile Computing and Networking* (MobiCom '99), Seattle, WA, USA, 1999. ACM Press

Kindberg, T.; Barton, J.; Morgan, J.; Becker G.; Caswell, D.; Debaty, P.; Gopal, G.; Frid, M.; Krishnan, V.; Morris, H.; Schettino, J.; Serra, B.; Spasojevic, M., 2000. People, Places, Things: Web Presence for the Real World, *3rd Annual Wireless and Mobile Computer Systems and Applications*, Monterey, USA, Dec. 2000

Leonhardt, U., 1998. Supporting Location-Awareness in Open Distributed Systems, PhD Thesis, University of London

Marmasse, N.; Schmandt, C., 2000. Location-aware Information Delivery with ComMotion, *Second International Symposion on Handheld and Ubiquitous Computing 2000*, Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer, 157-171

Open GIS Consortium, OpenLS Home Page, www.openls.org

Pradhan, S., 2000. Semantic Locations, *Personal Technologies*, Vol. 4, No. 4, 2000, 213-216

Roth, J., 2002a. A Communication Middleware for Mobile and Ad-hoc Scenarios, *Int. Conf. on Internet Computing* (IC'02), June 24-27, 2002, Las Vegas, Vol. I, CSREA press, 77-84

Roth, J., 2002b. Context-aware Web Applications Using the PinPoint Infrastructure, *IADIS Intern. Conference WWW/Internet 2002*, Lisbon, Portugal, Nov. 13-15, 2002, IADIS press, 3-10

Roth, J., 2003a. Flexible Positioning for Location-Based Services, *IADIS Intern. Conf. e-Society 2003*, Lisbon, Portugal, June 3-6, 2003, IADIS Press, 296-304

Roth, J., 2003b. Semantic Geocast Using a Self-organizing Infrastructure, *Innovative Internet Community Systems* (I2CS), Leipzig, Germany, June 19-21, 2003, Springer-Verlag, LNCS 2877, 216-228

U.S. Census Bureau, 2000. 108th Congressional Districts Census, 2000, TIGER/Line Files, http://www.census.gov/geo/www/tiger/index. html

Tomlin, C., D., 1990. Geographic Information Systems and Cartographic Modelling, Prentice Hall

UMTS Forum, 2000. Enabling UMTS/Third Generation Services and Applications, Report 11, http://www.umts-forum.org, Oct. 2000

Vivid Solutions, 2003. JTS Technical Specifications, http://www.vividsolutions.com, March 31, 2003

Vodafone Homepage, 2003. www.vodafone.com