IADIS International Journal on WWW/Internet
Vol. 1, No. 2, pp. 18-32
ISSN: 1645-7641

FLEXIBLE POSITIONING FOR LOCATION-BASED SERVICES

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

ABSTRACT

Location-based applications and services will get increasingly important for mobile users of the future. They take into account a mobile user's current location and provide location-dependent output. Often, location-based applications have to deal with raw location data of specific positioning systems (e.g. GPS) and perform further processing to get an appropriate representation. In this paper, we present a platform, which hides specific details of positioning systems and provides a uniform output containing both physical as well as semantic information. With the help of our platform, an application developer can concentrate on the application-specific details and has not to deal with position capturing and mapping. The corresponding infrastructure reflects a location domain model designed to cover the needs of mobile users. The domain model contains logical links and allows the expression of semantic relations between locations. As our infrastructure is self-organizing, it is flexible and easy to extend.

KEYWORDS

Ubiquitous computing, location-based services, positioning systems, wireless communication

1. INTRODUCTION

Determining a mobile user's current location will be one of the most important functions of future mobile computing environments. Location-awareness is a key issue for mobile, ubiquitous and handheld computing and many people expect a high potential of location-based services such as city guides or navigation systems for m-commerce scenarios.

Many applications use the satellite navigation system GPS (global positioning system) to determine the current location. GPS receivers are inexpensive and the corresponding location output is accurate, thus GPS is widely accepted. GPS however only works outdoors since the receiver must have a direct "view" to at least four GPS satellites. If a location-based service is designed for a high coverage, it has to use other (e.g. indoor) positioning systems. Accessing more positioning systems however, increases the complexity of a location-based service. As a solution, we suggest a flexible positioning platform, which relieves the service developer from position capturing and converting coordinates into different coordinate systems. Using such a platform, a developer can concentrate on the actual service function and has not to deal with sensors and low-level protocols to determine the location. In addition, such a platform can provide additional location information such as the semantic locations. Semantic locations transfer, in contrast to physical coordinates, a *meaning* of the location. E.g., instead of the GPS coordinates *N51°22.579/E007°29.616/169m*, it is often more meaningful to use the term "University of Hagen, building IZ, back door".

In this paper, we present the self-organizing, decentralized *Location Server Infrastructure* (*LSI*), which strongly separates the location-based service from positioning issues. We strongly considered scalability and accessibility issues. In our approach, we use a network of location servers, which automatically connect to each other. As the location data is distributed among a network of servers, our infrastructure is highly accessible for mobile users.

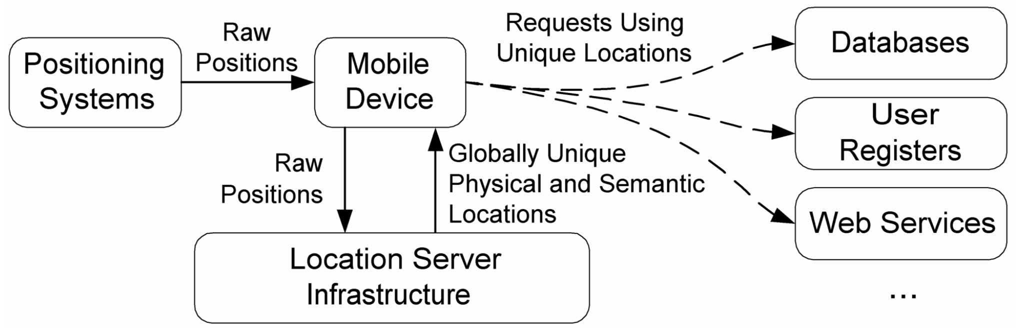

Positioning Systems
Raw Positions
Mobile Device
Requests Using Unique Locations
Databases
User Registers
Web Services
...
Raw Positions
Globally Unique Physical and Semantic Locations
Location Server Infrastructure

Fig. 1. The global data flow

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 the decentralized infrastructure. To achieve an optimal flexibility, the framework provides physical coordinates as well as semantic locations. As these data are globally unique with a well-defined format, they can easily be used as a search key to access databases, user registers or web services. 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-aware application. An application developer can thus concentrate on the actual application function and does not have to deal with positioning sensors or capturing protocols.

2. RELATED WORK

Many location-based applications have been developed over the last years. Tourist information systems are ideal examples for such applications. The systems CYBERGUIDE [1], GUIDE [4] and the PinPoint Tourist Guide [20] 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 [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 [5] allows the user to define complex 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. Leonardt describes a conceptual approach to handle multi-sensor input from different positing systems [11]. Cooltown [10] 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 could only be modified through the Nexus system. OpenLS [17] is an upcoming project and provides a high-level framework to build location-based services.

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.

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. Especially the relation of physical to semantic locations and the mapping of local sensor-data to global location information have a high influence on how location-based applications perform their service.

3. LOCATIONS AND POSITIONING SYSTEMS

Pradhan distinguishes three types of locations [19]: *physical* locations such as GPS coordinates, *geographical* locations such as "City of Hagen" and *semantic* locations such as "Jörg's office at the university". The notion of semantic locations is not new [11], but descriptions often tend to be very abstract. Semantic locations play a major role for applications, which do not focus on physical coordinates, but on the *meaning* of a location. A semantic location can be a railway station or airport, a city centre or a room inside a building, a river or mountain or a private apartment. Note that a user can reside at different semantic locations at the same time, e.g., being in a railway station, a user may be in a city centre as well. Some applications can process semantic locations much easier than physical ones, as they represent an entire area of physical locations. Semantic locations can easily be used as a search key for traditional databases. In this paper, we do not distinguish geographic and semantic locations any more, but view any location other than physical as a semantic location.

Different location-based applications or services need different location types: a navigation system for sailors needs physical locations, whereas a tourist guide may need semantic locations. The positioning systems, on the other hand, provide different types of location data, regardless which type of location currently is needed by the application. Table 1 shows some examples.

In addition to the location *type* (semantic or physical), we can classify positioning systems according to the *scope* (local or global). Satellite navigation systems such as GPS (Global Positioning System) provide *globally unique* physical locations. Indoor radio systems (such as [2, 7, 16]) in contrast provide physical locations which are valid only *locally*. They use a location such as a special corner of the building as a reference point.

A positioning system that provides global physical locations is based on the cellular phone network GSM (Global System for Mobile Communication) [6]: each GSM base station transmits a globally unique cell ID to the mobile phones. This mechanism is called cell of origin (COO). In addition, a mobile phone knows the distance to the base station in steps of 555 m with the help of timing measurements between base station and mobile phone (timing advance [14]). With this information, a mobile phone can look up its area, described as a circle segment with a certain thickness.

Some positioning systems already provide a kind of semantic location. Indoor infrared systems [24, 25] use the property of infrared light not to penetrate walls. An infrared beacon usually covers an entire room or hallway, which can be viewed as a fine-grained semantic location.

As a last example, we want to mention systems based on network infrastructures. Such systems are appropriate, if a specific network covers a limited (physical) area. Often network addresses reflect the structure inside an organization, as every department gets its own subnet address. Such subnet addresses are good candidates for semantic locations. Network participants inside a certain subnet can roughly determine their physical locations if they know the physical expansion of the subnet. A fine-grained approach, which maps network addresses to physical locations, is presented in [19].

Table 1. Positioning systems, location types and scopes

Positioning System	**Location Type**	**Scope**	**Example Location**
GPS	Physical	Global	5122.5790,N, 00729.6150,E,169.0M
Indoor Radio	Physical	Local	x=3.5m y=2.67m

GSM Cells	Physical (COO)	Global	CellID= 011B 28BF
			Dist= 1,6-2,2 km

Indoor Infrared	Semantic (COO)	Local	Room F.08
Subnet IDs	Semantic	Global	132.176.67.*

4. THE LOCATION SERVER INFRASTRUCTURE

The primary goal of our approach is to provide uniform location information, which is independent from the actual positioning system. Even if a user moves from one positioning system to another, a location-based application should not notice this. For each position, we want to provide a physical location as well as *all* semantic locations, which cover this position. We want to provide both location types since we do not want to impose any limitations to location-aware applications. Having both types, the application can choose the appropriate type (or even both) for the specific operating condition.

4.1 The Location Model

We use a location model primarily based on semantic locations to structure the physical space. Moreover, semantic locations are an ideal means to organize the logical network of servers.

We structure the entire space with so-called *hierarchies*. Hierarchies are built up of *domains*, each of it representing a semantic location which its corresponding physical extension (e.g. the physical bounding of a city centre). Each hierarchy has a root domain and a number of subdomains; each of it can in turn be divided into subdomains. We call a domain a *master* of the corresponding subdomains. Fig. 2 shows two hierarchies, a *de* hierarchy (the area of Germany, white boxes) and a *geo* hierarchy (geographic entities such as rivers and mountains, grey boxes). We call a link between a subdomain and a master domain a *relation*. Relations carry information about containment of one domain according to another, i.e. a subdomain is fully embedded into a master's domain.

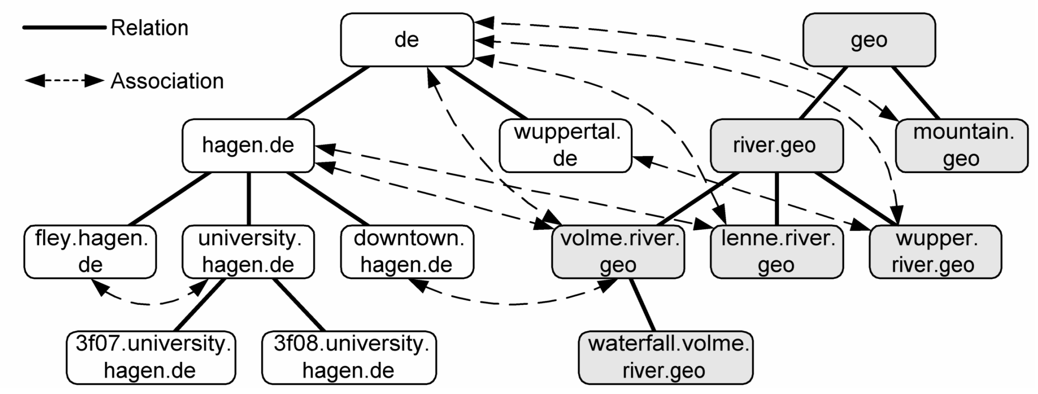


Fig. 2. The domain model

Domain names follow the Domain Name System (DNS) of the Internet [13]: the name of a subdomain extends the master's domain name according to the pattern *<subdomain>:=<extension>.<master>*.

In addition to relations, a domain can be *associated* to other domains. Domains are associated, if their corresponding physical areas overlap. Associated domains can be in different hierarchies or in the same hierarchy. The domain *downtown.hagen.de* is associated to *volme.river.geo*, because Volme is a river that flows through the downtown of Hagen. Associated domains can also be inside the same hierarchy (e.g. *fley.hagen.de* and *univer-sity.hagen.de*), i.e. we allow two subdomains of a domain to overlap.

Using such a model, we have to address organizational issues: we have to find useful domain names and divide hierarchies into useful subhierarchies. We have to decide how high a specific domain will be inside a hierarchy and which physical area it exactly covers. We want to put back such problems. We believe that we only can solve these problems, if we have a meaningful model and a powerful underlying technical infrastructure. Therefore, we concentrate in this paper on the technical and formal issues to build reasonable structures.

4.2 The Location Server Infrastructure

Storing maps and retrieving location information is traditionally a domain of spatial databases and geographic information systems. As an ad-hoc solution for our problem, we could use one huge database and store hierarchies with the corresponding domains on one server. A single database for a big number of potential clients, however, would be a bottleneck. In addition, information about local domains is usually available only locally and difficult to administrate in a central database.

Representing each domain by a separate server on the other hand would lead to an unacceptable high number of servers. An ideal solution lies between the two extremes. Our approach was strongly influenced by the Domain Name System of the Internet. The DNS has a similar function as our infrastructure: it maps symbolic host names to network addresses. DNS has a decentralized architecture: local administrators are allowed to define new local names and mappings and store them in local servers. We follow a similar approach and store location information in so-called *location servers*. Fig. 3 shows how location servers can be mapped to the location model.

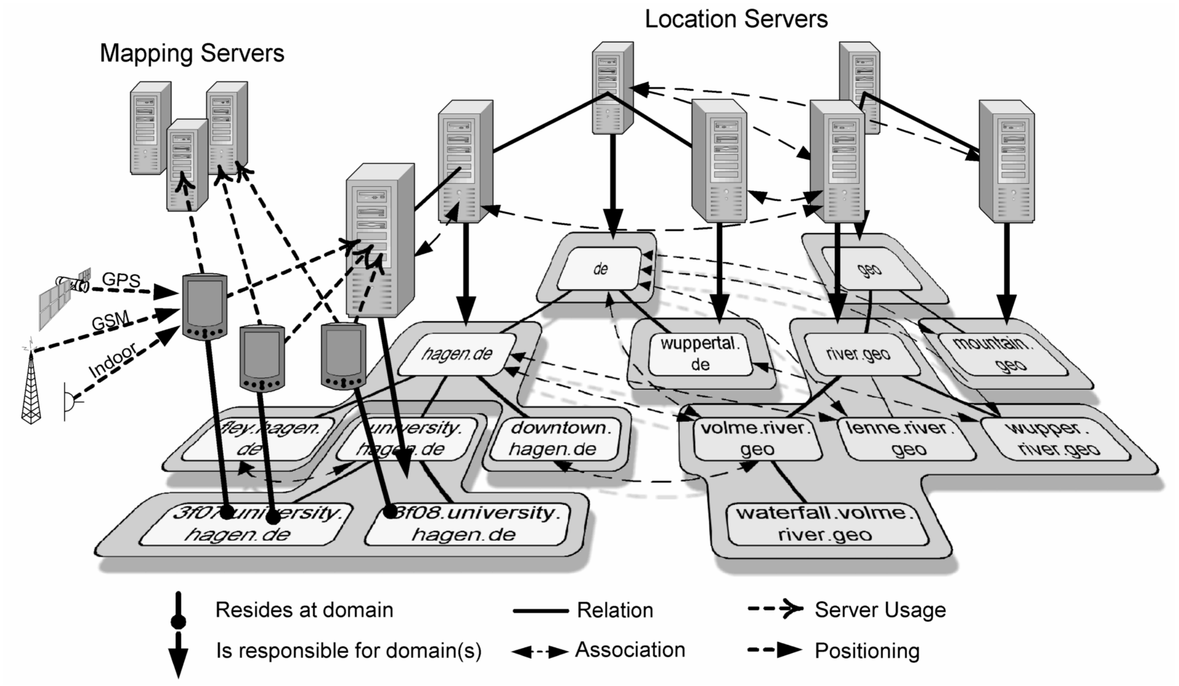


Fig. 3. The logical location server network

Each location server is responsible for a specific domain and all subdomains, for which no other location server is set up. In our example, the location server for *hagen.de* covers *fley.hagen.de* and *downtown.hagen.de*, but not *university.hagen.de*, as this domain has its own location server. Since mobile users are distributed among different location servers, this infrastructure is highly scalable. Our system does not overload top-level location servers.

As many positioning systems only provide local positioning data, we may need the help of so-called *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 installed indoor positioning systems. Mapping servers perform their tasks individually without the help of other mapping servers, thus they do not have to build up a logical network.

4.3 Resolving Positions

A mobile computer is connected to one or more positioning systems, which may provide raw position data with only local meaning. The primary task of our platform is to produce globally valid physical and semantic location data. For this, the mobile computer connects to the responsible location server (called the *local location server*, *LLS*) and the corresponding mapping server (*local mapping server*, *LMS*). In our example (fig. 3), the mobile nodes residing at the university of Hagen connect to the server *university.hagen.de*. Only this server knows the corresponding subdomains. In addition, the mapping server of the university knows the used positioning systems inside the university. The knowledge of positioning systems is important to perform the mapping of raw position data to globally valid positions. In detail, our servers perform the following mapping for mobile computers:

- local physical → global physical, e.g., *5.5m/6.3m → N51°22.581/E007°29.622/169m*
- local semantic → global semantic, e.g., *beacon_3F08 → 3f08.university.hagen.de*
- global physical → global semantic, e.g., *N51°22.578/E007°29.610/169m → 3f08.university.hagen.de*
- global semantic → global physical area, e.g., *3f08.university.hagen.de* → polygon(*N51°22.577/E007°29.614/169m*, ...)

Note that a semantic location cannot be mapped to single physical point, but to a physical *area* described by, e.g., a polygon. Fig. 4 illustrates the individual steps to get the required location data.

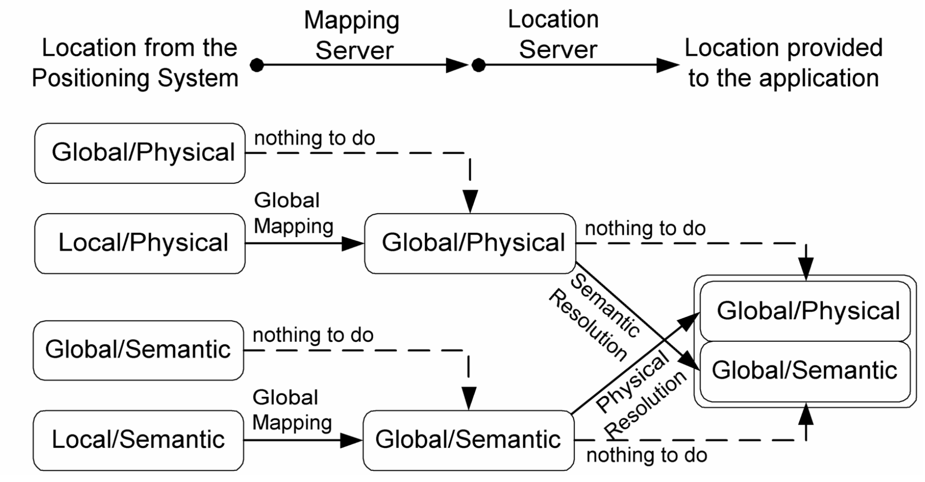


Fig. 4. Steps to perform a mapping of location data

With these mappings, our platform can provide physical positions (or areas) and semantic locations from any positioning system. If more than one positioning system is available, the client can choose the "best" one with the help of parameters such as *precision* or the *age* of location data. In addition, the platform can merge location data of different systems to increase precision.

A crucial point for a mobile node is to find the LLS and LMS when it enters a new region. One design goal was to run the system with a minimum configuration; particularly the user should not enter raw network addresses manually. When moving between locations, the user should not be aware, if the system performs a handover to new servers. A mobile client has to permanently supervise its location and possibly discover a new server. Our infrastructure supports the following discovery mechanisms:

- The mobile client can send lookup requests to the network, via, e.g. broadcast messages.
- The mobile client can use service discovery protocols such as SLP.
- The positioning system can distribute information about the corresponding servers via, e.g., infrared beacons.
- After a movement, the mobile node can ask the old LLS for the new one. For this, the location server uses relations and associations. In addition, an LLS distributes a list of potential LMSs inside its area.

A location server notices whenever a mobile client wants to resolve a position outside the covered area. In this case, it uses the relations to find a more suitable server. The server first redirects the request to its master, which in turn may redirect it to its master or a more suitable subdomain server.

We use the association links to resolve *all* semantic locations. If in the example above, a mobile user sails on the river Volme through the downtown of Hagen, the resolution process should return *volme.river.geo* and *downtown.hagen.de*. The LLS possibly only computes a subset of semantic locations. Only if all relevant location servers contribute their results, we get a complete list of semantic locations. For this, we first query the LLS for a list of associated location servers, and then query all associated for additional semantic locations.

4.4 Maintaining the Logical Network

The resolution mechanism requires the location servers to know their master, subdomains and associated location servers. Building and maintaining the corresponding entries is performed in three phases: *discovery*, *registration* and *re-registration*. The first two phases only run once when a server starts its service.

When a server starts up the first time, it has to look for other location servers. A location server can use for this discovery mechanisms as described in the previous section. Using network broadcast, a server transmits a discovery message with its own domain name and the specification of the physical area it covers. A server, which receives a discovery message, can easily decide, whether this message is relevant or not.

When all replies arrived, the new location server registers itself to other servers. We strongly separated the discovery from the registration procedure, as both steps use different communication paradigms. Discovery uses, e.g., multicast protocols which are only unidirectional, thus unsuitable for registration. The registration step in contrast uses a reliable bidirectional transport protocol, thus a registration can be acknowledged by the receiver. Once a registration was accepted, both servers store the relation or association in their local databases.

If a location server breaks down or disconnects from the network, it cannot explicitly inform other servers. Thus, we need a mechanism to ensure that inactive location servers are automatically removed from databases after a certain time. Active servers have to periodically re-register to all related and associated servers. If a re-registration is overdue, a server deletes this server from its list.

4.5 The Software Architecture and Implementation Details

Fig. 5 shows the software architecture. The main parts of the architecture are the client and server processes. An XML file configures the server process during startup. Inside a location server, a spatial engine does the processing concerning spatial information. The spatial engine decides, e.g., if a physical location resides in a certain area. The current implementation of the spatial engine uses polygonal representations of areas based on a 2.5 dimensional model, thus we can consider height data in our domains. A highly optimised platform [23] computes the required polygonal operations.

The mobile node hosts a client process, which fully runs in the background. Developers of location-based applications can use the LSI framework with the help of the LSI API.

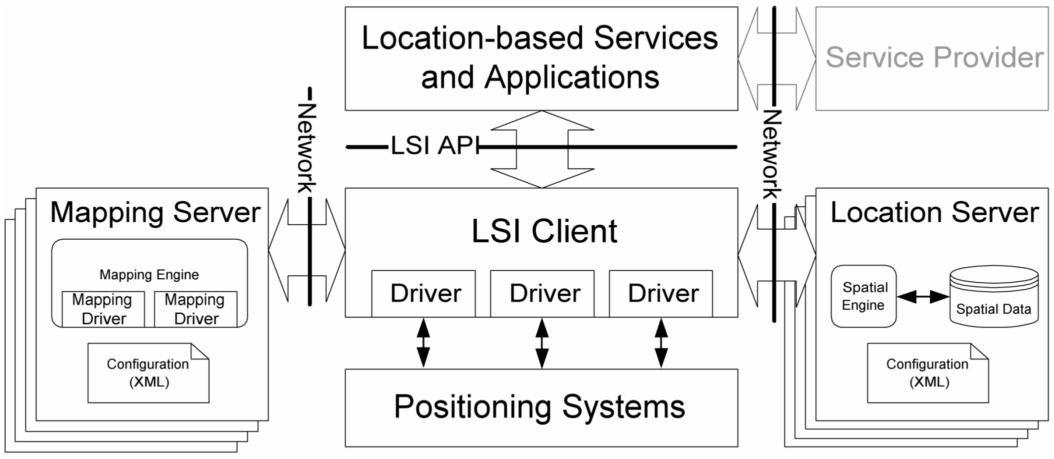


Fig. 5. The software architecture

Note that the communication between a mobile client and the service provider for the actual location-based service is not addressed by our platform. It depends on the location-based service or application to find an appropriate server and establish a service connection.

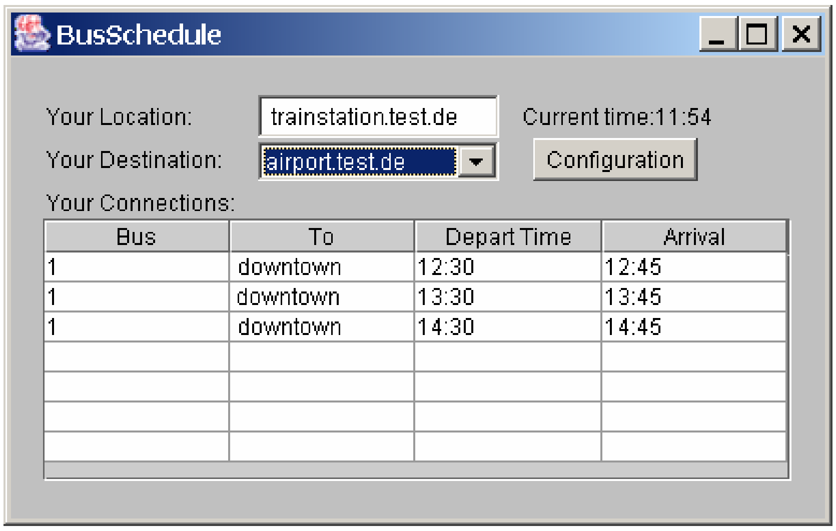
The platform communicates with positioning systems via *drivers*. Drivers run the capturing protocol with the positioning device. In addition to location data, client drivers provide information about the scope and type of location, thus the client can classify the positioning system without manual configuration. Drivers have a strongly defined driver interface. We can easily attach positioning systems to the client during runtime. Mapping servers get their knowledge about the mapping functions of positioning system by so-called *mapping drivers* that can easily be plugged into the server. We assume that each positioning system comes along with a client driver and a mapping driver, ideally developed by the positioning system manufacturer.

We implemented the client and server processes in Java, with some time-critical parts in C. The software system currently consists of 150 Java classes with approx. 70 000 lines of code.

A sample code to demonstrate the strength of LSI can be outlined as follows.

```
ResolvedPosition p=LSI.resolve();
for (int i=0;i<p.sempos.length;i++)
   if (p.sempos[i].endsWith("hagen.de"))
       ShowMessage("Now I'm in Hagen!"+
                   "My physical position is "+p.physpos.toString());
```

This program resolves the current position with the help of the API and checks, if the user resides at a certain semantic position. To avoid polling, an application can register itself as a *listener* for position-related events. As a result, the API calls a certain method of the application, whenever a specific location is entered or left.



Your Location: trainstation.test.de
Current time:11:54
Your Destination: airport.test.de
Your Connections:

Bus	To	Depart Time	Arrival
1	downtown	12:30	12:45
1	downtown	13:30	13:45
1	downtown	14:30	14:45

Fig. 6. A sample application

A more complex application that makes use of our infrastructure is the bus schedule (fig. 6). When a user starts the application, it determines the current semantic location. The user selects a destination and the application looks up appropriate bus connections. The application automatically informs the user when to change the bus or when he arrived at the final destination. The planning process fully operates on semantic locations, which are easy to look up in the timetables. Since the application has not to deal with the positioning process, it is very small (approx. 700 lines of code). The development needed less than one week.

To even more simplify the development, we created a platform on top of LSI called *PinPoint* [20], which allows using the powerful WWW infrastructure for location-based services. With PinPoint we developed, e.g., a tourist guide which runs in a Web browser of-the-shelf.

4.6 Further Details

Some further details of our platforms are the following:

Caching: Resolving a location may need a lot of network traffic. Thus, it is useful to store a result for further queries in a client's cache. A cache stores the domain name with the physical area replied from the resolution process. As long as the mobile node moves inside this area, it still belongs to the corresponding domain. An ad-hoc implementation however, leads to a problem as demonstrated in fig. 7.

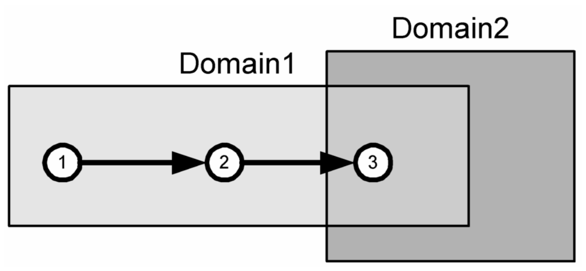


Fig. 7. The cache problem

In step 1, the mobile node resolves its location and caches *Domain1*. In steps 2 and 3, the mobile node has a cache hit and still expects to be in *Domain1*. However, in step 3, the mobile node enters additionally *Domain2*, thus both domains had to be replied but the cache still only replies *Domain1*. As a result, cache hits cannot be used without further processing. Our cache mechanism works as follows:

- A resolution request replies in addition to the actual area all associated domains with its physical areas. Associated domain entries are stored in a second cache. In our example: when the mobile node resolves the location 1, LSI replies *Domain1* and *Domain2* even though *Domain2* is currently not in reach.
- If a cache hit occurs, we have to look in the second cache to verify if no other domain covers the new position.

To react to changes of domain data (e.g., of mobile or temporary domains), each cache entry has a certain expiration time. A server defines the expiration time for its entries individually.

Filtering: A specific location based-application may only be interested in a small subset of all available domains. E.g., a tourist guide for a specific city may need only access to the subhierarchy of the city. If a mobile node only has to load specific domain information, we can drastically reduce the amount of network traffic. We configure the access to specific domain information with so-called domain filters, which contain a description of subhierarchies included or excluded from a resolution process. A domain filter could, e.g., be

include hagen.de
include river.geo
exclude university.hagen.de

Domain filters are both used for discovery and location resolution. A domain filter is highly application dependent, thus should be defined by the location-based application or service.

Compression: The number of associations can be very high and overload servers, especially top-level servers. We solve this problem with a *compression* mechanism. For this, associations are replaced by the highest server of the corresponding subhierarchy. In fig. 8, the *de* domain first is associated to *volme.river.geo*, *lenne.river.geo*, and *wupper.river.geo*. The compression replaces these associations by a single association between *de* and *river.geo*. This however, slows down the resolution process. Looking for overlapping domains, we now sometimes have to go down a hierarchy, thus a server has to carefully decide when to compress.

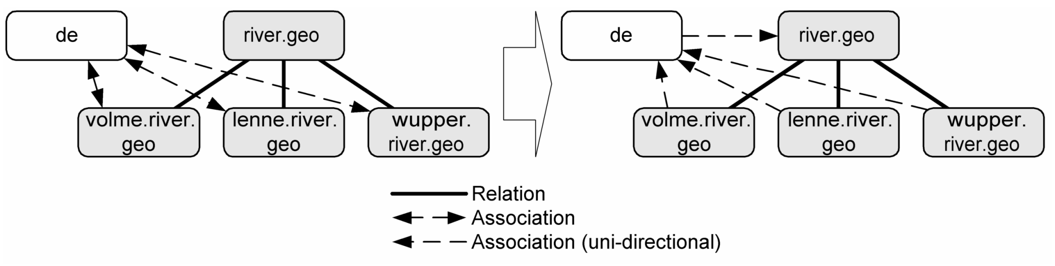


Fig. 8. Compressing associations

Security: In distributed systems, security is an important issue. Attackers could, e.g., define senseless domains or could install faulty location servers, which ignore registrations or send wrong resolution replies to mobile nodes. To protect a server or mobile node against malicious servers, a node can request a certificate of the correspondence node. According to established security infrastructures, we use a certification authority to administrate and sign certificates. Fig. 9 shows the steps to get a certificate.

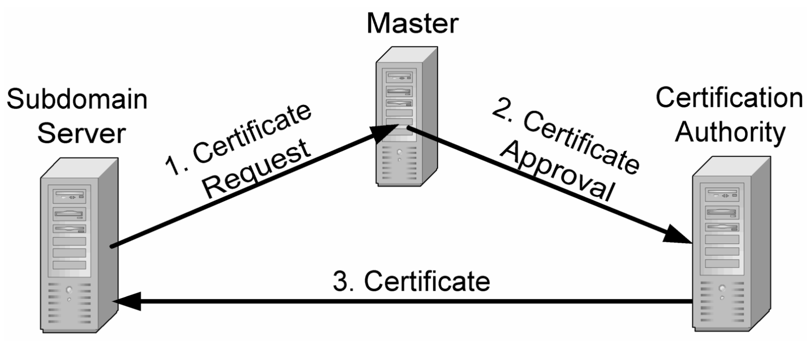


Fig. 9. Requesting a certificate

Certification authorities cannot directly proof requests of location servers as issues regarding the contents have to be considered. A certificate request is first passed to the master, which performs the following checks:

- Is the subdomain's name syntactically valid and does it indicate the subdomain relation to this master?
- Is the subdomain's area valid and completely inside the master's area?
- Is this the first request with this name?

Only if all checks are positive, a master accepts a request. A master performs these checks automatically. A domain administrator can be asked to perform further checks. E.g., a subdomain of *de* should be a city. The administrator can check, if the name is a city name and the area really describes the area of the specified city.

Approved requests (signed by the master) are passed to the certification authority (step 2). It checks, if the master already has a valid certificate, the request is properly signed, and if the original request comes from one of its subdomains. In a third step, the certificate is generated and signed by the certification authority.

Top-level servers send their requests directly to the certification authority, as they do not have a master. We assume that top-level servers play a unique role in our infrastructure and get their certificate from the certification authority by explicit appointment.

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. This operation is useful for domains with a fuzzy boundary such as city centres. For such domains, it is often more useful to return the distance to the user's position rather than the information about containment.

5. OPEN ISSUES

Although our system has reached a certain state of completeness, some issues are still open and subject of intensive future research.

Organizational Aspects: The technical platform of LSI 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 domain name or covered physical area, a domain has to satisfy informal conditions. E.g., if a city wants to register as a subdomain of *de*, one could require that the city 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 store, e.g., 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.

6. CONCLUSION

Currently, no positioning system is available which is accessible everywhere and which provides both physical as well as semantic locations. Our infrastructure makes location data available for arbitrary location-based applications and allows the integration of a wide range of positioning systems. Developers can use this infrastructure as a platform and do not have to deal with positioning capturing and resolution. The infrastructure is self-structuring and decentralized, thus highly accessible and scalable. In the corresponding location model, we define hierarchies and express logical links between domains, thus the system provides a semantic structure of the entire location space.

ACKNOWLEDGEMENT

I want to thank Nils Kunstmann, Jan Quadflieg and Daniel Weinand for implementing the LSI core classes, demo applications and tools, Hagen Barlag and Stephan Drautz for their work on the security infrastructure, and Thomas Hadig for his work on the proximity resolution algorithm. Further, I want to thank Daniela Keller and Dominic Heutelbeck for valuable in-depth discussions about LSI.

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] 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

[3] Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.: The Role of Connectivity in Supporting Context-Sensitive Appliations, Proc. of the first Intern. Symposium on Handheld and Ubiquitous Computing HUC '99, Karlsruhe Germany, Springer-Verlag, 1999, 193-207

[4] Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.; Efstratiou, C., 2000: Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences, in Proceedings of CHI'00 (Netherlands, 2000), ACM Press

[5] 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

[6] Ericsson, Mobile Positioning, http://www.ericsson.com/developerszone/

[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 Proceedings of The Fifth Annual International Conference on Mobile Computing and Networking (MobiCom '99), Seattle, WA, USA, 1999. ACM Press

[9] José, R.; Davies, N.: Scalable and Flexible Location-Based Services for Ubiquitous Information Access, Proc. of the first Intern. Symposium on Handheld and Ubiquitous Computing HUC '99, Karlsruhe Germany, Springer-Verlag, 1999, 52-66

[10] 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. 3 rd Annual Wireless and Mobile Computer Systems and Applications, Monterey CA, USA, Dec. 2000. p. 19

[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] Mockapetris, P.: Domain Names - Concepts and Facilities, Request for Comments 1034, November 1987

[14] Mouly, M.; Pautet, M.-B.: The GSM System for Mobile Communications, Published by the authors, ISBN 2-9507190-0-7, 1992

[15] Navas, J.; Imielinski, T.: GeoCast – Geographic addressing and routing, Proc. of the third annual ACM/IEEE internat. conf. on Mobile Computing and networking, Sept. 26-30, 1997, 66-76

[16] Nibble Location System, http://mmsl.cs.ucla.edu/nibble

[17] Open GIS Consortium, OpenLS Home Page, www.openls.org

[18] Padmanabhan, V., N.; Subramanian, L.: Determining the Geographic Location of Internet Hosts, Microsoft Technical Report MSR-TR-2000-110, Nov. 2000

[19] Pradhan, S.: Semantic Locations, Personal Technologies, Vol. 4, No. 4, 2000, 213-216

[20] Roth, J.: Context-aware Web Applications Using the PinPoint Infrastructure, IADIS International Conference WWW/Internet 2002, Lisbon, Portugal, Nov. 13-15 2002, IADIS press, 3-10

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

[24] Want, R.; Hopper, A.; Falcao, V.; Gibbson, J.: The Active Badge Location System, ACM Transactions on Information Systems, Vol. 10, No. 1, Januar 1992, 91-102

[25] WIPS Technical Documentation, Royal Institute of Technology, Schweden, http://2g1319.ssvl.kth.se/2000/group12/technical.html