PROXIMITY SERVICES WITH THE NIMBUS FRAMEWORK

Thomas Hadig
Stanford University
2575 Sand Hill Road, Menlo Park CA 94025, USA
hadig@stanford.edu

Jörg Roth
University of Hagen
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 location-dependent output. Currently, location-based applications are mainly provided by mobile phone networks, but we can generalize the idea to other usage scenarios. To support developers of such 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 one important operation provided by the framework called the *proximity resolution*, described by the question "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 highly scalable and accessible for mobile users.

KEYWORDS

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

1. INTRODUCTION

Applications or services that take into account the current location will become increasingly popular in the future. Especially mobile phone providers expect a huge market for such services [24]. Typical applications respond to queries like “Where is the nearest hotel?” or “Which of my friends is in proximity?” We call services that provide information about the nearer area of a mobile user *proximity services*. Proximity services could in principle run on a huge database storing all location information. We go into another direction and use a decentralized approach, which provides access to location information in two steps: first, we ask our framework for location *names* in the nearer area that fulfil certain formal criteria. As we use a well-defined name space, these names can in a second step be used to query relational databases or name services to get the required information.

To hide location capturing, naming and distribution issues from end-users and service developers, we created the Nimbus framework. With Nimbus, mobile users can switch between a variety of positioning systems, such as satellite navigation, positioning systems based on cell-phone infrastructures, or indoor positioning systems. A developer can concentrate on the actual service function and does not have to deal with positioning sensors, mapping and resolution algorithms.

2. RELATED WORK

Currently, location-based services, especially proximity services are a domain of mobile phone networks. Typical phone applications answer proximity questions about service points (e.g., gas stations, hospitals) or

persons (e.g. friends) in the nearer area with the help of SMS or WAP technology. The first marketable service platforms come from mobile phone providers. Services such as Nightguide or Loco Guide [26] serve as location-based information portals. Such services reach a huge number of users, but they have very coarse-grained tracking capabilities based on the GSM cell information.

The research community developed many location-based services and platforms over the last years. Tourist information systems are ideal examples for such services. CYBERGUIDE [1] and GUIDE [5] offer information to tourists, taking into account their current location. Usually, such systems are accompanied by a general development framework, which allows a developer to create other location-aware applications. Several frameworks deal with location data and provide a platform for location-based applications. Leonhardt describes a conceptual approach to handle multi-sensor input from different positioning systems [11]. 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.

Geographic information systems (GIS) and spatial databases provide powerful mechanisms to store and retrieve location data [23]. 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-based 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, top):

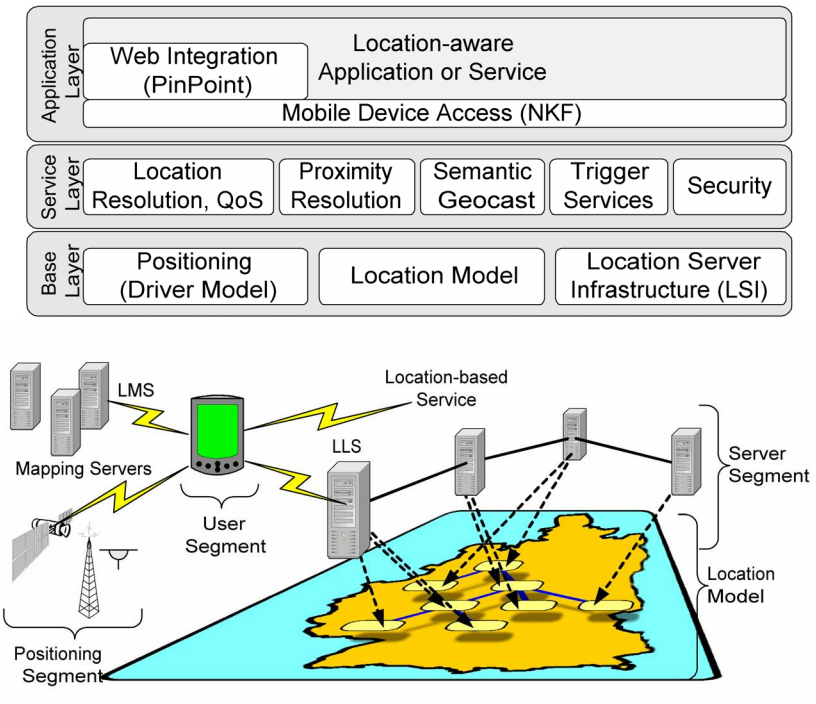

Fig. 1. The Nimbus framework (top) and the runtime infrastructure (bottom)

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* (*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 services are 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 [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 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 allows the usage of existing components such as Web browsers and Web servers without modification. As an example application, we developed a Web-based tourist guide with PinPoint.

Fig. 1 (bottom) shows the distributed runtime infrastructure which consists of three segments:

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 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 location data. Each location server is responsible for a specific part of the world's location data. We structure locations using *hierarchies* and *domains* (see below). 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.

3.1 The Nimbus Location Model

The Nimbus location model strongly considered a decentralized storage. Especially the resolution operations should be executed efficiently in a distributed federation of individual servers. 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 Lisbon" and semantic locations such as "Paul'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: first, semantic locations have a meaning to the user; in contrast, physical locations usually have no meaning at all to most people. Second, 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.

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 cover locations with similar meanings. One hierarchy could contain locations with a political meaning (e.g. states and cities); another could contain geographical locations (lakes and mountains). 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). 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` and `B`. 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.

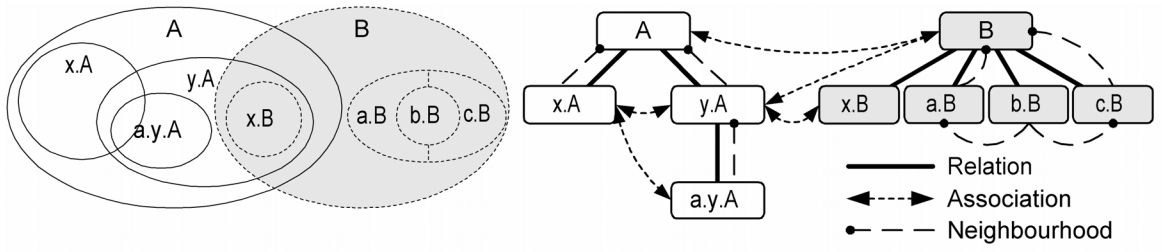

Fig. 2. Sample hierarchies

Domain names follow the Domain Name System of the Internet: 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.

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.

Of course, there are more links conceivable between domains. We could, e.g. link two domains, if a street or a subway line connects them. 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.

Each location server stores one or more domains. More precisely: a location server is responsible for a specific domain and all subdomains, for which no other location server exists. Domain administrators can map domains to location servers in a very fine-grained manner, thus they have an important tool for load balancing. In the Nimbus framework, we implemented algorithms that search the masters, subdomains, associated domains and neighbours automatically. This is done by a decentralized lookup procedure described in [20].

3.2 Performing the proximity search

Fig. 3 illustrates the two important search operations (called *resolutions*). The figure contains three two-dimensional hierarchies `A`, `B`, and `C`. We use the third dimension to represent the different hierarchies. The two resolution operations are:

- Given a location *p*. What domains contain *p*? (*location resolution*, fig. 3 left)
- Given a location *p*. What domains are inside a certain circle around *p*? (*proximity resolution*, fig. 3. right)

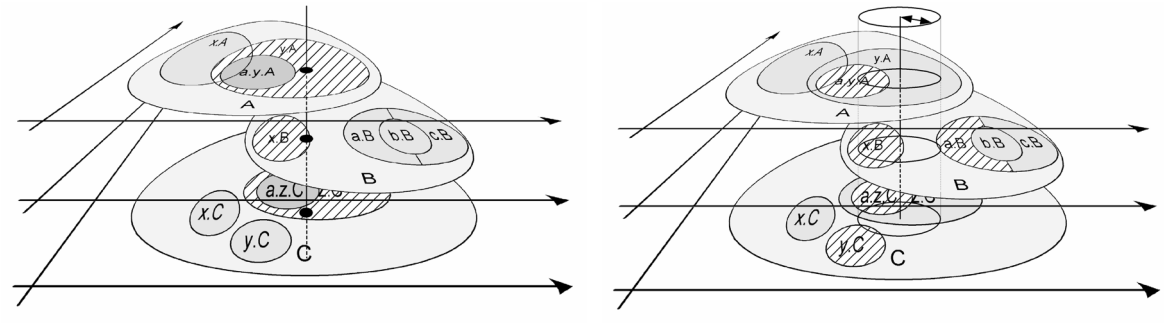

Fig 3. Resolution operations: location resolution (left), proximity resolution (right)

We are only interested in the most specific domains. E.g. if p resides in domain `y.A`, it is of course also inside the domain `A`, thus the solution `A` would not carry additional information. The proximity resolution is more complex than the location resolution as usually multiple domains (and thus servers) in one hierarchy are involved. More over, if we can carry out the proximity resolution, the location resolution is only a special case (i.e. circle with radius 0).

The basic idea of proximity resolution is to search all local domains and their neighbours in order to find semantic domains that match certain criteria. The algorithm is required to perform this search automatically and efficiently. The following definitions are used:

- A *match* is a semantic domain that fulfils a criterion. It is assumed that a filter using the semantic name of the domain can perform the test. 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. 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 filter also tests the master domains of the domain.
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.

We created a formalized representation of this algorithm and conducted a proof of correctness. The complete formalism and proof cannot be shown due to space considerations but is available from the authors. In addition, we estimated bandwidth and memory usage. We run several simulations 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 (up to 1.2 GHz CPUs and up to 512 MB RAM), each of it storing up to 200 sets of domain data. 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.

3.3 Acquiring and processing geographical data

In principle, there are three ways to acquire domain data:

- We could use an existing database of geographical data and convert it to our domain data format.
- We could create our own domain data.
- The infrastructure could learn domain data from mobile users.

Currently, we only practise the first two ways and currently investigate the potentials of the third one. Fig. 4 presents hierarchies built using an existing dataset. These definitions have been extracted from the TIGER/Line [22] dataset provided by 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. 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. We used this data to verify our algorithms and implementations. Note that for clearness, we do not present the associations and neighbourhood links in this figure.

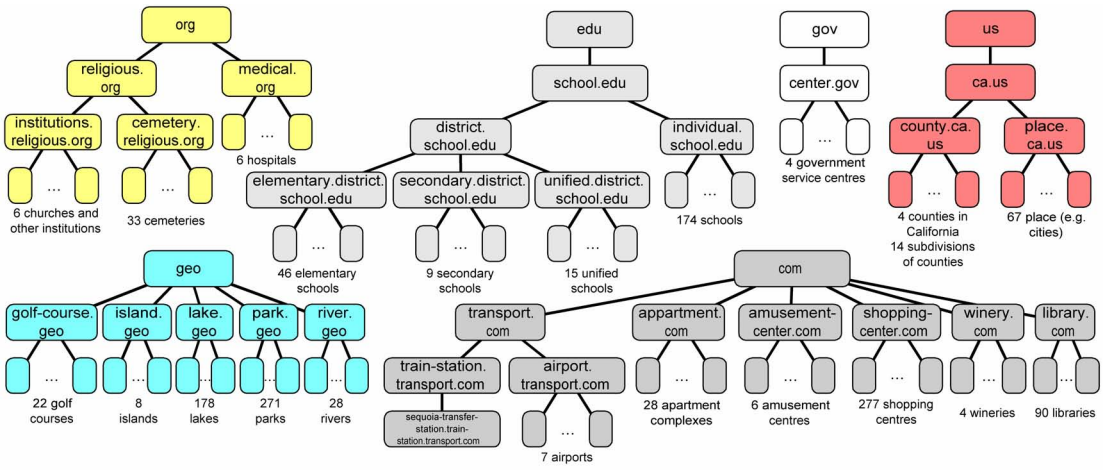

Fig 4. Overview of the domain definitions extracted from the TIGER/Line files

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. We store height data with the help of a 2.5D representation. Fig. 5 (right) shows a height profile of a city produced by a land survey office [10]. Height profiles contain a number of reference points, in, e.g., a grid of 10m. We developed an automated input procedure to process such height data and to store it in our domain data structure.

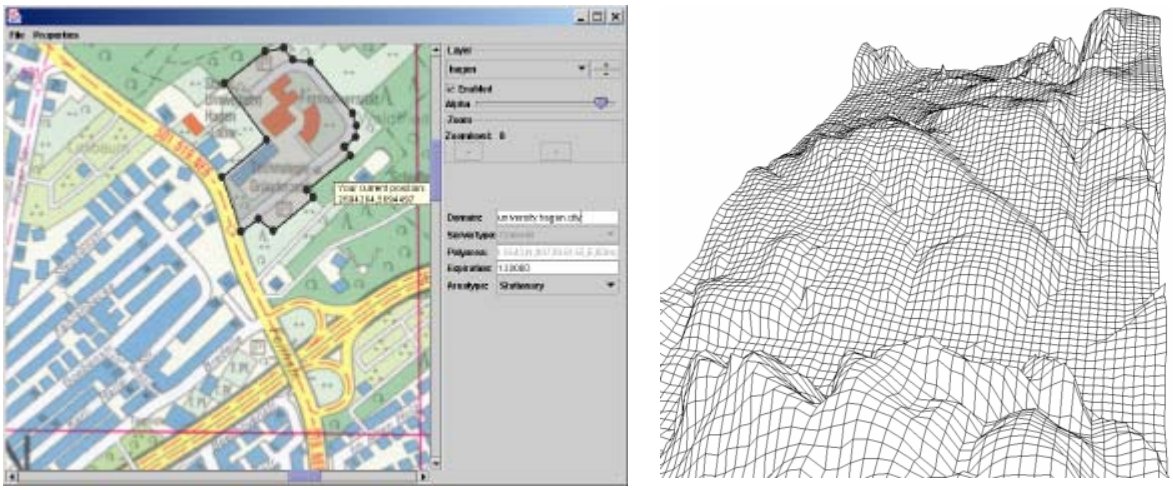

Fig. 5. The Domain Editor (left) and a height profile (right)

Fig. 5 (left) illustrates the second way to create domains: we can use our domain editor to create and edit domain data files. The user can create a polygonal domain definition with the help of a map (satellite or street map). In addition, height data of the corresponding area can be loaded to the domain to get a 2.5D representation.

We store domain information using XML files in which the most important entry is the polygon specifying the area and the height representation. To process polygonal data, we avoid heavyweight geo databases and use instead a lightweight toolkit to process polygonal data [25]. The toolkit handles all geometric operations in the runtime memory and can quickly check, if a point is inside or outside a polygon.

3.4 Future Work

Several extensions of the framework are planed. One important requirement in real usage would be to instantly set up new domains. For this, we investigate mechanisms to learn domain information from mobile users. In addition, we consider domains with limited access which end-users can check into the framework.

As a second extension, we want to consider mobile domains. Currently, we assume domains to have a long-term character and do not change too much over time. Some real-life domains such as trains or ships are mobile and permanently change their physical area. Such domains currently lead to high communication traffic. We are thus looking for new algorithms to handle mobile domains.

As a third direction, we want to introduce new notions of distances. Currently we only support the physical distance. Other *semantic* distances are conceivable, e.g. the time required to get to a certain domain by public transportation. These distances can no longer be computed by geometric operations on positions but have to be added to the location server configuration files.

4. CONCLUSION

In this paper, we presented a framework designed for mobile users accessing location information. We introduced a proximity search operation which provides a lookup of locations that fulfil certain criteria. Developers of proximity services 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 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

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, LNCS 2877, 216-228
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. 108th Congressional Districts Census 2000 TIGER/Line Files, http://www.census.gov/ geo/www/tiger/index.html, U.S. Census Bureau, 2000
23. Tomlin, C., D.: Geographic Information Systems and Cartographic Modelling, Prentice Hall, 1990
24. UMTS Forum, Enabling UMTS/Third Generation Services and Applications, Report 11, http://www.umts-forum.org, Oct. 2000
25. Vivid Solutions, JTS Technical Specifications, http://www.vividsolutions.com, March 31, 2003
26. Vodafone Homepage, www.vodafone.com, 2003
27. 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