Die HomeRun-Plattform für ortsbezogene Dienste außerhalb des Massenmarktes

Jörg Roth

Ohm-Hochschule Nürnberg, Joerg.Roth@Ohm-hochschule.de

Zusammenfassung

Ortsbezogene Dienste außerhalb des Massenmarktens haben spezielle Anforderungen und werden noch nicht hinreichend durch Rahmenwerke unterstützt. Die HomeRun-Plattform soll diese Lücke schließen. In diesem Papier wird die HomeRun-Komponente zur Speicherung von Geodaten beschrieben. Für die Verwaltung großer Mengen geographischer Daten setzen sich zwar zunehmend Datenbankerweiterungen für relationale Datenbanken durch; aufgrund der besonderen Rolle geometrischer und geographischer Datenstrukturen muss der Zugriff einer Anwendung auf Datenbankfunktionen jedoch signifikant erweitert werden. In HomeRun ist ein Ansatz realisiert, bei dem die räumliche Erweiterung nicht als Bestandteil der Datenbank verstanden wird, sondern als zusätzliche Software-Bibliothek zur Anwendung hinzugebunden wird. Sie kann dabei auf einer beliebigen Standard-Datenbank operieren. Neben einem geeigneten räumlichen Index zur Beschleunigung der Abfragen, wird eine objektorientierte Schnittstelle zur Nutzung der räumlichen Funktionen angeboten.

Schlüsselwörter: *Dienst-Plattform, räumliche Datenbank, räumlicher Index*

1. Einleitung

Ortsbezogene Dienste sind in der Entwicklung noch sehr aufwändig. Unter anderem müssen die Fragen der mobilen Kommunikation, der mobilen Benutzungsschnittstelle, der Positionsbestimmung und der Verwaltung von Geodaten geklärt werden. Durch diese Komplexität etablierten sich ortsbezogene Dienst bisher vor allem auf dem Massenmarkt, insbesondere im Umfeld des Mobilfunks, da hier die hohen Entwicklungskosten eher gerechtfertigt sind. Darüber hinaus haben Mobilfunkbetreiber in der Vergangenheit nicht nur das Kommunikationsnetz, sondern auch die Technik zur Positionsbestimmung basierend auf Mobilfunkzellen kontrolliert. Durch nicht zu vernachlässigende Kosten für die Lokalisierung war es für Anbieter außerhalb des Mobilfunknetzes häufig nicht attraktiv, Dienste zu entwickeln und anzubieten.

Mittlerweile stehen zunehmend mobile Endgeräte zur Verfügung, die ihre Position autonom auf der Basis der Satellitennavigation ermitteln können. Das Mobilfunknetz kann daher als reines Zugriffsnetzwerk für Dienste außerhalb der Zuständigkeit des Mobilfunkanbieters betrachtet werden. Dadurch ist es für kleine Betreiber zunehmend interessant, als Anbieter für ortsbezogene Dienste in Erscheinung treten [5]. Die Eigenschaften solcher Dienstangebote im Gegensatz zu Massenmarktdiensten sind:

- Die Kosten müssen gering gehalten werden. Dies umfasst die Entwicklungskosten, Ausstattungskosten (z.B. für die Server-Infrastruktur) und Lizenzkosten.
- Demgegenüber ist die Nutzerzahl relativ gering.
- Es wird häufig spezialisiertes Kartenmaterial benötigt. Viele Dienste integrieren zudem Benutzer-generierte Geodaten.

Die HomeRun-Plattform soll die Entwicklung ortsbezogener Dienste außerhalb des Massenmarktes vereinfachen.

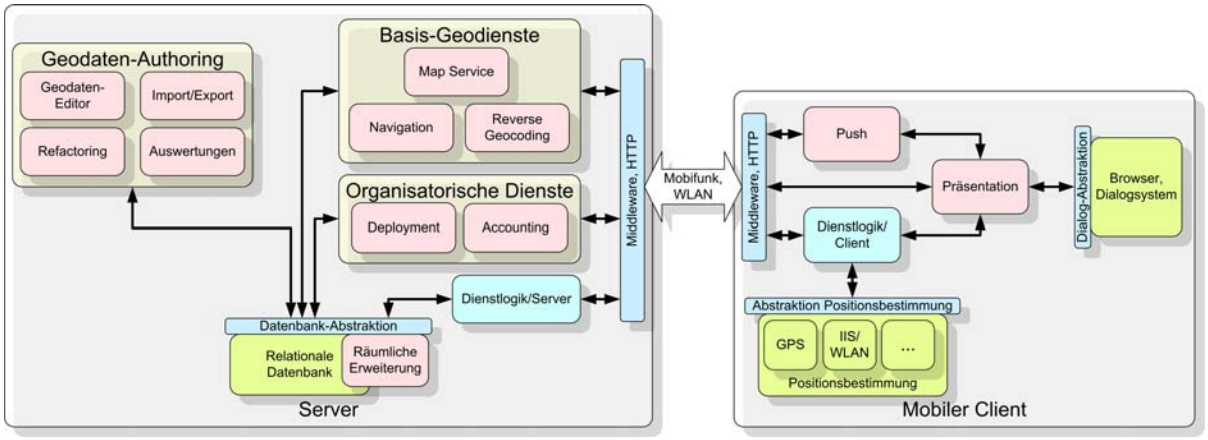

Abb.1: Die HomeRun-Architektur

Die Home-Architektur umfasst folgende Bausteine (Abb. 1):

- *Geodaten-Authoring*: hier können Geodaten aus verschiedenen Quellen importiert und fusioniert werden. Geodaten können darüber hinaus für den jeweiligen Dienst bearbeitet werden.
- *Datenbank mit räumlicher Erweiterung*: hier werden die Karten- und Benutzer-generierte Geodaten zur Laufzeit abgelegt und abgerufen.
- *Organisatorische Dienste* stellen Funktionen außerhalb des Ortsbezugs dar, z.B. Deployment und Accounting.
- *Basis-Geodienste* stellen Funktionen mit Ortsbezug (Kartendarstellung, reverse Geocoding, Navigation) zur Verfügung.
- Die *Client-Integration* umfasst die Kommunikation inkl. Push, die Anbindung an die Positionsbestimmung und Sensordatenfusion sowie die Präsentationslogik.

In diesem Papier soll die Komponente *Datenbank mit räumlicher Erweiterung* genauer beschrieben werden. Die Speicherung geometrischer und ortsbezogener Daten in relationalen Datenbanken stellt ohne spezielle Erweiterungen ein Problem dar, da Geometrien je nach Sichtweise, sowohl die Eigenschaft eines Spaltenattributs, als auch die Eigenschaft einer Relation besitzen. Als Lösung werden räumliche Erweiterung zu Datenbanken angeboten, die Geometrien als zusätzliche Datentypen betrachten, auf die spezielle geometrische Operationen und Abfragen angewendet werden können (Abb. 2a). In diesem Papier wird ein anderer Ansatz vorgeschlagen: ein so genanntes *Geospatial Add-on* (*GAO*) wird als Bibliothek zur Anwendung hinzugebunden und erlaubt ihr, Geometrien zu speichern und geometrischen Anfragen zu formulieren (Abb. 2b). Die eigentliche Datenbankfunktionalität wird aber auf einer relationalen Datenbank *ohne* räumliche Erweiterung ausgeführt.

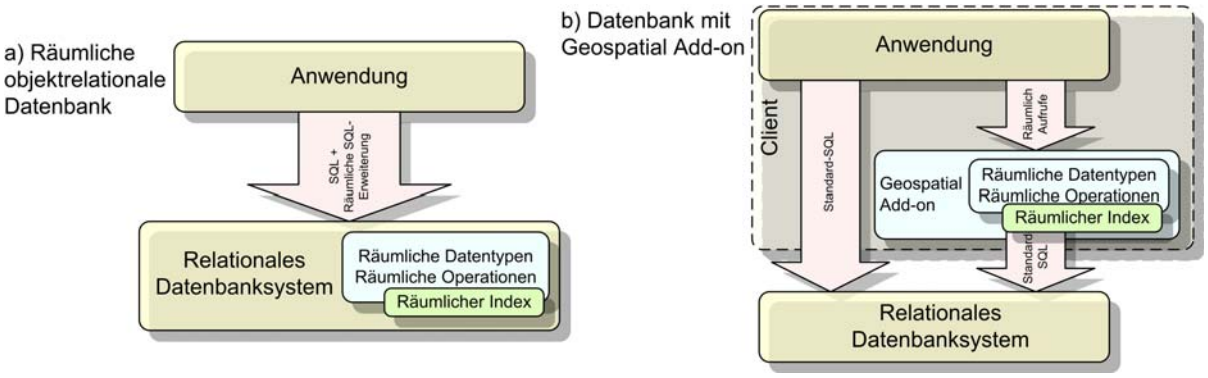

Abb.2: Varianten zur Verwaltung räumlicher Daten

Die Vorteile dieser Vorgehensweise:

- Zurzeit verwenden viele ortsbezogene Anwendungen und Dienste noch Standard-Datenbanken, indem ausschließlich Punkt-Geometrien (z.B. Points of Interests oder Koordinaten von Benutzern) gespeichert werden. In diesem Sonderfall können geometrische Anfragen (z.B. über die Entfernung von Punkten) über Standard-SQL abgewickelt werden. Der hier beschriebene Ansatz erlaubt eine nahtlose Migration solcher ortsbezogenen Anwendungen in Richtung komplexer Geometrien ohne große Modifikationen.
- Das unterlegte Datenbankmanagementsystem kann ohne Änderung der Anwendung ausgetauscht werden. So kann eine Anwendung beispielsweise prototypisch erst mit einem Datenbankmanagementsystem mittlerer Leistungsfähigkeit entwickelt werden, das später durch ein leistungsfähigeres Datenbankmanagementsystem ersetzt werden kann, ohne die Anwendung modifizieren zu müssen.
- Da das Add-on im Speicher der Anwendung ausgeführt wird, stehen die räumlichen Daten direkt der Anwendung als Objekte zur Verfügung. Damit übernimmt das Add-on das so genannte *Objektrelationale Mapping* (*ORM*), ohne dass man hierzu ein weiteres Rahmenwerk integrieren muss.
- Mittlerweile existieren Datenbanksysteme auch auf mobilen Endgeräten. So gibt es z.B. für die Endgeräteplattformen Symbian OS, Android und iPhone Realisierungen von SQ-Lite. Geodatenbanken sind dagegen typischerweise noch nicht für mobile Endgeräte portiert. Mit dem vorgeschlagenen Add-on können Anwendungen direkt auf dem Endgerät räumliche Daten ablegen und so beispielsweise ausgewählte Anteile einer größeren Geodatenbank lokal speichern.

In diesem Papier wird das Add-on beschrieben, das innerhalb des HomeRun-Projektes entwickelt wurde. Es verwendet einen eigenen räumlichen Index, den *Extended Split Index* und stellt eine Objekt-Schnittstelle für die Formulierung räumlicher Anfragen zur Verfügung.

2. Verwandte Arbeiten

Es gibt verschiedene räumliche Erweiterungen, die direkt in ein Datenbankmanagementsystem integriert wurden, z.B. *Oracle Spatial* [10], *PostgreSQL/ PostGIS* [11] oder *MySQL Spatial Extensions* [15]. Gegenüber den Standard-Datenbanken stellen die räumlichen Varianten zusätzlich geometrische Datentypen, geometrische Operationen und einen räumlichen Index zur Beschleunigung räumlicher Abfragen zur Verfügung. Als räumliche Indizes kommen häufig Verfahren zum Einsatz, die Geometrien baumartig organisieren, beispielsweise Quadtrees [4] oder verschiedene Varianten der R-Trees [6]. Die Verwaltung des Indexes wird vom Anwendungsentwickler weitgehend verborgen.

Die Schnittstelle der Anwendung zur Datenbank ist typischerweise SQL, durchgereicht zur Anwendung in Form von APIs wie ODBC und JDBC. Einen weiteren Komfort liefern Persistenz-Rahmenwerke wie Hibernate bzw. Hibernate Spatial [9], die die Inhalte der Datenbank direkt in den Objektraum der Anwendung einblenden. Hibernate Spatial verwendet so genannte *Provider*, um den Zugriff zur räumlichen Datenbank zu kapseln. Räumliche Abfragen werden mit einer SQL-Erweiterung HQL formuliert. Allerdings werden die Eigenschaften der räumlichen Datenbank nur durchgereicht, so dass keine vollständige Unabhängigkeit von dem unterlegten Datenbanksystem erreicht wird.

Zur Standardisierung räumlicher Erweiterungen und dem Zugriff über SQL hat das *Open Geospatial Consortium* (*OGC*) Vorschläge gemacht. Die so genannten *Simple Features* geben ein Rahmenwerk für Geometrien vor [7], indem Geometrie-Klassen wie z.B. LineString oder MultiPolygon und deren Abhängigkeiten definiert werden. Darüber hinaus wird festgelegt, wie diese Klassen auf SQL abgebildet werden [8]. Einen ähnlichen Ansatz verfolgt ISO SQL/MM Spatial [14], der auch auf den OGC Simple Features basiert.

Die existierenden räumlichen Datenbanken implementieren in unterschiedlichem Maße die durch die Standards vorgegeben Eigenschaften [3] – so fehlen in einigen Systemen bestimmte Eigenschaften oder entsprechen nicht den Vorgaben. Darüber hinaus sind der Satz geometrischer Funktionen sowie die Schnittstelle zum Zugriff auf die Geometrien noch sehr unterschiedlich ausgeführt. Bei der Anwendungsentwicklung muss man sich daher früh auf ein bestimmtes Datenbankmanagementsystem festlegen.

3. Das HomeRun-GAO

Eine der wesentlichen Komponenten für die Datenhaltung ist das *HomeRun-GAO*. Es hat folgende Eigenschaften:

- Die Funktionalitäten werden über eine objektorientierte Klassenbibliothek bereitgestellt (zurzeit in Java), die der Dienst oder die Anwendung nutzen kann. Diese ist mit ca. 80kB Binärcode extrem schlank.
- Ein neuartiger räumlicher Index genannt *Extended Split Index* erlaubt die geometrische Vorfilterung von Abfragen.
- Ein Serialisierungsmechanismus erlaubt die Ablage von Geometrien in einer Datenbankspalte. Zusätzlich wird die Index-Information in einer einzelnen weiteren Datenbankspalte gespeichert. Die GAO-Komponente stellt ein automatisches Objekt-Mapping für die Anwendung zur Verfügung.
- Die Bearbeitung der Geometrien geschieht ausschließlich im Objektraum der Anwendung. Hierzu wird eine externe *Geometry-Engine* auf der Basis der Java Topology Suite JTS [1] verwendet, die vollständig den Satz der OGC Simple Features implementiert. Geometrische Abfragen werden komfortabel über Funktionen zusammengestellt und automatisch auf Standard-SQL-Abfragen abgebildet.
- Die trotz Standardisierung vorhandenen Unterschiede verschiedener Datenbanksysteme werden in einem einzelnen Schnittstellen-Objekt behandelt. Zurzeit existieren DB-Zugriffe für die Datenbanken PostgreSQL, MySQL, Oracle, Microsoft SQL-Server, SQLite und HSQLDB. Weitere Schnittstellen können ohne großen Aufwand realisiert werden.

Typische räumliche Abfragen an das GAO ermitteln Datenreihen, die bestimmte räumliche Bedingungen erfüllen. Beispiele: *Welche Geo-Objekte umschließen einen bestimmten Punkt? Welche Geo-Objekte liegen vollständig in einem gegebenen Polygon? Welche schneiden eine bestimmte*

Linie? Als wesentliche Einschränkung können zurzeit keine Joins zweier Datenbanktabellen auf der Basis ihrer Geometrien durchgeführt werden, d.h. der eine Teil einer binären geometrischen Bedingung muss durch die anfragende Anwendung definiert werden. In der Praxis sind die verfügbaren Möglichkeiten für typische ortsbezogene Dienste völlig ausreichend. Hier werden beispielsweise alle Geo-Objekte für die Kartendarstellung ermittelt, die in einem bestimmten Kartenausschnitt liegen, oder es werden Objekte abgefragt, die auf einer bestimmten Benutzerposition liegen. Solch gestaltete Abfragen können durch das GAO bequem und effizient ausgeführt werden.

3.1 Der Extended Split Index

Auf einer baumartigen Struktur basierende räumliche Indizes haben einen signifikanten Nachteil: durch Einfügung, Änderung oder Löschung sind nicht nur Index-Werte der modifizierten Datenzeile betroffen – potentiell können durch eine Baum-Reorganisation die Index-Werte *aller* Zeilen einer Tabelle geändert werden. Für ein Add-on ist diese Herangehensweise kritisch, da es die Index-Werte über die Standard-Schnittstelle zur Datenbank modifizieren müsste. Während eine in die Datenbank integrierte räumliche Komponente einen effizienteren (da internen) Zugriff auf die Index-Strukturen hat, muss ein externes Add-on die Index-Werte wie eine typische Datenbankspalte betrachten. Großflächige Änderungen würden damit große Ausführungszeiten verursachen.

Ein weiterer Nachteil baumartiger Indizes ist, dass eine Suche durch das Abwandern eines Baumes realisiert wird, was mehrere Tabellenzugriffe erfordern würde. Für das Add-on wurde daher ein neuartiger räumlicher Index entwickelt, bei dem ein Index-Wert ausschließlich von der jeweiligen Datenzeile abhängt. Es ist in keinem Fall eine Reorganisation mehrerer Datenzeilen notwendig.

Der Index basiert auf der Idee des *Split Index* [2] und wurde in der aktuellen Fassung zum *Extended Split Index* weiterentwickelt. Als Grundidee wird die zweidimensionale Lage und Größe einer Geometrie auf eine kleine Menge eindimensionaler Index-Werte abgebildet, die effizient über Intervall-Suchen der Datenbank recherchierbar sind. Die zweidimensionale Suche wird dadurch auf etablierte eindimensionale Index-Mechanismen innerhalb von Standard-Datenbanken abgebildet.

Wichtig ist bei dieser Index-Struktur, wie generell bei Indizes, dass die Menge der Kandidaten, die eine bestimmte geometrische Eigenschaft erfüllen, schon bei der ersten Abfrage signifikant eingeschränkt werden kann. Die Liste der Kandidaten muss außerhalb der Datenbank noch exakt geometrisch überprüft werden. Die Feinfilterung der Kandidatenmenge geschieht im Objektraum der Anwendung.

Der Extended Split Index erlaubt die Verwaltung eines *endlichen* zweidimensionalen Bereiches. Die Erweiterung auf mehr Dimensionen ist denkbar. Zur Speicherung weltweiter geographischer Daten würde man Längengrad-Breitengrad-Koordinaten verwenden. Die Grundidee ist, ganzen Zahlen eindeutig bestimmte Teilflächen (genannt *Kacheln*) des Bereiches zuzuordnen. Die Zuordnung wird in Abb. 3 illustriert.

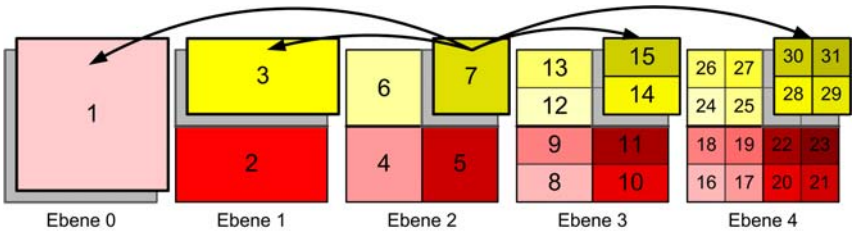

Abb.3: Idee des Extended Split Index

Diese Zuordnung ähnelt der *z-Kurve* [16], allerdings wird eine Kachel nicht in vier, sondern zwei Flächen zerlegt, zusätzlich werden die Kacheln fortlaufend über die Ebenen hinweg nummeriert. Einer vorgegebenen Geometrie werden später alle Kachelnummern verschiedener Ebenen zugeordnet, die diese Geometrie schneiden. Durch die Nummerierung ergibt sich eine einfache Zuordnung einer Kachel zu allen anderen Kacheln auf verschiedenen Ebenen, die diese schneiden. Zu einer gegebenen Zahl z (in Abb. 3 z.B. 7), ist die Menge aller *umgebenden* Kacheln

$$\left\{\left\lfloor\frac{z}{2}\right\rfloor,\left\lfloor\frac{z}{4}\right\rfloor,\left\lfloor\frac{z}{8}\right\rfloor,\ldots 1\right\}. \tag{1}$$

Umgekehrt werden *alle eingebetteten* Kacheln zu einer Kachel z durch die Menge

$$\{2\cdot z, 2\cdot z+1\}\cup\{4\cdot z,\ldots 4\cdot z+3\}\cup\{8\cdot z,\ldots 8\cdot z+7\}\cup\ldots \tag{2}$$

repräsentiert. Damit kann man einfach über den Vergleich von Kachelnummern erkennen, ob Geometrien potentiell überlappen. Eine ausführliche Diskussion der Zusammenhänge von Kachelnummern ist in [12] zu finden. Der Index wird wie folgt verwendet:

- Zu jeder Geometrie wird in der Datenbank eine weitere Spalte für die Kachelnummer eingerichtet. Die Einrichtung geschieht für den Anwendungsentwickler unsichtbar.
- Für jede Einfüge- oder Änderungsoperation der Geometrie wird die Kachelnummer berechnet und gespeichert.
- Werden Geometrien gelöscht, sind keine weiteren Maßnahmen notwendig.
- Bei Abfragen, die selbst Geometrien enthalten, wird die Kachelnummer der Abfragegeometrie ermittelt und die Abfrage um die Einschränkung der Index-Werte gemäß der Formeln 1 und 2 erweitert.

Die Einschränkung der Index-Werte hängt von der eigentlichen geometrischen Fragestellung ab. Bei Abfragen wie `CONTAINS`, `OVERLAPS`, `WITHIN` oder `EQUALS` werden zuerst alle Geometrien betrachtet, die eine Kachelnummer im erwarteten Bereich besitzen und danach über die Geometry-Engine exakt überprüft, ob die gesuchte geometrische Eigenschaft vorliegt. Eine Ausnahme bildet lediglich die Abfrage `DISJOINT` (alle Geometrien, die mit der Abfrage *keinen* gemeinsamen Punkt haben). Hier sind alle Geometrien, die eine Kachelnummer *außerhalb* des erwarteten Bereichs haben, ohne weitere Überprüfung ein Resultat. Da dadurch u.U. sehr große Ergebnismengen zurückgeliefert werden können, ist diese Operation mit Vorsicht anzuwenden.

Die Vorgehensweise bei der Index-Abfrage soll an einem Beispiel illustriert werden: Gesucht werden alle Städte in Deutschland mit einer Einwohnerzahl kleiner als 100 000. Der nichtgeometrische Anteil der Abfrage wird repräsentiert durch die Abfrage

```
SELECT * FROM CITIES WHERE INHABITANS<100000
```

Die Kachelnummer der durch die Grenzen von Deutschland repräsentierten Geometrie sei 55; `IND` sei der Spaltenname für die Kachelnummer. Dann lautet die Abfrage insgesamt

```
SELECT * FROM CITIES WHERE INHABITANS<100000 AND
        (IND=55 OR IND=27 OR IND=13 OR
         IND=6 OR IND=3 OR IND=1
         OR IND BETWEEN 110 AND 117
         OR IND BETWEEN 220 AND 227
         OR IND BETWEEN 440 AND 447)
```

Es ist dabei zu klären, ob durch die zusätzlichen Bedingungen die Länge eines SQL-Abfragestrings eine maximale Länge nicht überschreitet. Bei typischen Datenbanksystemen ist das allerdings unkritisch.

3.2 Weitere Einschränkung der Kandidatenmenge

Der bisher dargestellte Index hat noch ein Problem: Selbst kleine Geometrien können eine sehr niedrige Kachelnummer erhalten, wenn sie eine Trennlinie einer kleinen Ebenenzahl schneiden. So haben beispielsweise selbst sehr kleine Objekte, die auf dem Äquator liegen, die Kachelnummer 1. Solche Objekte werden immer in die Feinfilterung durch die Geometry-Engine einbezogen und führen zu einem konstanten Laufzeit-Anteil jeder Abfrage.

Als Lösung werden parallel *zwei* Kachelnummern berechnet. Die Index-Raster sind so gegeneinander verschoben, dass in der Regel nicht beide Indizes gleichzeitig eine kleine Kachelnummer produzieren. Bei Abfragen werden die beiden Anteile, die sich auf die jeweiligen Kachelnummern beziehen durch `AND` verknüpft. Die höhere Kachelnummer schränkt die Resultatmenge dabei am größten ein.

Die Suche nach dem optimalen Versatz ist nicht trivial. Einerseits soll an den Stellen, wo einer der Indizes eine kleine Kachelnummer produziert, der andere eine große produzieren. Andererseits darf der Versatz nicht zu groß sein, damit nicht zu viel Verschnitt an den Rändern entsteht. Eine ausführliche Analyse dazu ist in [13] dargestellt. Für jede Dimension ergibt sich ein optimaler Versatz von 1/6 der Gesamtgröße. Damit reduziert sich die abgedeckte Fläche auf 69% der Ausgangsfläche. Dies stellt aber bei der verfügbaren Auflösung kein Problem dar. Analysen mit realen Daten zeigen, dass durch den versetzten Index kleine Kachelnummern vermieden werden und die Ebenenzahl im Durchschnitt um 2 erhöht wird.

Weitere Performance-Analysen ergaben, dass für Gesamtlaufzeiten die Zeiten in der Datenbank ausschlaggebend sind, also nicht die Zeit für die Feinfilterung [13]. Zusammenfassend ergibt die Auswertung, dass der Extended Split Index ein effizienter Ansatz mit geringen Einfügekosten und hinreichend kleiner Kandidatenmenge bei geometrischen Suchen ist. Die Laufzeiten sind akzeptabel und praxistauglich.

4. Zusammenfassung und Ausblick

Das HomeRun Add-on erlaubt ortsbezogenen Anwendungen und Diensten, räumliche Daten in Standard-Datenbanken zu verwalten. Das Add-on führt den neuartigen Extended Split Index ein, der Index-Werte isoliert für jede Datenzeile berechnet, somit auf die Reorganisation ganzer Index-Spalten verzichten kann. Der Datenzugriff erfolgt über den Objektraum der Anwendung.

Aktuell sind Abfragen nur anhand von Vergleichen mit einer gegebenen Geometrie möglich. Ein erstes zukünftiges Ziel ist, Abfragen über *abgeleitete* Eigenschaften der Geometrie (z.B. Flächeninhalt oder Durchmesser) zu realisieren. Eindimensionale abgeleitete Eigenschaften lassen sich leicht in weitere Spalten ablegen, müssen jedoch bei jeder Geometrie-Änderung

korrigiert werden. Hier ist ein geeigneter Mechanismus zu entwickeln, damit nicht zu viele zusätzliche Spalten ständig mitgeführt werden müssen.

Langfristig sollen weitere Funktionen, die aus dem nicht-geometrischen Bereich bekannt sind, auch für geometrische Attribute übernommen werden. Die größte Herausforderung wird dabei sein, Geometrien aus mehreren Tabellen effizient in einer einzelnen Anfrage zu kombinieren (insb. das geometrische Join), ohne große Zwischenergebnisse im Objektraum der Anwendung generieren zu müssen.

5. Literaturverzeichnis

[1] Aquino J.: JTS Topology Suite, Technical Specifications, Vivid Solutions, 2003

[2] Beyer M.: Konzeption und Realisierung eines Geospatial Add-Ons für SQL, Diplomarbeit, Ohm-Hochschule Nürnberg, März 2009

[3] Brinkhoff T.: Open-Source-Geodatenbanksysteme, Datenbank-Spektrum 22/2007, 37-43

[4] Finkel R., Bentley J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys, Acta Informatica 4 (1): 1974, 1–9

[5] Additional information is available in the Gartner report "Location- Based Services Subscriber and Revenue Forecast, 2006-2011" (ID Number: G00153022)

[6] Guttman A.: R-Trees: A Dynamic Index Structure for Spatial Searching, Proc. of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, June 1984, 47-57

[7] Herring J. (ed.): OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture, OGC, 2005

[8] Herring J. (ed.): OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option, OGC, 2006

[9] Hibernate Spatial Tutorial, 2009, http://www.hibernatespatial.org

[10] Murray C.: Oracle Spatial Developer's Guide, Oracle, Aug. 2009

[11] Neufeld K.: PostGIS Manual, 2009

[12] Roth J.: Verwaltung geographischer Daten mit Hilfe eines Add-ons für Standard-Datenbanken, Use In Context, GI Informatik 2009, Lübeck, 29. Sept. 2009

[13] Roth J.: The Extended Split Index to Efficiently Store and Retrieve Spatial Data With Standard Databases, IADIS International Conference Applied Computing 2009, Rom (Italien), 19.-21. Nov. 2009

[14] Stolze K.: SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems, BTW 2003, Leipzig, Feb. 2003

[15] MySQL 5.0 Reference Manual, Sun Microsystems, 2009

[16] Tropf H., Herzog H.: Multidimensional Range Search in Dynamically Balanced Trees, Angewandte Informatik, 2/1981, 71–77