Accessing Location Data in Mobile Environments – The Nimbus Location Model

Jörg Roth

University of Hagen
Department for Computer Science
58084 Hagen, Germany
Joerg.Roth@Fernuni-hagen.de

Abstract. Location-based applications and services are getting increasingly important for mobile users. They take into account a mobile user's current location and provide a location-dependent output. Often, location-based 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-based services, 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 the location model, which takes into account the requirements of clients in mobile environments. A domain model contains logical links and allows the expression of semantic relations between locations. A decentralized and self-organizing runtime infrastructure offers operations to resolve the current location efficiently.

1 Introduction

Applications or services which take into account the current location will become increasingly popular in the future. Especially mobile phone providers expect a huge market for such services [23]. Typical applications answer questions like "Where is the nearest hotel?" or "Who of my friends is in proximity?". Further examples are city guides or navigation systems. Currently, the development of such services is cost-intensive due to the heterogeneity of positioning techniques, positioning systems and location data.

To support developers of location-based services we created the *Nimbus* framework. Nimbus provides a common interface to location data and hides 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 has not to deal with positioning sensors or capturing protocols.

In this paper we present the Nimbus location model. After discussing related work we introduce the formal model. We strongly believe that a model has to consider the usage in a real scenario, thus our model supports the efficient access to location data in a network environment. In addition, Nimbus efficiently supports three-dimensional

F. Crestani et al. (Eds.): Mobile and Ubiquitous Info. Access Ws 2003, LNCS 2954, pp. 256–270, 2004.

locations with the help of a 2.5D approach. We conclude with the presentation of the underlying server infrastructure and discuss open issues.

2 Related Work

Many location-based applications and services have been developed in the last years. Tourist information systems are ideal examples for such applications. The systems CYBERGUIDE [1] and GUIDE [5] offer information to tourists, taking into account their current location. Usually, such systems come along 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 [21]. ComMotion [12] 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 [6] 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 application. In [11] Leonardt describes a conceptual approach to handle multi-sensor input from different positing systems. Cooltown [9] 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 [8] 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 [15] is an upcoming project and provides a high-level framework to build location-based services.

The first marketable service platforms come from the mobile phone providers. Services such as *Nightguide* or *Loco Guide* [25] serve as location-based information portals based on WAP technology. Such services reach a huge number of users, but they suffer from an insufficient location mechanism still based on the GSM cell information.

Geographic information systems (GIS) and spatial databases provide powerful mechanisms to store and retrieve location data [22]. 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

Many existing frameworks either rely on a specific positioning system such as GPS or only provide a very high-level concept to integrate other positioning systems. 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. 1):

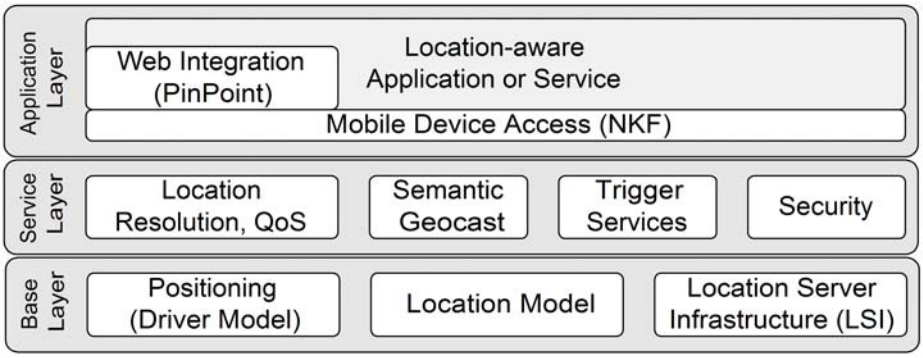

Fig. 1. The Nimbus framework

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 e.g. 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 actual focus of this paper, the *location model*, contains a formalism to describe locations and a set of rules to model the world. Finally, the *Location Server Infrastructure* (*LSI*) [19] 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 second layer, the *service layer,* provides higher-level location services. The most important service is called *location resolution*: an application uses this service to ask for the current location. In contrast to positioning systems, the location provided by this component contains globally unique physical as well as semantic locations. 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. An important service of this layer is the *semantic geocast* [20] which extends the original idea of geocasting by Imielinski and Navas [13]. 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* (*NKF*) [17] was especially 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* [18]. The World Wide Web is a powerful platform to develop location-based services, but currently makes

no use of the client's current position. PinPoint integrates location information into the HTTP data stream and still allows the usage of existing components such as Web browsers and Web servers without modifications. 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 efficiently be executed in a distributed federation of individual servers.

3.1.1 Structuring the Space with Domains and Hierarchies

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 (e.g., [11, 21]), but descriptions often tend to be very abstract. Pradhan distinguishes three types of locations [16]: *physical* locations such as GPS coordinates, *geographical* locations such as "City of Hagen" and *semantic* locations such as "Jörg's office at the university". In this paper, we do not distinguish geographical and semantic locations, but regard any location other than a physical one as a semantic location. In simplified terms: physical locations 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 some important advantages:

- Semantic locations have a meaning to the user; in contrast, physical locations usually have no meaning at all to most peoples.
- Semantic locations can easily be used as a search key for traditional databases, tables or lists. In contrast, to look up physical locations, we need spatial databases with the ability to deal with geometric objects such as polygons.

In this section, we want to describe the concept of semantic locations more precisely. We especially want to relate semantic 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.

- locations with a political meaning: countries, states, cities, districts;
- geographical locations: continents, oceans, mountains, rivers, lakes, forests;
- mobile locations: trains, planes, cars;
- temporary locations: construction zones, fairs;
- other locations: campus, malls, city centres.

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 in so-called *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 only letters and digits. 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.

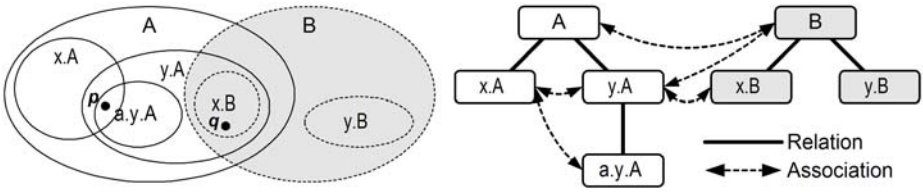

Fig. 2. Sample hierarchies

Fig. 2 shows an example with two hierarchies. Even though relations are directed, we use undirected lines in the figures as the directions are obvious.

3.1.2 Associations

In principle, the model is now expressive enough to specify realistic sets of semantic 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. 2 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 *semantic 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 $d_1 \sim d_2$, iff

- 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 the superfluous linking of masters to their subdomains as they always share an area. Associated domains can be in different hierarchies or in the same hierarchy (see fig. 2). 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, thus we can avoid the time-consuming search through all hierarchies.

We can however 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, i.e.

- $\left(d_1.c \setminus \bigcup_{e_1 \in C, d_1 \rhd e_1} e_1\right) \cap \left(d_2.c \setminus \bigcup_{e_2 \in C, d_2 \rhd e_2} e_2\right) \neq \{\}.$ (*condition 3*)

Note that if condition 3 is true, condition 2 is true as well, thus we can use condition 3 as definition for associations. We introduce the abbreviation

$$\Delta(d) = \left(d.c \setminus \bigcup_{e \in C, d \rhd e} e\right)$$

for the domain's area without the subdomains' area and finally get the short definition

$d_1 \sim d_2$ iff $\Delta(d_1) \cap \Delta(d_2) \neq \{\}$.

In fig. 2, 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 that condition 3 does not 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` ∩ `B` outside of `x.B`.

3.1.3 A Realistic Example

Fig. 3 shows a realistic semantic coordinate system. The figure shows a small part of a huge set of domains of two hierarchies: a `city` hierarchy contains the cities, districts etc. (white boxes) and a `geo` hierarchy contains geographical entities such as rivers and mountains (grey boxes). As an example `downtown.hagen.city` is associated to `volme.river.geo`, because Volme is a river which flows through the downtown of Hagen.

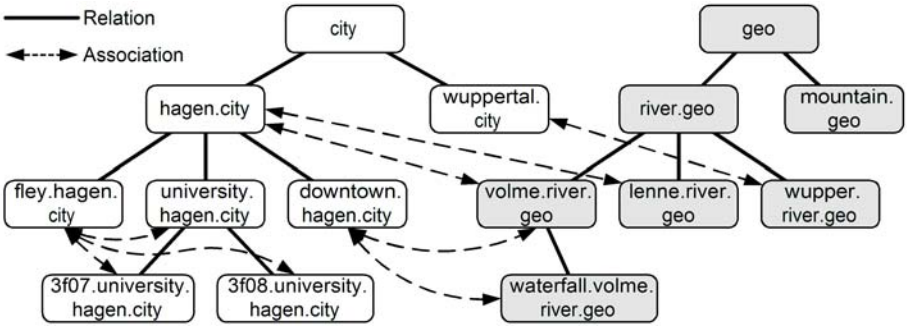

Fig. 3. A realistic semantic coordination system

Of course, there are more links conceivable between domains. We could, e.g. link two domains, if they have a common border. Bauer et al. proposed additional symbolic links to express topological aspects or to express proximity, which may be different from geometric distance [2]. We can store such links as meta data in a domain record, but they do not have any influence on the infrastructure described later. For the operations described in the next section, relations and associations are sufficient.

3.1.4 Operations on the Model

The primary 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 (e.g. [26]) directly produce semantic location output. Having both types, the application can choose the appropriate type (or even both types) for the specific operating condition.

To produce these location data, we have to perform one initial step: in our model, we currently can only express globally unique locations. Thus we have to perform a mapping, if the positioning system provides only local information. E.g., indoor radio systems (such as [4, 7, 14]) produce locally valid physical location output. They use, e.g., a special corner of the building as a reference point. So-called *mapping servers* set up for these positioning systems know the specific coordination system and transform from local to global locations.

Having a global valid location, we then can perform a resolution operation to get the respective other type. We distinguish two resolution operations:

- *Physical resolution*: Given a semantic location by its name *n*. What is the physical extension *d.c* of the domain *d* with this name?
- *Semantic resolution*: Given a physical location *p*. What are the names $\{n_i\}$ of the domains d_i which contain *p*?

The physical resolution is simple, as we only have to look up the appropriate domain and return *d.c*. The cost intensive part is the lookup mechanism which will lead to a network search function in the distributed implementation. We discuss this in a later

section. The more complex operation is the semantic resolution, as multiple hierarchies and domains may be involved. The algorithm to provide this resolution 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. Again, the cost-intensive part is the lookup.

At this point, we want to outline a proof of the correctness of this algorithm. We want to show that $d.name \in names$ iff $p \in \Delta(d)$:

Step 1: we have to show that all names collected by the algorithm correspond to domains which actually contain p. This is obviously true, as this condition is part of the lookup and if statement.

Step 2: we have to show that there is not any solution that the algorithm does not collect. Assumption: there is such a solution domain h. Thus there must be no subdomain of h containing p (otherwise h would not be a solution, but this subdomain), i.e. $p \in \Delta(h)$. Further $p \in \Delta(d)$, which is ensured by the first statement of the algorithm. As a result $p \in (\Delta(h) \cap \Delta(d)) \neq \{\}$, therefore condition 3 (see above) is true and thus h and d are associated. As $h \sim d$ and $p \in \Delta(h)$, the algorithm would have collected h which is a contradiction to the assumption above.

3.2 Storing Domain Data – The Third Dimension

Until now, we make two demands on domain data:

- We can precisely specify an area $d.c$.
- There is an effective test, whether a point p is inside an area $d.c$ or not.

Since we only have a finite storage space, an area $d.c$ is usually approximated. Storing geographical data is a task of geographic information systems. Typical geo databases centrally store a large amount of geographical data. In our case however, we want to store only a small number of domains at a specific site. As a result, we can avoid heavyweight geo databases and use instead a lightweight toolkit to process polygonal data [24]. The toolkit handles all geometric operations in the runtime memory and especially 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 conveniently can edit these XML files with the help of a graphical domain editor.

Two-dimensional polygons are sufficient for many domains. Unfortunately, our world is three-dimensional, thus for some domains it is necessary to take into account the third dimension. Some examples: Offices inside a building may have the same 2D coordinate; to map a physical location to the corresponding office, we have to consider the height. Another example is a street crossing another street via a bridge. On a bridge the 2D coordinates match both streets, thus we need the height to make a decision.

In principle, we could store a domain in three dimensions with the help of a volume model similar to those we find in CAD systems (fig. 4, left). With such a model we could specify arbitrary three-dimensional domains, but we have to consider two important disadvantages:

- As we cannot use the simple polygon inclusion test, it would be very cost-intensive to check if a point is inside a domain.
- It would be circumstantial to specify the three-dimensional domains, as most sources for domain data are basically two-dimensional, e.g. maps or ground plans from land registers.

As a solution, we avoid a full 3D representation and use a 2.5D representation as shown in fig. 4 (right). We request a domain to have a polygonal projection on a reference surface. As reference surface we use the WGS84 ellipsoid [27], which roughly can be viewed as an approximation of the earth's surface.

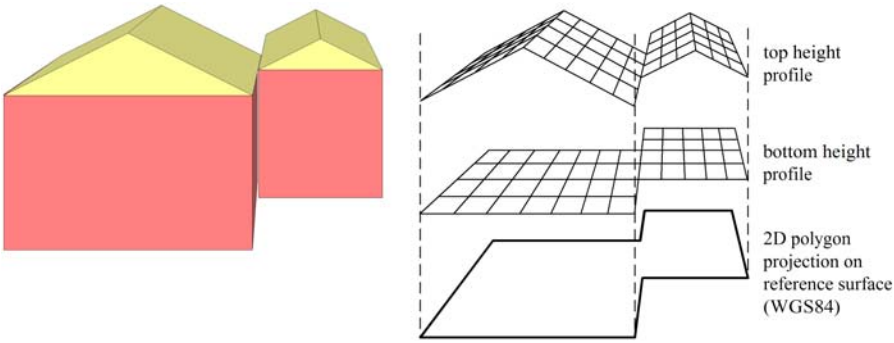

Fig. 4. Representing domains in three dimensions: full 3D (left), 2.5D (right)

We specify the third dimension of a domain with the help of two height profiles – a top and a bottom height profile. A height profile can be

- a surface specified by an array of reference points,
- a constant height, i.e. a surface parallel to the reference surface, or
- unspecified, i.e. the domain either extends to the sky or to the earth's centre.

Using the 2.5D representation, the check if a point is inside a domain is simple: We first check, if the 2D projection is inside the projected polygon of the domain with a simple polygon inclusion test. If not, the result is negative. Then we compute the domain's height values at the projected 2D position. If the height is inside the

height interval, the result is positive. Note that the height interval can be open at one or two sides, if the corresponding height profiles are unspecified.

As not all three-dimensional volume elements can be projected to a polygon and limited by a maximum of two height intervals, we cannot express some figures with our representation, e.g. some irregular polyhedrons. However, most conceivable realistic domains fulfil this requirement, thus we do not loose too much expressiveness.

The question is how to set the height profiles in reality. First, there is a huge class of domains, where the height is uncritical, e.g. countries or states. We either could leave these height profiles unspecified or set spacious constant heights such as [-500km...10000km].

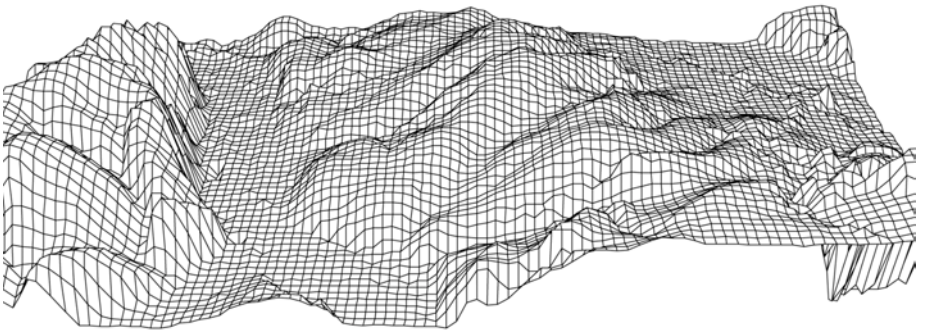

Fig. 5. Height profile of the city of Hagen

For buildings we can use constant heights, e.g., [178m...181m] for a specific floor. For the domains that require precise height profiles, e.g. streets, we can use height data as provided by land survey offices. Fig. 5, e.g., shows a height profile of the city of Hagen produced by a German land survey office [10]. Height profiles contain a number of reference points, in, e.g., a grid of 10m. We can easily compute height values between the reference points with the help of interpolation functions.

3.3 Performing Operations – The Location Server Infrastructure

We now switch from the abstract location model to an infrastructure storing model data. A location model is useless, unless we do not provide mechanisms to effectively run the required resolution operations. In principle, we could use one huge database and store hierarchies with the corresponding domains on a single server. A single database would be a bottleneck for a huge number of potential clients. In addition, information about local domains is usually available locally and difficult to administrate in a central database. As a solution, we use a distributed system of *location servers* each storing a number of domains.

3.3.1 The Infrastructure

Fig. 6 shows the distributed infrastructure which consists of three *segments*:

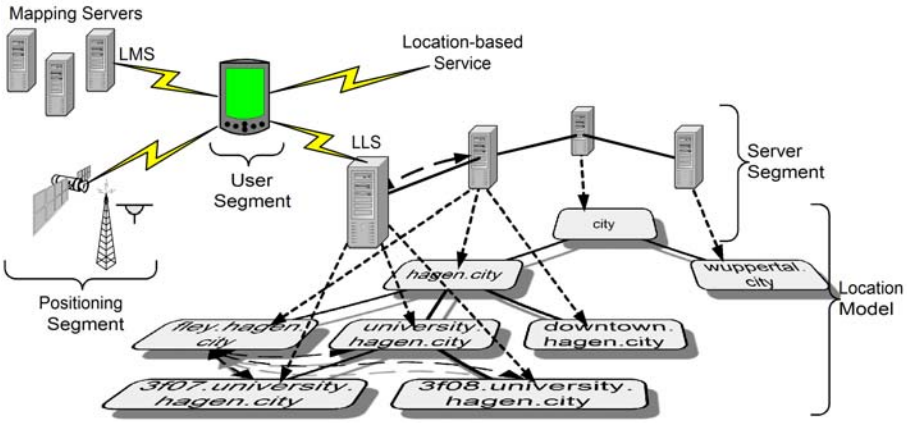

Fig. 6. The infrastructure

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 allow 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. Note that our infrastructure does not cover the network part of a location-based service. It depends on the mobile part to establish a connection to a specific server and to use the service. We developed a lightweight runtime system for the mobile nodes. We especially 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. In our example, the location server for `hagen.city` covers `fley.hagen.city` and `downtown.hagen.city`, but not `university.hagen.city`, as this domain has its own location server. 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. Especially, our system does not overload top-level servers.

The entire system is self-organizing. The location servers establish the links of relations and associations among each other automatically. Thus, a new location server

simply has to be configured using an XML file and turned on. A discovery procedure connects the server to its domain masters and looks up associated servers.

3.3.2 Looking Up Servers

Until now, we mentioned two different types of lookup: mobile clients looking up either an LMS or LLS, and a location server looking up other location servers (i.e. its master or associated servers). The latter lookup is called the *inter-server lookup*. As we do not have any central instance, the lookup has to run in a distributed manner. As location servers usually have a long lifetime, the links of relations and associations have a long lifetime as well (except for mobile domains, see open issues). As a result the inter-server lookup is uncritical and we can use well-established discovery mechanisms used in peer to peer networks, which may need a certain amount of time without significant drawbacks for the system.

Looking up the LMS and LLS is more critical: if a mobile node moves to a new location, the mobile user expects to use the service without interruption. Ideally, the user should not be aware, when the system performs a handover to a new LMS and LLS. For this, a mobile client automatically supervises its location and possibly discovers new servers. Our infrastructure supports the following lookup mechanisms:

- The mobile client can send lookup requests via broadcast messages using UDP multicast, or if available, Multicast IP.
- The mobile client can use service discovery protocols such as SLP or the network directories DHCP and DNS to ask for a server. For this, we defined new record types.
- The positioning system can distribute information about the LMS or LLS. Systems, which broadcast beacons, could e.g. distribute the corresponding network addresses in the beacon's payload.
- A mobile node can ask an arbitrary location server (e.g., the old LLS) for the new LLS.

The last point is very important: in principle, a mobile node has only once to know a location server to get the current LLS. A propagation mechanisms presented in [20] ensures that, after a certain (small) number of subsequent queries, an LLS will be found. The only prerequisite: there must be an uninterrupted sequence of relations and associations between the location servers, which in real environments is usually true.

We heavily can improve the lookup procedure using caches. When a mobile client looked up a server, it can store these data for a certain amount of time. Whenever it enters a specific area again, the lookup can thus be done without any network interaction.

3.3.3 Performing Resolution Operations

Having the lookup, we now can outline the distributed resolution operation, which is a distributed version of the algorithm presented in section 3.1.4:

- We first lookup an LMS which maps any local location data to global ones.
- We then lookup an LLS which returns one appropriate domain for a specific location and all associated servers.
- Subsequent queries to associated servers complete the resolution request.

This mechanism can entirely be controlled by the mobile client. We call this the *outbound mode*. In the outbound mode, the mobile node carries out a location resolution by subsequent queries to a number of location servers. This is efficient, as long as the mobile client is able to connect to every relevant location server. In some scenarios however, this is not possible: a mobile node may be separated from the global network by a firewall, which only allows pre-defined hosts to connect outside hosts. Or a mobile node using a cell phone network could have quick access to a server inside the phone network, but connections to servers outside are slow and cost intensive. In these cases, we use the so-called *inbound mode*: the mobile node only connects to one LLS, which in turn performs all subsequent queries to other location servers. Once an LLS queried the associated servers, the results are cached for future use.

3.4 Further Details

In this section, we summarize some further details concerning the location model:

Filtering: A specific location based-application may only be interested in a subset of all available domains. E.g., a bus schedule application may need semantic locations representing bus stations and not `geo` domains. If a mobile node only has to load specific domain information, we can drastically reduce the amount of network traffic. For this, we integrated so-called *domain filters*, which contain a description of subhierarchies included or excluded from the resolution process.

Compression: The number of associations can be very high for top-level servers. A request could lead to a large list of associated domains and cause heavy load on the server, especially in the inbound mode. We solve this problem with a compression mechanism: if the list of associations exceeds a certain limit, we connect a server to a top node of all associated domains. In fig. 2 we have an association between `A` and `B`. When compressing, we remove the association between `B` and `y.A` and just store one unidirectional association from `y.A` to `B`. To still get correct results, we need to modify the resolution algorithm: we now sometimes have to go down a hierarchy when checking associated candidates. As a benefit, we shift away processing load from top-level servers.

Proximity: The described model only resolves locations which are *inside* a certain area. We further could ask the system for domains in the nearer area. We call this operation the *proximity resolution*. We developed an algorithm, which collects all domains inside a certain circle with a minimum of network transactions.

3.5 Open Issues

Even though Nimbus reached a high level of completeness, we have some open issues:

Organizational Aspects: The technical platform is entirely decentralized. Nevertheless, for a specific hierarchy, we need a central organization to supervise the registration of subhierarchies. This problem is similar to the registration of Internet domain names. In addition to formal parameters such as the domain name, covered physical area etc., a domain has to satisfy informal conditions. E.g., if a city wants to register as a subdomain of `city`, one could require that it has a certain number of inhabitants. Our system currently does not support such issues and concentrates on the technical infrastructure. We could consider a second infrastructure to help organizations to control hierarchies and administrate additional information about domains.

Secret Domains: In the current implementation, our system stores domains with a public character. Every user can access all domains as domains such as rivers or cities have a certain meaning for the public. Some domains however should not be open for everyone, e.g. barracks in a military area. We are currently working on appropriate access control mechanisms.

Mobile Domains: Domains such as trains or ships permanently change their location. In principle, our system supports such domains, but they cause a high amount of updates messages. We currently work on a mechanism which avoids huge traffic and at the same time ensures consistency.

4 Conclusion

In this paper, we presented a location model especially designed for mobile users accessing location information. We introduced two resolution operations which provide a unique location data independently of the underlying positioning systems. We considered semantic locations and modelled three-dimensional locations with the help of a 2.5D approach. 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 positioning capturing and resolution. As the corresponding infrastructure is self-organizing and decentralized, it is highly accessible and scalable.

References

1. 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
2. Bauer, M.; Becker, C.; Rothermel, K.: 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
3. Beigl, M.; Zimmer, T.; Decker, C.: A Location Model for Communicating and Processing of Context, Personal and Ubiquitous Computing, Vol. 6, No. 5, Dec. 2002, 341-357

4. Bahl, P.; Padmanabhan, V., N.: User Location and Tracking in an In-Building Radio Network, Microsoft Research Technical Report MSR-TR-99-12, Febr. 1999
5. Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.; Efstratiou, C., 2000: Developing a Context-aware Electronic Tourist Guide, in Proc. of CHI'00, ACM Press
6. Dey, A., K.; Abowd, G., D., 2000: CybreMinder: A Context-aware System for Supporting Reminders, Second International Symposion on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer-Verlag, 187-199
7. Hightower, J.; Boriello, G.; Want, R.: SpotON: An Indoor 3D Location Sensing Technology based on RF Signal Strength, Technical Report #2000-02-02, University of Washington, Febr. 2000
8. Hohl, F; Kubach, U.; Leonhardi, A.; Schwehm, M.; Rothermel, K.: Nexus - an open global infrastructure for spatial-aware applications. In Proc. of the 5th Intern. Conference on Mobile Computing and Networking (MobiCom '99), Seattle, WA, USA, 1999. ACM Press
9. 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, Proc. 3rd Annual Wireless and Mobile Computer Systems and Applications, Monterey CA, USA, Dec. 2000
10. Land Survey Office North Rhine-Westphalia, http://www.lverma.nrw.de (in German)
11. Leonhardt, U.: Supporting Location-Awareness in Open Distributed Systems, PhD Thesis, University of London, 1998
12. Marmasse, N.; Schmandt, C., 2000: Location-aware Information Delivery with ComMotion, Second International Symposion on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer, 157-171
13. Navas, J.; Imielinski, T.: GeoCast – Geographic addressing and routing, Proc. of the 3rd ACM/IEEE inter. conf. on Mobile Computing and networking, Sept. 26-30, 1997, 66-76
14. Nibble Location System, http://mmsl.cs.ucla.edu/nibble
15. Open GIS Consortium, OpenLS Home Page, www.openls.org
16. Pradhan, S.: Semantic Locations, Personal Technologies, Vol. 4, No. 4, 2000, 213-216
17. Roth, J.: 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
18. Roth, J.: Context-aware Web Applications Using the PinPoint Infrastructure, IADIS Intern. Conference WWW/Internet 2002, Lisbon, Portugal, Nov. 13-15 2002, IADIS press, 3-10
19. Roth, J.: Flexible Positioning for Location-Based Services, IADIS International Conference e-Society 2003, Lisbon, Portugal, 3-6 June 2003, IADIS Press, 296-304
20. Roth, J.: Semantic Geocast Using a Self-organizing Infrastructure, Innovative Internet Community Systems (I2CS), Leipzig, June 19-21, 2003, Springer-Verlag
21. Schilit, B.; Adams, N.; Want, R., 1994: Context-Aware Computing Applications, Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 1994
22. Tomlin, C., D.: Geographic Information Systems and Cartrographic Modeling, Prentice Hall, 1990
23. UMTS Forum, Enabling UMTS/Third Generation Services and Applications, Report 11, http://www.umts-forum.org, Oct. 2000
24. Vivid Solutions, JTS Technical Specifications, http://www.vividsolutions.com, March 31, 2003
25. Vodafone Homepage, www.vodafone.com, 2003
26. Want, R.; Hopper, A.; Falcao, V.; Gibbson, J.: The Active Badge Location System, ACM Transactions on Information Systems, Vol. 10, No. 1, Jan. 1992, 91-102
27. WGS 84 - Implementation Manual, EUROCONTROL European Organization for the Safety of Air Navigation, Brussels, Belgium, Febr. 1998