Detecting Identifiable Areas in Mobile Environments

Jörg Roth
University of Hagen
Department for Computer Science
58084 Hagen, Germany
+49 2331/987-2134

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. To support developers of location-based services, the Nimbus framework hides specific details of positioning systems and provides uniform output containing physical as well as symbolic location information, which often are more suitable for applications and users. One basic problem solved by Nimbus is the Distributed Location Resolution Problem: given a location; which identifiable areas cover this location, if the area information is distributed among different servers. This paper presents a solution of this problem that runs in the distributed, self-organizing Nimbus infrastructure. The effectiveness is demonstrated with the help of performance analyses.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *distributed systems*

General Terms

Algorithms, Measurement, Performance

Keywords

Location-based service, symbolic location, self-organizing infrastructure

1. INTRODUCTION

The answer to the question "*Where am I?*" is becoming increasingly important for mobile users. In the future, it will not only be sufficient to know the coordinates of the current location, but also to know information *about* the current location. Typical questions will be: "*Where is the nearest hotel?*" or "*Which bus takes me to the railway station?*".

Location-based services often need *symbolic* location information, but most positioning systems only provide *physical* ones. In this paper we present the Nimbus platform, which provides a solution for the *distributed location resolution problem*: given a location and a set of *distributed* computers, each storing a certain set of *identifiable areas*; which areas contain this location? Nimbus introduces a naming scheme and classifies these areas according to hierarchies. If an application gets well-defined area names for a certain location it can easily lookup location-based information in e.g. databases or file systems. The following example motivates the benefits of this approach.

Fig. 1 presents a location-based bus planner, developed with the Nimbus framework. The user simply selects the destination and the application computes the appropriate timetable for the selected destination, while taking into account the current time and location. This application is not primarily interested in the physical coordinates, but wants to know at which bus station the user is waiting. As the bus stations' locations are stored decentrally (e.g. inside different bus station registers for the respective city and the specific bus company), we have to look up these data in a distributed manner.

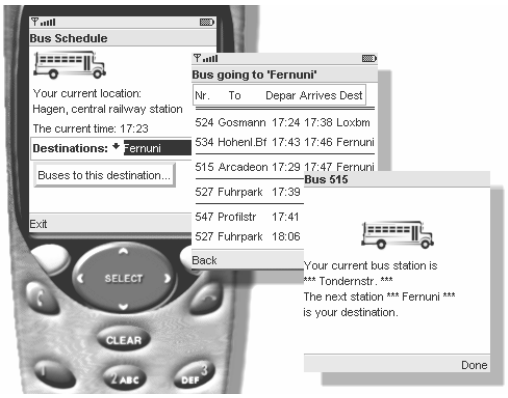

Figure 1. The Nimbus bus planner

Areas such as bus stations, offices, streets, forests, city centers etc. are often called *symbolic* or *semantic locations* [9, 11, 14]. Some people expect a lot of semantics and intelligence behind these words, even though the term semantic location in its original meaning only describes an area that has a certain identity for a user or application. Thus, in Nimbus we use the term *identifiable area* instead. Such an area can uniquely be identified by a *name*. The naming scheme provides a rough classification of the area, but the actual semantics behind it is only perceptible inside the respective application.

In principle, one huge database on a single server could store all identifiable areas, but these are typical performance bottlenecks. Moreover, mobile users want to access location data locally and not over potential far distances. Finally, information about local areas is usually available locally and is difficult to administrate in a central database. As a solution, Nimbus provides a distributed system of *Location Servers* that store the information about identifiable areas in a decentralized manner. Location servers are connected to each other in a peer-to-peer style.

SAC'06, April, 23-27, 2006, Dijon, France.

2. RELATED WORK

Several frameworks deal with location data and provide a platform for location-based application. The *Open Geospatial Consortium* [10] provides a high-level framework to build location-based services. *Cooltown* [7] 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.

Another platform that supports developers of location-based services is *LORE* [2]. It contains three key components: a location server providing a location sensing API, a moving object database and a spatial publish/subscription engine. All components are built according a traditional client/server architecture.

Nexus [6] introduces so-called *augmented areas* to formalize location information. They represent spatially limited areas, which may contain real as well as virtual objects, where the latter can only be modified through the Nexus system. Different locations can be modeled in the so-called *augmented world model*: static objects like houses or streets, mobile objects such as users or cars and virtual objects such as virtual post-its or virtual kiosks.

A number of platforms focus on location sensing. The *Location Stack* [5] provides a layered approach to access multiple positioning systems. Lower layer contain the low-level hardware, e.g. GPS pseudo range or ultrasound time measurements. The upper layer combine low-level sensor data, fuse multiple location information and perform probabilistic reasoning about relationships between multiple objects.

A similar approach follows the *POS* component inside the *PoLoS* project [12]. It is responsible for the integration of positioning systems into the platform. Specific positioning systems are modeled using wrappers. Even though platforms such as Location Stack and PoLoS simplify the process of location sensing, they do not consider the distributed nature of information *about* locations and focus on the location sensing and fusion function.

Geographic information systems (*GIS*) and spatial databases provide powerful mechanisms to store and retrieve location data [15]. Such systems primarily concentrate on accessing large amounts of spatial data. In the 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.

Even though, some of the platforms above provide services to discover identifiable areas, they do not solve the DLRP in its general meaning. E.g. in Nexus, a requesting client has to discover the appropriate server for the specific area via a *central* register. Requests concerning multiple servers have to be carried out one after the other. LORE is built according to a fully centralized client/server architecture and does not support any distributed storage of location information. In the following, we describe a fully distributed infrastructure that efficiently solves the DLRP without any central instance.

3. NIMBUS

The Nimbus framework supports developers of location-aware applications. The development is simplified with the help of general services. The primary goal is to provide the most enriched location information including identifiable areas. Once the enriched location information is delivered to the application, the mobile client can use arbitrary mechanisms to execute the location-based service. Established mechanisms such as databases, component services, web services or simple socket connections can be used to get further information about the location (fig. 2).

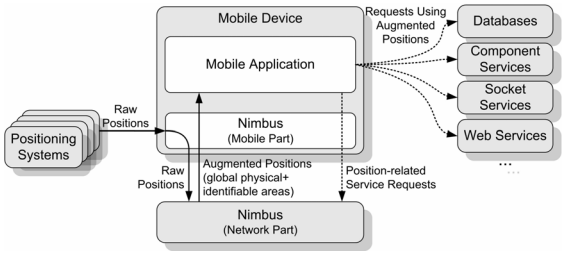

Figure 2. Data flow with the Nimbus framework

The Nimbus framework has a mobile part installed on the mobile system and a network part providing information about the location. The mobile part takes raw positions and requests the network part to provide an augmentation. In principle, different physical coordinate systems could be used simultaneously, but Nimbus maps any local coordinates to WGS 84 coordinates [3] in order to use globally unique locations. In addition, the mobile client receives identifiable area information, which now can easily be processed by the application. As the areas are identified using a predefined name scheme, they can easily be used for direct queries in database tables and lists and can be parts of file names. Physical locations are still useful for all kinds of geometric queries, e.g. asking for distances between locations or asking for directions.

Besides the main function of augmenting locations, Nimbus provides several services strongly related to location-based services such as geocasting services not presented in this paper.

3.1 The Nimbus Location Model

Even though the formal location model is defined independently of the later implementation, a decentralized storage of location data in the later infrastructure is considered. We want to describe the concept of identifiable areas in Nimbus more precisely. Let P denote the set of all physical locations inside the observed area. A reasonable set P could be the set of all WGS 84 coordinates from a certain depth below the earth's surface up to a certain altitude in the atmosphere.

We call an area $S \subseteq P$ with a certain meaning an *identifiable area* of P. Let $C \subset 2^P$ be the set of all identifiable areas. To identify areas not only by their geometry, they have a *name*. An area with its corresponding name is called a *domain*. For a domain d, $d.name$ denotes the domain name, $d.c$ the domain geometry. Further D denotes the set of all domains.

Names of identifiable areas could in principle be human-understandable, but there is no *single* human-understandable representation which is suitable for *all* conceivable applications. Thus, the area names are optimized for the internal Nimbus operations. Once the name space is clearly defined, transforming a name to its final representation often means simply looking up a string inside a table.

In principle, C could be an arbitrary subset of 2^P. Looking at real-world scenarios, however, space is often structured hierarchically, e.g. a room is inside a building, a building is in a city and a city is

in a country. Thus, C is divided into *hierarchies*. A hierarchy contains domains with a similar meaning, e.g. domains of cities or geographical domains. Each hierarchy has a *root domain* and a number of *subdomains*; each of them can in turn be divided into subdomains. A top node of a subhierarchy is called a *master* of the corresponding subdomains. A link between a subdomain and its master is called a *relation* (denoted *master* $\triangleright$ *subdomain*). Names of subdomains are built similar to Internet host names, i.e. `<sub>.<master>`. Fig. 3 presents sample hierarchies.

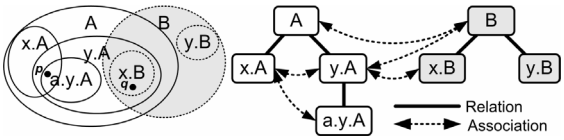

Figure 3. Example hierarchies

As the names contain all higher-level names, domains can be uniquely identified, even though the corresponding real-world objects are named equally (e.g. a monument and street with the same name).

3.2 The Distributed Location Resolution Problem

As our initial goal, we want to solve the *distributed location resolution problem* (*DLRP*):

> Given a location p and a set of distributed computers, each storing a certain set of identifiable areas $d_i.c$. Which areas (identified by their name $d_i.name$) contain p?

E.g., in fig. 3 point p resides in the domains `A`, `x.A`, `y.A` and `a.y.A`. Since a master always fully encloses a subdomain, the results `A` and `y.A` do not carry additional information. A sufficient answer to the question above is "`x.A` and `a.y.A`".

Answering such a question, could be performed by browsing through all hierarchies, from the root down to the smallest domains containing p. This, however, would cause a large number of requests and a considerable amount of network traffic in distributed infrastructures. Therefore, a second relationship between domains is introduced, the *association*. Two domains d_1, d_2 are associated, denoted $d_1 \sim d_2$, if they share an area and are not related (related domains always share an area). With associations, the question above can be answered: if one domain d_0 is available that contains the location p, then only the domains $d \sim d_0$ can potentially contain p. In turn, no other domains have to be checked and thus the time-consuming search through all of the hierarchies can be avoided.

The number of candidates can be reduced even more because applications are only interested in the most specific domains. If, in the example above, an application wants to know which domains contain the point q, it is only interested in the domains `y.A` and `x.B`, and not in `A` or `B`, even though their areas overlap. Associations thus only link two domains if their shared area is not fully covered by their respective subdomains, more formally:

$$\left(d_1.c \setminus \bigcup_{e_1 \in D, d_1 \triangleright e_1} e_1.c\right) \cap \left(d_2.c \setminus \bigcup_{e_2 \in D, d_2 \triangleright e_2} e_2.c\right) \neq \{\} \qquad (1)$$

With this formula we get a definition of association. We introduce the abbreviation

$$\triangle(d) = d.c \setminus \bigcup_{e \in D, d \triangleright e} e.c \qquad (2)$$

for the domain's area excluding the subdomains' area and finally get the short definition

$$d_1 \sim d_2 :\Leftrightarrow \triangle(d_1) \cap \triangle(d_2) \neq \{\}. \qquad (3)$$

After these preparations, we now could sketch a solution for the DLRP:

```
DLRP(p) → set of names {
    look up a domain d0 with p ∈ △(d0)
    names ← {d0.name}
    for all d ~ d0 do
        if p ∈ △(d)
            names ← names ∪ {d.name}
    return names
}
```

If an arbitrary domain d_0 is available that fulfils the first condition, the algorithm can efficiently loop through the associated domains. Note that the algorithm assumes that at every conceivable location, at least one domain d_0 is available that covers this location. Thus, a full coverage of domains in the intended service area is required.

We conducted a formal proof of correctness of this algorithm. The proof cannot be shown due to space considerations but is available from the author.

3.3 The Infrastructural Perspective

Each location server can store a certain amount of domains, called a *cluster*. Associations and relations can be defined for clusters and thus for servers in a canonical manner: two clusters are related (associated) if there exist domains from each cluster that are related (associated). The relation between clusters forms the so-called *cluster tree*.

Fig. 4 shows the infrastructure based on the storage of clusters. This example uses more realistic hierarchies imported from land survey offices (see section 4).

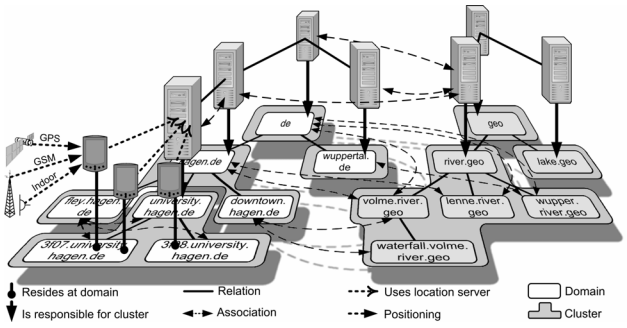

Figure 4. The distributed Nimbus infrastructure

To reduce the communication between the servers, they proactively update the sets of associations and the sets $\triangle(d)$. As a result, all information required by the algorithm above is stored locally on each server, also by the server that contains d_0. A client thus has only to discover that server (called the *Local Location Server*, *LLS*), which then can execute the algorithm's main loop.

As mobile users are distributed among different location servers, this infrastructure is highly scalable. One can add a server and

restructure the respective hierarchy, thus it is possible to add server power to the infrastructure in a simple way.

Two different types of lookup are required. First, mobile clients have to look up their LLS. In addition, as the federation of location servers is self-organizing, location servers have to look up other related and associated location servers. Nimbus uses a peer-to-peer-like lookup procedure not presented in this paper. It is based on a combination of a routing algorithm to redirect lookup messages and a broadcast search. A mechanism to avoid reply storms is integrated.

3.4 Caching

The clients store results of former resolution requests in a cache. Before performing a new request over the network, the client first checks, if any key corresponds to the new request. To take into account that domains may change over time, each cache entry has a certain expiration time after which a cache entry is removed.

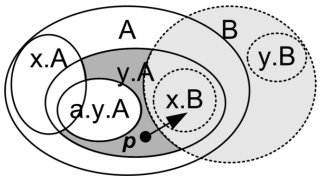

Figure 5. Cache example

It is not reasonable to use p as a key for the cache as it is not likely to have exactly the same location p for a second resolution. Fig. 5 illustrates a further issue. A user resides at location p and gets the area `y.A`. He then follows a path to a location inside `x.B`. A naive implementation would use $\triangle$(`y.A`) as the cache key. As long as the user resides in the grey area, the cache resolution produces the right result. However, when entering the area inside `B` and being still in $\triangle$(`y.A`), he would get a cache hit and again the result `y.A`. This is only partly true, as we additionally have to consider the second area `B` or `x.B` as a result. The correct solution to produce cache entries is more complex. Let

$$\blacklozenge(p) = \{q \in P \mid DLRP(q) = DLRP(p)\} \tag{4}$$

denote the area of same resolution results as p. Then we get the following formula to compute a cache key:

$$\blacklozenge(p) = \bigcap_{\substack{e \in D: e \sim d_0 \vee e = d_0 \\ p \in \Delta(e)}} \Delta(e) \setminus \bigcup_{\substack{f \in D: f \sim d_0 \\ p \notin \Delta(f)}} \Delta(f) \tag{5}$$

for a domain d_0 with $d_0.name \in DLRP(p)$. Note that the location model requires at least one domain that covers each conceivable location, thus a domain d_0 exists. Further note that all involved domain areas can easily be collected by the resolution algorithm, thus the computation of $\blacklozenge(p)$ does not lead to more network queries. Moreover, once a domain e or f is collected, the pairs ($e.name$, $\triangle(e)$) or ($f.name$, $\triangle(f)$) respectively can be stored inside a second cache to build future $\blacklozenge$ areas. A formal proof of correctness for equation (5) is available from the author.

3.5 From Points to Area Locations

The algorithm in section 3.2 is based on a simplification: positioning systems provide a single point in space as location output, i.e. a location is presented as a point $p \in P$. For many positioning systems in mobile scenarios this is not adequate.

Location measurements usually have a certain accuracy, thus the actual position can be inside a certain area around the measured location. Advanced probabilistic approaches [1, 4] model locations by a statistical distribution of measurements. For two dimensions we get a probability distribution for a certain measurement.

Probabilistic approaches have some disadvantages: They usually assume a normal distribution of sensor values, but systems based on cells can only specify the location by the cell area with a uniform distribution [13]. Moreover, the numerical approximation of the location probability and appropriate resolution operations for weak clients is difficult and requires huge storage space.

As a solution, Nimbus uses an area model to describe location sensor output. A specific location output provided by a positioning system is modeled by an area $A \subset P$, where the user's current location is anywhere inside A. From a strict point of view, the corresponding A to a measurement would often be very large and thus meaningless, as even accurate positioning systems sometimes produce very inaccurate measurements. To get a useful representation, the Nimbus sensor model uses an area A which defines the area of most probable locations (e.g. a 95% probability). The definition of A primarily depends on the accuracy of the system. As an advantage, an area is a very compact representation of a location sensor output. In addition, this model leads to very efficient processing algorithms.

Using the area sensor model, the resolution algorithm has to be modified. As we cannot determine the user's location exactly, we only compute a *probability* for a user to be inside a specific domain. The only definite statement about the user's location in fig. 6 is that he or she resides in the domain `y.A`. For other domains we can only declare degrees of residing in these domains. E.g., there is a 50% probability that the user is located in domain `B`.

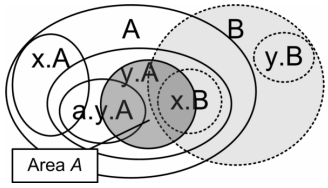

Figure 6. Location example

The probability $pr(A, d)$ that the actual location is located in a domain d is

$$pr(A,d) = \frac{vol(A \cap d.c)}{vol(A)} \tag{6}$$

where *vol* denotes the size of the surface or volume respectively. The formula assumes a uniform distribution of possible locations over A as requested by the location sensor model. In principle, we are also interested in an alternative probability

$$pr_{\Delta}(A,d) = \frac{vol(A \cap \Delta(d))}{vol(A)} \tag{7}$$

expressing the probability to reside in the $\triangle$ area of d. It is closer to the intention of the point variation of the resolution algorithm, as areas of subdomains are not counted in higher-level domains. Using $pr_{\triangle}(A, d)$ on the other hand, there may be no domain with 100% probability, even though the location is enclosed in a domain. E.g., in fig. 6 there is no domain with $pr_{\triangle}(A, d) = 100\%$. In contrast $pr(A,$ `y.A`$) = 100\%$ is a useful information.

An algorithm that produces a list (*d.name*, *pr*(*A*, *d*)) with *pr*(*A*, *d*)>0 can be outlined as follows:

```
DLRP(A) → table of (name, ratio) {
   initialize table nameToRatio()
   while A ≠ {}    {
      choose a p ∈ A   // any p in A will do
      n ← DLRP(p)
      add names of superior domains to n // ✪
                                        // remove to get pr△
      r ← vol(A ∩ ♦(p)) / vol(A)
      for each name ∈ n
         add or assign r to nameToRatio(name)
      A ← A \ ♦(p)
   }
   return nameToRatio
}
```

The algorithm works as follows:

- It starts with a complete area *A* and performs a resolution for an arbitrary point *p* inside *A* with the help of the existing point variation of the algorithm.
- We get a partial result. The area around *p* that leads to the same result can be removed from *A*. This area is obviously the area ♦(*p*) introduced for the caching mechanism.

If the new *A* is not empty, the algorithm chooses a new point *p* and proceeds in the same way. This algorithm computes values *pr*(*A*, *d*). We can get a list of the $pr_{\triangle}(A, d)$, if we remove statement ✪, as then only the resolved domains and not any superior domains are considered.

Again, a formal proof of correctness of the algorithm above is available from the author.

One could argue that the steps shown above lead to many transactions over the network. The efficiency of this algorithm can be motivated as follows:

- Often, the area *A* is small compared to the size of the domains (think of GPS positioning). In many cases, the result of a single iteration fully covers *A*.
- Once resolved, a certain movement of *A* is allowed without any further network transactions. Very often, the same ♦ areas are involved and the algorithm has only to adapt the ratio values.
- When *A* leaves the area that can be fully processed by cached areas, often only a single further query is required.

The area variation of the resolution can also be used for proximity requests. A user, e.g., may want to know which hotels are inside a circle of 5 km around the current location. With the so-called geometric *buffer* operation, the current location area *A* can be 'blown up' the respective search area.

4. EVALUATIONS

The efficiency of the Nimbus mechanisms is evaluated with the help of several simulations which use real imported data from the German land survey office. In Germany, the ATKIS database provides geospatial data for governmental purposes [8]. For the Nimbus project, data covering an area of 84 km^2 was purchased. It contains approx. 80000 raw database records which were converted to approx. 8000 Nimbus domains. The data set contains geographic objects, nature and agriculture, industrial areas and plants, inhabited areas, infrastructure and traffic. The resulting data structure contains three hierarchies: A `de` hierarchy contains cities, parts of cities, properties, sites and buildings. A `geo` hierarchy contains domains with a geographical meaning, currently rivers and lakes. Finally, a `traffic` hierarchy contains roads and railways.

4.1 Real Network Measurements

The first measurements consider several clients simultaneously requesting resolutions over the network. All caches were switched off. Once receiving a resolution result, the next request is sent without any delay. This causes a high load to the servers (2.8 GHz, Windows XP).

The experiments are conducted for 1 to 3 servers (fig. 7). As expected, the approach is scalable: the more location servers are used, the more the curve is flat. Whenever in reality a location server is overloaded, new subservers can easily be introduced.

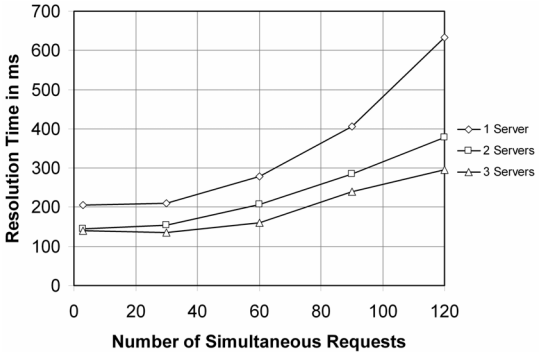

Figure 7. Resolution time for simultaneous requests

It is important to note that in reality a much higher number of users per location server is needed to cause a specific load as clients use a cache and usually do not permanently perform resolution requests. Typical applications allow 10-20 times more users to impose the same load as presented in fig. 7.

4.2 Cache Efficiency

Further simulations measure the cache efficiency, assuming that a mobile user permanently moves with a specific speed on random paths. Real paths such as streets are not considered, thus the simulated paths are more disordered than in reality. The mobile user performs a location resolution (point variation) every second. The cache contains only one single entry which stores the current ♦ area. This is worse than in reality, where a user can cache several areas, but it provides a meaningful worst case estimation.

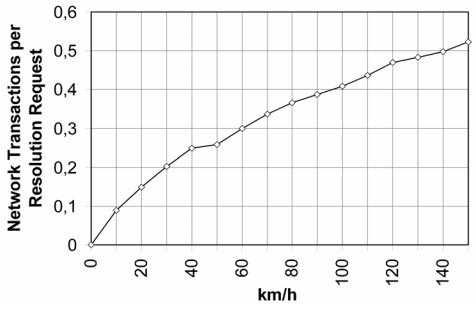

Figure 8. Number of network transactions per request

Fig. 8 shows the result: the cache provides reasonable values even for speeds in the range of a car. The author is fully aware that the results only indicate a trend as they highly depend on the size of ♦ areas and thus on the specific domain data.

It is important to note that in reality the cache will be far more efficient. Consider tourists visiting a city: they may visit the same areas several times (e.g., the city centre, their hotel). Thus after initially loading the ♦ areas and respective △ areas, the tourists can work off line for a certain time.

4.3 Area Resolution

The next simulation performed multiple area resolution requests for circular areas with a specific radius. We measured the cache efficiency related to the area resolution. It is based on similar settings as in the cache measurements above. The cache only stores those ♦ areas which were the result of the last area resolution – this is again a reduction of the real expected results.

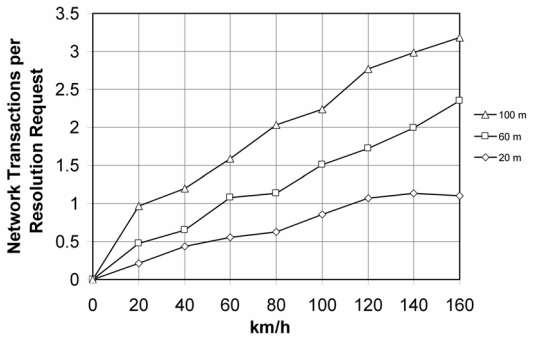

Figure 9. Transactions per area resolution request

Fig. 9 shows the results. For the size of 20 m (approx. the GPS accuracy) we have about one network resolution even for higher speeds. For the speed of a pedestrian, we have less than 10% network resolutions. Larger areas need more requests, but as a promising result the average amount of 3 network transactions for a 100 m area and 150 km/h is still low. To sum up all results of our experiments:.The approach is scalable, especially servers are not overloaded; and mechanisms to avoid traffic from mobile clients over potentially weak wireless connections are effective.

5. CONCLUSION AND FUTURE WORK

The main goal of Nimbus is to offer enriched location information that is easy to process. Nimbus provides globally unique physical locations, independently of the underlying positioning systems and identifiable areas according to a predefined name space. Nimbus reached this goal with a distributed, self-organizing infrastructure, easily to be administrated, designed according to scalability issues. Performance evaluations show the efficiency and scalability of the approach.

Even though Nimbus reached a high level of completeness, there are some open issues. In the current implementation, Nimbus stores domains with a public character. Every user can access all domains as, e.g., 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.

A second issue results in mobile domains: trains or ships, e.g., permanently change their location. In principle, Nimbus supports such domains, but such domains 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. REFERENCES

[1] Burgard, W.; Fox, D.; Hennig, D.; Schmidt, T.: Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids, *IAAI*, Vol. 2, 1996, 896-901

[2] Chen, Y.; Chen, X. Y.; Rao, F. Y.; Yu, X. L.; Li, Y.; Liu, D.: LORE: An infrastructure to support location-aware services, *IBM Journal of Research & Development*, Vol. 48, No 5/6, Sept. 2004, 601-615

[3] EUROCONTROL European Organization for the Safety of Air Navigation: *WGS 84 - Implementation Manual*, Brussels, Belgium, Febr. 1998

[4] Fox, D.; Burghard, W.; Dellaert, F.; Thrun, S.: Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, *Proc. of the 16th National Conf. on AI (AAAI)*, Orlando, USA, 343-349, 1999

[5] Hightower, J.; Brumitt, B.; Borriello, G.: The location stack: A layered model for location in ubiquitous computing, in *proc. of the 4th IEEE Workshop on mobile Computing Systems & Applications (WMCSA 2002)*, Callicoon, NY, USA, June 2002. IEEE Computer Society Press, 22-28

[6] 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

[7] 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

[8] Land Survey Office North Rhine-Westphalia, http://www.lverma.nrw.de

[9] Leonhardt, U.: *Supporting Location-Awareness in Open Distributed Systems*, PhD Thesis, University of London, 1998

[10] Open Geospatial Consortium, *OGC Home Page*, http://www.opengeospatial.org/

[11] Pradhan, S.: Semantic Location, *Personal Technologies*, Vol. 4, No. 4, 2000, 213-216

[12] Prigouris, N.; Papazafeiropoulos G.; Marias, I.; Hadjiefthymiades S.;Merakos, L.: Building a Generic Broker for Location Retrieval, *Proc. of the IST Mobile & Wireless Communications Summit*, Portugal, June 2003

[13] Roth, J: Data Collection, in Jochen Schiller, Agnès Voisard (eds), *Location-Based Services*, Morgan Kaufmann Publishers, May 2004

[14] Schilit, B.; Adams, N.; Want, R.: Context-Aware Computing Applications, *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, 1994

[15] Tomlin, C., D.: *Geographic Information Systems and Cartrographic Modeling*, Prentice Hall, 1990