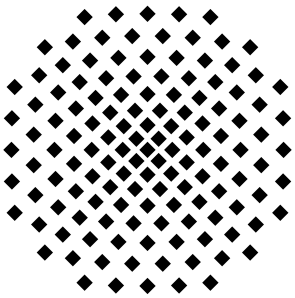


Universität Stuttgart

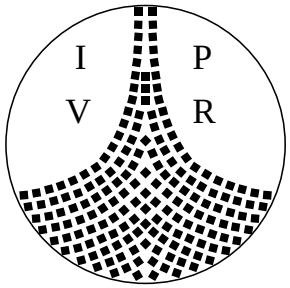
Fakultät Informatik

Prüfer:	Prof. Dr. B. Mitschang
Betreuer:	Dipl.-Inf. Daniela Nicklas Dipl.-Ing. Darko Klinec
begonnen am:	1.8.2001
beendet am:	1.2.2002
CR-Klassifikation:	C.2.4, H.2.4, H3.5, H.2.2

Diplomarbeit Nr. 1958

Enabling technology for an indoor location aware information system

Stjepan Grzan



Institut für Parallele und Verteilte
Höchstleistungsrechner (IPVR)
Abteilung Anwendersoftware
Breitwiesenstr. 20-22
70565 Stuttgart

AS

Kurzfassung

Das Nexus-Projekt entwickelt eine Plattform, die als Infrastruktur für ortsbasierte Anwendungen dienen soll. Ein wichtiger Teil dieser Plattform sind Spatial Model Server. In der Diplomarbeit soll für einen kleinen, abgeschlossenen Teil des Datenmodells ein hochperformanter Spatial Model Server implementiert werden, der seine Objekte weitgehend im Hauptspeicher hält. Außerdem soll ein Positionierungssystem für Innenräume unterstützt werden. Es werden hierfür Positionierungssysteme betrachtet und Konzepte bei Hauptspeicherdatenbanksystemen vorgestellt. Es werden danach die Anforderungen an einen Indoor SpaSe betrachtet und der Aufbau des Prototypen beschrieben. Obwohl der Prototyp fast keine Optimierungen benutzt, so ist die Geschwindigkeit für das Einsatzszenario (Fakultätsgebäude) bereits ausreichend. Bei der Verwendung größerer Objektmengen zeigten sich Geschwindigkeitsprobleme. Die Geschwindigkeitsprobleme entstanden nicht durch den Zugriff auf die Objekte, der ohne Indexe erfolgte, sondern durch die langsame und speicherintensive Umwandlung der Objekte aus dem internen Format in das XML-Format.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Mobile Computer können von Anwendern mit sich geführt werden und sind an unterschiedlichen Orten einsetzbar. Daraus ergeben sich mögliche Anwendungen, die speziell für mobile Anwender ausgelegt sind. Ein Beispiel hierfür sind ortsbasierte Anwendungen, bei denen die ausgegebene Information vom Aufenthaltsort des Anwenders abhängt. Navigationssysteme berechnen ausgehend von der momentanen Anwenderposition den idealen Weg zu einem Ziel und führen den Anwender dorthin. Dabei erhält der Anwender Richtungsanweisungen, die von seiner momentanen Position abhängen. Die Positionsbestimmung muss nicht durch den Anwender erfolgen. Die Miniaturisierung hat Positionierungssysteme ermöglicht, die auch in mobile Computer eingebaut werden können. GPS-Sensoren ermöglichen beispielsweise eine relativ genaue Positionsbestimmung außerhalb eines Gebäudes und können in mobile Computer eingebaut werden.

Die Daten für solche Anwendungen müssen nicht auf einem mobilen Computer gespeichert sein, sondern können über Funknetze übertragen werden. Es ist so möglich, Dienste zu entwickeln, die speziell von ortsbasierten Anwendungen benutzt werden. Eine Anwendung könnte als Museumsführer dienen, der Informationen über Räume oder Ausstellungsstücke in der Nähe des Benutzers ausgibt. Die Datenmenge wäre wahrscheinlich zu groß für viele mobile Computer. Der Museumsführer ist daher nur realisierbar, wenn auf externe Datenquellen zugegriffen werden kann.

1.2 Das Nexus Projekt

Das Nexus Projekt erforscht Konzepte und Methoden zur Unterstützung von ortsbasierten Anwendungen mit mobilen Anwendern. Hierfür wird eine offene Plattform entwickelt, die als Infrastruktur für ortsbewusste Anwendungen dienen soll. Ähnlich wie beim WWW soll jeder in der Lage sein neue Dienste in die Plattform integrieren zu können.

Am Nexus Projekt ist das Institut für parallele und verteilte Höchstleistungsrechner mit den Abteilungen verteilte Systeme und Anwendersysteme, das Institut für Photogrammetrie und das Institut für Nachrichtenvermittlung und Datenverarbeitung beteiligt.

1.3 Aufgabenstellung

Im nächsten Kapitel wird die Nexus Architektur vorgestellt. In ihr gibt es sogenannte Spatial Model Server (SpaSe), die als eine Art Web-Server für ortsbasierte Informationen arbeiten sollen. Dafür wurde ein gemeinsames Datenmodell definiert, das Augmented World Model, dessen Objekte über eine definierte XML-Schnittstelle abgefragt werden können.

Eine mögliche Implementierung eines solchen SpaSe ist der Einsatz einer geographischen Datenbank. In dieser Arbeit soll ein anderer Ansatz untersucht werden: für einen kleinen, abgeschlossenen Teil des Datenmodells soll ein

hochperformanter SpaSe implementiert werden, der seine Objekte weitgehend im Hauptspeicher hält. Außerdem soll ein Positionierungssystem für Innenräume unterstützt werden. Die Räume können beispielsweise mit Infrarotsendern ausgestattet werden. Wenn diese Sensoren und ihre Sichtbarkeitsbereiche im SpaSe modelliert sind, kann die Position aus einer Verschneidung der Bereiche bestimmt werden.

Das Einsatzszenario ist das Innere des Fakultätsgebäudes. Modelliert werden dabei Räume und Sender.

1.4 Aufbau der Arbeit

Kapitel 2 beschreibt die Nexus Architektur und ihre Komponenten. Kapitel 3 gibt einen Überblick über Positionierungssysteme und beschreibt wie sie durch einen SpaSe unterstützt werden können. Da der SpaSe seine Daten komplett im Hauptspeicher verwalten kann, werden in Kapitel 4 Hauptspeicherdatenbanksysteme vorgestellt und daraus Schlüsse für den Aufbau des Prototypen geschlossen. Die Anforderungen an den SpaSe werden in Kapitel 5 dargestellt und dessen Aufbau in Kapitel 6. Kapitel 7 evaluiert den Indoor SpaSe. Die Ergebnisse der Diplomarbeit werden in Kapitel 8 zusammengefasst und ein Ausblick auf zukünftige Entwicklungen gegeben.

2 Nexus

2.1 Zielsetzung

Das Nexus-Projekt entwickelt eine offene Plattform, die Dienste für ortsbasierte Anwendungen mit mobilen Anwendern bereitstellt. Die Plattform verwaltet ein Modell der realen Welt mit realen und virtuellen Objekten, das Augmented World Model genannt wird. Ähnlich wie im WWW soll jeder in der Lage sein, Informationen für das Modell bereitzustellen, indem jeder Server, der eine Nexus-Schnittstelle besitzt, in das Nexus-System integriert werden kann. Externe Datenquellen wie WWW-Server sollen ebenfalls integriert werden können. Es entsteht hierdurch eine große Heterogenität der Daten. Nexus benutzt einen Föderationsansatz, um die Heterogenität kontrollieren zu können.

Ein Beispiel für virtuelle Objekte sind E-Notes, die an einem anderen Objekt „kleben" und Informationen enthalten. Man kann sie beispielsweise an Pflanzen kleben und die Zeitpunkte eintragen, an denen sie zuletzt gegossen wurden.

Die Nexus-Plattform soll bei unterschiedlichen Anwendungsszenarien einsetzbar sein.

- In Zukunft wird die Bereitstellung von Informationen mit einem Ortsbezug sehr wichtig sein. Hier sind die Informationen an einen Ort, ein Gebiet oder ein Objekt der realen Welt gebunden. Anwender suchen die Informationen, die zu einem bestimmten Gebiet oder Objekt gehören. Ein solches System ist das in der Einleitung erwähnteBeispiel des Museumsführers, bei dem die Anwender Informationen zu Räumen in der Nähe der Anwenderposition und zu Ausstellungsstücken in ihrer unmittelbaren Nähe erhalten.
- Die Mobilität von Anwendern und Objekten ermöglicht weitere Anwendungsszenarien bei denen die Informationen von mobilen Objekten gespeichert werden. Ein Beispiel wäre eine Anwendung, die es ermöglicht die Position von Freunden innerhalb eines Museums zu finden. Ein anderes Beispiel ist ein Taxiunternehmen, das ein zu einem Kunden nächste Taxi sucht, damit dieses den Kunden abholt.
- Navigationssysteme sind die heutzutage verbreitetsten Systeme und werden bereits in vielen Fahrzeugen verwendet, damit die Anwender möglichst schnell zu ihrem Ziel finden. Da bei solchen Systemen auch aktuelle Daten wie momentane Staus eine wichtige Rolle spielen können, müssen die Navigationsdaten laufend aktualisiert werden und Ereignisse wie Unfälle oder Staus berücksichtigt werden.
- Die bisher vorgestellten Anwendungsszenarien gehen von einem eher passiven Anwender aus, der zwar Informationen empfängt, aber keine Änderungen an der Umwelt vornimmt. Eine besondere Herausforderung besteht, wenn die Anwender die Möglichkeit bekommen ihre Umwelt zu verändern. Anwender könnten Daten im Nexus-System verändern oder neue Daten einfügen. Ein Beispiel hierfür sind die vorgestellten E-Notes oder die Möglichkeit, Nachrichten an bestimmten Positionen zu hinterlas-

sen. Dies könnte für Warnschilder oder für die Realisierung einer virtuellen Schnitzeljagd benutzt werden, bei der keine realen Markierungen zurückgelassen werden, sondern virtuelle. Es ist auch denkbar, dass Anwender reale Objekte steuern und einen PDA benutzen, um ihre Modelleisenbahn anzusteuern oder die Heizung in einem Raum zu verändern.

- Kommunikationsmethoden mit Ortsbezug ermöglichen neue Anwendungen. Es gibt Fälle, in denen es sinnvoll ist, Nachrichten an Personen in einem bestimmten Gebiet zu senden. Ein typisches Beispiel sind Katastrophenwarnungen, die an Personen in der betroffenen Region geschickt werden. Brennt ein Gebäude, so kann man alle Personen innerhalb des Gebäudes und in dessen Nachbarschaft benachrichtigen. Ein anderes Beispiel sind Nachrichten über interessante Ereignisse, die nur für Kunden innerhalb eines bestimmten Gebiets interessant sind. Besucher eines Freizeitparks können die Information erhalten, dass man nun für die nächste Aufführung einer bestimmten Show hineingelassen wird. Man könnte auch Besucher des Fakultätsgebäude über die Öffnung des Computermuseums benachrichtigen.

2.2 Architekturüberblick

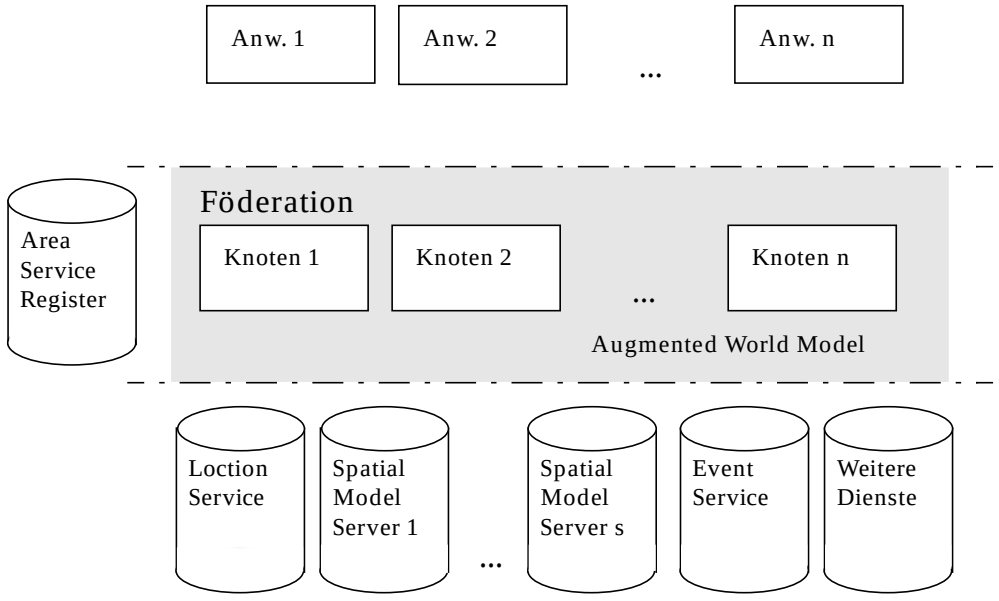


Abbildung 2-1: Nexus-Architektur (entlehnt aus [Nicklas et al 2001])

Abbildung 2-1 zeigt die Nexus-Architektur. Sie besteht aus drei Teilen: Den Nexus-Anwendungen, den Nexus-Knoten, die den Zugriff auf die Föderation ermöglichen, und den Nexus-Diensten. Die Nexus-Anwendungen stellen Anfragen an Nexus-Knoten. Die Nexus-Knoten stellen den Anwendungen eine einheitliche Sicht auf die Daten zur Verfügung: Die Augmented World. Sie müssen ankommende Anfragen analysieren und daraus Anfragen an die Nexus-Dienste erzeugen. Die Ergebnisse der Nexus-

Dienste werden von den Nexus-Knoten integriert und in eine für die Anwendungen lesbare Form umgewandelt. Nexus-Anwendungen greifen in der Regel nur über Nexus-Knoten auf die Nexus-Dienste zu und müssen so keine Informationen über Verteilung und Aufbau der Dienste kennen. Die Föderation verwendet das Area Service Register, um die für eine Anfrage zuständigen Server zu finden. Es gibt mehrere unterschiedliche Nexus-Dienste. Die wichtigsten Komponenten der Nexus-Dienste sind der Location Service (siehe Abschnitt 2.5.2 „Location Service"), die Spatial Model Server des Spatial Model Service (siehe Abschnitt 2.5.1 „Spatial Model Service") und der Event Service (siehe Abschnitt 2.5.3 „Event Service").

2.3 Nexus-Anwendungen

Die typische Nexus-Anwendung ist ortsbasiert und befindet sich auf einem mobilen Rechner. Da sie ortsbasiert ist, hängt ihre Funktionalität in der Regel von der Position des Anwenders ab und muss durch die Anwendung mit Hilfe einer Sensorkomponente ermittelt werden. Da sie auf einem mobilen Rechner ist, kommuniziert sie drahtlos mit der Nexus-Plattform. Abbildung 2-2 zeigt die Architektur eines Nexus-Clients auf dem eine Nexus-Anwendung läuft.

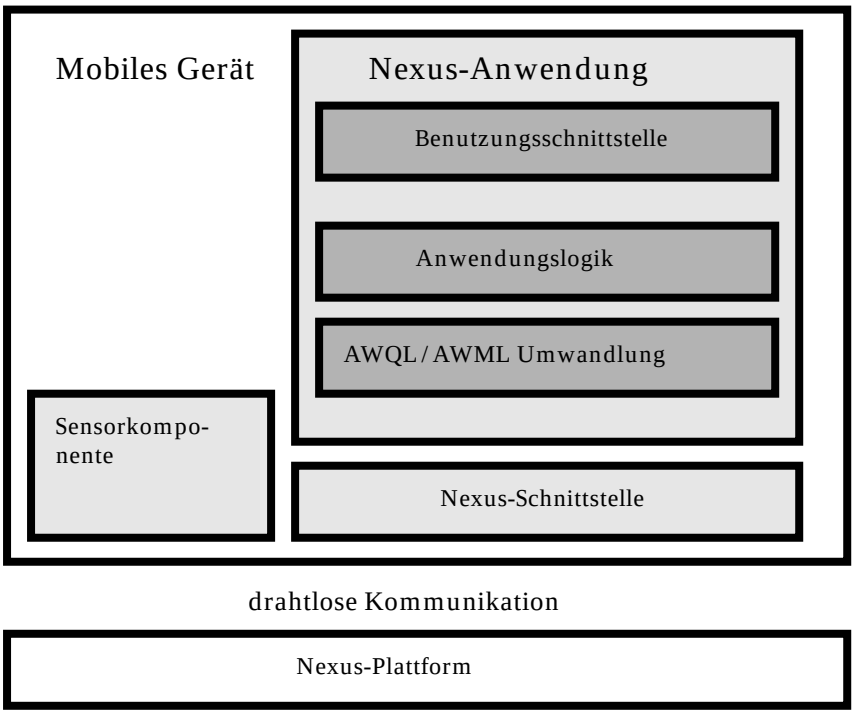


Abbildung 2-2: Aufbau eines Nexus-Clients (übersetzt aus [Nicklas & Mitschang 2001])

Schnittstellen zur Sensorkomponente und die Nexus-Schnittstelle werden im Idealfall für das mobile Gerät bereitgestellt und müssen nicht von den Anwendungen implementiert werden. Eine Nexus-Anwendung besteht aus einer Benutzungsschnittstelle, der Anwendungslogik und einer Komponente für die Umwandlung von AWQL und AWML. Sie greift zur Kommunikation

mit der Nexus-Plattform auf die Nexus-Schnittstelle zu. Viele mobile Geräte besitzen nur sehr kleine Bildschirme und nur einfache Eingabemechanismen. Es ist deswegen nicht sinnvoll, viele Objekte darzustellen oder aufwendige Eingaben zu tätigen. Viele Anwendungen müssen sehr leicht verständlich sein, da sie von Anwendern benutzt werden, die keine oder geringe Erfahrung mit der Anwendung haben. Die Anwendungslogik ist in der Regel nicht sehr kompliziert und recht klein, da viele mobile Geräte nur eine geringe Rechenleistung besitzen und der Funktionsumfang bei einer einfachen und schnell erlernbaren Bedienung nicht sehr groß ist. Benutzungsschnittstelle und Anwendungslogik können allerdings auch den Umfang herkömmlicher Desktopanwendungen erreichen, wenn sie von erfahrenen Benutzern bedient werden sollen und die mobilen Geräte ähnliche Leistungsmerkmale und Bildschirmgrößen wie Desktopsysteme aufweisen. Für den Zugriff auf die Nexus-Schnittstelle müssen Anfragen in einer Anfragesprache, der Augmented World Query Language (AWQL), erstellt werden. Das Ergebnis ist in der Augmented World Modeling Language (AWML).

Beispiele sind Anwendungen zur Darstellung ortsbasierter Informationen oder Systeme, die als Navigationshilfe dienen. Eine Anwendung könnte ein Museumsführer sein, den ein Museum bereitstellt. Ein Museumsführer könnte eine Tour durch das Museum ermöglichen, indem er zu den einzelnen Ausstellungsstücken führt und Informationen zu dem gerade betrachteten Ausstellungsstück ausgibt. Er ist ein Beispiel für eine Anwendung, die von unerfahrenen Anwendern ausgehen muss. Ein Museumsführer wird wahrscheinlich speziell für ein Museum erstellt und unterscheidet sich daher in seinem Aussehen und seiner Bedienung von anderen Programmen. Die meisten Besucher eines Museums waren entweder noch nie oder schon sehr lange nicht mehr in dem Museum und kennen die Anwendung daher noch nicht.

2.4 Föderation

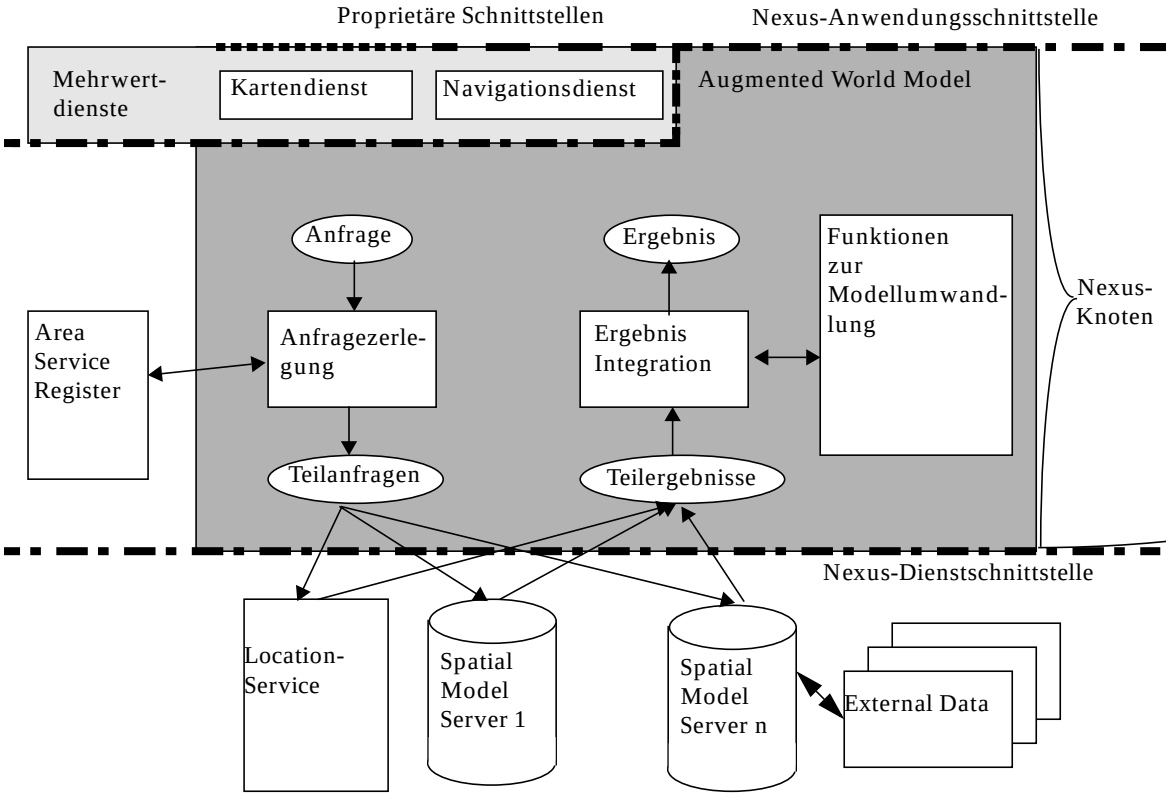


Abbildung 2-3: Aufbau der Föderation (übersetzt aus [Nicklas et al 2001])

Die Föderation stellt den Nexus-Anwendungen eine einheitliche Sicht auf die Daten der Nexus-Dienste zur Verfügung. Der Zugriff erfolgt über Nexus-Knoten, die auch Mehrwertdienste bereitstellen können. Den Aufbau der Föderation zeigt Abbildung 2-3 .

2.4.1 Nexus-Knoten

Bei Nexus-Knoten handelt es sich um Rechner, die den Nexus-Anwendungen eine einheitliche Sicht auf die Daten der Nexus-Dienste (das Augmented World Model) anbieten.

Nexus-Anwendungen greifen über die Anwendungsschnittstelle auf einen Nexus-Knoten zu. Der Nexus-Knoten verteilt die Anfrage auf die notwendigen Nexusdienste, indem er Ortsinformationen auswertet, die in jeder Anfrage einer Nexus-Anwendung enthalten sind. Dadurch kann er die Gebiete der benötigten Informationen ermitteln. Diese Gebiete werden an das Area Service Register gesendet, um die zuständigen Server für diese Gebiete zu erhalten. Der Nexus-Knoten erhält dabei auch Informationen über die gespeicherten Objektklassen des Servers. Er stellt über die Nexus-Dienstschnittstelle Anfragen an die benötigten Server. Die erhaltenen Antworten werden mit Hilfe von Funktionen zur Modellumwandlung in die benötigte Form umformuliert und zusammengeführt.

Anfragen an die Föderationsschicht können an jeden beliebigen Nexus-Knoten verschickt werden, da die Fähigkeiten jedes Knoten gleich sind. Die Nexus-Anwendungen sind dadurch nicht auf einen bestimmten Nexus-Knoten angewiesen, sondern können immer den sinnvollsten und damit in der Regel den räumlich nächsten Nexus-Knoten benutzen. Es werden dadurch Konzepte für die Duplizierung und nicht für die Verteilung benötigt. Da eine Duplizierung von Daten in einem globalen System sehr aufwendig ist, speichern die Nexus-Knoten keine Daten und stellen so sicher, dass eine Anfrage bei jedem Nexus-Knoten das gleiche Ergebnis liefert.

Mehrwertdienste können auf einen Nexus-Knoten geladen werden und ermöglichen so eine Erweiterung der Funktionalität. Aus Sicherheitsgründen müssen diese Dienste auch über die Nexus-Anwendungsschnittstelle auf einen Nexus-Knoten zugreifen.

2.4.2 Mehrwertdienste

Mehrwertdienste befinden sich auf einem Nexus-Knoten und werden über eine proprietäre Schnittstelle von den Nexus-Anwendungen angesprochen. Beispiele hierfür sind Kartendienste, die Karten erzeugen, oder Dienste für die Navigation. Diese Dienste benutzen die Anwendungsschnittstelle, um Daten von der Föderation zu erhalten. Drittanbieter können so die Nexus-Plattform um Dienste ergänzen, indem sie diese in einen Nexus-Knoten integrieren. Es ist auch denkbar, dass Nexus-Anwendungen Dienste auf einen Nexus-Knoten laden. Beispielsweise könnten so Daten von einer Nexus-Anwendung auf einen Nexus-Knoten ausgelagert werden oder Anwendungslogik auf einem Nexus-Knoten ausgeführt werden. Dies kann in mehrfacher Hinsicht von Nutzen sein. Der Nexus-Knoten kann so selbst sehr leistungsarme Clients haben, da große Teile der aufwendigen Anfrageerstellung und AWML Umwandlung nicht mehr von den Clients gemacht werden muss. Die Kommunikation kann in einem sehr viel einfacherem Format z.B. mit Hilfe eines RPC Mechanismus durchgeführt werden. Weiterhin kann die Menge der AWQL Aufrufe und damit die Menge der übertragenen Daten reduziert werden. AWQL ist sehr einfach, weshalb mehrere AWQL Anfragen für den Erhalt gewünschter Daten notwendig sein können. Anstatt dass ein Client mehrere Anfragen stellt und die Ergebnisse analysiert, kann dies der Nexus-Knoten mit Hilfe eines Mehrwertdienstes durchführen. Die Kommunikation zwischen Nexus-Anwendung und Nexus-Knoten kann so reduziert werden. Die Reduktion der Kommunikation führt bei mobilen Geräten nicht nur zu einem geringeren Leistungsbedarf, sondern auch zu einem geringeren Energieverbrauch und damit zu einer längeren Laufzeit. Da die Kommunikation auch über kostenpflichtige Funknetze durchgeführt werden kann, führt eine Reduzierung der Kommunikation auch zu einer Reduzierung der Kosten.

Mehrwertdienste werden nicht zu den Nexus-Diensten gezählt, da sie föderierte Daten bearbeiten oder auf Quellen außerhalb des Nexus-Systems zugreifen und eine beliebige Schnittstelle zu den Anwendungen

anbieten können. Außerdem bieten Nexus-Dienste ihre Dienste der Föderation an, während Mehrwertdienste als Zielgruppe Nexus-Anwendungen haben.

2.4.3 Area Service Register

Das Area Service Register speichert Informationen über Augmented Areas. Eine Augmented Area ist eine logisch zusammenhängende Menge von NexusObjekten, die auf einem SpaSe gespeichert ist. Sie ist ein beliebig ausgewähltes Gebiet und darf sich mit anderen Augmented Areas überlappen. Augmented Areas enthalten Objekte, die auch auf bestimmte Typen z.B. Räume begrenzt sein können. Alle Objekte einer Augmented Area liegen komplett innerhalb des Gebiets. Objekte dürfen innerhalb einer Augmented Area nur einmal vorkommen. Da sie in unterschiedlichen Augmented Areas vorhanden sein können, kann es aber mehrere Repräsentationen eines Objekts geben. Eine Augmented Area ist einem Spatial Model Server zugeordnet und befindet sich mit allen ihren Objekten auf einem Spatial Model Server. Eine Augmented Area befindet sich deshalb immer nur auf einem Spatial Model Server, wobei ein Spatial Model Server mehrere Augmented Areas verwalten kann. Das Area Service Register speichert die Ausdehnung der Augmented Areas, die Klassen der enthaltenden Objekte und den Namen des sie verwaltenden Spatial Model Servers.

Da die Föderation wegen der sonst notwendigen Duplizierung keine Daten speichert, ist das Area Service Register nicht Teil der Föderation. Es ist ein Dienst, der sich auf der gleichen Ebene wie die Föderation befindet, und stellt das verteilte Gedächtnis dar, mit dem die Föderation die Augmented Areas und Spatial Model Server zuordnen kann.

2.5 Nexusdienste

2.5.1 Spatial Model Service

Spatial Model Server verwalten Informationen zu statischen Objekten und ermöglichen räumliche Anfragen über die Objekte. Spatial Model Server registrieren die von ihnen verwalteten Augmented Areas beim Area Service Register und geben hierfür das abgedeckte Gebiet und Metainformation über die gespeicherten Daten (das Augemented World Schema) an.

Erhalten Nexus-Knoten Anfragen zu statischen Objekten, so stellen sie entsprechende Anfragen an Spatial Model Server. Spatial Model Server können Events erzeugen und müssen hierfür mit dem Event Service zusammenarbeiten.

Sie können nicht nur Daten für Nexus-Anwendungen speichern, sondern auch Daten für andere Dienste. Das Positionierungssystem beispielsweise kann benötigte Daten in einem Spatial Model Server speichern (siehe Kapitel Positionierung), Karten können für den Map Service gespeichert werden und Navigationsdienste können Navigationsknoten in einem Spatial Model Server speichern. Daten in externen Datenquellen, wie Webservern oder Datenbanken, können mit Hilfe eines Spatial Model Servers, der die Daten mit Objekten

verknüpft, in die Augmented World integriert werden. Anfragen an einen Spatial Model Server werden mit Hilfe der Augmented World Query Language gestellt.

2.5.2 Location Service

Der Location Service verwaltet die Informationen zu mobilen Objekten wie z.B. Personen oder Fahrzeugen. Spatial Model Server sind für die Speicherung mobiler Objekte ungeeignet, da statische Objekte und mobile Objekte unterschiedlich verwaltet werden müssen ([Nicklas & Mitschang 2001]). Die Position muss bei mobilen Objekten sehr oft verändert werden, während dies bei statischen Objekten nicht notwendig ist. Die Positionsinformation eines mobilen Objekts muss nicht persistent gespeichert werden, da eine gespeicherte Position nach einem Neustart veraltet sein könnte. Die aktuelle Position lässt sich außerdem sehr schnell durch Sensoren ermitteln. Die Position statischer Objekte wird in der Regel nicht überwacht. Es ist deshalb nicht möglich, eine aktuelle Position für ein statisches Objekt mit Hilfe eines Sensors zu erhalten. Außerdem können sich mobile Objekte aus einer Augmented Area herausbewegen und müssen dann einem anderem Server übergeben werden. Selbst wenn sie sich in ein Gebiet bewegen, das nicht von einer Augmented Area abgedeckt wird, dürfen die Daten nicht verloren gehen. Durch die separate Speicherung der mobilen Objekte können Spatial Model Server und Location Server auf die Anforderungen der unterschiedlichen Objektarten optimiert werden. Spatial Model Server haben in erster Linie nur Anfragen und ihre Daten werden nur sehr selten geändert. Ihre Datenstrukturen und Funktionen müssen auf lesenden Zugriff optimiert sein. Location Server müssen die Position der mobilen Objekte sehr oft verändern und benötigen Datenstrukturen und Funktionen, die für schreibenden Zugriff ausgelegt sein müssen. Ein weiterer Unterschied besteht in den Event Typen, die Location Server und Spatial Model Server unterstützen (siehe Abschnitt 2.5.3 „Event Service”).

2.5.3 Event Service

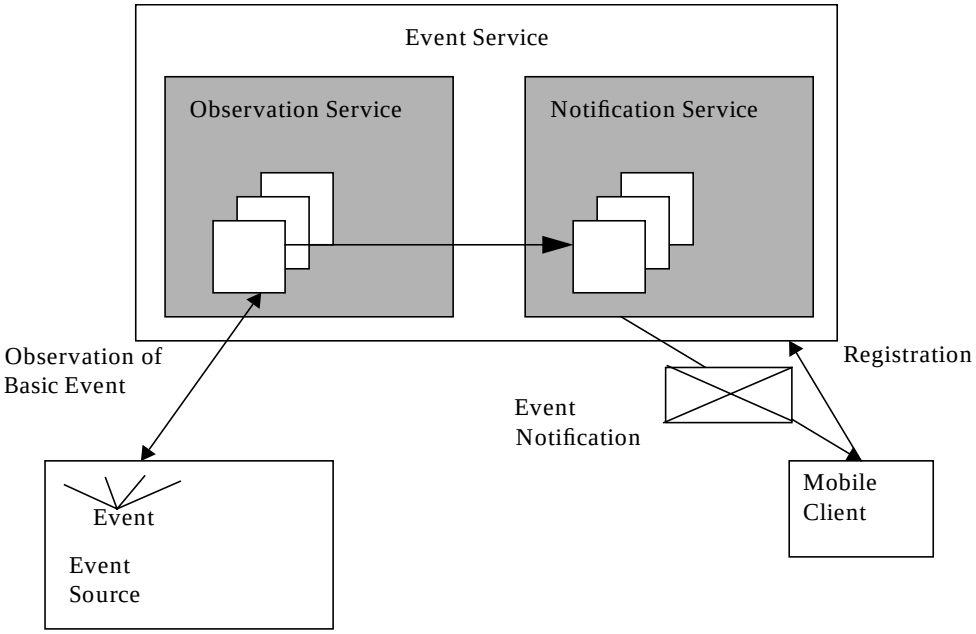


Abbildung 2-4: Architektur des Event Service (aus [Bauer 2001])

Der Event Service ermöglicht eine ereignisbasierte Kommunikation. Ein *Event* ist das Auftreten eines Ereignisses. Die Benachrichtigung über das Auftreten ist eine *Event Notification*. Der Event Service besteht aus dem *Observation Service* und dem *Notification Service*. Abbildung 2-4 zeigt die Architektur des Event Services. Ein Dienst, der Events beobachten und Event Notifications erzeugen kann, wird *Event Source* bezeichnet. Hierzu gehören Spatial Model Server und Lokationsserver. Die Event Sources registrieren ihre beobachtbaren Events beim Event Service. Hierfür wird für jeden dieser Event Typen eine eindeutige ID (*Template ID*) erzeugt. Möchte ein Client benachrichtigt werden, wenn ein bestimmtes Event eintritt, so registriert er sich beim Event Service. Der Event Service registriert diesen Event dann beim Event Source und erhält von diesem eine *Predicate ID*. Anhand der Predicate ID werden die Events verteilt. Sobald ein Event eintritt, erhält der Observation Service eine Nachricht vom Event Source. Die Nachricht enthält eine *Notification ID* und weitere Daten, die beim Eintreten eines Ereignisses von diesem Typ versendet werden sollen. Die Notification ID wird bei jedem Auftreten eines Events neu generiert und ist eindeutig, damit man die aufgetretenen Events voneinander unterscheiden kann. Der Observation Service entscheidet, ob das Event weitergeleitet werden soll und benachrichtigt dann den Notification Service, der die Verteilung der Events durchführt und alle für diesen Event registrierten Clients benachrichtigt. Ein Beispiel verdeutlicht den Ablauf:

Eine Event Source soll einen Event generieren, wenn ein Element eingefügt wird. Wird das Event registriert, so soll man auch die Klasse der Elemente angeben, bei denen eine Notification versendet werden soll. Dieser Event wird beim Event Service registriert. Hierfür muss eine Template ID erzeugt werden und eine Beschreibung des Events und seines Parameters (die Objektklasse).

Ein mobiler Client möchte nun benachrichtigt werden, wenn Objekte von der Klasse Sender eingefügt werden. Er registriert sich beim Event Service und gibt als Parameter „Sender“ an. Der Event Service registriert diesen Event dann bei der dazugehörigen Event Source mit dem Parameter Sender. Es wird eine eindeutige Predicate ID generiert. Wird nun ein Sender in die Event Source eingefügt, so wird ein Event ausgelöst. Für die Benachrichtigung generiert die Event Source dann eine Notification ID und schickt an den Event Service eine Event Notification. Die Event Notification enthält die Predicate ID. Mit deren Hilfe erkennt der Event Service, an welchen mobilen Client die Nachricht verschickt werden muss und verschickt sie an den mobilen Client.

Ein Event einer Event Source ist ein Basic Event. Die einzige Einschränkung eines Basic Events besteht in den Implementierungsmöglichkeiten bei der Event Source. In zukünftigen Erweiterungen des Event-Service soll es möglich sein, Plug-In Trigger beim Event Service und bei den Event Sources zu registriern. Es wird dabei ein Programm übergeben, dass komplexe Events im Inneren einer Event Source überwachen kann.

Die Basic Events können zu komplexeren Events kombiniert werden, wozu eine Definitionssprache benutzt werden kann. Basic Events sind die Elemente dieser Sprache. Der Observation Service beobachtet diese komplexen Events.

Der verwendete Mechanismus für die Eventdefinition unterscheidet sich von der Eventdefinition bei aktiven Datenbanken. Bei aktiven Datenbanken werden Ereignisse mit Hilfe von ECA-Regeln (oft auch Trigger oder Alerter genannt) definiert. ECA sind die Abkürzungen für Event, Condition und Action. Bei einer ECA Regel gibt man ein Ereignis und eine Bedingung, an bei der das Event ausgelöst werden soll, und beschreibt, was dann geschehen soll. Ein Beispiel:

Beim Einfügen eines Senders, dessen ausgesendete ID bereits von einem anderen Sender benutzt wird, soll eine Benachrichtigung an den Einfügenden erfolgen, bei der auch die Daten der anderen Sender mit dieser ID enthalten sind. Das Event ist hier das Einfügen eines Senders, die Bedingung ist eine ausgesendete ID, die bereits mindestens ein anderer Sender besitzt. Die Aktion wäre das Aussenden einer Nachricht an den Einfügenden. In dieser Nachricht müssten die Daten der anderen Sender mit der gleichen ausgesendeten ID enthalten sein.

In Nexus werden Ereignis und Bedingung zu einem Ereignis zusammengefasst. Das Ereignis aus dem obigen Beispiel wäre dann: „Einfügen eines Senders mit bereits verwendeter ID”.

Gründe, die gegen eine Verwendung von ECA-Regeln sprechen sind [Gespräch mit Bauer 2002]:

Als Aktion wird nur eine Benachrichtigung benötigt und nicht, wie bei aktiven Datenbanksystemen auch Änderungsoperationen. Komplexe Events, die über mehrere Event Sources verteilt sein können, kann nur der Observation Service erkennen. Er führt keine Aktionen aus, da es eine Trennung zwischen Event-Service und Aktionskomponenten geben soll.

Eine einfache und klare Trennung zwischen Ereignis und Bedingung ist auf Ebene des Event Services nicht möglich. Der Event Service müsste hierfür erkennen können, ob es sich bei der Kombination aus Ereignis und Bedingung um ein neues Event handelt oder ob es bereits Events gibt, die aus dieser Kombination von Ereignis und Bedingung bestehen.

Eine Bedingung ist nach [Dittrich & Gatziu 2000] „ein Prädikat über dem Datenbankzustand". Da Nexus ein verteiltes System ist, ist der „Datenbankzustand" über viele Systeme verteilt und in der Regel nicht nur auf dem Event Service zu finden. Der Event Service muss mit anderen Diensten kommunizieren, um eine Bedingung zu überprüfen. Hierdurch kann es zu Leistungsproblemen beim Event Service und bei den Event Sources kommen. Es ist auch denkbar, dass die Netzlast hierdurch sehr stark erhöht wird. Werden aber nur Events verschickt und der Event Service betrachtet nur Kombinationen aus Events, so ist er in der Lage ohne weitere Anfragen an eine Event Source einen komplexen Event zu erkennen. Er muss hierfür nur bisher aufgetretene Events speichern. Die notwendige Kommunikation ist daher sehr viel geringer als bei der Verwendung von Bedingungen.

Eine Verwendung von ECA-Regeln zwischen Event Source und Event Service ist ebenfalls nicht sinnvoll.

- Betreiber möchten die angebotenen Events auf bestimmte Eventarten begrenzen, was bei der Verwendung von ECA-Regeln schwierig ist.
- Eine Event Source muss nicht auf einer Datenbank aufbauen, die das Triggerkonzept unterstützen. Ein Implementierung von ECA-Regeln in solch eine Event Source ist aufwendig und rentiert sich oft nicht.
- Das Triggerkonzept wird zwar im SQL 99 Standard definiert, allerdings implementieren viele DBMS nur eine Untermenge dieses Triggerkonzeptes. Die Definition eines Triggers ist daher nicht portabel (vgl. [CerI, Cochrane & Widom 2000]).
- Bei der Verwendung von ECA-Regeln können nur Events erzeugt werden, die beschrieben werden können, während die von Nexus verwendete Methode alle Events ermöglicht, die implementiert werden können. Man betrachte beispielsweise folgendes Basic Event: ein Event soll erst ausgeben werden, nachdem ein Benutzer an einem Desktop-System das Event gesehen hat und sein Einverständnis für die Ausgabe dieses Events gegeben hat. Dies ist wohl implementierbar, aber wie soll man dieses Event beschreiben, wenn die Beschreibungssprache von einem Datenbanksystem ausgeht und andere Systeme nicht kennt?

Spatial Model Server und Location Server unterscheiden sich auch bei den Events, die sie auslösen. Ein Location Server kann räumliche Events bereitstellen. Dürfen sich zwei Objekte nur bis zu einer bestimmten Entfernung nähern, so kann man mit Hilfe eines Events ein Warnsystem erstellen, das bei einer bestimmten Entfernung der beiden Objekte Alarm schlägt. Räumliche Events sind bei einem Spatial Model Server wenig sinnvoll, da sich die Position von statischen Objekten nicht ändert. Das Event wird dadurch entweder bei der Registrierung sofort ausgelöst oder es wird nie ausgelöst. Spatial Model Server dürften deshalb in der Regel nur „technische" Events haben, die beim Einfügen, Löschen oder Ändern eines oder mehrerer Objekte ausgelöst werden. Location Server werden stattdessen beide Arten unterstützen müssen.

2.6 Augmented World Model

Das Augmented World Model ist ein objekt - orientiertes Informationsmodell für mobile ortsbasierte Anwendungen. Es handelt sich dabei um ein Modell der realen Welt, das um virtuelle Objekte angereichert (augmented = angereichert) wird (vgl. [Nicklas et al 2001]). Es entsteht durch die Zusammenführung aller Augmented Areas [Hohl et al 1999]. Die Klassen der Objekte werden in sogenannten Class Schemas festgelegt. Da es unmöglich ist, sämtliche möglichen Klassen für sämtliche möglichen Nexus-Anwendungen zu definieren, existiert ein Standard Class Schema, das die Basis für alle Schemas ist, und bei Bedarf erweitert werden kann. Class Schemas werden als XML-Schema definiert. Für den Datenaustausch wurden zwei Sprachen definiert: Eine Sprache zur Darstellung der Objekte einer Augmented World (die Augmented World Modeling Language) und eine Anfragesprache (die Augmented World Query Language). Die Sprachen sind XML-Varianten. Für die Positionsinformation wird bei beiden Sprachen GML verwendet. GML ist eine XML Variante für zweidimensionale geographische Daten. Spezifikation und Informationen zu GML können unter [GML 2.0] gefunden werden. Neben AWQL und AWML, die näher vorgestellt werden, existieren noch weitere Sprachen:

- Die Change Report Language (CRL), die bei Einfüge, Lösch- und Änderungsoperationen den Erfolg oder Misserfolg der Operation auf den betroffenen Objekten enthält.
- Die Augmented Area Description Language (AADL), die Augmented Areas beschreibt
- Die Map Predicate Language (MapPL) wird bei der Generierung von Karten benutzt und beschreibt wie die benötigte Karte auszusehen hat.

2.6.1 Standard Class Schema

Das Standard Class Schema definiert die grundlegenden Klassen für die Nexus-Anwendungen. Hierbei sind die Klassenhierarchie und die Attribute der Klassen festgelegt. Die enthaltenen Klassen entstammen aus Use Cases und Szenarios wie sie in ([Nicklas & Mitschang 2001]) beschrieben sind. Das exakte Aussehen wird in [Meßmer 2001] festgelegt und besteht aus über 120 Klassen. Bei der Definition der Attribute entstand das Problem, dass man entweder ein möglichst einfaches oder ein möglichst vollständiges Modell erstellen kann. Bei einem einfachen Modell enthalten die

Klassen nur die notwendigsten Attribute, was dazu führt, dass reale Anwendungen die Klassen um weitere Attribute erweitern müssen und die Dienste aufgrund der vielen unteschiedlichen Schemas nur sehr wenig Daten austauschen können. Bei einem möglichst vollständigen Modell enthalten die Klassen sehr viele Attribute. Da die Nexus-Dienste alle Klassen mit allen Attributen unterstützen sollen, ist der Aufwand für die Entwicklung eines solchen Dienstes sehr hoch. Ein weiteres Problem entsteht, wenn der Nexus-Dienst keine sinnvollen Werte für bestimmte Attribute einer Klasse finden kann. Was soll man bei einem Attribut Baujahr angeben, wenn man das Baujahr nicht kennt und es für die Nexus-Anwendung auch unwichtig ist? Statt nun für jedes Attribut ein Null Value einzuführen, wird das Konzept der optionalen Attribute eingeführt. Ein optionales Attribut muss nicht implementiert werden. Benötigt ein Nexus-Dienst ein solches Attribut, so ist der Name und der Typ bereits definiert und kann einfach übernommen werden. Dies wird von XML sehr gut unterstützt, da auch hier die Elemente so definiert werden können, das sie optional sind. Ein großer Teil der Attribute im Standard Schema sind optionale Attribute. Hierdurch sind die Dienste in der Lage sehr viele Daten austauschen zu können, während die Dienste einfach bereitgestellt werden können.

2.6.2 Extended Class Schema

Da das Standard Class Schema nicht alle möglichen Klassen mit allen möglichen Attributen enthalten kann und soll, wird ein Mechanismus benötigt um das Standard Class Schema zu erweitern. Hierfür werden Extended Class Schemas benutzt. Benötigt man weitere Klassen, so lässt man diese einfach von der am nächsten kommenden Klasse des Standard Class Schema erben. Man benutzt hierfür den Vererbungsmechanismus bei XML-Schemas.

2.6.3 Augmented World Modelling Language (AWML)

Die Augmented World Modelling Language (AWML) wird für die Darstellung der Informationen über die Augmented World. Ein AWML-Document ist ein XML-Dokument. Es besteht aus einer Folge von Objekten aus der Augmented World. Die Spezifikation befindet sich in [Großmann et al 2001]. AWML hat zwei wichtige Funktionen. Es dient zum Datenaustausch zwischen unterschiedlichen Diensten und wird deshalb für das Einfügen von Elementen in einen Spatial Model Server benutzt. Es wird außerdem bei AWQL-Anfragen zur Modellierung des Ergebnisses benutzt. Ein Beispiel:

```
<awml>
  <nexusobject type="room" nol="nexus://..." kind="virtual">
    <number>2.008</number>
    <!-- extent gibt den Umriss des Raums an und ist ein Polygon.
         Polygone werden in GML angegeben -->
    <extent>
      <gml:polygon>
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coordinates>
              40.0,40.0 40.0,35.0 35.0,35.0 35.0,40.0 40.0,40.0
            </gml:coordinates>
          </gml:LinearRing>
```

```
        </gml:outerBoundaryIs>
      </gml:polygon>
    </extent>

    <-- Bei der Position des Raums wird ein Punkt innerhalb von
        extent verwendet. Bei AWQL kann man Objekte nach Entfernung
        sortieren, wobei neben extent auch position verwendet
        werden kann. Position sollte daher sinnvoll gesetzt sein.
        Ein GML Point repräsentiert die Position. -->
    <position>
      <gml:Point>
        <gml:Coord>
          <gml:X>37.0</gml:X> <gml:Y>37</gml:Y>
        </gml:Coord>
      </gml:Point>
    </position>
  </nexusobject>

  <!-- Beim folgenden Element handelt es sich um einen Sender mit
     dem Namen Sender 1, der die ID 1 ausstrahlt und dessen
     Sichtbereich angegeben wird -->
  <nexusobject type="emitter" nol="nexus://..." kind="real">
    <name>Sender1</name>
    <emittedid>1</emittedid>
    <!-- der Sichtbereich ist ein Polygon im GML Format -->
    <fieldOfVision>
      <gml:polygon>
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coordinates>
              40.0,40.0 40.0,35.0 35.0,35.0 35.0,40.0 40.0,40.0
            </gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:polygon>
    </fieldOfVision>
    <-- Die Position an der sich der Sender befindet -->
    <position>
      <gml:Point>
        <gml:Coord>
          <gml:X>37.0</gml:X> <gml:Y>37</gml:Y>
        </gml:Coord>
      </gml:Point>
    </position>
  </nexusobject>
</awml>
```

Abbildung 2-5: AWML Beispiel

2.6.4 Augmented World Query Language (AWQL)

Anfragen an die Föderation werden mit Hilfe der Augmented World Query Language gestellt, in der man die Eigenschaften der gesuchten Objekte und gewünschte Operationen angibt. Das Löschen und Verändern von Objekten ist ebenfalls mit AWQL möglich. AWQL ist eine objekt - orientierte Anfragesprache. Dies bedeutet, dass es Konzepte wie Polymor-

phismus kennt. Verlangt man beispielsweise alle Objekte der Klasse Raum, so wird man auch alle Objekte mit einer Unterklasse von Raum erhalten. AWQL wird benutzt, da SQL komplexer ist und die Aufgabe der Föderierung erschwert würde. XML-Anfragesprachen wie QUILT dienen dazu Daten aus XML-Dokumenten zu extrahieren (vgl. [Nicklas et al 2001]. Die exakte Definition befindet sich in [Großmann et al 2001].

AWQL hat folgende Bestandteile:

- Man kann eine Liste mit Nexus Augmented Area Locator (NAAL) angeben und so die betroffenen Rechner direkt bestimmen. Hierdurch kann eine Anwendung der Föderation die sonst notwendige Suche nach den betroffenen Spatial Model Servern ersparen und den Suchraum auf Daten spezieller Anbieter beschränken.
- Mit Hilfe eines Scope Elements, das eine Liste von Extended Class Schemas enthält, wird angezeigt welche Schemas die Nexus-Anwendung versteht.
- Ein `<restriction>` Element enthält die Bedingungen, die ein Objekt erfüllen muss, damit es zur Ergebnismenge gehört. Als Bedingungen kann man mit den Elementen `<equal>` bestimmen, welchen Wert ein bestimmtes Attribut haben soll. Das Element `<in>` wird benutzt wenn ein Attribut einen Wert aus einer angegebenen Wertemenge besitzen soll. Mit Hilfe der geometrischen Relationen `<inside>` und `<overlap>` lassen sich Objekte auswählen, die innerhalb eines Polygons sind bzw. sich mit einem Polygon überschneiden. Mit Hilfe von `<and>`, `<or>` und `<not>` lassen sich Anfragen mit mehreren Bedingungen erzeugen.
- Da nicht alle Anwendungen alle Werte eines Objektes benötigen, lassen sich die Angaben mit Hilfe des Elements `<filter>` auf die gewünschten Elemente eines Objekts beschränken. Hierfür gibt man mit `<includes>` die gewünschten Elemente und mit `<excludes>` die unerwünschten Elemente an. Das Element `<includeallother>` dient dazu alle Elemente einzutragen, die nicht durch das `<excludes>` Element ausgeschlossen wurden. `<excludeallother>` entfernt alle Attribute eines Objektes, die nicht in `<includes>` stehen.
- Ein `<closest>`-Prädikat begrenzt die Objekte auf die n nächsten Objekte zu einer Position.
- Das `<generalization>` Element dient dazu, Details eines Attributes zu entfernen.
- Mit Hilfe des `<aggregation>` Elements lassen sich Objekte zu einem größeren Objekt verschmelzen
- Ein `<update>` Element enthält die neuen Werte der Attribute, die geändert werden sollen.

Die folgende Anfrage soll als Ergebnis alle Sender und Räume zurückliefern, die sich innerhalb eines bestimmten Gebiets aufhalten. Es sollen nur die 10 nächsten Objekte zurückgeliefert werden. Außerdem soll nur das Attribut Position im Ergebnis zurückgeliefert werden.

```
<awql>
  <restriction>
```

```
    <and>
      <-- Die Objekte müssen Sender oder Räume sein -->
      <or>
        <equal>
          <attr name =”type”/>
          <nexusdata>emitter</nexusdata>
        </equal>
        <equal>
          <attr name = “type”/>
          <nexusdata>room</nexusdata>
        </equal>
      </or>
      <-- Die Objekte müssen sich innerhalb des Polygons
          befinden -->
      <inside>
        <gml:polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>
                40.0,40.0 40.0,35.0 35.0,35.0 35.0,40.0 40.0,40.0
              </gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:polygon>
      </inside>
    </and>
  </restriction>
  <-- Nur die 10 nächsten Objekte zur angegebenen Positon sollen
      zurückgeliefert werden -->
  <closest num = “10“>
    <nexusdata>
      <position>
        <gml:Point>
          <gml:Coord>
            <gml:X>37.0</gml:X> <gml:Y>37</gml:Y>
          </gml:Coord>
        </gml:Point>
      </position>
    </nexusdata>
  </closest>
  <-- Alle Attribute bis auf position sollen herausgefiltert
      werden -->
  <filter>
    <includes>
      <attr name = “position”/>
    </includes>
    <excludeallother/>
  </filter>
</awql>
```

Abbildung 2-6: Beispiel einer AWQL Anfrage

Das Ergebnis der Anfrage könnte die obige Beispiel-AWML sein.

2.6.5 Zusammenfassung

Das Nexus-Projekt entwickelt eine offene Plattform, die Dienste für ortsbasierte Anwendungen mit mobilen Anwendern anbietet. Sie besteht aus Nexus-Anwendungen, Nexus-Knoten und Nexus-Diensten. Nexus-Knoten bieten den Nexus-Anwendungen eine Sicht auf das Augmented World Model an. Das Augmented World Model ist ein Modell der realen Welt, das um virtuelle Objekte angereichert ist. Das Augmented World Model ist in Augmented Areas aufgeteilt. Die Objekte des Augmented World Models werden in AWML modelliert. Nexus-Anwendungen stellen Anfragen in AWQL an einen Nexus-Knoten, der daraus AWQL Anfragen an Nexus-Dienste macht. Das Ergebnis lesender Anfragen sind Objekte, die in AWML beschrieben sind. Die statischen Objekte der Augmented Areas werden von Spatial Model Servern gespeichert. Spatial Model Server können eine Event Source für den Event Service sein und Basic Events anbieten. Im Gegensatz zu einem Location Service sind räumliche Events nicht sinnvoll, sondern nur „technische” Events. Die gespeicherten Daten eines Spatial Model Servers werden nicht nur von Nexus-Anwendungen benötigt, sondern können auch Daten für andere Komponenten der Nexus-Plattform speichern. Ein solches Beispiel ist die Sensorkomponente der mobilen Geräte. Sie kann zur Positionierung Informationen benötigen, die in einem Spatial Model Server gespeichert werden können. Das nächste Kapitel beschreibt Positionierungssysteme und zeigt welche Daten von den Systemen in einem Spatial Model Server gespeichert werden können.

3 Positionierung

Dieses Kapitel beschreibt auf welche Art und Weise eine Positionierung in Nexus erfolgen kann. Hierfür wird die Positionierung in Nexus beschrieben, Positionierungssysteme vorgestellt und mögliche Bezugssysteme beschrieben. Danach werden die Positionierungssysteme und Bezugssysteme ausgewählt, die durch einen Spatial Model Server für Indoor Bereiche unterstützt werden können.

3.1 Positionierung in Nexus

In Nexus ist die Sensorkomponente auf den mobilen Rechnern für die Positionierung zuständig. Es existieren Positionierungssysteme bei denen die Sensorkomponente eine Positionierung nicht selbständig durchführen kann, sondern die Hilfe weiterer Systeme benötigt. Benötigt die Sensorkomponente zur Positionsbestimmung Informationen über statische Objekte, so lassen sich diese Informationen in einem Spatial Model Server speichern. Die Genauigkeit und die Geschwindigkeit, in der sie die Position benötigen, hängen von der Anwendung ab. Ein Wanderer der nur hin und wieder seine Position feststellen möchte, wird wahrscheinlich eine Abweichung von 100 Metern und eine Positionierungsdauer von mehreren Sekunden bis zu wenigen Minuten akzeptieren, eine Anwendung für die Landesvermessung müsste genauer sein und eine Ungenauigkeit von wenigen Metern oder noch weniger haben. Bewegen sich Anwende, so ist oft eine sehr viel schnellere Positionierung notwendig. Dies gilt beispielsweise bei Navigationssystemen. Benötigt man z.B. 10 Sekunden um eine Position zu bestimmen, so macht das bei einer Person, die sich mit einem Meter pro Sekunde fortbewegt 10m aus. Sollte das Programm die Person zu einem Raum führen, so ist es sehr wahrscheinlich, dass die Person an dem Raum vorbeiläuft. Dauert die Positionsbestimmung lange und müsste sie für jede Anfrage durchgeführt werden, so würden viele Anwendungen keine Akzeptanz finden, da die Wartezeit zu groß wäre. Die Positionierung muss also sehr schnell, d.h. innerhalb wenigen Sekunden, erfolgen.

3.2 Positionierungssysteme

3.2.1 Überblick

Es gibt Outdoor-Systeme, die große Gebiete abdecken können und Indoor-Systeme, die für die Anwendungen innerhalb eines Gebäudes ausgelegt sind. Die Sensorkomponente wird mehrere Positionierungssysteme unterstützen müssen, da Outdoor-Systeme innerhalb eines Gebäudes nicht oder nicht sinnvoll verwendet werden können. Indoor-Systeme können zwar prinzipiell auch außerhalb eines Gebäudes benutzt werden, werden dort aber nur für kleine Gebiete eingesetzt. Bei Indoor Systemen werden für die Positionierung in der Regel Sender oder Empfänger aufgestellt. Die Anschaffungskosten und Wartungskosten sind bei großen Gebieten sehr hoch und höher als bei einem entsprechenden Gebiet innerhalb eines Gebäudes, da die Systeme sehr viel robuster sein müssen, da sie beispielsweise Witterungseinflüssen ausgesetzt sind. Eine flächendeckende Verfügbarkeit solcher Systeme ist nicht gewährleistet, da es für viele Gebiete keine Betreiber gäbe. Neben den Systemen, die

ausschließlich für eine Positionierung innerhalb oder außerhalb eines Gebäudes geeignet sind, gibt es noch Systeme, die sowohl innerhalb als auch außerhalb eines Gebäudes verwendet werden können. Es existieren darüberhinaus Hilfssysteme, die alleine für eine Positionierung ungeeignet wären, aber durch die andere Positionierungssysteme unterstützt werden können.

Systeme lassen sich auch dahingehend unterteilen, ob die Position eines Objekts von einer Systemkomponente beim Objekt ermittelt wird oder von einer Systemkomponente, die sich nicht auf dem Objekt befindet. Befindet sich die Systemkomponente auf dem Objekt, so wird von Selbstpositionierung gesprochen. Hierbei empfängt eine Systemkomponente beim Objekt Signale aus der Umgebung und kann so die Position ermitteln. Wird die Position nicht vom Objekt, sondern von einer externen Komponente bestimmt, so spricht man von externer Positionierung oder Tracking. Das Objekt sendet Signale aus, die von der externen Komponenten empfangen werden. Die externe Komponente kann hierdurch erkennen, welches Objekt die Signale aussendet und wo sich das Objekt befindet. Möchte das Objekt seine Position erfahren, so muss es hierzu mit der externen Komponenten oder einer anderen Komponente kommunizieren, die die Position speichert. Selbstpositionierungssysteme und Trackingsysteme unterscheiden sich in ihrem Aufbau. Beim Tracking fallen alle Positionsdaten extern an und werden in der Regel an ein zentrales System verschickt, das die Daten speichert, auswertet und an das Objekt schickt, falls dieses die eigene Position anfordert. Hierdurch entstehen Skalierungs- und Sicherheitsprobleme. Bei einer Selbstpositionierung fallen die Positionsdaten lokal beim Objekt an und werden dort verwaltet. In der Regel kennt keine weitere Komponente die Position des Objekts. Selbstpositionierung hat dadurch keine Skalierungs- und Sicherheitsprobleme wie Tracking. Die prinzipiell verteilte Datenhaltung hat den Nachteil, dass Anfragen über die Position von Objekten nur schwer zu realisieren sind [Schiele 1999]. Eine Anfrage, ob sich zwei Objekte innerhalb eines Bereichs befinden, ist nur möglich, wenn man die Position von beiden Objekten erhalten kann. Dies kann aus Sicherheitsgründen verwehrt sein oder weil eine Kommunikation mit der Nexus-Anwendung nicht möglich ist.

Es existieren unterschiedliche Positionierungsmethoden. Wird ein räumlicher Bereich in Zellen unterteilt, so spricht man von einer zellenbasierten Methode. Bei der Methode der Triangulation über Signallaufzeiten oder Signalstärken werden die Zeit bis ein Signal sein Ziel erreicht oder die Signalstärke gemessen. Mit Hilfe von Triangulation mehrerer dieser Werte wird auf die Position geschlossen. Bei einer Bewegungsmessung ist der Startpunkt eines Objekts bekannt und die Bewegungen eines Objekts werden überwacht. Da Messungenauigkeiten auftreten, verschlechtert sich die Positionierung bei der Verwendung über einen längeren Zeitraum. Die Systeme müssen daher ihre interne Position mit der realen Position an Synchronisationspunkten abgleichen. Eine andere Methode besteht darin, die Umgebung zu analysieren und so auf die eigene Position zu schließen. Die Beobachtung von Ereignissen in der realen Welt kann auch zu einer Positi-

onierung benutzt werden. Wird beispielsweise das Einloggen an einem Terminal registriert, so kann man, wenn ein Benutzer sich einloggt, darauf schließen, dass er sich bei dem Terminal aufhält.

Die Positionsinformation kann unterschiedliche Bedeutung haben. Bei Systemen wie GPS handelt es sich um eine geometrische Position. Bei einer symbolischen Positionierung ist die empfangene Position einem Objekt zugeordnet. Die empfangene Position kann z.B. bedeuten, dass man innerhalb eines Raums ist oder vor einem Objekt steht. Nicht bei jedem Positionierungssystem wird die Position direkt ermittelt. Werden Sender benutzt, so muss ein Sender nicht unbedingt eine Position ausstrahlen, sondern er kann auch nur eine eindeutige ID ausstrahlen. Die ID ist einer geometrischen oder symbolischen Position zugeordnet. Empfängt ein Client eine solche ID, so muss er erst die dazugehörige Position ermitteln.

Systeme mit denen sich Positionen ermitteln lassen, können neben der Positionsbestimmung noch weitere Aufgaben erfüllen. Sender von Positionierungssystemen müssen nicht nur Positionsinformationen bereitstellen, sondern können bei einigen Systemen auch zusätzliche Information aussenden. Kommunikationsnetzen wie z.B. GSM dienen der Datenübertragung und ermöglichen auch eine Positionierung.

Neben den hier vorgestellten kontaktfreien Positionierungssystemen, gibt es auch kontaktbehaftete Positionierungssysteme, bei denen der Anwender beispielsweise eine Chipkarte oder eine Magnetkarte in einen Chipkartenleser bzw. Magnetkartenleser einschieben muss. Das bekannteste kontaktbehaftete Positionierungssystem ist wohl die Stechuhr mit der festgehalten wird, zu welcher Zeit ein Arbeitnehmer im Betrieb gearbeitet hat.

3.2.2 Outdoor-Systeme

GPS/DGPS

GPS besteht aus 24 Satelliten, die Signale aussenden. Zur Positionierung muss ein GPS-Empfänger mindestens 4 Satelliten empfangen können. GPS hatte für zivile Nutzung ursprünglich nur eine Genauigkeit zwischen 40 und 100 Metern. Am 2. Mai 2000 wurde die Selective Availability (eine künstliche Verfälschung) abgeschaltet, aufgrund dessen die Position genauer ermittelt werden kann. Die Genauigkeit beträgt für zivile Nutzer nun zwischen 12 und 30 Metern. DGPS ist genauer. Bei einem Test im Bohnenviertel in der Stuttgarter Innenstadt hatte DGPS eine durchschnittliche Genauigkeit von 3 Metern und die größte Abweichung betrug 10 Meter [Fritsch, Klinec & Volz 2001]. Diese Genauigkeit ist für Anwendungen außerhalb von Gebäuden ausreichend. Bei GPS handelt es sich um ein System zur Selbstpositionierung, das eine Triangulation über Signallaufzeiten als Positionierungsmethode verwendet. Es handelt sich um eine geometrische Positionierung, für die keine weiteren Systeme notwendig sind. Ein Problem von GPS ist, dass bei herkömmlichen GPS-Empfänger die Sicht zu den Satelliten nicht durch Gebäude blockiert sein darf, da sonst die Signale nicht empfangen werden können.

Das Global Positioning System (GPS) und Differential GPS (DGPS) sind die verbreitetsten Systeme zur Positionsbestimmung außerhalb von Gebäuden. Es ist weltweit verfügbar, wetterunempfindlich und die Empfangsgeräte können sehr klein sein. GPS und DGPS werden im Rahmen von Nexus als Hauptpositionierungssensoren außerhalb von Gebäuden verwendet.

GSM Netzwerk

GSM (Global System for Mobile Communications) ist ein Standard für Funknetze und ist in Europa sehr verbreitet. Der GSM Dienst ist ein Netzwerk, das aus Zellen aufgebaut ist. Es kann nicht nur zur Kommunikation, sondern auch zur Positionierung dienen. Der GSM Provider weiß bei der Kommunikation in welcher Zelle sich das mobile Gerät aufhält. Wird diese Information zur Verfügung gestellt, so kann man die Position auf eine Zelle genau bestimmen. Die Größe einer Zelle beträgt in dicht bebauten Gebieten ungefähr 100 m. In ländlichen Regionen können die Zellen auch mehrere Kilometer groß sein. Es ist damit ungenauer als GPS oder DGPS und nur in den Fällen sinnvoll, in denen keine genaueren Positionierungssysteme vorhanden sind, das Gerät nur GSM unterstützt oder die Genauigkeit ausreicht. Eine Verbesserung lässt sich erzielen, indem man die Signallaufzeit zwischen Antenne und GSM-Telefon verwendet wird und durch einen „Schnitt von Kreisen" auf die Position geschlossen wird.

Interessanterweise hat die FCC (US Federal Communications Commision) beschlossen, das ab dem 1. Oktober 2001 die Position einer Person, die die 911 Notrufnummer anruft, bestimmt werden können muss [FCC E911]. Dies gilt allerdings nur für Besitzer eines modernen Handys. Die Genauigkeit muss in 67% aller Fälle bei 50 m liegen. Es dürften daher im Bereich der Funknetze und Handys, sehr bald bessere Positionierungssysteme für den amerikanischen Markt verfügbar sein. Leider sind die technischen Probleme so groß, das dies bisher nicht verwirklicht werden konnte und die Netzbetreiber einen Aufschub dieser Frist erhalten haben. Der Termin ab dem 95% aller Personen, die die 911 Notrufnummer wählen, bestimmt werden muss, ist bestehen geblieben und ist der 31. Dezember 2005.

3.2.3 Indoor-Systeme

Indoor GPS

Innerhalb eines Gebäudes können die GPS-Signale der GPS-Satelliten nicht durch herkömmliche GPS-Empfänger empfangen werden, da das Signal zu schwach ist. Möchte man GPS innerhalb eines Gebäudes benutzen, so gibt es zwei Möglichkeiten, entweder verwendet man sehr viel bessere GPS Empfänger oder man stellt Systeme auf, die GPS-Signale aussenden.

Die Verwendung verbesserter Empfänger ermöglicht es, auch innerhalb eines Gebäudes die GPS-Signale zu empfangen. Die Positionsgenauigkeit ist allerdings in der Regel nicht ausreichend, da Mehrwegeffekte die Genauigkeit verschlechtern und trotz der verbesserten Empfänger nicht überall ein GPS-Signal empfangen werden kann. Die Genauigkeit wäre aber selbst ohne Störungen zu schlecht, da bei einer Genauigkeit zwischen

10 und 30 m keine raumgenaue Positionierung möglich ist. Selbst wenn DGPS möglich wäre, ist keine raumgenaue Positionierung möglich. Das Problem besteht nicht nur in der maximalen Abweichung von 10 m. Da die Wände nur wenige Zentimeter dick sind und die Gänge nur wenige Meter breit sind (2-3 m), ist selbst bei 3 m eine raumgenaue Positionierung oft nicht möglich, da man nicht sicher sein kann, ob sich der Anwender auf einem Gang, in einem Raum oder in einem der angrenzenden Räume befindet. Abbildung 3-1 zeigt ein solches Beispiel bei dem eine Position prinzipiell mehreren Räumen zugeordnet werden kann. Der Anwender kann sich hierbei sowohl in einem der Räume 1 bis 4 oder auf dem Gang aufhalten.

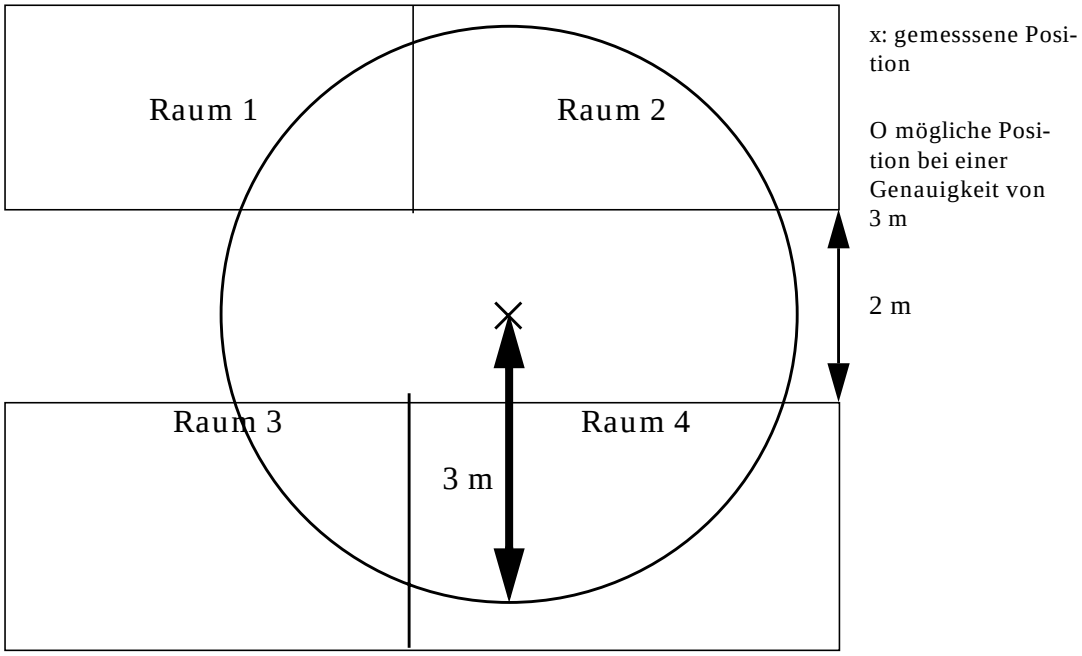


Abbildung 3-1: Positionierungsproblem mit Indoor GPS

Es wird deshalb bei solchen Indoor Systemen nicht nur die Positionierung mit Hilfe von GPS unterstützt, sondern auch durch weitere Systeme. Ein solches System ist Assisted GPS [Djuknic & Richton 2001].

Ein System bei dem Sender aufgestellt werden, die GPS Signale aussenden wird beispielsweise in [Kee et al 2001] beschrieben. Hier hat man Probleme mit Mehrwegeffekten, der Signalstärke bei unterschiedlichen Entfernungen und Probleme mit der Uhrdrift von Computeruhren. Es wurden Tests mit statischen und bewegten Objekten innerhalb eines Raums durchgeführt, die Positionsangaben mit einer Abweichung von wenigen Millimetern ermöglichten. Hierbei befanden sich Sender und Empfänger in einem Raum. Befinden sich die Sender und Empfänger an unterschiedlichen Räumen werden die Mehrwegeffekte sehr viel stärker und die Positionierung erheblich verschlechtert.

Infrarot Systeme

Infrarot Systeme können sowohl als Tracking Systeme als auch als Selbstpositionierungssysteme verwendet werden. Das Active Badge System ist ein Tracking System, das von Olivetti Research Ltd. entwickelt wurde. Es dient zur Positionierung von Personen innerhalb eines Gebäudes. Jede Person trägt dabei einen kleinen Sender, der die Form eines Namensschildchen hat und wird bei Personen „Active Badge" und bei Objekten „Equipment Badge" genannt. Der Sender ist ein Infrarot Sender, der alle 10 s eine ID aussendet. Im Gebäude befinden sich Sensoren, die die ID empfangen. Jeder dieser Sensoren ist mit einem Positionierungsrechner verbunden, der mit Hilfe der ID des Sensors und des Senders die Position einer Person bestimmen kann. Es ist fraglich ob es ein geeignetes Indoor Positionierungssystem für Nexus ist, da die Kosten schnell steigen und die Anwenderakzeptanz möglicherweise nicht sehr groß sein wird. Das System ist daher nur für wenige Einsatzszenarien wie Flughäfen oder Museen sinnvoll [Fritsch, Klinec & Volz 2001].

Bei Selbstpositionierungssystemen auf Infrarot Basis werden Infrarotsender aufgestellt, die eine ID aussenden. Empfängt der Anwender eine solche ID, so muss sie in der Regel in eine Position umgewandelt werden, da es sich um eine symbolische Positionierung handelt. Es gibt Systeme wie z.B. das der Firma Eyeled bei dem nur eine wenige Byte große ID ausgesandt wird. Andere Systeme sind in der Lage sehr viel größere Datenmengen auszusenden. Es lassen sich dann zusätzliche Daten aussenden, wie z.B. Informationen über das Objekt an dem sich der Sender befindet. Ein Beispiel für ein solches Indoorpositionierungssystem ist das IRREAL System des REAL Projekts [Wahlster et al 2001]. Prinzipiell ist es hierbei auch möglich die geometrische Position auszusenden, wodurch ein mobiler Client sich direkt positionieren kann.

Der Anwender benötigt einen mobilen Rechner mit einer Infrarotschnittstelle, um die ID zu empfangen. Die meisten mobilen Rechner besitzen nur eine Infrarotschnittstelle und können nur einen Sender auf einmal empfangen. Da die IDs in der Regel nur aus wenigen Bytes bestehen und die Übertragung relativ schnell sein kann (mehrere tausend Baud), wird nur ein Bruchteil einer Sekunde für die Übertragung einer ID benötigt. Man kann dies Nutzen, um mehrere Sender zu empfangen. Hierfür müssen die Sender nachdem sie eine ID ausgesendet haben eine kurze Zeit warten bis sie wieder ihre ID aussenden. Innerhalb dieses Intervalls kann die ID eines anderen Senders gelesen werden. Bei einer Positionsbestimmung muss man nur eine kurze Zeit warten bis man mehrere IDs empfangen hat und kann mit Hilfe dieser IDs eine Positionsbestimmung durchführen. Prinzipiell wäre es auch möglich die IDs in unterschiedlichen Frequenzen zu senden und so mehrere Sender gleichzeitig empfangbar zu machen. Standardkomponenten in mobilen Geräten unterstützen dies allerdings nicht.

Eigenschaften von Infrarotstrahlen entsprechen denen von Licht. Infrarotstrahlen können reflektiert werden. Mit Hilfe von Spiegeln kann man einen Infrarotstrahl in einen Bereich reflektieren, der nicht direkt beleuchtet wer-

den kann. Die Strahlen werden in der Regel ungewollt reflektiert und verursachen so Positionierungsprobleme. Da sie leicht blockiert werden können, kann man die Strahlen sehr leicht bündeln. Sehr oft trennen Hindernisse unterschiedliche Gebiete, wodurch der Empfang eines Senders automatisch auf ein solches Gebiet begrenzt ist. Eine raumgenaue Positionierung ist hiermit möglich. Hindernisse wie Tische sorgen aber auch dafür, dass nicht alle Teile eines Raumes ausgeleuchtet werden können. Außerdem besteht die Gefahr, dass eine Person ein oder mehrere Sender verdeckt und eine Positionierung verhindert. Durchsichtige Gegenstände wie Glasscheiben blockieren die Strahlen nicht. Die Sender können daher mit Hilfe von Glas oder durchsichtigem Kunststoff leicht geschützt werden. Viele Räume haben Fenster, wodurch die ID ungewollt in Räume anderer Gebäude gesendet werden kann. Da die Strahlen gerichtet sind, müssen die Empfänger immer in die Richtung der Infrarotstrahlen gehalten werden. Hierdurch kann man die ungefähre Richtung ermitteln, in die der Anwender blickt. Manche Geräte haben ihre Infrarotempfänger an der Seite und können die Infrarotstrahlen von Sendern auf der gegenüberliegenden Seite nicht empfangen. Sie müssen gedreht werden, was eine Bedienung eines Programms sehr erschwert. Da IR-Geräte bei unterschiedlichen Geräten an unterschiedlichen Seiten vorkommen, ist es nicht möglich Infrarotsender so aufzubauen, dass sie unter Garantie empfangen werden können.

Ein weiteres Problem besteht, wenn eine Anwendung nur Infrarotsignale akzeptieren kann, wenn sie nach einem Protokoll gesendet werden, die Infrarotsender die Werte nach einem anderen oder gar keinem Protokoll senden.

Da viele mobile Geräte wie z.B. die Palm-PDAs und viele Laptops standardmäßig mit einer Infrarotschnittstelle ausgestattet sind, handelt es sich hierbei um eine sehr wichtige Positionierungsmöglichkeit für Nexus. Die Kosten für Sender und Empfänger sind relativ niedrig, solange man mit der Qualität von Standardprodukten auskommt.

Ultraschall Systeme

Es gibt Trackingsysteme und Selbstpositionierungssysteme auf Ultraschallbasis. Trackingsysteme können eine hohe Genauigkeit (Millimeter genau) erreichen und werden beispielsweise benutzt, wenn Bewegungen für Filmanimationen digitalisiert werden sollen. Systeme mit solch einer hohen Genauigkeit, und hohen Updateraten haben aber ihren Preis. Solche Systeme kosten selbst bei Räumen die nur wenige Meter groß sind zwischen ein paar tausend und mehreren zehntausend Euro. Solche Systeme sind nur für die Positionierung in kleinen Bereichen sinnvoll.

Bei einer Selbstpositionierung wird die Zeit gemessen bis ein ausgesendetes Signal wieder beim Gerät angekommen ist und wird in der Natur beispielsweise von Fledermäusen benutzt. Es wird hierbei eine relative Positionierung verwendet, bei der die Abstände zu Hindernissen gemessen werden. Kennt man die Umgebung, so kann man auch die absolute Positon feststellen. Eine Positionierung mit Ultraschall kann, wie man am Beispiel von Fledermäusen sehen kann, sehr genau sein. Ultraschall hat Eigenschaften, auf die bei einer

Verwendung geachtet werden muss. Die Zeit zwischen Aussenden und Empfang eines Signals sind sehr kurz, für kurze Abstände (<20 cm) können Probleme bei der Positionsbestimmung auftreten. Ein weiteres Problem sind Echos, die falsche Messwerte erzeugen können. Ist kein Objekt innerhalb des notwendigen Abstands, so ist auch keine Positonierung möglich. Die Positionierung kann durch andere Ultraschallquellen wie Ultraschallsender anderer Rechner gestört werden.

Verwendet werden Ultraschallsensoren beispielsweise von den Nomad Super Scout Robotern, die als Fußballroboter im Robotiklabor der Fakultät Informatik an der Universität Stuttgart verwendet werden. Die effektive Reichweite ihrer Ultraschallsensoren liegt zwischen 15 und 650 cm. In Systemen wie PDAs oder Laptops werden in der Regel keine Ultraschallsensoren eingebaut, weshalb diese Positionierungsmethode nicht von vielen Nexus-Anwendungen verwendet werden kann.

Funksysteme

Funksysteme können ähnlich wie Infrarot Systeme aus Sendern und Empfängern bestehen. Eine weitere Möglichkeit besteht in der Verwendung von Funknetzen. Funknetze wie Bluetooth oder WLAN können ähnlich wie GSM zur Positionierung benutzt werden. Bei Bluetooth ist eine Kommunikation nur innerhalb eines Radius von 10 m um einen Bluetooth Empfänger möglich. Ein Positionierungssystem entsteht, wenn man ein aus Zellen bestehendes Netzwerk erstellt, bei dem in einem Raster von 10 bis 20 m ein Bluetooth Empfänger steht [Klinec &Volz 2000]. Wie bei GSM kann man mit Hilfe der Zelle die Position des Anwenders ermitteln. Hierbei ist nicht nur ein Trackingsystem, sondern auch ein Selbstpositionierungssystem realisierbar, wenn die Bluetooth Master Station in regelmäßigen Abständen einen Broadcast der Zellen ID oder des von der Zelle abgedeckten Gebiets durchführt.

Verwendung von symbolischen Markierungen

Objekte lassen sich markieren, indem man Symbole oder Muster auf ihnen hinterlässt, das dann interpretiert werden muss, um das Objekt zu erkennen. Ein Beispiel hierfür sind Raumnummern vor den Türen vieler Räume. Markiert man bestimmte Positionen und Objekte mit solchen Symbolen ist eine Positionierung möglich. Es handelt sich hierbei um eine symbolische Positionierung. Die Objekte müssen für solche Systeme mit auf einfache Bildverarbeitung optimierten Symbolen markiert werden. In der Regel entspricht ein Symbol einer ID. Möchte sich ein Anwender positionieren, so muss er ein Symbol mit Hilfe einer Kamera oder eines Scanners einlesen. Nachdem die ID ermittelt wurde, wird dann die dazugehörige symbolische oder geometrische Position aus einer Datenbasis gelesen. Beispiele sind Systeme auf Barcode Basis oder CyberCode, das von der Firma Sony entwickelt wurde [Rekimoto & Ayatsuka 2000]. CyberCode Muster wurden speziell für eine einfache Bildverarbeitung entwickelt. Ein Muster entspricht dabei einer ID. Zum Einlesen einer solchen ID wird eine Kamera benutzt. Solche Systeme sind kostengünstig, da die Cybercodes auf herkömmliches Papier oder auf Aufkleber gedruckt werden können. Sie sind

aber nicht auf Papier beschränkt, da man die Muster auch durch einen Fernseher anzeigen lassen kann. Ein solches Muster muss nicht groß sein. Sie lassen sich dadurch zur Markierung von kleinen und großen Objekten einsetzen. Da sie keine Energie benötigen, können sie auch für die Markierung von Objekten eingesetzt werden, bei denen Sender mit Stromquellen nicht sinnvoll sind. Ein Beispiel ist die Markierung von Aktenordnern oder von Büchern in einer Bücherei.

Ein Nachteil solcher Systeme ist, dass sie ausreichend beleuchtet sein müssen. Weiterhin werden Kameras für das Einlesen der Muster benötigt, die aber nur selten in mobilen Geräten vorhanden sind. Außerdem kann man nur ein Muster auf einmal einlesen.

Tracking ist ebenfalls möglich, wenn man die zu positionierenden Objekte mit Symbolen, wie z.B. einem Namensschild, ausstattet damit ein Bilderkennungsprogramm dieses erkennt und das Objekt positionieren kann.

3.2.4 Mischsysteme und Hilfssysteme

Digitaler Kompass

Ein digitaler Kompass ermittelt die Richtung, in die sich ein Anwender bewegt, oder in die mit einem Gerät gezeigt wird. Ein digitaler Kompass wird von anderen Positionierungssystemen als Hilfssystem benötigt.

Pedometer

Pedometer registrieren die Schritte eines Anwenders. Mit Hilfe der Schrittlänge kann man die zurückgelegte Entfernung messen. Sie sind sowohl innerhalb als auch außerhalb eines Gebäudes verwendbar. In Kombination mit weiteren Systemen lässt sich ein Positionierungssystem erstellen. Ein Pedometer benötigt als Positionierungssystem noch die Richtung in die sich ein Anwender bewegt und ein Gerät zur Höhenmessung für nichtebene Gebiete. In [Konishi & Shibasaki 2001] wird ein solches Positionierungssystem beschrieben. Für die Richtungsmessung wird ein digitaler Kompass benutzt, und ein Barometer misst über den Luftdruck die Höhe des Anwenders.

Pedometer ermitteln keine absoluten Positionen, sondern relative zu einem Ausgangspunkt. Die Genauigkeit eines solchen Systems hängt von der damit zurückgelegten Strecke ab, da sich Fehler akkumulieren können. Bei [Konishi & Shibasaki 2001] war die Genauigkeit ohne Hilfsmittel zwischen 5 und 10%. Soll die Genauigkeit innerhalb eines Bereichs bleiben, so müssen diese Systeme Hilfssysteme benutzen. Eine Möglichkeit besteht darin, Sysnchronisationspunkte festzulegen, an denen das Positionierungssystem die interne Position mit der tatsächlichen Position abgleicht. In [Konishi & Shibasaki 2001] wird statt Synchronisationspunkten die Methode des Map Matching verwendet. Hierdurch lässt sich die Genauigkeit der Positionierung stark verbessern, da Ungenauigkeiten immer wieder reduziert werden können.

Bei Fahrzeugen werden Radsensoren verwendet, die ähnlich wie bei Fahradtachos die Umdrehungen der Räder messen.

Dead Reckoning

Beim Dead Reckoning wird mit Hilfe der letzten Position, der Richtung und Geschwindigkeit eines Objekts auf seine aktuelle Position geschlossen. Dead Reckoning kann zur Steigerung der Genauigkeit benutzt werden. Beispielsweise könnte man bei zellenbasierten Systemen, so abschätzen, ob sich die Position an einem Rand oder in der Mitte befinden müsste. Dead Reckoning kann auch die Häufigkeit reduzieren, in der eine Position bestimmt werden muss. Eine Positionierung mit Hilfe eines Sensors wäre dann nur notwendig, wenn die Genauigkeit einer mit Dead Reckoning ermittelten Position nicht mehr ausreichen dürfte. Dead Reckoning ist bei Trackingsystemen sehr sinnvoll, um den Kommunikationsbedarf zwischen einem sich positionierenden Objekt und dem Trackingsystem zu reduzieren. Bei Nexus wird Dead Reckoning nicht nur bei Positionierungssystemen benutzt, sondern kann auch vom Location Service benutzt werden, um die notwendige Anzahl von Aktualisierungen auf die Position mobiler Objekte zu reduzieren [Leonhardi & Rothermel 2001]. Ein Kombination mit Map Matching Verfahren ermöglicht eine weitere Verbesserung der Genauigkeit und reduziert die notwendige Kommunikation weiter [Leonhardi, Nicu & Rothermel 2002].

Map Matching

Beim Map Matching werden die ermittelten Positionen mit Hilfe von Karteninformationen auf eine sinnvolle Position auf der Karte abgebildet.Map Matching ist ein Hilfssystem, das die Genauigkeit von Positionierungssystemen verbessert. Befindet man sich beispielsweise auf einer Straße und benutzt GPS zur Positionierung, so kann man eine Position erhalten, die eigentlich innerhalb eines Gebäudes wäre. Mit Hilfe von Map Matching wird stattdessen eine sinnvolle Position außerhalb des Gebäudes benutzt. Merkt man sich für ein Map Matching Algorithmus auch den zurückgelegten Weg des Benutzer, so kann man die Genauigkeit weiter verbessern, da nur Positionen möglich sind, die auf dem Weg erreichbar sind. Man kann beispielsweise erkennen, dass der Anwender in einen Seitengang abbiegt und nicht einen Raum betritt, wenn der Raum an der Stelle keine Tür hat.

Abbildung 3-2 zeigt, wie sich die Genauigkeit mit der Verwendung von Map Matching steigert, da nur sinnvolle Positionen akzeptiert werden.

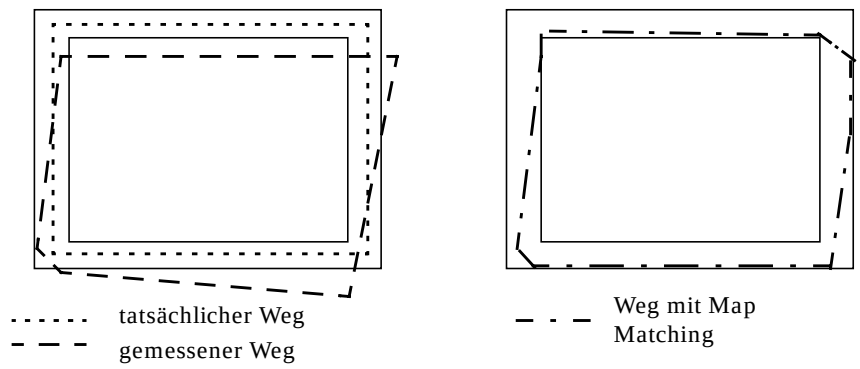


Abbildung 3-2: Verbesserung der Positionierung durch Map Matching

Map Matching ist prinzipiell sowohl für Outdoor wie Indoor-Systeme geeignet und wird beispielsweise für Anwendungen zur Fahrzeug Navigation benutzt [Fritsch, Klinec & Volz 2001].

Bildverarbeitung

Bildverarbeitung ist eine Technologie die bei CyberCode oder Barcodesystemen benötigt wird. Man kann sie aber auch alleine einsetzen und sowohl Selbstpositionierungs- als auch Trackingsysteme erstellen. Bei Selbstpositionierungssystemen trägt der Anwender eine Kamera und nimmt die Umgebung auf. Hierbei werden Objekte aus den aufgenommenen Bildern erkannt und so auf die Position des Anwenders geschlossen. Bei Trackingsystemen werden die Objekte anhand charakteristischer Merkmale erkannt. Ein Beispiel für ein solches Trackingsystem ist das System, das in England bei der Videoüberwachung zur Gesichtserkennung verwendet wird und automatisch gesuchte Straftäter erkennt und ihre Position bestimmt.

Bildverarbeitung wird nur sehr selten allein eingesetzt, da die notwendige Rechenleistung und die benötigten Datenbestände für viele Anwendungen sehr hoch ist und die Fehleranfälligkeit ebenfalls sehr hoch sein kann. Außerdem muss ein Modell der Umgebung existieren, um die entsprechende Position des Anwenders bestimmen zu können.

Positionierung durch Anwender

Statt Positionierungssysteme zu verwenden, kann man den Anwender seine Position angeben lassen. Der Anwender kann seine Position auf einer Karte markieren oder die Raumnummer des nächsten Raums angeben. Die Genauigkeit hängt hierbei sehr stark vom Anwender ab. Sie ist sowohl innerhalb eines Gebäudes wie außerhalb eines Gebäudes möglich und, da es für viele Regionen sehr genaue Karten gibt, fast überall einsetzbar. Allerdings müssen die Karten für die zumeist kleinen Bildschirme der mobilen Geräte angepasst werden. Ein Problem bei solch einer Positionierung ist die geringe Aktualisierungsfrequenz, die sich mit solch einer Positionierung realisieren lässt.

3.2.5 Kombination mehrerer Systeme

Die Kombination mehrerer Systeme ist notwendig, um eine Indoor und Outdoor Positionierung zu ermöglichen. Die Kombination mehrerer Systeme ermöglicht es, die Genauigkeit zu erhöhen und die Schwächen der einzelnen Systeme auszugleichen [Fritsch, Klinec & Volz 2001].

Die Kombination von GPS und dem Pedometersystem aus [Konishi & Shibasaki 2001] hat gegenüber GPS den Vorteil, dass es in Regionen ohne GPS funktioniert. Durch GPS besitzt das Pedometersystem außerhalb von Gebäuden die Möglichkeit seine interne Position auf den Wert des GPS Systems zu setzen und kann so ohne echte Synchronisierungspunkte die maximale Abweichung begrenzen.

Eine Kombination von GPS und Selbstpositionierung des Anwenders ermöglicht im Vergleich zu GPS eine sehr viel höhere Genauigkeit und ist auch in Regionen ohne GPS Empfang einsetzbar. Das GPS Signal kann verwendet

werden, um den Kartenausschnitt zu finden, auf dem sich der Anwender positionieren soll. Das für den Anwender umständliche und fehleranfällige Suchen des richtigen Kartenausschnitts ist dadurch überflüssig. Die maximale Abweichung der Position ist durch GPS begrenzt und deswegen auch bei Anwendern sinnvoll, die sich auf Karten nur schlecht positionieren können.

Eine Kombination eines Outdoor-Systems wie GPS mit einem Indoor-System, wie Infrarotsendern, ist auch für die Positionierung außerhalb eines Gebäudes sinnvoll. In manchen Gebieten ist die Positonierung mit GPS nicht genau genug oder eine symbolische Positonierung wäre sinnvoller. Haben mehrere Sehenswürdigkeiten nur einen kurzen Abstand voneinander, so kann man mit GPS wahrscheinlich nicht bestimmen vor welcher Sehenswürdigkeit sich ein Anwender aufhält. Man kann an dieser Stelle eine sehr viel genauere Positionierung oder eine symbolische Positionierung benutzen, indem man an den Objekten Infrarot Sender oder CyberCode Markierungen anbringt.

3.2.6 Sichtbereich

Die verbreitetsten Positionierungssysteme benutzen Sender. Sender können nur innerhalb ihres Sichtbereichs empfangen werden. Die Größe und Form der Sichtbereiche hängt von der Art der Technologie und der Umgebung ab. Infrarotsender besitzen einen kegelförmigen und Funksender einen kreisförmigen Sichtbereich. Die Umgebung beeinflusst den Sichtbereich. Gegenstände können das ausgesendete Signal schwächen, blockieren oder reflektieren (z.B. IR und Spiegel).

Ist ein Sender innerhalb eines Raumes, so verändert sich sein Sichtbereich, wenn die Tür des Raums offen oder geschlossen ist. Man kann dies mit sicheren und unsicheren Bereichen modellieren. In einem sicheren Bereich empfängt man den Sender mit einer hohen Wahrscheinlichkeit, während im unsicheren Bereich die Wahrscheinlichkeit niedrig ist.

Sind Sender Räumen zugeordnet (symbolische Positionierung), so entstehen Probleme, wenn Sender auch außerhalb des zugehörigen Raumes empfangen werden können. In so einem Fall wäre es sinnvoll den sicheren Bereich als den Raum und den unsicheren Bereich als den Sichtbereich außerhalb des Raumes zu definieren.

Die Anzahl der empfangbaren Sender hängt von der eingesetzten Technologie ab. So kann ein Gerät für CyberCodes immer ein Muster auf einmal einlesen, selbst wenn sich die Sichtbereiche überschneiden z.B. bei sich gegenüberstehenden Mustern. Bei anderen Technologien dürfen sich Sichtbereiche nicht überschneiden, da sich die Sender gegenseitig stören würden. Manche lassen eine Überschneidung zu. Dies hat kleine (Bluetooth) oder gar keine Konsequenzen. Auch eine Mischung der eingesetzten Technologien kann entweder zu einer gegenseitigen Störung führen (zwei Funksysteme) oder problemlos sein (IR und Funk).

Sichtbereiche können aus einer oder mehreren geschlossenen Flächen bestehen. Abbildung 3-3 zeigt wie ein Hindernis Teile eines Sichtbereichs verdeckt und so zwei Flächen (Gebiet 1 und Gebiet 3) entstehen. Senden Sender 1 und Sender 2 die gleiche ID aus, so entsteht ein *virtueller Sender* dessen Sichtbereich aus den Gebieten 1 bis 3 besteht.

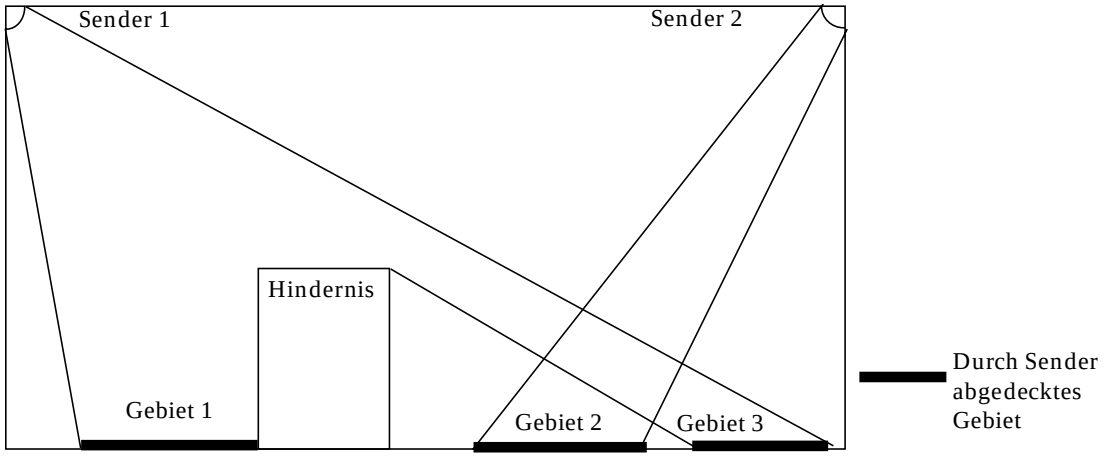


Abbildung 3-3: Entstehung mehrerer Sichtbereiche

3.3 Bezugssysteme

Bezugssysteme werden benötigt, um Daten an einer bestimmten Stelle im Raum eindeutig zu fixieren. In aller Regel werden hierfür Koordinatenssysteme benutzt.

Ein kartesisches Koordinatensystem besteht aus zwei oder drei senkrecht aufeinander stehenden Achsen. Punkte werden relativ zu einem Null-Punkt beschrieben, der festgelegt werden muss. Ein solcher Null-Punkt oder Referenzpunkt kann eine bestimmte Position in einer Stadt, einem Gebäude oder auf einem mobilen Objekt sein. Wenn man die Erdoberfläche in eine Ebene abbildet, so ist das nicht verzerrungsfrei möglich. Verwendet man ein zweidimensionales kartesisches Koordinatensystem, so sind die Verzerrungen für viele Anwendungen nur für kleine Gebiete akzeptabel. Stattdessen werden Abbildungen der Erdoberfläche und Koordinatensysteme benutzt, die diese Verzerrungen auf ein akzeptables Niveau vermindern. Ein Beispiel ist das Gauß-Krüger Koordinatensystem, das die Erdoberfläche in Hauptmeridiane einteilt. Das Gauß-Krüger Koordinatensystem wird hauptsächlich in Deutschland eingesetzt, auch wenn andere Länder ähnliche Koordinatensysteme besitzen. Ein weltweit einheitliches Referenzkoordinatensystem ist WGS84.

Statt Koordinaten lassen sich auch andere Beziehungen verwenden. Beispiele sind indirekte Beziehungen, bei denen ein Ort einen Identifikator erhält. Identifikatoren sind Straßennamen, Postleitzahlen oder Hausnummern. Symbolische Beziehungen sind Angaben wie „vor”, „hinter”, „innerhalb” und werden bei der symbolischen Positionierung verwendet. Eine empfangene ID kann beispielsweise bedeuten, dass man sich innerhalb eines Raumes oder vor

einem Objekt befindet. Im Vergleich mit Koordinatensystemen ist der Umgang mit solchen Bezugssystemen für viele Menschen einfacher [Bartelme 2000].

3.4 Unterstützung im Spatial Model Server

3.4.1 Unterstützte Positionierungssysteme

Positionierungssysteme können von einem Spatial Model Server unterstützt werden, wenn sie bei der Positionierung Informationen über statische Objekte benötigen.

Systeme, bei denen die benötigten Positionsdaten direkt empfangen werden, benötigen keinen Spatial Model Server für die Positionierung. Systeme, bei denen die empfangenen Daten erst umgewandelt werden müssen, können einen Spatial Model Server für das Umwandeln der Daten benutzen. Eine Umwandlung ist notwendig wenn zwar Positionsdaten empfangen werden, aber in eine andere Form umgewandelt werden müssen, z.B. eine symbolische Position in eine geometrische Position. Wird von der Sensorkomponente statt einer Postion eine ID empfangen, so muss diese ID in eine Positon umgewandelt werden. Diese Umwandlung erfolgt mit dem Sichtbereich des Senders mit der empfangenen ID und kann durch eine Anfrage an einen Spatial Model Server erfolgen.

Für Trackingsysteme können Informationen zu den aufgestellten Empfängern gespeichert werden. Allerdings speichern sie diese Informationen in der Regel selbst und benötigen einen Spatial Model Server bestenfalls als Backupsystem aus dem sie die Koordinaten auslesen können.

Bei GPS und GSM werden geometrische Positonen ermittelt. Die Verwendung eines Spatial Model Servers zur Outdoor-Positionierung ist nur sinnvoll, wenn die geometrische Position in eine symbolische umgewandelt werden soll. Spatial Model Server werden daher nur selten für eine Outdoor Positionierung benötigt. Dies ist mit einer der Gründe warum der in der Diplomarbeit zu entwickelnde Spatial Model Server für Daten eines Gebäudes benutzt werden soll. Bei Indoor Systemen, deren Sender nur eine ID aussenden, können Spatial Model Server die ID umwandeln. Beispiele hierfür sind Infrarotsysteme oder die Systeme die symbolische Markierungen benutzen.

Bei Systemen wie Pedometern oder bei Selbstpositionierung mit Ultraschall wird ein Spatial Model Server nicht für die Positionierung benötigt. Da diese Systeme aber nur relative Positionen ermitteln, kann es Synchronisierungspunkte geben, an denen sie ihre interne Positon auf die tatsächliche Position einstellen. Die Synchronisierungspunkte können selbst wieder Sender sein, die nur eine ID ausstrahlen, die umgewandelt werden muss. Der Spatial Model Server kann für diese Systeme die Synchronisierungspunkte speichern.

Viele mobile Rechner können nur sehr wenige Daten aufnehmen. Ändern sich die Daten nicht sehr schnell, so kann der Spatial Model Server diese

bereitstellen. Beispielsweise kann man die Karteninformationen für Map Matching und Verfahren, bei denen sich der Anwender selbst positioniert, speichern, wodurch ein mobiler Rechner nur die notwendigen Kartenausschnitte besitzen muss. Theoretisch wären auch Daten für die Bildverarbeitung in einem Spatial Model Server speicherbar. In der Regel sind die Datenmengen aber zu groß um übertragen zu werden und AWQL verhindert eine Analyse der Daten durch einen Spatial Model Server.

3.4.2 Unterstützte Koordinatensysteme

AWQL benutzt zur Positionierung ein Koordinatensystem, weshalb als Bezugssystem ein Koordinatensystem festgelegt werden muss. Andere Bezugssysteme lassen sich zwar zusätzlich verwenden, werden aber nicht direkt unterstützt.

Bei den Koordinatensystemen müssen Spatial Model Server, die Outdoor Positionierung ermöglichen wollen, Koordinatensysteme für Regionen oder Weltkoordinatensysteme verwenden. Ein lokales kartesisches Koordinatensystem, ist wegen der Verzerrungen nur für kleine Gebiete wie für Gebäude sinnvoll. Eine einfache kartesische Abbildung hat in diesem Fall gegenüber Abbildungen für große Gebiete Vorteile. Die Koordinaten sind leichter zu verstehen und werden bei Gebäudekarten verwendet. Eine Position eines Raums kann direkt übernommen werden.

Das lokale kartesische Koordinatensystem benutzt eine relative Positonierung, während bei Koordinatensystemen für große Gebiete eine absolute Positionierung benutzt wird. Statische Objekte, die sich auf einem mobilen Objekt befinden, erscheinen bei Positionierung relativ zum mobilen Objekt als statische Objekte und können so von einem Spatial Model Server verwaltet werden. Kabinen auf einem Schiff haben eine feste Position auf dem Schiff und bewegen sich nur, wenn sich das Schiff bewegt. Bei einer absoluten Positionierung erscheinen sie als mobile Objekte und müssen vom Location Service verwaltet werden. Bei einem lokalen Koordinatensystem wird der Location Service entlasstet, da die Positionen der statischen Objekte nicht mehr aktualisiert werden müssen. Ein weiterer Vorteil entsteht für die Verteilung der Anfragen durch die Föderation, die im Fall von statischen Objekten nur einen Spatial Model Server anfragen muss und nicht zusätzlich den Location Service.

3.4.3 Daten zu einem Sender

Der SpaSe speichert die Position der Sender (Punkt), den Sichtbereich und die ausgesendete ID. Die Sichtbereiche werden vom Anwender in den SpaSe eingetragen. Der SpaSe führt keine Konsistenzüberprüfung durch. Die Sichtbereiche werden als geometrisches Modell dargestellt. Man kann z.B. ein Dreieck, einen Kreis, ein Polygon oder mehrere Polygone als Sichtbereich verwenden. Eine Aufteilung in sichere und unsichere Bereiche wird nicht durchgeführt, da das genaue Aussehen nicht von einem Spatial Model Server bestimmt wird, sondern durch die Erfordernisse der Sensorkomponente. Es könnten neben den geometrischen Daten noch zusätzliche Daten wie Wahrscheinlichkeiten notwendig sein.

Es lassen sich noch weitere für die Diplomarbeit erstmal unbedeutende Daten speichern, wie z.B. die eingesetzte Batterie, die letzte Wartung, Sendefrequenz etc. Eine Verifizierung des Modells wird vom SpaSe nicht durchgeführt. Es wird keine Unterteilung von sicherem und unsicheren Bereich gemacht, da die Aufteilung aus Sicht des Spatial Model Servers keine zusätzliche Funktionalität ermöglicht. Statt einem Sichtbereich würden dann zwei Sichtbereiche existieren.

3.5 Zusammenfassung

Die Positionierung in Nexus muss in wenigen Sekunden erfolgen, damit ortsbasierte Anfragen schnell bearbeitet werden können. Es gibt unterschiedliche Einteilungsmöglichkeiten von Positionierungssystemen Eine wichtige ist die Einteilung in Outdoor und Indoor Systeme. GPS-Systeme sind die wichtigsten Outdoor-Systeme und Infrarot-Systeme sind die wichtigsten Indoor-Systeme, da sie am billigsten sind und von vielen mobilen Geräten unterstützt werden.Es existieren Mischsysteme, die sich sowohl innerhalb eines Gebäudes als auch außerhalb eines Gebäudes verwenden lassen. Die Verwendung von Hilfssystemen kann die Positionsgenauigkeit verbessern. Die Kombination mehrere Systeme ermöglicht es Probleme und Einschränkungen der einzelnen Systeme zu umgehen.

Der Spatial Model Server wird bei der Positionierung mit Outdoor Systemen wie GPS nicht benötigt. Er wird nur bei Indoor-Systemen eingesetzt, deren Sender keine Position, sondern nur eine ID aussenden, die in eine Position umgewandelt werden muss. Das verwendete Bezugssystem ist ein lokales Koordinatensytem, da der Spatial Model Server Objekte innerhalb eines Gebäudes verwalten soll.

Die Begrenzung auf Daten eines Gebäudes ermöglicht es die Daten im Hauptspeicher zu halten. Das folgende Kapitel zeigt am Beispiel von Hauptspeicherdatenbanken, wie sich Systeme, die nur Hauptspeicher benutzen, von Systemen unterscheiden, die für die Benutzung von externen Speicher optimiert sind.

4 Hauptspeicherdatenbanken

Spatial Model Server und Datenbanksysteme besitzen Ähnlichkeiten. Der entwickelte Spatial Model Server wird für Innenräume verwendet und kann seine Daten im Hauptspeicher halten. Hauptspeicherdatenbanksysteme halten ihre Daten ebenfalls im Hauptspeicher. Hierdurch ergeben sich im Vergleich mit herkömmlichen Datenbanksystemen Unterschiede in der Architektur, in den Datenstrukturen und ihrem Verwendungszweck.

4.1 Vergleich Datenbanksystem und Spatial Model Server

Spatial Model Server ähneln Datenbanksystemen. Beide Systeme verwalten Daten für andere Systeme und sind in der Regel nicht auf spezielle Anwendungen optimiert. Die gespeicherten Daten werden mit einer Anfragesprache ausgelesen oder verändert. Die Datenmengen sind in der Regel zu groß um komplett im Hauptspeicher gehalten zu werden, weshalb Spatial Model Server und Datenbanksysteme in der Regel für den Zugriff auf externen Speicher optimiert sein müssen. Die Anforderungen an die Geschwindigkeit können sehr hoch sein, wenn sehr viele Anwendungen auf sie zugreifen können und die Verarbeitung einer Anfrage sehr schnell erfolgen muss.

Spatial Model Server sind allerdings für die Verwendung in der Nexus-Plattform gedacht. Ihre Anfragesprache AWQL ermöglicht räumliche Anfragen, ist aber sehr viel einfacher als SQL. Die Daten von Spatial Model Servern werden in der Regel selten geändert. Sie müssen daher in der Regel auf lesenden Zugriff optimiert sein.

Es gibt Anwendungsgebiete, bei denen der Hauptspeicher heutiger Systeme ausreicht, um die Daten eines Spatial Model Servers komplett im Hauptspeicher verwalten zu können. Werden hohe Geschwindigkeitsanforderungen gestellt, so kann es notwendig sein alle Daten im Hauptspeicher halten zu müssen. Solch ein Spatial Model Server sollte auf den Zugriff auf Hauptspeicher optimiert sein. Ein Beispiel dafür ist der entwickelte Spatial Model Server für Innenräume, der für die Positionierung von Anwendern benötigt wird. Konzepte und Methoden bei herkömmlichen Datenbanksystemen sind deswegen nicht immer geeignet. Es gibt aber auch Datenbanksysteme, die auf die effiziente Verwendung von Hauptspeicher optimiert sind: Die im folgenden vorgestellten Hauptspeicherdatenbanksysteme.

4.2 Hauptspeicherdatenbanksysteme

Hauptspeicherdatenbanksysteme (HSDBS) verwalten Daten ausschließlich im Hauptspeicher. Der Hauptspeicher kann dabei durch virtuellen Speicher erweitert sein. Der Sekundärspeicher wird nur für virtuellen Speicher, für die Datensicherung (und evtl. für das Recovery) und das Laden der Daten beim Start des Betriebs benutzt.

HSDBS und herkömmliche Datenbanksysteme unterscheiden sich voneinander, da herkömmliche Datenbanksysteme auf den Zugriff auf Festplatten optimiert sind und so die Vorteile von Hauptspeicher nicht voll ausnutzen.

4.3 Eigenschaften von Hauptspeicher

Hauptspeicher unterscheidet sich in folgenden Punkten von Festplattenspeicher [Garcia-Molina & Salem 1992]:

- Die Zugriffszeit auf Hauptspeicher ist sehr viel kleiner als auf Festplatten
- Hauptspeicher ist normalerweise flüchtig
- Hauptspeicher ist nicht blockorientiert
- Sequentieller Zugriff hat bei Hauptspeicher eine geringere Bedeutung als bei Festplatten
- Hauptspeicher wird vom Prozessor direkt angesprochen. Softwarefehler sind für Hauptspeicher eine größere Bedrohung als für Festplatten

Dazu kommt:

- Hauptspeicher hat einen höheren Preis
- Hauptspeicher ist sehr viel schneller. SDRAM hat eine Bandbreite von ca. 1GB/s, DDR-Ram von ca. 2 GB/s und RAMBUS von ca 3. GB/s. Im Vergleich zu diesen bei PCs verbreiten Hauptspeichersystemen ist ein Festplattensystem wie RAID U160 SCSI mit seiner maximalen Bandbreite von 160 MB/s langsam. Einzelne Festplatten können solch eine Bandbreite normalerweise nicht einzeln ausfüllen. Sie erreichen 160 MB/s nur wenn sie im Verbund betrieben werden und sie große Datenmengen sequentiell einlesen können.

4.4 Einfluss der Hauptspeichergröße

Normalerweise haben herkömmliche Datenbanksysteme nicht alle Daten im Hauptspeicher, sondern puffern nur einen Bruchteil dieser Daten. Man kann aber auch bei herkömmlichen Datenbanken alle Daten in den Hauptspeicher laden. Erhöht man den Hauptspeicher für die Puffer, so erhöht sich die Geschwindigkeit der Datenbank. Die Geschwindigkeit ist bei kleinem Puffer durch die Geschwindigkeit des Sekundärspeichers und bei großem Puffer durch die Geschwindigkeit der CPU bestimmt.

Durch die höhere Geschwindigkeit können mehr Transaktionen pro Sekunde durchgeführt werden. Die höhere Ausführungsgeschwindigkeit ermöglicht es, Redundanzen zu entfernen, die in Datenbanken aus Geschwindigkeitsgründen oft enthalten sind. Die Speicherersparnis sorgt für eine bessere Ausnutzung des Speichers, der aufgrund der Hauptspeicherbeschränkungen vieler Systeme notwendig ist. 32-Bit Systeme können normalerweise nur zwischen zwei und vier GB Hauptspeicher adressieren. Der maximale Speicher von 64-Bit Systemen wird dagegen nur durch den maximal möglichen Speicherausbau der Hardware und der

maximal möglichen Speicheradressierung durch das Betriebssystem begrenzt. Momentan können die Systeme bis zu 64 GB Speicher verwalten [TimesTen 1999a].

4.4.1 Einfluss auf die Architektur

Abbildung 4-1 zeigt das vereinfachte Schichtenmodell von [Härder & Rahm 1999]. Ein Datenbanksystem besteht danach aus Datensystem, Zugriffssystem und Speichersystem.

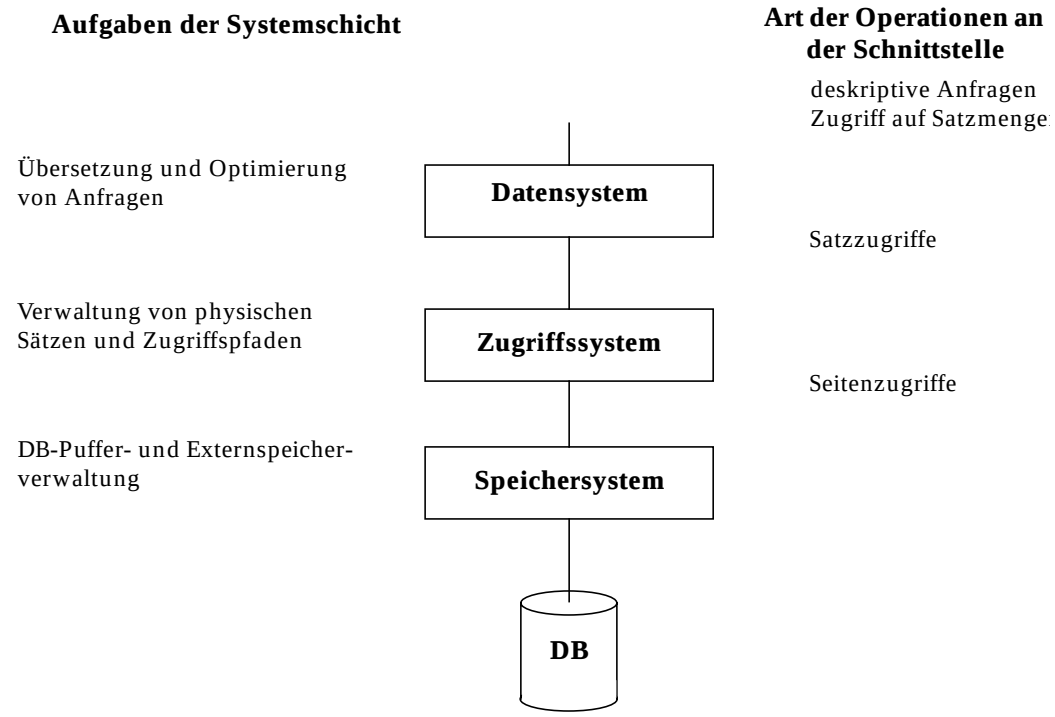


Abbildung 4-1: Vereinfachtes Schichtenmodell

Anfragen werden mit Hilfe des Datensystems in eine interne Darstellung übersetzt und zu einer möglichst effizient ausführbaren Anfrage optimiert. Hierdurch entsteht eine Aufrufreihenfolge von Funktionen an das darunterliegende Zugriffssystem. Das Zugriffssystem bietet einen satzweisen Zugriff auf die verwalteten physischen Sätze und Zugriffspfade an und bildet sie auf die Seiten des Speichersystems ab. Das Speichersystem verbirgt durch die Zugriffseinheit „Seite" alle Aspekte der Externspeicherverwaltung. Das Speichersystem besitzt bei herkömmlichen Datenbanksystemen einen Puffer und benutzt zur Leistungssteigerung eigene Funktionen zum Zugriff auf den externen Speicher.

Da HSDBS die Daten im Hauptspeicher halten, greifen sie nur zum Abspeichern oder bei einem Neustart auf Sekundärspeicher zu. Deswegen müssen sie das Verhalten von Sekundärspeicher nicht berücksichtigen. Es werden keine Puffer benötigt und der Zugriff auf den Sekundärspeicher kann über Betriebssystemfunktionen erfolgen.

Der Anteil der Schichten an der Ausführungszeit unterscheidet sich bei HSDBS und herkömmlichen DBS. Die Informationen für Datensystem und Zugriffssystem können im Hauptspeicher gehalten werden, weshalb die Algorithmen hauptspeicherbasiert sind; z.B. muss man bei der Verarbeitung einer SQL-Anfrage keine Daten auslagern und das Zugriffsystem entscheidet welche Zugriffsstruktur benutzt werden soll, ohne Daten auszulagern. Das Speichersystem greift dagegen auf Sekundärspeicher zu, da es nur einen Bruchteil der gespeicherten Daten im Hauptspeicher halten kann. Nach [Härder & Rahm 1999] beträgt der Aufwand pro E/A-Operation 2500 Instruktionen in der CPU und 15 bis 30 ms für den Zugriff auf den Sekundärspeicher. Die Kosten hierfür sind sehr hoch und treten oft auf, da nur kleine Teile in den Puffern sind.

Die anderen Schichten versuchen die Anzahl der Plattenzugriffe zu minimieren und können sehr viel Rechenleistung für Optimierungen aufwenden. Die CPU-Zeit, die so verwendet wird, ist für die Ausführungszeit unbedeutend, da die Ausführungszeit von den Zugriffen auf Sekundärspeicher bestimmt wird.

Bei HSDBS muss das Speichersystem nicht auf Sekundärspeicher zugreifen. Der Anteil an der Ausführungszeit reduziert sich und die anderen Schichten gewinnen an Bedeutung.

Ein Beispiel zeigt diesen Unterschied. Tabelle 4-1: „Technische Daten”enthält die technischen Daten für den Benchmark. Die Rechenleistung der CPU entspricht in etwa einer 1 GHz CPU und ist für heutige Verhältnisse etwas langsam. Die mittlere Zugriffszeit dagegen ist sehr niedrig und dürfte wohl nur durch sehr teuere Festplatten oder mit RAID zu erreichen sein.

Tabelle 4-1: Technische Daten

Rechenleistung der CPU	1 Milliarde Operationen/s
mittlere Zugriffszeit	1 ms (entspricht 1 Millionen Operationen)
Anzahl Elemente	1 000 000
B-Baumhöhe	3
B-Baumbreite	100 Elemente
Anzahl Operationen pro B-Baum Ebene	7 Vergleiche + 28 sonst Operationen vereinfacht 35 Operationen
Anzahl Operationen für Elementsuche	35 * 3 = 105

Es werden nun zwei Szenarien bei unterschiedlicher Pufferbelegung betrachtet. Der Aufwand für das Zugriffssystem wird vereinfacht als Aufwand für den Zugriff auf die Indexe dargestellt und die Kosten für einen Vergleich sollen den Kosten einer normalen Operation entsprechen. Der Aufwand für Speichersystem, Puffersystem und Betriebssystem wird zum Aufwand für die Plattenzugriffe vereinfacht.

In Szenario 1 wird eine einfache Anfrage unoptimiert ausgeführt. Szenario 2 betrachtet den Aufwand für die Anfrage, wenn sie stark optimiert wird. Der Aufwand für die Verknüpfung der Teilergebnisse sei in dieser Betrachtung vernachlässigbar.

Tabelle 4-2 zeigt das Ergebnis bei Szenario 1. Das Datensystem benötigt nur wenige Operationen, um die Anfrage in eine interne Darstellung umzuwandeln. Es wird 20 mal auf B-Baum Indexe zugegriffen. Sowohl bei einem System ohne Puffer als auch bei normalem Ausbau sind die Festplattenzugriffe wegen der hohen Zugriffszeit für die Geschwindigkeit der Ausgabe bestimmend. Allerdings wird auch die Anzahl der Operationen in erster Linie durch die Festplattenzugriffe bestimmt. Sowohl Datensystem als auch Zugriffssystem sind vom Aufwand für die Anfrage praktisch bedeutungslos und die CPU wird nur sehr schlecht ausgelastet. Ein weiterer interessanter Aspekt ist, dass eine Ausführung ohne Puffer 14671 mal länger dauert als eine Ausführung im Puffer. Die Ausführungsdauer kann sehr stark variieren. Echtzeitsys-

Tabelle 4-2: Szenario 1: Keine Anfrageoptimierung

	kein Puffer	Hälfte der Daten im Puffer	komplett im Puffer
Fehlseitenrate	100%	50%	0 %
Anzahl Operationen (Ops) für Datensystem	2.000	2.000	2.000
20 * Elementsuche Ops	105 * 20 = 2.100	105 * 20 = 2.100	105 * 20 = 2.100
Anzahl Plattenzugriffe	20 * 3 = 60	10 *3 = 30	0
Anzahl Ops für Plattenzugriffe	60 * 2.500 = 150.000	30 * 2.500 = 75.000	0
Anzahl Operationen	2.000 + 2.100 + 150.000 = 154.100	2.000 + 2.100 + 75.000 = 79.100	2.000 + 2.100 = 4.100
Aufwandsverteilung Ops (DS, ZS, PS)	1,3%, 1,4%, 97,3%	2.5%, 2,7%, 94,8%	48,7%,51,3%,0%
Zeit für Ops	0,154 ms	0,079 ms	0,0041 ms
Zeit für Plattenzugriffe	60 ms	30 ms	0 ms
Zeit insgesamt	60.154 ms	30,079 ms	0.0041 ms
Aufwandsverteilung (DS, ZS, PS)	0,003%, 0,003%, 99,99%	0,006%, 0,006%, 99,97 %	48,7%, 51,3%, 0 %
CPU-Auslastung (CPU-Zeit / Zeit insgesamt)	0.154 / 60.154 = 0.256%	0.079 / 30.079= 0.262%	100 %

teme können mit diesem Verhalten Probleme haben, da sie dies im vornherein nicht abschätzen können. Es zeigt sich auch, dass man sehr große Puffer braucht bis das Verhalten von der Rechenleistung und nicht von der Festplattengeschwindigkeit bestimmt wird. Selbst wenn die Hälfte der Daten im Puffer ist, ist die CPU-Auslastung gering.

Im zweiten Szenario wird die Anfrage stark optimiert. Das Datensystem verbraucht jetzt 30000 Operationen und benötigt so nur noch 10 Anfragen.

Tabelle 4-3: Szenario 2: Mit Optimierung

	kein Puffer	Hälfte der Daten im Puffer	komplett im Puffer
Fehlseitenrate	100%	50%	0 %
Anzahl Ops für Datensystem	30.000	30.000	30.000
10 * Elementsuche Ops	105*10 = 1.050	105*10 = 1.050	105*10 = 1.050
Anzahl Plattenzugriffe	10 *3 = 30	5*3 = 15	0
Anzahl Ops für Plattenzugriffe	30 * 2.500 = 75.000	15 * 2.500 = 27.500	0
Anzahl Operationen	30.000 + 1.050 + 75.000 =106.050	30.000 + 1.050 + 27.500 = 58.550	30.000 + 1.050 = 31.050
Aufwandsverteilung Ops (DS, ZS, PS)	28%, 1%,71%	51%, 2%, 47%	97%, 3%, 0%
Zeit für Ops	0,106 ms	0,058 ms	0,031 ms
Zeit für Plattenzugriffe	30 ms	15 ms	0
Zeit insgesamt	30,106 ms	15,058 ms	0,031 ms
Aufwandsverteilung (DS, ZS, PS)	0,09%, 0,003%, 99,9%	0,2%, 0,007%, 99,79%	97%. 3%, 0%
CPU-Auslastung (CPU-Zeit / Zeit insgesamt)	0,35%	0,38%	100%

Die Ausführungsdauer hat sich bei den Fällen ohne Puffer und normalen Puffer praktisch halbiert. Die CPU-Auslastung ist dabei nur gering gestiegen. Die Ausführungsdauer der Anfragen hängt immer noch von der Zugriffsgeschwindigkeit der Platten ab. Bei der Aufwandsverteilung bei den Operationen (Ops) ist der Anteil des Datensystems stark gestiegen. Bei der Aufwandsverteilung mit Plattenzugriffen ist der Anteil aber immer noch unter einem Prozent.

Liegen die Daten komplett im Hauptspeicher, so verlangsamt die Optimierung die Ausführungsdauer um den Faktor 7,5. Die Optimierung war daher nicht sinnvoll und wäre selbst, wenn danach keine weitere Operationen nötig wären, langsamer als die nicht-optimierte Anfrage.

Das Datensystem und das Zugriffssystem von HSDBS können nicht wie bei herkömmlichen DBS CPU-Zeit benutzen und davon ausgehen, dass es die Ausführungszeit einer Transaktion nicht wesentlich verändert.

4.4.2 Einfluss auf das Datensystem

Das Datensystem übersetzt und optimiert die Anfragen und wandelt das Ergebnis in eine für Externe lesbare Form um. Bei HSDBS kann der Optimierer einfach gehalten werden. Da er keinen Puffer/E/A-Zugriffe berücksichtigen muss, sind nur Berechnungskosten wichtig.

Dem Optimierer steht allerdings auch sehr viel weniger CPU-Zeit zur Verfügung als bei herkömmlichen DBS, da sonst die Gefahr besteht, dass die Optimierung länger dauert als die eingesparte CPU-Zeit. Deshalb sollten nur Optimierungen vorgenommen werden, die wenig CPU-Zeit kosten und sich mit Sicherheit auch lohnen.

Die Geschwindigkeit lässt sich steigern, wenn als Anfragesprache statt SQL effizient verarbeitbare Anfragesprachen benutzt werden können. Zusätzlich kann man Stored-Procedures verwenden, bei denen die Anfrageverarbeitung und -optimierung wegfällt.

4.4.3 Einfluss auf Indexe

Die Indexe bei herkömmlichen DBS müssen die Anzahl der Plattenzugriffe minimieren, während HSDBS Indexe benötigen, die auf minimale Rechenzeit und Speicherverbrauch optimiert sind. Die Indexe für herkömmliche DBS enthalten Kopien von den gesuchten Daten, da sie so zusätzliche Plattenzugriffe vermeiden. Bei Verwendung von Hauptspeicher ist die Verwendung von Zeigern auf die Daten sinnvoller, da so Speicher gespart wird und die Geschwindigkeitsvorteile durch den wahlfreien Zugriff auf Hauptspeicher nicht relevant sind.

Im folgenden werden B-Bäume als typischer DBS Index und AVL und T-Bäume als Hauptspeicherindexe betrachtet. An den B-Bäumen kann man auch erkennen, wie Implementierungsdetails durch festplattenbasierten oder hauptspeicherbasierten Zugriff beeinflusst werden.

Beim lesenden Zugriff auf Festplatten entscheidet die Anzahl der Plattenzugriffe und damit die Höhe des Baums über die Geschwindigkeit. Bei Hauptspeicher hängt sie von der Anzahl der benötigten CPU-Operationen ab. Es zählen dann nicht nur die Höhe h des Baums, sondern auch die Anzahl der Operationen pro Knoten. Die Anzahl der Operationen hängt von der Anzahl der zu betrachtenden Elemente pro Knoten v ab und wird im Folgenden als Vergleich bezeichnet. Da die tatsächliche Anzahl der Vergleiche von der Realisierung der Suchoperation und der notwendigen Anzahl von Hardwarevergleichen abhängt, ist die tatsächliche Anzahl der Hardwarevergleiche um einen Faktor k höher. k hängt unter anderem davon ab welche und in welcher Anordnung die Vergleichsoperationen(<, >, =) benutzt werden und wie oft man die Elemente dabei vergleichen muss. Es kann einen großen Unterschied machen, ob man den Vergleich zweier Elemente für jede Bedingung durchführt und damit drei Elementvergleiche macht oder ob man zwei Elemente vergleicht und als Ergebnis einen Wert erhält, der angibt ob ein Element kleiner, gleich oder größer ist als das andere. Bei T-Bäumen wird man dieses Problem haben. Da alle drei Indexarten eine Suche in O(logn) realisieren, ist ein

Vergleich mit Hilfe des O Kalküls nicht sinnvoll. Sie werden deshalb etwas genauer betrachtet. Hierbei wird in erster Linie der Worst Case betrachtet, d.h. das gesuchte Element ist in einem Blatt und niemals in einem inneren Knoten. Dies führt dazu, dass B-Bäume etwas besser dastehen als im durchschnittlichen Fall, da bei B-Bäumen fast alle Elemente in einem Blatt sind, während bei einem AVL-Baum nur maximal 50% +1 Element Blätter sind.

Das Einfügen, Löschen und Verändern von Elementen wird nur oberflächlich betrachtet, da für den Indoor SpaSe in erster Linie die Suchanfragen wichtig sind.

Die Ausnutzung des Speichers der verschiedenen Indexstrukturen wird für kleine und große Elemente betrachtet.

Es werden folgende Abkürzungen verwendet:

- n: Anzahl der Elemente im Index
- k: Anzahl der Elemente eines Knotens im B-Baum
- m: festgelegte Anzahl Elemente jedes Knotens in B- oder T-Baum
- h: Höhe des Baums
- v: Anzahl der Vergleiche
- v_{pk}: Anzahl der Vergleiche pro Knoten
- e: Speicherverbrauch eines Elements
- z: Speicherverbrauch eines Zeigers

B-Bäume

B-Bäume haben folgenden Eigenschaften ([Härder & Rahm 1999]):

- Jeder Weg von der Wurzel zum Blatt hat die gleiche Länge h.
- Jeder Knoten (außer Wurzel und Blätter) hat mindestens k + 1 Söhne. Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne.
- Jeder Knoten hat höchstens 2k+1 Söhne

Abbildung 4-2 zeigt einen solchen B-Baum.

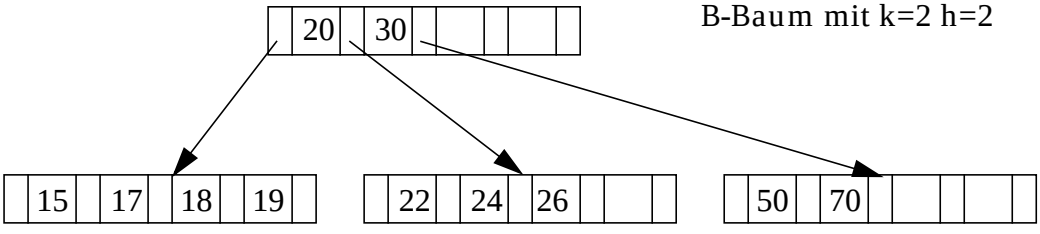


Abbildung 4-2: B-Baum

Beim Zugriff auf Festplatten hängt die Geschwindigkeit einer Suchanfrage in erster Linie von der Höhe h des B-Baums ab. Die Höhe eines B-Baums mit n Elementen (n>1) ist begrenzt durch (vgl. [Härder & Rahm 1999]):

$$\log_{2k+1}(n+1) \le h \le 1 + \log_{k+1}\left(\frac{n+1}{2}\right)$$

Dies ist deutlich kleiner als die Höhe von AVL oder T-Bäumen, weshalb sie die bevorzugte Indexart bei herkömmlichen DBS sind.

Die Anzahl der Vergleiche v hängt von der Höhe und der Anzahl der Vergleiche pro Knoten vpk ab:

$$v = v_{pk} h$$

Die Anzahl der Vergleiche pro Knoten hängt von der angewendeten Suchmethode ab. Benutzt man lineare Suche (O(n)) so ist die Anzahl der Vergleiche zwischen 1 und der Anzahl der Elemente im Knoten m d.h. zwischen k und 2k Elementen.

Bei binärer Suche beträgt die Anzahl der Vergleiche bei m Elementen nie mehr als $\log_2(m) + 1$ Vergleiche [Sedgewick 1992].

$$\log_2(k) \le v_{pk} \le \log_2(2k)$$

Zur Vereinfachung der Rechnung werden nur B-Bäume betrachtet, bei denen alle Knoten m Elemente haben und die inneren Knoten m+1 Söhne haben. Das Ergebnis sollte sich nur sehr wenig verfälschen, da bei B-Bäumen in erster Linie eine niedrige Höhe gewünscht ist, die sich zwischen der maximalen Anzahl Söhnen und der minimalen Anzahl bei großem k kaum unterscheidet. Der Unterschied zwischen k=20 und k = 40 ist

$$\log_{40} n = \frac{\log_{20} n}{\log_{20} 40} = 0.81 \log_{20} n$$

und damit 19 Prozent. Da h sehr klein ist (3-4) unterscheiden sich beide Höhen fast nicht.

Die zweite Vereinfachung ist, dass nur die Fälle betrachtet werden, bei denen das gesuchte Element in einem Blatt ist. Für m = 40 sind dies 97,5 Prozent der Fälle.

Die Anzahl der Vergleiche beträgt bei linearer Suche im Durchschnitt:

$$v_{lin} = \frac{m}{2} \log_m n$$

und bei binärer Suche:

$$v_{bin} = (\log_2 m + 1) \log_m n = \log_2 n + \frac{\log_2 n}{\log_2 m}$$

In der Literatur findet man beide Varianten: lineare Suche bei [Cormen et al 1990] und binäre Suche bei [Lehman & Carey 1986a].

An der Auswahl des Suchverfahrens kann man erkennen, wie sehr selbst Implementierungsdetails von technischen Rahmenbedingungen beeinflusst werden. Binäre Suche benötigt weniger Vergleiche als lineare Suche, weshalb in der Regel binäre Suche auch schneller ist. Jedoch können technische Einflüsse lineare Suche bevorteilen. Die Parallelisierbarkeit einer linearen Suche ist sehr gut, während sie bei einer binären Suche schlecht ist, da unklar ist welches Element als nächstes zu vergleichen ist.

Ein weiterer Einflussfaktor ist die Realisierung des Datenzugriffs. Lineare Suche realisiert einen sequentiellen Zugriff auf die Daten des Knotens, während binäre Suche einen wahlfreien Zugriff darstellt. Bei Datenbanken könnten beide Zugriffsformen vorhanden sein. Muss erst der komplette Knoten in den Hauptspeicher kopiert sein, damit darauf zugegriffen werden kann, so ist ein wahlfreier Zugriff möglich und binäre Suche schneller. Werden die Daten allerdings sequentiell in den Speicher kopiert und der Zugriff auf die bereits eingelesenen Elemente ermöglicht, so wird bei linearer Suche bereits verglichen, während man bei binärer Suche auf das mittlere Element warten muss. Wenn Vergleiche nicht langsamer sind als das Einlesen eines Elements, dann ist lineare Suche schneller oder gleich schnell wie binäre Suche denn es gilt:

- Findet man bei binärer Suche das gesuchte Element, so würde lineare Suche dieses Element zur gleichen Zeit einlesen d.h. lineare Suche ist gleichschnell.
- Liest man bei binärer Suche ein Element, das größer als das gesuchte Element ist, so hat man dies bei linearer Suche bereits eingelesen und gefunden d.h linearer Suche war schneller.
- Liest man bei binärer Suche ein Element, das kleiner als das gesuchte Element ist, so müssen weitere

Elemente eingelesen werden.

Da Daten im Sekundärspeicher sequentiell eingelesen werden und die Lesegeschwindigkeit geringer ist als im Hauptspeicher, kann dies trotz höherer CPU-Belastung durch mehr Vergleiche und der Zugriffskontrolle zwischen Pufferverwaltung und dem Zugriff auf den Knoten zu einer höheren Geschwindigkeit führen. Selbst im Fall, dass sich die Daten im Festplattencache befinden, müssten die Daten durch einen relativ langsamen Bus übertragen werden (z.B. kann U160 SCSI maximal 160 MB/s übertragen, was selbst im Vergleich zur Geschwindigkeit von SDRAM recht klein ist (ca. 1GB/s)).

Einfüge- und Löschoperationen sind relativ effizient ausführbar, da hierfür nur die Stelle, an der die Operation durchgeführt werden soll, gesucht werden muss und die Operation darauf ausgeführt wird. Ein Umstrukturieren des Baumes ist nur selten notwendig.

Die Speicherausnutzung eines B-Baums hängt sehr stark von der Belegung der Knoten ab. Jeder Knoten enthält Speicher für 2k+1 Zeiger z und 2k Elemente e. Für die Speicherausnutzung s gilt deshalb:

$$\frac{ke}{(2k+1)z+2ke} \le s \le \frac{2ke}{(2k+1)z+2ke}$$

Für e = z gilt:

$$\frac{k}{4k+1} \le s \le \frac{2k}{4k+1}$$

s liegt also grob zwischen 25% und 50% Speicherausnutzung.

Für große Elemente (e>>z) gilt:

$$\frac{1}{2} \le s \le 1$$

Theoretische Untersuchungen zeigen, dass man mit einer Belegung von 69% rechnen kann [Yao & De Jong 1978]. In der Praxis wird oft auf die Behandlung eines Unterlaufs verzichtet, wodurch die tatsächliche Belegung schlechter ist. Die Belegung lässt sich allerdings durch verändern der Split-Technik auch erhöhen [Härder & Rahm 1999].

AVL-Baum

AVL-Bäume sind balancierte binäre Bäume. Sie entsprechen einer binären Suche, bei der allerdings keinerlei mathematischen Berechnungen benötigt werden. Deshalb sind sie schneller als binäre Suche [Lehman & Carey 1986a].

Ihre Höhe beträgt maximal $\log_2 n$ +1. Da die Suche einer binären Suche entspricht, benötigt es in etwa $\log_2 n$ + 1 Vergleiche.

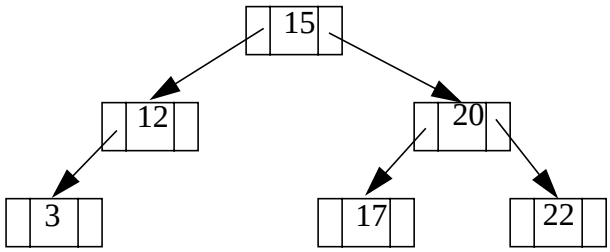


Abbildung 4-3: AVL-Baum

Wird ein Element eingefügt, gelöscht oder verändert, so dauert dies im Durchschnitt länger als bei den anderen Datenstrukturen. Die Position, an der die Operation ausgeführt werden soll, wird zwar am schnellsten gefunden, aller-

dings muss mit einer im Vergleich zu den anderen Bäumen hohen Wahrscheinlichkeit eine neue Balancierung des Baums vorgenommen werden.

Ein Knoten eines AVL-Baums enthält zwei Zeiger, das Element und den Balancefaktor (-1,0,1). Der Balancefaktor habe zur Vereinfachung die Größe eines Zeigers. Für die Speicherausnutzung gilt dann:

$$s = \frac{ne}{n(e+3z)}$$

Für e = z beträgt die Speicherausnutzung 25%. Ist e sehr viel größer als z, so ist die Speicherausnutzung beinahe 100%.

T-Baum

T-Bäume sind eine von Lehman und Carey entwickelte Datenstruktur [Lehman & Carey 1986a], [Lehman & Carey 1986b]. Sie sind eine Mischung aus B-Baum und AVL-Baum. Wie AVL-Bäume besitzen sie nur zwei Zeiger für den linken und den rechten Kindknoten. Knoten enthalten wie B-Bäume mehrere Elemente in sortierter Reihenfolge. Die Anzahl der Elemente darf ein Minimum nicht unterschreiten.

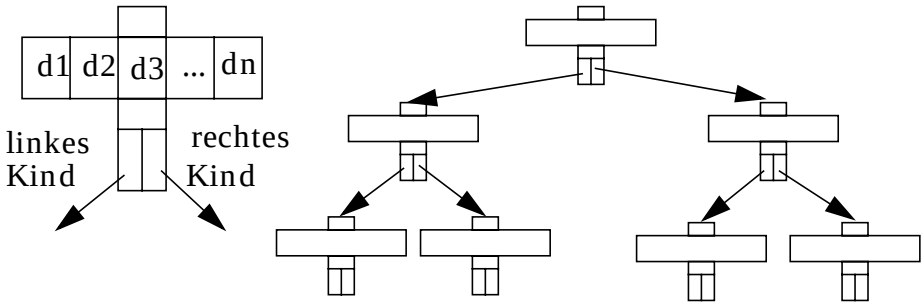


Abbildung 4-4: T-Bäume

Das Balancieren entspricht dem Verfahren von AVL-Bäumen, kommt allerdings seltener vor, da nicht jedesmal für ein Insert, Delete oder Update ein Knoten erstellt oder gelöscht werden muss.

Die Höhe eines T-Baums mit durchschnittlich m Elementen pro Knoten beträgt $\log_2 (n / m)$.

Bei der Suche wird zu erst das kleinste Element des Knotens mit dem gesuchten Element verglichen und, falls das Element kleiner ist, der linke Kindknoten betrachtet, falls nicht, das größte Element des Knotens und evtl. der rechte Kindknoten betrachtet. Ist das Element größer als der

kleinste Knoten und kleiner als der größte Knoten, so wird innerhalb des Knotens mit binären Suche gesucht. Die Anzahl der Vergleiche v entspricht:

$$v = \log \frac{n}{2m} + 1 + \log_2 m + 1 = \log_2 n + 2$$

T-Bäume vergleichen zuerst das kleinste Element und dann das größte Element, während bei AVL-Bäumen und bei binärer Suche ein Element mehrfach verglichen wird. Dies hat bei kleinen Elementen nur geringe Geschwindigkeitsnachteile. Bei großen Elementen dagegen kann es zu einem großen Nachteil werden. Beispielsweise wird in Java bei einem Vergleich zweier Zeichenketten zurückgegeben, ob das erste Element kleiner, größer oder gleichgroß ist. Bei AVL-Bäumen muss man nur dieses Ergebnis auswerten, um die nächste Ebene zu erreichen. Bei T-Bäumen muss man das Element im Worst Case mit zwei Zeichenketten vergleichen. Im Worst Case entspricht die Anzahl der Vergleiche:

$$v = 2\log_2 n - \log_2 m + 2$$

Vergleich der Indexe

Tabelle 4-4: Vergleich der Indexe

Operationen	AVL-Baum	T-Baum	B-Baum
Kleine Elemente			
Elementsuche	am schnellsten	gut	am schlechtesten
Änderungsoperation	am langsamsten	am besten	gut
Speicherverbrauch	am schlechtesten	am besten	gut
Große Elemente			
Elementsuche	am schnellsten	gut	gut
Änderungsoperation	am schnellsten	gut	gut
Speicherverbrauch	am besten	gut	am schlechtesten

AVL-Baum, T-Baum und B-Baum werden in Tabelle 4-4 beim Einsatz bei kleinen und bei großen Elementen miteinander verglichen. In [Lehman & Carey 1986a] und [Lehman & Carey 1986b] werden diese Indexe und diverse Hashverfahren miteinander verglichen. Die gespeicherten Elemente waren klein (Zahlen). Die Hashverfahren hatten dabei entweder einen im Vergleich hohen Speicherverbrauch oder ein schlechtes Update Verhalten. Da kleine Elemente benutzt wurden, hatte der T-Baum im Vergleich zu AVL und B-Bäumen die beste Speicherausnutzung. Dies entspricht auch der obigen theoretischen Betrachtung. Bei Änderungsoperationen war er schneller als die anderen beiden Indexarten. AVL-Bäume machten die schnellste Elementsuche, waren aber bei Einfüge- und Löschoperationen schlechter als T- oder B-Bäume. Wegen der guten Speicherausnutzung und der relativ hohen Geschwindigkeit

bei Such-, Einfüge- und Löschoperationen, sah Lehmann den T-Baum als besten Index an. Die obige theoretische Betrachtung bestätigt diese Aussage für kleine Elemente. Speichert der T-Baum große Elemente, so verändert sich das Ergebnis. Der AVL-Baum hat dann die beste Speicherausnutzung und benötigt weniger Vergleiche als der T-Baum. Da dann die Vergleiche bei den Änderungsoperationen einen sehr viel höheren Aufwand bedeuten, reduziert sich der Nachteil gegenüber den T- und B-Bäumen, den AVL-Bäume durch die Rebalancierung haben. Der Unterschied zwischen T- und B-Bäumen reduziert sich bei den Such- und Änderungsoperationen auf großen Elementen, da der Vergleich auf kleiner und größer als zwei Elementvergleiche angesehen werden muss. Der Speicherverbrauch ist bei T-Bäumen gut, während er bei B-Bäumen sehr schlecht ist. Bei großen Elementen ist der AVL-Baum daher der beste Index.

4.4.4 Einfluss auf das Speichersystem

Das Speichersystem besitzt bei herkömmlichen DBS eine Pufferverwaltung, aus der die Daten kopiert werden müssen. HSDBS verzichten auf eine Pufferverwaltung. Der Zugriff auf die Daten erfolgt direkt und ohne Kopieroperationen oder aufwendige Verwaltungsoptionen. Die CPU benötigt dadurch nur einen Bruchteil der Rechenleistung. Ein weiterer Vorteil ist die Vereinfachung dieser Komponente.

4.4.5 Einfluss auf das Transaktionssystem

Transaktionen müssen auch bei HSDBS dem ACID Prinzip entsprechen. Allerdings unterscheiden sich einige Implementierungsdetails. Der erste Unterschied besteht in der Anzahl der parallel laufenden Transaktionen. Bei herkömmlichen DBS lässt sich die Anzahl der verarbeiteten Transaktionen pro Minute erhöhen, indem man mehrere Anfragen parallel verarbeitet. Da die einzelnen Prozesse sehr oft und auch relativ lange warten müssen bis Daten eingelesen wurden, kann der Prozessor Teile einer anderen Anfrage in der Zwischenzeit bearbeiten. Die Geschwindigkeit wird gesteigert, indem beim Zugriff auf die Daten eine feine Nebenläufigkeitskontrolle benutzt wird, damit auch bei schreibendem Zugriff nur die Daten gesperrt sind, die geändert werden und möglichst wenige parallele Transaktionen blockiert werden.

Die CPU-Auslastung ist bei HSDBS hoch, da die CPU auf kein anderes Gerät warten muss. Werden Anfragen nun von mehreren Prozessen parallel verarbeitet, so sinkt wegen der nun notwendigen Nebenläufigkeitskontrolle und der Prozessverwaltung durch das Betriebssystem die Anzahl Transaktionen pro Minute. Der Bedarf an einer feinen Nebenläufigkeitskontrolle beim Zugriff auf die Daten ist viel geringer, da der Zugriff auf die Daten sehr schnell ist und ein großer Teil der Rechenleistung auch beim Parsen der Anfrage und des Ergebnisses verbraucht wird. Die Anzahl der verwendeten Prozesse sollte deshalb kaum größer sein als die Anzahl der CPUs.

Die Anzahl der Transaktionen kann sehr hoch sein (100.000 TPS), wodurch hohe Anforderungen an das Transaktionssystem gestellt werden. Die

Rechenleistung, die für das Transaktionssystem benutzt wird, gewinnt sehr viel stärker an Bedeutung, aber auch die Datenmengen sind sehr viel größer als bei herkömmlichen Datenbanken. Liefert man beispielsweise 100 Byte für ein Commit zurück, so werden bei 100.000 TPS ca. 10 MB/s an Daten erzeugt. Ein Fast Ethernet Netzwerk (100 Mbit/s = 12,5 MB/s) wäre allein durch Commits fast ausgelastet. Eine Möglichkeit dies zu reduzieren ist die Verwendung von Group Commits (Commits, die mehre Transaktionen bestätigen).

Ein besonders großes Problem ist es die Dauerhaftigkeit von Transaktionen bei einer hohen Anzahl von Transaktionen zu gewährleisten. Persistenz kann durch Hauptspeicher nicht erreicht werden, da Hauptspeicher flüchtig ist, Hardwarefehler zu einem Totalverlust der Daten führen und durch die CPU direkt angesprochen werden kann und so die Wahrscheinlichkeit eines Datenverlusts bei einem Softwarefehler höher ist als bei Festplatten. Die Flüchtigkeit von Hauptspeicher lässt sich eliminieren, indem man Batterien benutzt, die den Speicher auch bei Stromausfall mit Strom versorgen. Hardwarefehler lassen sich durch Fehlererkennungs und -korrekturmaßnahmen und dem Einsatz von Redundanzen reduzieren. Allerdings müssen auch bei Einsatz solcher Techniken Sicherungskopien angelegt werden: Softwarefehler bedrohen trotzdem die Daten und Rambausteine lassen sich zur Laufzeit nur bei spezieller und teuerer Hardware auswechseln. Außerdem benötigen Rambausteine im Gegensatz zu Festplatten Strom, um ihre Daten zu speichern. Ein Ausfall der Sicherungsmechanismen ist daher mit Datenverlusten verbunden (vgl. [Garcia-Molina & Salem 1992]).

Bei hohen Updateraten wird der Zugriff auf die Festplatten sehr schnell zu einem Geschwindigkeitsproblem, wie folgendes Beispiel zeigt:

Bei jeder Transaktion werden 1 KB an Daten geschrieben, der wahlfreie Zugriff auf eine Festplatte dauert 1 ms und die Daten können mit 160 MB/s geschrieben werden. Speichert man die Daten sofort auf der Festplatte, so kann man wegen der langsamen Zugriffszeit im Durchschnitt nur 1000 Transaktionen/s durchführen. Schreibt man nur eine Logdatei und liest diese bei einem Fehler wieder ein, so kann man bis zu 160000 Transaktionen/s durchführen. Allerdings ist dabei der Speicherverbrauch sehr groß: Nach 10.000 s = 2,7 h sind 1,6 Terrabyte an Daten geschrieben worden. Sinnvoller als diese beiden Varianten ist es Sicherungspunkte zu verwenden, an denen die Datenbank gesichert wird und so die Größe der Logdatei wieder verkürzt werden kann und auch die Zeit für ein Recovery verkürzt werden kann.

Systeme wie z.B. TimesTen ermöglichen es, unterschiedliche Einstellungen für die Dauerhaftigkeit einer Transaktion zu verwenden. Die Spanne reicht dabei von garantierter Dauerhaftigkeit bis hin zu Einstellungen völlig ohne Sicherungen auf Festplatte, aber dafür höchster Geschwindigkeit [TimesTen 1999b].

Ändern sich die Daten in nur sehr geringem Umfang, so kann man auch parallel zur HSDBS eine normale DBS zur Speicherung der Daten benutzen. Die HSDBS dient dann zur effizienten Verarbeitung von Anfragen, während die DBS für die Persistenz zuständig ist.

4.5 Weitere Besonderheiten

Durch speziell für HSDBS ausgelegete Joins sind Geschwindigkeitsvorteile möglich ([De Witt et al 1984], [Lehman & Carey 1986b], [Manegold, Boncz & Kersten 1999]). Weiterhin kann man Zeiger für Joins benutzen, wodurch ein Join praktisch keine Kosten verursacht. Dies lässt sich bei den verwendeten Datenmodellen einer HSDBS ausnutzen. Beispielsweise speichert Monet die Spalten einer Tabelle getrennt in sogenannten BATs ab und verknüpft die Spalten bei Bedarf wieder [Boncz & Kersten 1994]. Dies hat den Vorteil, dass nur die Teile der Tabelle im Hauptspeicher sein müssen, die auch tatsächlich benötigt werden. Wenn nicht alle Daten der Datenbank in den Hauptspeicher passen, so kann man so die Daten, die im Hauptspeicher sein müssen und die zu ladende Datenmenge auf die notwendigen Spalten reduzieren. Hierdurch kann selbst bei Anwendungen, die Datenmengen besitzen, die weit größer als der Hauptspeicher sind, der Einsatz von HSDBS sinnvoll sein. Monet kann durch diese Zerlegung die I/O Aktivitäten reduzieren und kann so große Geschwindigkeitsvorteile im Vergleich zu einem herkömmlichen relationalen Datenbanksystem erlangen, die die ganze Tabelle einlesen. In [Boncz, Rühl & Kwakkel 1998] konnte Monet ein relationales Datenbanksystem um Größenordnungen schlagen. Die Systeme mussten auf einer großen Tabelle mehrfach Scans durchführen. Hierbei wurden nur wenige Spalten der Tabelle betrachtet. Beide Systeme hatten die gleiche Speicherausstattung, die groß genug war, um die für eine Anfrage notwendigen Spalten einer Tabelle im Hauptspeicher halten zu können. Monet hat nur die für die Anfrage notwendigen Spalten eingelesen, während die relationale DBS erst bei der Verwendung einer temporären Tabelle die Daten in Speicher halten konnte. Allerdings musste die relationale Datenbank hierfür die Daten der ganzen Tabelle einlesen und hat daher eine sehr viel größere Datenmenge eingelesen. Die relationale Datenbank benötigte bei der unoptimierten Anfrage Stunden. Bei einer Verwendung der temporären Tabelle benötigte es 3 Minuten für das Erstellen der temporären Tabelle und machte danach keine Festplattenzugriffe. Die komplette Verarbeitung der Anfrage dauerte 22 Minuten. Monet benötigte für das Einlesen der Daten und für die Verarbeitung der Daten zusammen 39 Sekunden.

4.6 Die Geschwindigkeit von HSDBS

Die Anbieter von HSDBS geben an, dass ihre Datenbanken im Vergleich zu herkömmlichen DBS mit sämtlichen Daten im Puffer zwischen 10-100 (TimesTen) oder bis zu 80 fach (Lucent) schneller sind. In den angegebenen Benchmarks erreichen sie allerdings „nur” einen Faktor zwischen 10 und 17. Monet konnte bei dem oben erwähnten Benchmark eine Geschwindigkeitssteigerung bis zum 80-fachen erreichen und war selbst dann noch schneller, wenn die Datenmenge nicht in den Hauptspeicher passte. Da sich diese Datenbanken sehr stark voneinander unterscheiden, kann es bei dem Vergleich sehr leicht passieren, dass diese Geschwindigkeitssteigerungen auch durch andere Komponenten, wie z.B. Anfrageoptimierern zustandekommen und so den Vergleich verfälschen.

Innerhalb von IBM's Starburst Projekt wurden zwei Komponenten für die Speicherung der Daten implementiert: Eine für die Speicherung auf Festplatten und eine für die Speicherung im Hauptspeicher. Hierdurch konnten beide Komponenten miteinander verglichen werden, ohne dass die anderen Komponenten den Benchmark durch unterschiedliches Verhalten beeinflussen konnten. Die beiden Komponenten unterschieden sich in den verwendeten Indexarten (B+ Bäume vs T-Bäume) und dem Zugriff auf die Daten (Puffer vs ohne Puffer). Die Daten befanden sich bei beiden Komponenten vollständig im Hauptspeicher. Als Benchmark wurde der Wisconsin TenK Benchmark benutzt. Die Hauptspeicherkomponente war zwischen 2.3 und 7.1 mal schneller [Lehman, Shekita & Cabrera 1992].

4.7 Anwendungsgebiete

HSDBS können eingesetzt werden, sobald alle Daten oder zumindest ein großer Teil der Daten in den Hauptspeicher passen. Wenn es einen Teil der Daten gibt, auf die sehr häufig zugegriffen wird (Hot Spot), so kann es bereits reichen, diese im Hauptspeicher zu halten und die selten benötigten Daten auszulagern, um eine HSDBS sinnvoll verwenden zu können. Ihre Einsatzgebiete sind Systeme in denen wie z.B. bei PDAs nur Hauptspeicher vorhanden ist, sehr hohe Transaktionsraten benötigt werden z.B in Telekommunikationsnetzwerken oder Echtzeitanwendungen, bei denen die Verzögerungen bei einem Plattenzugriff zu groß wären.

4.8 Neuere Entwicklungen

In den letzten Jahren hat sich die Zugriffslücke zwischen CPU und Hauptspeicher zu einer Bremse für die CPU entwickelt, da die Geschwindigkeit der CPUs stark gewachsen ist, während der Hauptspeicher kaum schneller wurde. Deshalb müssen nun die Operationen, die von einer DBMS verwendet werden nun auch dies berücksichtigen und nicht nur die verwendete CPU-Zeit. Beispielsweise beschleunigten sich bei Starburst bei der Verwendung der Hauptspeicherkomponente auch die anderen Programmteile, die identisch zur Festplattenkomponente waren, da der CPU-Cache nun mehr Programmteile dieser Komponenten enthalten konnte.

Hauptspeicherdatenbanksysteme hatten bislang immer Systeme mit großem Speicherausbau und hohen Performanceansprüchen als Zielgruppe. Durch das Aufkommen von PDAs und dem Bedarf von mobilen Datenbanksystemen, die nur Hauptspeicher zur Verfügung haben, sind nun auch sehr kleine Systeme Zielgruppen für HSDBS geworden. Da PDAs anders gestaltet sind als herkömmliche Systeme, müssen die HSDBS angepasst werden.

Durch die PDAs ist ein großer Bedarf an nicht flüchtigem, relativ schnellem und wenig stromverbrauchenden Speicher entstanden. Entwicklungen wie MRAM oder FRAM haben das potential die flüchtigen DRAMs zumindest bei den PDAs abzulösen [Mengel & Henkel 2001]. Durch solche nicht-flüchtigen Speicher und Schutzmechanismen vor Datenveränderungen durch abstürzende Programme könnte der Zugriff auf Festplatten für HSDBS unnötig werden.

4.9 Bedeutung für den Indoor SpaSe

Der Indoor SpaSe sollte relativ einfach aufgebaut sein und benötigt keine Pufferverwaltung. Der Optimierer sollte nur einfache und effektive Optimierungen vornehmen, da sonst die Zeit für das Ermitteln einer Optimierung länger ist als die daraus resultierende Einsparung. Es sollte ein direkter Zugriff auf die Objekte verwendet werden und daher möglichst Referenzen statt Kopien benutzt werden. Eine Möglichkeit für die Sicherung der Daten sollte im Indoor SpaSe möglich sein. Es bedarf nur einfacher Mechanismen, da Datenverluste nur geringe Schäden verursachen und die Wahrscheinlichkeit eines Datenverlusts im Endeffekt hauptsächlich in Softwarefehlern besteht, die sich auch in Recovery-Algorithmen befinden können. Der Indoor SpaSe sollte nur wenig Parallelität benötigen, da er eine sehr hohe CPU-Auslastung erzeugen wird und das Verwalten von Threads selbst sehr viel CPU-Aufwand kostet. Der Indoor SpaSe soll nur wenig Daten enthalten. Optimierungen dürften bei der geringen Anzahl von Objekten nicht notwendig sein. In Szenarien bei denen mehr Objekte benötigt werden, sollten AVL-Bäume zur Beschleunigung eingesetzt werden, da bei Zeichenketten der Vergleich relativ lange dauert und in erster Linie nur lesender Zugriff nötig ist. Die Kosten für ein rebalancieren sind deshalb unwichtig. Die Speicherauslastung ist wegen der geringen Datenmenge unbedeutend. Sind sehr hohe Transaktionsraten verlangt, so sind Optimierungen nicht nur durch Verwendung von effizienten Indexen sinnvoll. Optimierungen beim Datensystem können ebenfalls sinnvoll und notwendig sein.

4.10 Zusammenfassung

Spatial Model Server und Datenbanksysteme besitzen Ähnlichkeiten. Für einen hochperformanten und hauptspeicherbasierten Spatial Model Server sind daher auch Konzepte und Methoden bei Hauptspeicherdatenbanksystemen sinnvoll.

Hauptspeicherdatenbanken und herkömmliche Datenbanken sind nach unterschiedlichen Gesichtspunkten optimiert und unterscheiden sich in vielen Punkten. Hauptspeicherdatenbanksysteme benötigen keinen Puffer und haben eine einfache Externspeicherverwaltung, wodurch sie eine unterschiedliche Architektur besitzen. Der Anteil der einzelnen Komponenten an der Ausführungszeit einer Anfrage unterscheidet sich bei ihnen. Die unterschiedlichen Optimierungskriterien führen dazu, dass sich ihre Indexe und verwendeten Funktionen beispielsweise zur Realisierung eines Joins unterscheiden. Die hohe Geschwindigkeit führt bei Hauptspeicherdatenbanksystemen zu weiteren Problemen. Ein Beispiel hierfür sind die Probleme des Transaktionssystems, um die Persistenz der Daten zu garantieren ohne zu einer Systembremse zu werden. Das Ergebnis ist eine deutlich höhere Geschwindigkeit der Hauptspeicherdatenbanksysteme im Vergleich mit herkömmlichen Datenbanksystemen, die sämtliche Daten im Puffer halten. Entwicklungen wie die Verbreitung von PDAs, die nur Hauptspeicher besitzen führen zu neuen Anwendungsgebieten für Hauptspeicherdatenbanksysteme.

5 Anforderungen an den Indoor Spatial Model Server

Spatial Model Server haben unterschiedliche Aufgaben zu erfüllen und haben dabei verschiedene Kommunikationspartner. In diesem Kapitel werden die Aufgaben des entwickelten Indoor Spatial Model Servers beschrieben und dazugehörige Anwendungsfälle mit Hilfe von Use Cases vorgestellt. Danach werden die hierfür zu speichernden Daten beschrieben und die nicht-funktionalen Anforderungen an den Indoor Spatial Model Server vorgestellt. Besondere Aufmerksamkeit wird dabei den Leistungsanforderungen gewidmet.

5.1 Aufgaben

SpaSe verwalten statische Objekte, müssen Anfragen in AWQL verarbeiten und können als Event Source für den Event Service dienen. Der Indoor SpaSe wird Daten für Nexus-Anwendungen und für die Sensorkomponente verwalten. Die Daten sollen gelesen und verändert werden können. Als Event Source werden Events angeboten, die Änderungen der Daten signalisieren. Indoor SpaSe werden für die Verwaltung von Objekten innerhalb eines Gebäudes oder eines kleinen Gebietes eingesetzt. Sie können für kommerzielle und nicht-kommerzielle Zwecke verwendet werden. Sie können Daten für Museumsführer speichern oder Informationen zu den Topfpflanzen in einer Wohnung verwalten. Das Einsatzszenario ist im Rahmen der Diplomarbeit das Innere des Fakultätsgebäudes. Der Indoor SpaSe soll dabei ein Positionierungssystem für Innenräume unterstützen und Raumdaten für Nexus-Anwendungen bereitstellen. Besucher der Fakultät sollen hierdurch in der Lage sein, sich im Gebäude zu positionieren und Informationen zu einzelnen Räumen zu erhalten.

5.2 Schnittstellen zu anderen Systemen

SpaSes bieten eine Schnittstelle für AWQL-Anfragen an. Innerhalb der Nexus-Architektur stellen Nexus-Anwendungen und Sensorkomponenten ihre Anfragen an einen Nexus-Knoten, der diese Anfragen bearbeitet und an die davon betroffenen SpaSes sendet. SpaSes erhalten daher nur von Nexus-Knoten AWQL-Anfragen. Da noch keine Nexus-Knoten existieren, wird der SpaSe im Rahmen der Diplomarbeit mit mobilen Rechnern direkt kommunizieren. Die Schnittstelle des SpaSes muss hierfür nicht angepasst werden, da sowohl die Schnittstelle eines Nexus-Knoten als auch die eines Spatial Model Servers AWQL verstehen. Es gibt zwei Komponenten auf den mobilen Rechnern, die AWQL-Anfragen aussenden. Die Sensorkomponente sendet Anfragen, um eine Positionsbestimmung durchzuführen und die Nexus-Anwendungen stellen Anfragen, um Daten über statische Objekte zu erhalten.

Spatial Model Server benötigen in der Regel eine Schnittstelle, mit der Administrationsaufgaben durchgeführt werden können. Diese Schnittstelle kann beliebig aussehen.

Ein SpaSe muss sich beim Area Service Register mit seinen verwalteten Augmented Areas registrieren.

Der SpaSe registriert seine beobachtbaren Eventtypen beim Event Service. Er bietet eine Schnittstelle für den Event-Service an, über die Basic Events registriert werden können. Wird ein Event ausgelöst, so wird ein Basic Event direkt zum Event-Service versendet.

Die Abbildung 5-1 zeigt die Kommunikation des Indoor Spatial Model Servers mit den anderen Komponenten der Nexus-Architektur. Der Spatial Model Server hat als Kommunikationspartner Nexus-Knoten, das Area Service Register, den Event Service. Nexus-Anwendungen und die Sensorkomponente greifen direkt auf den SpaSe zu, wenn kein Nexus-Knoten verfügbar ist. Darüberhinaus werden Administratoren auf den Indoor SpaSe zugreifen.

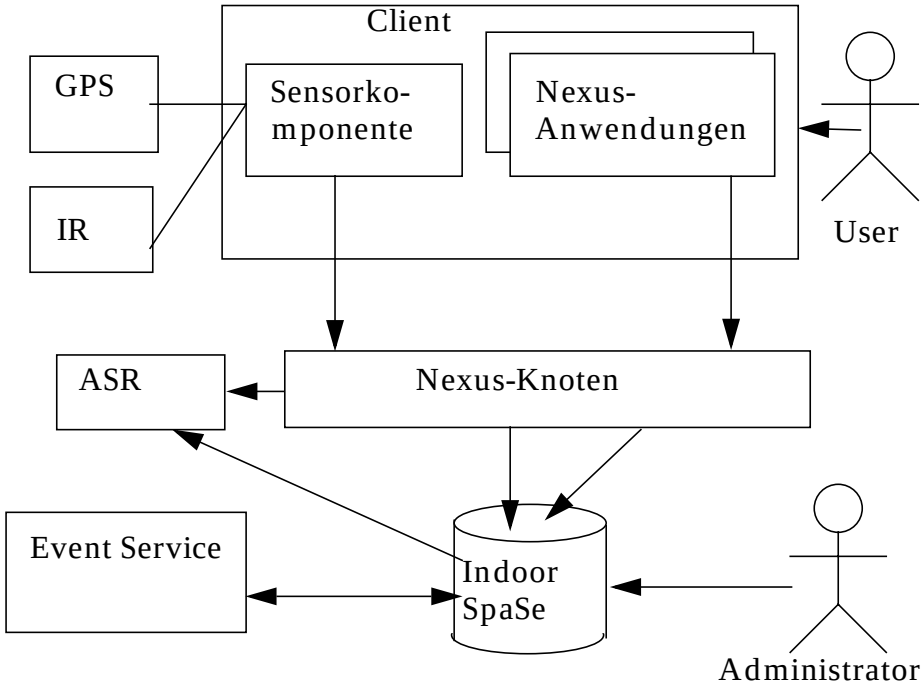


Abbildung 5-1: Kommunikation innerhalb der Nexus-Architektur

5.3 Funktionale Anforderungen

Die Kommunikationspartner des SpaSe benötigen den Indoor SpaSe für unterschiedliche Aufgaben. Die Anwendungsfälle lassen sich in die folgenden Gruppen einteilen:

- Anfragen von Nexus-Anwendungen
- Positionierungsanfragen durch die Sensorkomponente
- Verwaltung des SpaSe durch einen Administrator
- Kommunikation mit dem Event Service

Abbildung 5-2 zeigt die Akteure und die von ihnen verwendeten Use Cases.

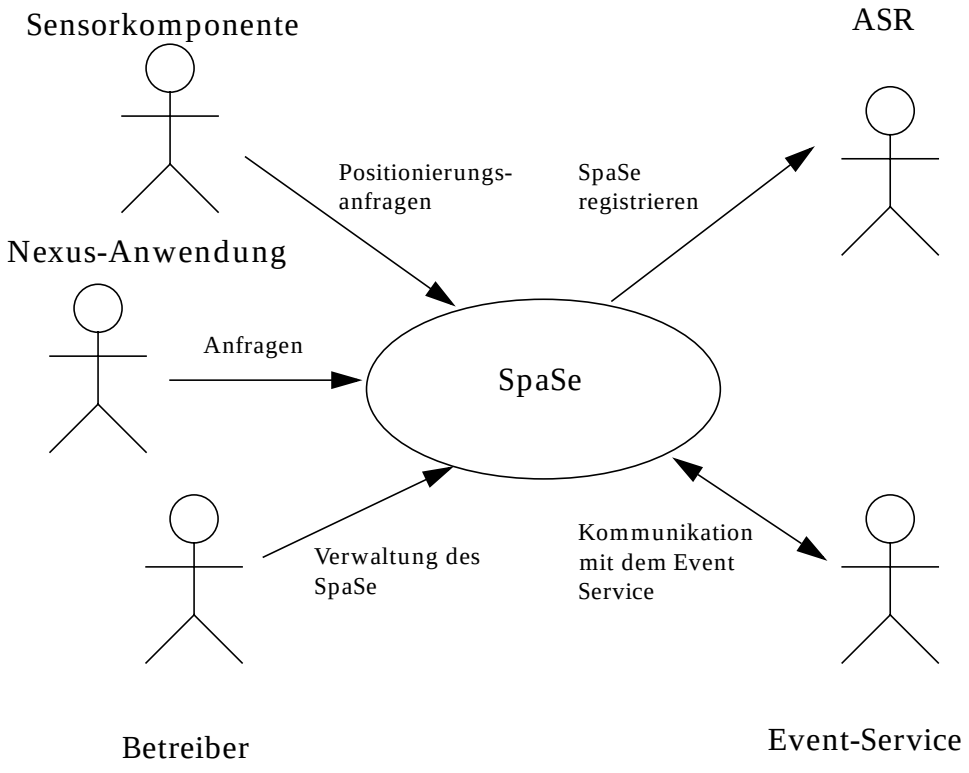


Abbildung 5-2: Die Akteure und ihre Use Cases

5.3.1 Use Case 1: Anfragen von Nexus-Anwendungen

Use Case 1.1: räumliche Anfragen

Bei räumlichen Anfragen müssen die gesuchten Objekte neben allgemeinen Bedingungen auch räumliche Bedingungen erfüllen. Da Nexus-Anwendungen ortsbasierte Anwendungen sind, sind räumliche Anfragen die häufigsten Anfragen an einen SpaSe.

Räumliche Bedingungen sind das Enthalten sein oder der Schnitt einer Position oder Ausdehnung eines Objekts mit einem angegebenen Polygon. Hierdurch kann man z.B. alle Räume anfordern die sich innerhalb eines angegebenen Polygons befinden. Eine andere Art von räumlicher Bedingung ist die Möglichkeit, ein Ergebnis auf die n räumlich nächsten Objekte zu einem Punkt zu beschränken. Ein Beispiel ist eine Anfrage nach dem nächstgelegenem Drucker.

Use Case 1.2: Anfragen ohne Ortsbezug

In AWQL lassen sich Anfragen formulieren, in denen die gewünschten Objekte nicht auf ein Gebiet beschränkt werden. Solche Anfragen enthalten keinerlei Einschränkungen auf ein Gebiet und verlangen auch keine entfernungsabhängige Sortierung der Objekte zu einer Position. Anfragen an einen SpaSe können keine echten ortsunabhängigen Anfragen sein, da er nur die

Daten bestimmter Gebiete verwaltet und die gewünschten Objekte automatisch auf die verwalteten Gebiete beschränkt sind. Solche Anfragen sind für SpaSes kein Problem und können ohne Probleme ausgewertet werden. Anfragen ohne Ortsbezug können von Nexus-Anwendungen auch direkt an den SpaSe gestellt werden. Nexus-Anwendungen können z.B. Anfragen nach allen Räumen stellen oder nach Räumen mit einer bestimmten Raumnummer. Nexus-Knoten hingegen haben mit solchen Anfragen Probleme, da sie die Anfrage nicht auf die wenigen SpaSes eines abgedeckten Gebiets beschränken können. Theoretisch müssten dann alle SpaSes mit den gewünschten Klassen für die Anfrageverarbeitung benutzt werden, weshalb Nexus-Knoten solche Anfragen nicht zu lassen.

SpaSes erhalten von Nexus-Knoten nur Anfragen ohne Ortsbezug, wenn die geforderten räumlichen Einschränkungen der Objekte im SpaSe ohnehin erfüllt sind und daher entfernt wurden. Dies kann geschehen, wenn eine Nexus-Anwendung alle Augmented Areas eines SpaSe verlangt oder eine räumliche Einschränkung für alle Objekte erfüllt ist.

5.3.2 Use Case 2: Positionierungsanfragen

Positionierungsanfragen werden von der Sensorkomponente auf den mobilen Rechnern gestellt. Die Sensorkomponente kann räumliche Anfragen und, bei direktem Zugriff auf einen SpaSe, Anfragen ohne Ortsbezug stellen. Der Ortsbezug kann in der Regel nicht auf die aktuelle Position gemacht werden, da die Sensorkomponente diese erst durch die Anfrage ermitteln wird. Es werden dann Gebiete wie z.B. der Gebäudeumriss benutzt, in denen das gewünschte Objekt sicher enthalten ist. Als Position kann möglicherweise nur eine veraltete Position benutzt werden. Aufgrund ihrer Bedeutung werden die wichtigsten Positionierungsanfragen im folgenden vorgestellt.

Use Case 2.1: Ein Sender

Die Sensorkomponente empfängt einen Sender. Die ID dieses Senders wird an den SpaSe geschickt. Zurückgeliefert werden soll der Sichtbereich des Senders. Der SpaSe liefert den Bereich zurück, in dem man den Sender empfangen kann.

Use Case 2.2: Zwei Sender und Schnittberechnung durch Client

Die Sensorkomponente empfängt zwei Sender. Die IDs dieser Sender werden an den Server geschickt. Verlangt werden die Sichtbereiche der Sender. Der Spatial Model Server liefert die Sichtbereiche der Sender zurück. Der Client macht daraufhin eine Schnittberechnung.

Use Case 2.3: Zwei Sender und Schnittberechnung durch Server

Wie beim Use Case 3 werden zwei Sender empfangen und deren IDs an den SpaSe gesendet. Es soll nun aber der geschnittene Sichtbereich zurückgeliefert werden.

Use Case 2.4a: Sender eines Bereichs liefern

Es sollen nur die Sender in einem bestimmten Bereich geholt werden. Solche Anfragen würde man benutzen, um das sogenannte Hoarding zu realisieren. Beim Hoarding werden zusätzlich zu den benötigten Informationen noch weitere Informationen geladen, damit die Daten bei einer späteren Anfrage auf dem mobilen Rechner sind und nicht von einem anderen System geladen werden müssen. Hierdurch reduziert sich die Anzahl der notwendigen Anfragen.

Use Case 2.4b: Alle Sender des SpaSe liefern

Die Sensorkomponente soll die Positionierung ohne Kommunikation mit dem SpaSe durchführen. Dafür benötigt die Sensorkomponente alle Sender mit ihren Positionen, ihren Sichtbereichen und ihren ausgestrahlten Signalen. Die Sensorkomponente stellt deshalb eine AWQL-Anfrage an den SpaSe. Der SpaSe liefert alle Sender, Sichtbereiche und ausgestrahlten Signale.

Hierbei handelt es sich aus Sicht des SpaSes nur um eine Variante zu Use Case 2.4a, da der SpaSe nur Sender der verwalteten Bereiche zurückliefern kann. Auch diese Anfrage kann für Hoarding benutzt werden.

Use Case 2.5: Symbolische Positionierung

Bei einer symbolischen Positionierung ist eine ID einem Objekt zugeordnet. Es werden dann die Daten zu dem Objekt geladen. Ein Beispiel wäre die Verbindung von ID und Raumnummer. Empfängt die Sensorkomponente eine solche ID, so stellt sie eine Anfrage nach dem Raum mit der Nummer der ausgesendeten ID. Der SpaSe sucht nach dieser ID und liefert den Raum mit den dazugehörigen Daten zurück.

Use Case 2.6: Umwandlung symbolische zu geometrische Position

Empfängt die Sensorkomponente eine symbolische ID und benötigt die geometrische Position, so muss sie diese umwandeln. Ist eine ID einem Raum zugeordnet, so stellt sie eine Anfrage an den SpaSe, damit dieser die Position oder den abgedeckten Bereich des Raums zurückliefert.

Use Case 2.7: Umwandlung geometrische in symbolische Position

Für eine Umwandlung einer geometrischen Position in eine symbolische Position benötigt die Sensorkomponente alle Objekte, deren Position oder deren abgedeckter Bereich sich mit einer geometrischen Position oder einem bestimmten Bereich überschneidet. Die Sensorkomponente sendet hierfür eine Anfrage an den SpaSe, der dann Daten zu keinem, einem oder mehreren Objekten zurücksendet. Das Ergebnis kann auf eine bestimmte Anzahl von Objekten begrenzt sein und nach der Entfernung sortiert werden. Die Sensorkomponente sucht dann das sinnvollste Objekt aus. Hierfür kann neben der Entfernung auch der Objekttyp ausschlaggebend sein.

5.3.3 Use Case 3: Verwaltung des SpaSe

Administratoren verwalten den SpaSe und müssen hierbei den Betrieb des SpaSe sicherstellen und die gespeicherten Daten verwalten. Die Schnittstellen und die Möglichkeiten für die Verwaltung eines SpaSes können sich zwischen

SpaSes unterscheiden. Die folgenden Use Cases beschreiben Anwendungsfälle, die bei SpaSes in aller Regel anfallen.

Use Case 3.1: Start des SpaSe
Der SpaSe Administrator startet den SpaSe. Der SpaSe lädt Konfigurationsdaten und Programmdaten und registriert sich beim ASR. Danach können Anfragen an den SpaSe gestellt werden.

Use Case 3.2: Konfiguration
SpaSes können für die Konfiguration unterschiedliche Schnittstellen und Möglichkeiten bereitstellen. Es könnten Informationen zum SpaSe bereitgestellt werden oder auch Einstellungen des SpaSe verändert werden. Hierbei könnte man beispielsweise bestimmen, welche Indexe benutzt werden sollen. Die Schnittstelle kann auch Befehle an den SpaSe anbieten, die ihn z.B. dazu veranlassen, die verwalteten Augmented Areas auf einer Festplatte abzuspeichern.

Use Case 3.3: SpaSe runterfahren
Der SpaSe soll runtergefahren werden. Deswegen gibt der Betreiber dem SpaSe den Befehl runterzufahren, woraufhin sich der SpaSe aus dem ASR austrägt und herunterfährt. Hierbei müssen dann eventuell Konfigurationsdaten oder die verwalteten Daten in AWML-Dateien gespeichert werden.

Use Case 3.4: Einfügen eines neuen Objekts
Ein neues Objekt soll in den SpaSe eingefügt werden. Dies geschieht durch ein AWQL-Insert Befehl. Ein Beispiel:

Der Administrator des SpaSe stellt einen neuen Sender auf. Durch einen AWQL-Insert Befehl wird der Standort, der Sichtbereich und die ausgestrahlte ID des Sensors dem SpaSe mitgeteilt. Der SpaSe liefert ein Change Report zurück.

Use Case 3.5: Verändern eines Objekts
Ein bestehendes Objekt soll mit Hilfe eines AWQL-Update Befehls verändert werden. Ein einfaches Beispiel:

Der Sichtbereich eines Senders hat sich verändert, da ein Gegenstand die Sicht behindert. Der Administrator sendet den neuen Sichtbereich an den SpaSe, der den neuen Sichtbereich intern einstellt und ein Change Report zurückliefert.

Beim Verändern eines Objekts kann es zu Konsistenzproblemen kommen, wenn andere Objekte eine Kopie dieses Objekts sind oder auf dieses Objekt verweisen. Als Verweis auf ein Objekt werden eindeutige IDs verwendet. Solche eindeutige IDs sind Nexus Object Locators (NOL), die ein Attribut vieler Objekte sind. Wird eine solche ID verändert, so besitzen die Verweise die alte ID. Ein Beispiel verdeutlicht das Problem:

Ein Sender hat eine NOL mit dem Wert „nol:1”. Ein anderes Objekt speichert einen Verweis auf alle Sender und benutzt hierfür NOLs. Der Verweis auf den Sender hat den Wert „nol:1”. Der Sender erhält ein Update auf die NOL. Der NOL-Wert ist nun „nol:2”. Die Verweise enthalten immer noch den Wert „nol:1”. Die Suche nach einem Objekt mit einem solchen NOL-Wert wird nun entweder erfolglos sein oder liefert ein falsches Objekt zurück, wenn ein anderes Objekt in der Zwischenzeit den Wert „nol:1” für seine NOL erhalten hat.

Use Case 3.6: Löschen eines Objekts

Ein bestehendes Objekt soll durch ein AWQL-Delete Befehl gelöscht werden. Wie in Use Case 3.5 kann es zu Problemen mit der Konsistenz kommen, da andere Objekte auf dieses Objekt verweisen können. Ein Beispiel:

Ein Sender wird entfernt. Der Administrator löscht diesen nun aus dem SpaSe durch einen AWQL-Delete Befehl. Der Server liefert einen Change Report zurück.

Use Case 3.7: Einfügen einer neuen Klasse

Klassen können ebenfalls auf unterschiedliche Art und Weise in SpaSes eingefügt werden. Es ist auch möglich, dass es SpaSes gibt, die auf bestimmte Klassen beschränkt sind und keine weiteren Klassen aufnehmen können. In Abschnitt 5.5 „Besonderheiten des Prototyps” wird auf Probleme beim Einfügen einer neuen Klasse genauer eingegangen. Eine neue Klasse lässt sich beim Indoor SpaSe nur durch Verändern des Programmcodes erzeugen. Ein Beispiel für einen neuen Sendertyp zeigt den Ablauf:

Der Administrator benötigt einen neuen Sendertyp im SpaSe. Es handelt sich dabei um ein Funk-System, bei dem zusätzlich zu Position, Sendebereich und ausgestrahlter ID noch die Frequenz benötigt wird. Hierfür erstellt er eine neue Klasse mit dem Namen „Funksender”. Er definiert ein Extended Class Schema und muss im Quellcode neue Klassen für die Verarbeitung und Speicherung von Objekten der Klasse Funksender erstellen. Zusätzlich müssen noch weitere Stellen im Quellcode geändert werden. Die Konfigurationsdateien müssen ebenfalls geändert werden, damit die Einträge des ASR stimmen. Der SpaSe wird neukompiliert, der laufende SpaSe heruntergefahren und der ausführbare Code ersetzt. Danach startet der Betreiber den SpaSe neu.

5.3.4 Use Case 4: Kommunikation mit dem Event-Service

Nicht jeder SpaSe dient als Event Source für den Event Service. Der entwickelte Indoor SpaSe kann als solcher eingesetzt werden.

Use Case 4.1: Bekanntmachen eines Event Templates

Ein neuer Event Typ muss beim Event Service registriert werden. Hierfür wird eine Template ID für den Event Typ erzeugt. Zusätzlich zur Template ID werden weitere Informationen über den Event mitgeliefert. Hierzu gehören die Parameter des Events und eine Beschreibung des Events. Die Informationen erstellt der Administrator des SpaSe. Diese Informationen werden dann an den Event Service versendet. Ein Beispiel wäre ein Update Event bei dem der Parameter ein Objekttyp ist.

Use Case 4.2: Registrieren eines Basic-Events
Der Event-Service registriert einen Basic Event bei einem SpaSe. Der Event Service muss hierfür die notwendigen Parameter des Events angeben. Der SpaSe erhält zusätzlich die Adresse zu der die Event Notification gesendet werden soll. Der SpaSe erzeugt dann eine Predicate ID und gibt sie an den Event Service weiter.

Use Case 4.3: Versenden einer Event Notification
Tritt ein Event ein, so sendet der SpaSe die zugehörige Event Notification an den Event-Service oder an die vom Event Service angegebene Adresse.

5.4 Komplexe Use Cases

Die bisher vorgestellten Use Cases sind einfache Anwendungsfälle und zeigen die Anwendungsfälle aus Sicht des SpaSes. Dieser Abschnitt soll zwei komplexere Use Cases vorstellen, damit man das Zusammenspiel der einfachen Use Cases, aus Sicht von Nexus-Anwendungen sieht.

5.4.1 Besucher der Fakultät

In diesem Use Case läuft ein Besucher der Fakultät durch das Gebäude. Als Positionierungssystem werden Infrarot Sender benutzt. Allerdings gibt es einen Bereich, der keine Infrarot Sender enthält. Die Anwender müssen ihre Position dann selbst markieren. Die verwendete Nexus-Anwendung zeigt die Benutzerposition an und liefert bei Bedarf Informationen zu Räumen.

Der Besucher betritt die Fakultät. Die Sensorkomponente empfängt einen Sender und sendet eine AWQL Anfrage (Use Case 2.1: Ein Sender). Die Position wird auf einer Karte angezeigt. Ein paar Meter weiter empfängt die Sensorkomponente einen zweiten Sender und stellt eine AWQL Anfrage für beide Sender (Use Case 2.2: Zwei Sender und Schnittberechnung durch Client). Der Anwender läuft weiter und geht in den Hörsaal V20.01. Hier möchte er wissen in welchem Raum er sich befindet. Die Nexus-Anwendung verlangt eine symbolische Positionierung. Die Sensorkomponente stellt eine AWQL Anfrage an den SpaSe, die die geometrische Position in eine symbolische umwandelt (Use Case 2.5: Symbolische Positionierung). Es wird Raum V20.01 zurückgeliefert. Der Besucher verlässt den Hörsaal und geht zum Computer Museum, das noch geschlossen hat. Der Besucher informiert sich über das Computer Museum (Use Case 1.1: räumliche Anfragen) und möchte informiert werden, wenn sich das Computer Museum öffnet, und registriert den dazugehörigen Event. (Use Case 4.2: Registrieren eines Basic-Events). Der Besucher bewegt sich danach in einen Bereich ohne Positionierungssystem. Er möchte sich positionieren, findet sich aber auf der Karte nicht zurecht. Die Nexus-Anwendung kann die Position ermitteln, wenn man eine Raumnummer angibt. Daher gibt er eine Raumnummer an. Es wird eine Anfrage an den SpaSe gestellt, um die geographischen Daten des Raums zu erhalten. (Use Case 2.6: Umwandlung symbolische zu geometrische Position). Danach wird der Raum markiert. Das Computer Museum öffnet. Der SpaSe wird aktualisiert (Use Case 3.5: Verändern eines Objekts) und dabei ein Event ausgelöst. Es wird

eine Event Notification an den Besucher verschickt (Use Case 4.3: Versenden einer Event Notification)

5.4.2 Administrator testet die Sichtbereiche

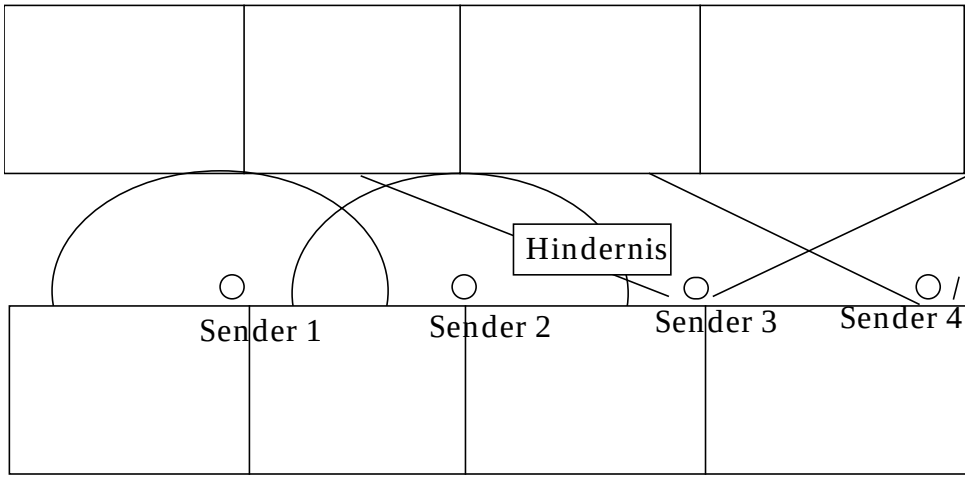


Abbildung 5-3: Karte mit Sichtbereichen der Sensoren

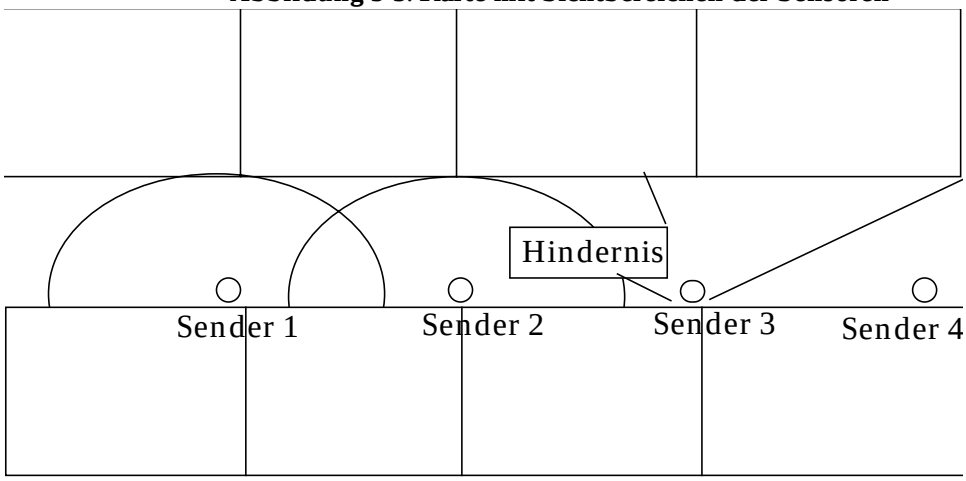


Abbildung 5-4: tatsächliche Sichtbereiche

Ein Administrator möchte die Sichtbereiche von Sendern in einem Gang überprüfen. Er benutzt eine Nexus-Anwendung, die ihm alle Sichtbereiche auf einer Karte anzeigt und die Sichtbereiche der empfangenen Sender markiert. Er geht den Gang entlang und lässt sich alle Sichtbereiche auf seiner Karte anzeigen. Jedesmal wenn er einen neuen Sender empfängt oder einen Sender nicht mehr empfängt, verändern sich die Markierungen. Abbildung 5-3 zeigt die Karte mit den Sichtbereichen, wie sie im SpaSe modelliert sind und Abbildung 5-4 zeigt die tatsächlichen Sichtbereiche.

Da der Administrator für den Test eine Karte benötigt, in der die Sichtbereiche der Sender eingezeichnet sind, stellt die Nexus-Anwendung eine Anfrage, die ihm die Karte und die Sichtbereiche der Sensoren im Gang liefert. Der SpaSe liefert dabei die Sichtbereiche aller Sensoren in dem Gang (Use Case 1.1: räum-

liche Anfragen), während die Karte von einem anderen Dienst bereitgestellt wird. Die Nexus-Anwendung verknüpft nun die Karte mit den Sichtbereichen und stellt eine Karte mit Sichtbereichen auf dem Display dar.

Der Administrator empfängt zu Beginn keinen Sender und tritt dann in das Sendefeld von Sender 1. Nun schickt die Sensorkomponente eine Anfrage an den SpaSe, in der sie Sender 1 angibt und als Antwort den Bereich verlangt, der nur von Sender 1 abgedeckt wird (Use Case 2.1: Ein Sender). Die Nexus-Anwendung markiert den Bereich auf der Karte.

Er läuft weiter und empfängt nun auch Sender 2. Der mobile Client sendet eine Anfrage, in der er Sender 1 und Sender 2 angibt und als Antwort den von diesen abgedeckten Bereich verlangt (Use Case 2.2: Zwei Sender und Schnittberechnung durch Client). Er läuft weiter und empfängt nur noch Sender 2. Kurz darauf empfängt er nun Sender 2 und Sender 3. Er geht ein paar Schritte und Sender 3 verschwindet wieder, obwohl er eigentlich empfangen werden müsste. Der Sender scheint in Ordnung zu sein, allerdings wird der Sichtbereich durch einen Gegenstand verkleinert. Der Betreiber ermittelt nun den neuen Sichtbereich und sendet ein Update an den SpaSe (Use Case 3.5: Verändern eines Objekts).

Er läuft weiter und stellt fest, dass er Sender 4 nicht empfangen kann. Der Grund ist allerdings nicht ein Hindernis, sondern ein Defekt des Senders. Der Sender wird abmontiert und aus dem SpaSe gelöscht (Use Case 3.6: Löschen eines Objekts).

Nachdem er einen neuen Sender (Sender 5) geholt hat, montiert er ihn an die Position von Sender 4 und aktiviert ihn. Der mobile Client läuft noch und sendet an den SpaSe eine Anfrage mit Sender 5. Da der SpaSe allerdings noch keine Informationen über Sender 5 hat, kann er auch den Sichtbereich nicht zurückliefern. Der Betreiber bemerkt das Problem und möchte Sender 5 einfügen. Dabei fällt ihm auf, dass der SpaSe diese Senderart noch nicht kennt. Er erzeugt nun die Senderart (Use Case 3.7: Einfügen einer neuen Klasse), wofür er den SpaSe herunterfahren (Use Case 3.3: SpaSe runterfahren) und wieder starten muss (Use Case 3.1: Start des SpaSe). Danach fügt er den Sender 5 zusammen mit Sichtbereich, ID und sonstigen Informationen in den SpaSe ein (Use Case 3.4: Einfügen eines neuen Objekts).

5.5 Besonderheiten des Prototyps

Zum Zeitpunkt der Diplomarbeit existieren weder ASR, Event-Service, Sensorkomponente noch Nexus-Anwendungen, mit denen der SpaSe kommunizieren könnte. Daher sind einige Schnittstellen noch nicht definiert. Betroffen ist hiervon besonders der Event Service.

Der Event Service befindet sich in einem sehr frühen Stadium, weshalb die Schnittstelle des Event Service und die Schnittstelle einer Event Source

noch nicht definiert ist. Auch das Aussehen von Template IDs, Predicate IDs oder Notification IDs ist nicht definiert.

Das ASR wird nicht unterstützt, da keine Kommunikation möglich ist und die eingetragenen Daten nicht automatisch erzeugt werden können. Das Aussehen von Augmented Areas wird durch Anwender bestimmt. Eine Erzeugung der Augmented Areas, indem das von den Objekten abgedeckte Gebiet angegeben wird führt zu Problemen, da das verwaltete Gebiet auch sehr viel größer sein kann.

Der Prototyp besitzt nur wenige Möglichkeiten zur Konfiguration über eine externe Schnittstelle. Einstellungen werden stattdessen im Quellcode vorgenommen. Neue Objektklassen lassen sich beim Prototypen nur durch Änderungen am Quellcode realisieren. Ein Problem ist, dass man bei XML eine Datendefinitionssprache wie DTD oder XML Schemas interpretieren muss. Ein weiteres Problem besteht darin, dass man in XML zwar Daten aber keine Methoden definieren kann und so beispielsweise keine eigene Entfernungsberechnung implementieren kann.

Eine Schnittberechnung zweier Sichtbereiche durch AWQL (Use Case 2.3) scheint auf den ersten Blick nicht möglich zu sein, da AWQL nicht in der Lage ist, die Attribute mehrerer Objekte miteinander zu verknüpfen. Es gibt allerdings prinzipiell die Möglichkeit, dies durch die Entwicklung eigener Klassen zu realisieren. Eine solche Klasse muss eine Methode anbieten, die als Parameter die zu schneidenden Sichtbereiche erhält. Das Ergebnis der Methode wäre dann der geschnittene Sichtbereich. Solche Klassen sind nicht im Standard Class Schema enthalten und unterscheiden sich von anderen Klassen dadurch, dass sie in erster Linie die Möglichkeiten durch AWQL Anfragen erweitern. Der Prototyp enthält keine solchen Klassen.

5.6 Die verwalteten Daten

Der Indoor SpaSe speichert nur wenige Klassen. Die Angaben für die Position oder Ausdehnung eines Objekts sind zwei Dimensional. Die Höhe wird durch die Angabe des Stockwerks definiert, auf dem sich das Objekt befindet. Für die Positionierungssysteme werden Sender gespeichert. Die Sender (engl. emitter) benötigen nur wenige Daten:

- eine global eindeutige ID vom Typ Nexus Object Locator (NOL)
- kind gibt an, ob es sich um einen realen oder einen virtuellen Sender handelt
- die Position (pos) des Senders
- das Stockwerk (floor), in dem er sich befindet
- den Sichtbereich des Senders (fieldOfVision)

Als Daten für räumliche Anfragen dienen Räume. Nexu-Anwendungen können so Informationen zu Räumen erhalten. Viele Daten werden bereits in anderen Datenquellen gespeichert. SpaSes müssen diese Daten nicht speichern, sondern können zur Vermeidung von Redundanzen Verweise wie URLs benutzen. Die graphische Darstellung und die Informationen zu den

Räumen im Fakultätsgebäude sind bereits über den Webserver zugänglich. Der Indoor SpaSe speichert hierfür nur URLs ab. Räume müssen nicht immer real sein, sondern können auch virtuell sein, wenn man beispielsweise eine Vorführung in mehreren Räumen macht, so kann man statt einer Angabe aller Räume auch einen virtuellen Raum erstellen, der aus den realen Räumen besteht. Virtuelle Räume können auch sinnvoll sein, um Gebiete einzuteilen oder um virtuelle Objekte, wie virtuelle Bilder an eine Wand zu hängen.

Als Daten für Räume werden benötigt:

- eine global eindeutige ID vom Typ Nexus Object Locator (NOL)
- kind gibt an, ob es sich um einen realen oder einen virtuellen Raum handelt
- die Position (pos) sollte ein Punkt innerhalb des Raums sein und wird benötigt, da eine räumliche Sortierung von Objekten oft über ihre Position gemacht wird
- die Ausdehnung (extent) des Raums
- das Stockwerk (floor), in dem er sich befindet
- die Raumnummer (number) und ein Name (name) des Raums
- das Attribut illumination state gibt an, ob in einem Raum Positionierungsinformation empfangen werden kann oder nicht
- Das Attribut representation ist eine URL zur graphischen Darstellung des Raums (In der Regel ein Kartenausschnitt)
- Das Attribut info ist eine URL zu Rauminformationen

Das Gebäude das der Indoor SpaSe verwalten soll, wird ebenfalls gespeichert. Hierdurch können Nexus-Anwendungen Gebäudeinformationen erhalten. Wichtig sind hierbei:

- eine eindeutige ID
- Informationen zu einer graphischen Darstellung (representation) und sonstigen Informationen (info).
- das Attribut kind: Sie können sowohl real als auch virtuell sein. Virtuelle Gebäude können z.B. für die Bauplanung eingesetzt werden, um das Aussehen eines zukünftigen Gebäude vorzuführen
- eine Adresse

Abbildung 5-5 zeigt, wie die Klassen von den Klassen der Augmented World abgeleitet werden. Hierdurch erben die Klassen zusätzliche Attribute die sie nicht benötigen oder nur selten gebraucht werden, wie z.B. das Erstellungsjahr für Räume oder Navigationsknoten.

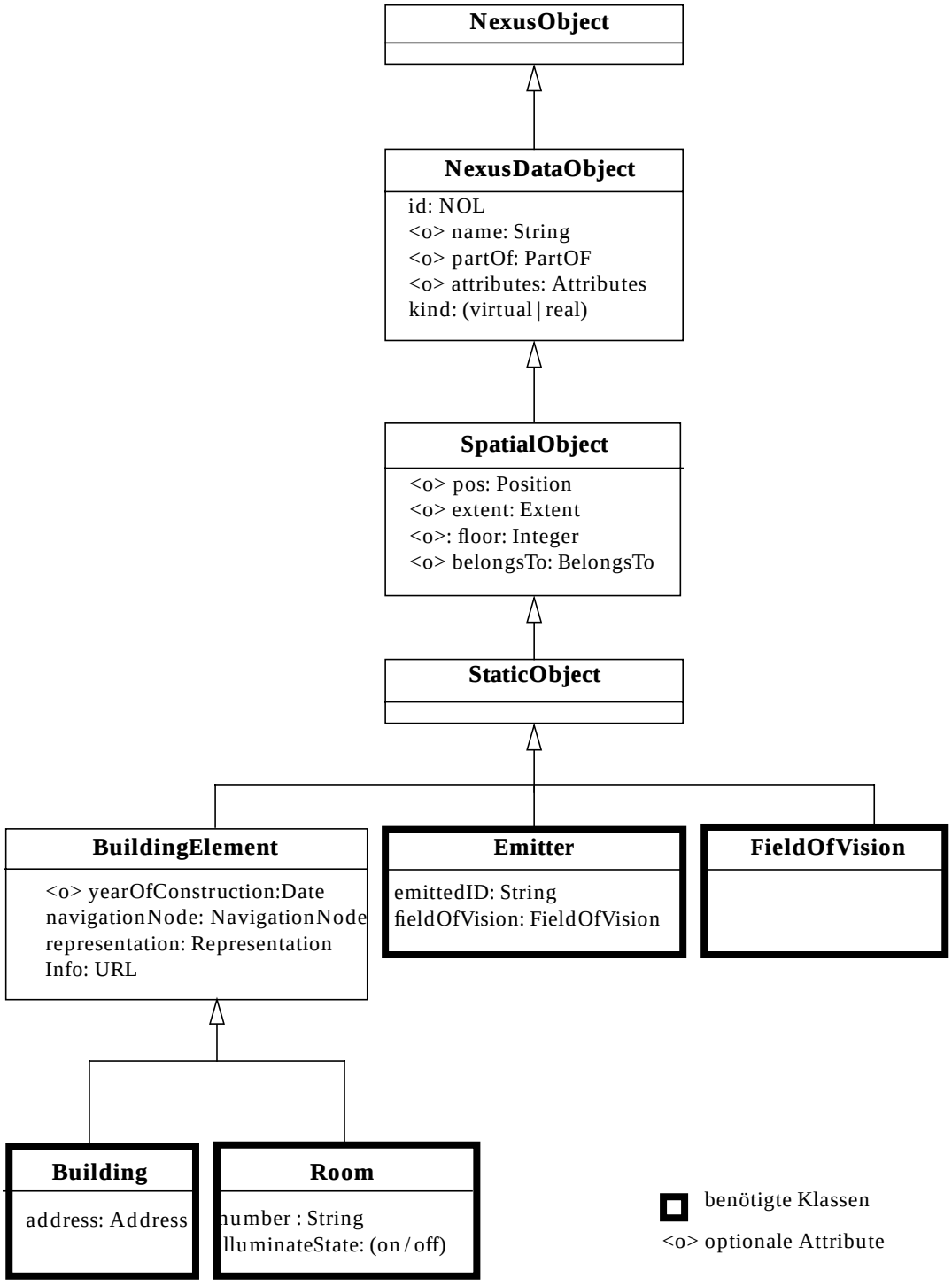


Abbildung 5-5: Die Klassen innerhalb der Augmented World

5.7 Nicht-Funktionale Anforderungen

Die Nicht-Funktionalen Anforderungen an einen SpaSe entsprechen denen von Web Servern. Die Anwendungsgebiete können sehr unterschiedlich sein

und stellen unterschiedliche Anforderungen. An ein SpaSe für das eigene Haus werden andere Anforderungen gestellt als an einen SpaSe, der Tourismusinformationen für eine ganze Großstadt bereitstellen muss. Die folgenden Angaben beziehen sich auf die Anforderungen an einen Indoor SpaSe. Die Leistungsanforderungen werden genauer betrachtet als die anderen Nicht-Funktionalen Anforderungen, da sie für den Prototypen am wichtigsten sind.

5.7.1 Leistungsanforderungen

Die Datenmenge

Die Datenmenge muss sich im Hauptspeicher befinden können. Das Szenario ist das Innere eines Gebäudes und begrenzt so die Anzahl der realen Objekte. Da man wahrscheinlich nur selten Büromaterial wie Bleistifte oder Kugelschreiber modelliert, liegt die Anzahl sinnvoller reale Objekte in der Regel sehr viel niedriger. Bedenkt man, dass die Daten in der Regel manuell eingetragen werden müssen, so ist hier wahrscheinlich eine weitere Obergrenze von ungefähr 1000 Objekte.

Tabelle 5-1: Die Datenmenge

Speicher pro Objekt	1 KB
maximaler Hauptspeicherverbrauch	100 MB - 1 GB
maximale Anzahl Objekte	100.000 - 1.000.000
Anzahl sinnvoller realer Objekte in einem Gebäude	500 - 10.000 Objekte
maximale Anzahl Objekte (manuell eingetragen)	1000
Anzahl Objekte des Indoor SpaSe	701 (500 Sender, 200 Räume, 1 Gebäude)

Anfragen durch Nexus-Anwendungen

Der Anwender akzeptiert bei diesen Anfragen eine etwas längere Wartezeit und wird im Durchschnitt alle fünf Minuten eine solche Anfrage stellen.

Tabelle 5-2: Anfragen durch Nexus-Anwendungen

durchschnittliche Anzahl Anfragen pro Anwender	1 Anfrage alle 5 Minuten
Anzahl Besucher	100
Anzahl Anfragen pro s	0.33
Durchschnittliche Antwortzeit	3s
Maximale Antwortzeit	< 10s
durchschnittliche Größe einer Anfrage	2 KB
Antwortgröße	1 KB - 500 KB

Tabelle 5-2: Anfragen durch Nexus-Anwendungen

durchschnittliche Antwortgröße	5 KB
durchschnittliche Datenmenge	2.33 KB / s

Positionierungsanfragen

Positionierungssysteme benötigen unterschiedliche Positionierungsanfragen und stellen unterschiedliche Anforderungen an einen Indoor SpaSe. Die folgenden Positionierungssysteme werden nicht betrachtet, da sie entweder geringere Leistungsanforderungen haben oder die Verwendung eines SpaSes nicht sinnvoll ist.

- Trackingsysteme benötigen den SpaSe nur zur Initialisierung der Datenbasis. Dies ist nur sehr selten notwendig. Die ausgesendete Datenmenge hängt sehr stark von den gespeicherten Daten ab und kann zwischen wenigen hundert Kilobyte und mehreren Megabyte liegen. Dies ist wahrscheinlich kein Problem für einen SpaSe.
- Pedometersysteme benötigen SpaSes, um die Daten zu Synchronisierungspunkten zu erhalten. Sie stellen daher nur selten Anfragen.
- Für Map Matching speichern SpaSes Karteninformationen. Die übertragene Datenmenge hängt von dem Verfahren ab und muss nur selten übertragen werden.
- Die Datenmenge für Bildverarbeitung ist wahrscheinlich zu groß um über ein Netzwerk transportiert werden zu können.

Für die Positionierungsanfragen werden Systeme mit Sendern betrachtet, die eine ID ausstrahlen, die vom SpaSe umgewandelt werden muss.

Die Leistungsanforderung hängt davon ab, ob Caching, Hoarding oder keines von beiden benutzt wird.

- Wenn der Client kein Hoarding oder Caching benutzt, so muss die Sensorkomponente jedesmal eine Anfrage an den Server stellen, um die Daten für die Positionsbestimmung zu erhalten. Die Eingabemenge ist recht klein. Die Ausgabemenge wird durch die Komplexität der Sichtbereiche bestimmt, die Indoor einfach sein sollten und nach einer Schätzung von Darko Klinec zu 95% weniger als 8 Ecken haben werden.
- Wird Caching benutzt, reduzieren sich die Anfragen an den SpaSe. Wieviel hängt dabei von der Größe des Cache und den Laufwegen der Anwender ab. Die übertragenen Datenmengen pro Anfrage entsprechen denen, wenn kein Cache benutzt wird.
- Wird Hoarding benutzt reduzieren sich die Anfragen, die an den SpaSe gesendet werden. Lädt man alle Sender, so muss nur noch eine einzelne Anfrage bearbeitet werden. Die Anfragen an den SpaSe sind wahrscheinlich relativ klein, da nur die IDs der gewünschten Sender oder ein Bereich, in dem sich die Sender befinden übertragen werden müssen. Sollen alle

Sender übertragen werden, so kann die zurückgesendete Datenmenge ziemlich groß werden, da alle Sichtbereiche und ausgesendeten IDs übertragen werden müssen.

Es ist noch unklar, ob und in welcher Form Caching und Hoarding benutzt werden. Deshalb kann man nicht sagen, ob es eher viele kleine Anfragen an den SpaSe geben wird oder nur wenige große. Weitere Faktoren beeinflussen die Anzahl der Positionierungsanfragen. Die Sensorkomponente könnte z.B. nur bei Anfragen der Anwendungen nach einer Position verlangen oder nur, wenn der Anwender steht. Die folgenden Angaben sind für das Fakultätsgebäude geschätzt. Die Positionierung sollte in 1 bis 3s möglich sein. Wird Hoarding benutzt, so kann eine solche Anfrage auch etwas länger dauern. Maximal 10 Sekunden sollten akzeptabel sein.

Verarbeitungsgeschwindigkeit der Positionierungsanfragen

Tabelle 5-3: Allgemeine Daten

Nettobandbreite	5 MBit / s (625 KB / s)
Anzahl Besucher	100
maximale Anzahl gleichzeitig betretender Besucher	5
Durchschnittsgeschwindigkeit eines Besuchers	1 m / s (3.6 km / h)
Anzahl Sender	500
Dichte der Sender	Alle 5m
Größe AWQL-Anfrage	1 KB
Flächeninformation pro Sender	400 Byte
Overhead einer AWML Antwort	100 Byte

Tabelle 5-3 zeigt die Daten, die den Berechnungen zugrunde liegen. Ich gehe von einem 11 Mbit Funk-LAN aus, da der SpaSe direkt mit den Nexus-Anwendungen kommuniziert. Die maximale Bandbreite ist netto 5 MBit / s (625 KB / s) (vgl.[Ahlers & Ziegler 2001]). Wird ein Nexus-Knoten verwendet, so kann man davon ausgehen, dass der SpaSe mit diesem durch ein schnelleres Netz wie z.B. Fast Ethernet (100 Mbit / s) verbunden ist. Die folgenden Annahmen sind für das Gebäude geschätzt und sind wahrscheinlich eine obere Grenze. Eine solche Anzahl Besucher hat die Fakultät wahrscheinlich nur sehr selten (Tag der offenen Tür). Die Anzahl der Sender ist wahrscheinlich zu groß, da bei einem Stückpreis von 50 - 100 Euro für einen Sender die Kosten 25.000 - 50.000 Euro betragen würden.

Tabelle 5-4 zeigt die Anforderungen bei der Verwendung von Hoarding. Die

Tabelle 5-4: Verwendung von Hoarding

Maximale Antwortzeit	10 s
Größe einer Antwort	400 * 500 + 100 Byte = 200100 Byte
Maximale Anzahl Antworten	3,125 Antworten / s
Bearbeitungszeit Worst Case	2 s

Anwender tolerieren eine größere Wartezeit bei Hoarding, als bei den anderen Anfragen, da dies nur einmal notwendig ist. Durch die Größe einer Antwort ermöglicht das Funknetz maximal 3.125 Antworten pro Sekunde. Da aber nur sehr wenige Besucher gleichzeitig das Gebäude betreten und die maximale Antwortzeit relativ lange dauern kann, hat der Indoor SpaSe selbst im Worst Case 2 Sekunden Zeit um eine solche Anfrage zu bearbeiten.

Ohne Hoarding und Caching wird die Anzahl der zu verarbeitenden Anfragen von der Anzahl der Personen und der Positionierungsmethode bestimmt. Hierbei kann eine laufende Positionierung erfolgen oder nur bei Bedarf einer Nexus-Anwendung. Bei einer laufenden Positionierung stellt die Sensorkomponente eine Anfrage, sobald sie einen neuen Sender empfängt. Dies hängt von der Bewegungsgeschwindigkeit des Anwenders und der Dichte der Sender ab. Im Durchschnitt sendet die Sensorkomponente alle 5 Sekunden eine Anfrage und benötigt in weniger als 5 Sekunden eine Antwort, da sonst die gesendete Position veraltet ist. Tabelle 5-5 zeigt, die daraus resultierenden Anforderungen. Mit 20 Anfragen pro Sekunde sind Positionierungsanfragen bei weitem die häufigsten Anfragen. Die Bearbeitungszeit für die Anfragen ist mit durchschnittlich 50 ms sehr kurz.

Tabelle 5-5: Kein Hoarding

Größe einer Anfrage	1 KB
Größe Antwort	500 Byte
maximale Antwortzeit	5s
Durchschnittliche Anzahl Anfragen pro Anwender	Alle 5s eine Anfrage
durchschnittliche Anzahl Anfragen	20 / s
Anzahl Anfragen im Worst Case	100
durchschnittliche Datenmenge	30 KB / s (20 KB Anfragen, 10 KB Antworten)
Datenmenge Worst Case	150 KB / s (100 KB Anfragen, 50 KB Antworten)
Bearbeitungszeit Durchschnitt	50 ms
Bearbeitungszeit Worst Case	10 ms

Die Anforderungen sinken, wenn Positionierungsanfragen nur in Verbindung mit Anfragen von Nexus-Anwendungen gestellt werden. Tabelle 5-6 zeigt die Anforderungen bei einer solchen Positionierung. Die Anfragen und Antworten entsprechen denen bei einer laufenden Positionierung ohne Hoarding. Die durchschnittliche Antwortzeit sollte unter der Antwortzeit bei normalen Anfragen liegen, da sie zu diesen dazuaddiert werden muss. Die durchschnittliche Anzahl Anfragen entspricht denen bei normalen Anfragen.

Tabelle 5-6: Positionierungsanfragen nur bei Bedarf

Größe einer Anfrage	1 KB
Antwortgröße	500 Byte
Durchschnittliche Antwortzeit	<3 s
Maximale Antwortzeit	< 10 s
durchschnittliche Anzahl Anfragen pro s	0.33
durchschnittliche Datenmenge	0.5 KB / s

Positionierungsanfragen und Anfragen durch Nexus-Anwendungen müssen gemeinsam betrachtet werden, um die Anforderungen an den Indoor SpaSe zu bestimmen. In Tabelle 5-7 werden die Anforderungen der Nexus-Anwendungen mit den Anforderungen durch Positionierungsanfragen verknüpft. Da Anfragen beim Hoarding nur sehr selten vorkommen sind hier die Anforderungen in etwa die Anforderungen bei den Anfragen durch Nexus-Anwendungen. Bei einer laufenden Positionierung ohne Hoarding oder Caching werden die Anforderungen durch die Positionierung bestimmt. Die Anforderungen sind sehr viel höher als die Anforderungen, wenn die Positionierung nur bei Bedarf erfolgt.

Tabelle 5-7: Anforderungen durch Positionierung und normale Anfragen

Positionierungsmethode	kein Hoarding	nur bei Bedarf
durchschnittliche Anzahl Anfragen pro Sekunde	20.33	0.66
durchschnittliche Datenmenge	32 KB / s	2.8 KB / s

Die Verwaltung des Systems

Die Befehle für die Konfiguration des Servers müssen nicht sehr schnell verarbeitet werden, da sie nur sehr selten ausgeführt werden. Die Häufigkeit mit der Funktionen zur Verwaltung der Objekte (Insert, Update, Delete) genutzt werden, hängt davon ab, ob der Betreiber den Inhalt des SpaSe festlegt, der SpaSe startet oder sich im normalen Betrieb befindet.

Während der Zeit in der ein Administrator den Inhalt des SpaSe festlegt und Objekte in den SpaSe einfügt, wird in erster Linie der Insert Befehl

benutzt. Werden die Objekte manuell durch Insert Befehle eingetragen, so beträgt die Zeit zwischen zwei Insert Befehlen wahrscheinlich mehrere Minuten. Ein Insert Befehl muss größere Objektmengen verarbeiten, wenn die Daten aus einem anderen Programm konvertiert werden. Da sich der SpaSe nicht im Betrieb befindet kann die Verarbeitung eines solchen Insert Befehls auch mehrere Minuten dauern. Update und Delete Befehle sind in dieser Zeit eher selten.

Beim Start eines SpaSe werden Update und Delete Befehle wohl nicht ausgeführt. Hier werden nur Daten eingelesen. Die Daten können vom SpaSe aus einer Datei eingelesen werden oder durch Insert Befehle erhalten werden.

Insert und Delete Befehle sind im normalen Betrieb eher selten, da der SpaSe nur statische Objekte speichert, die normalerweise nicht so schnell entstehen und wieder verschwinden. Bei Updates hängt es davon ab, wie oft die nicht zur Positionierung dienenden Daten geändert werden. Die räumlichen Daten eines Objektes werden im SpaSe nur sehr selten geändert (statische Objekte).

Die Kommunikation mit dem Event Service

Die Kommunikation zwischen Event Service und SpaSe ist selten, da nur Änderungen auf den Daten Events auslösen. Das Registrieren eines Basic Events benötigt nur wenige Daten und eine Event Notification enthält wenige Daten (<100 Byte). Solange nur wenige Events gleichzeitig ausgelöst werden, sollte die Kommunikation mit dem Event Service vernachlässigbar sein. Es ist allerdings auch denkbar, dass sehr viele Event Notifications verschickt werden müssen. In solchen Fällen kann es zu Geschwindigkeitsproblemen beim SpaSe oder dem Event Service kommen.

5.7.2 Zuverlässigkeit/Robustheit

Der Indoor SpaSe wird von der Sensorkomponente zur Positionierung verwendet. Da es sich hierbei um eine Basisfunktionalität von ortsbasierten Anwendungen handelt, muss der Indoor SpaSe Anfragen in der geforderten Zeit zuverlässig verarbeiten können.

Nexus-Anwendungen haben in der Regel keine Benutzungsschnittstelle bei denen AWQL direkt eingegeben wird, da die meisten Anwender kein AWQL kennen und Texte bei vielen Systemen nur umständlich eingegeben werden können. Nexus-Anwendungen erstellen mit Hilfe der Benutzungseingaben AWQL Anfragen und überprüfen die Eingabe. Sie erstellen daher Anfragen, die nur selten Fehler enthalten. Der Indoor SpaSe wird allerdings bei der Entwicklung von Nexus-Anwendungen verwendet werden. Hierbei erstellen die Entwickler die Anfragen selbst. Die Anfragen können dabei sehr leicht Tippfehler enthalten. Da der Indoor SpaSe direkt mit den mobilen Geräten kommuniziert, werden solche Anfragen nicht von Nexus-Knoten zurückgewiesen. Der Indoor SpaSe muss daher sehr robust sein und fehlerhafte Anfragen zurückweisen.

5.7.3 Sicherheit

Authentifikation und Autorisationsmechanismen sind für SpaSes wichtig. Herkömmliche SpaSes dürften keine Verschlüsselung benötigen, da der Zugriff auf die Daten in der Regel öffentlich ist. Der Prototyp des Indoor SpaSe unterstützt keine Sicherheitsmechanismen.

5.8 Zusammenfassung

Der Indoor Spatial Model Server speichert Daten für die Positionierung und Informationen für Nexus-Anwendungen ab. Der Prototyp beschränkt sich dabei auf Informationen zu Sendern, Räumen und Gebäuden. Als Kommunikationspartner dienen die Nexus-Anwendungen und die Sensorkomponenten, die direkt oder über einen Nexus-Knoten an den SpaSe senden. Der Indoor SpaSe wird seine Augmented Areas beim ASR eintragen und dient als Event Source für den Event Service. SpaSes bieten auch eine Schnittstelle für Administratoren an.

Die Positionierungsanfragen dienen als Grundlage für die räumlichen Anfragen der Nexus-Anwendungen. Ihre Leistungsanforderungen hängen von der verwendeten Positionierungsmethode ab. Wird Hoarding benutzt, so reduzieren sich die Anforderungen an den SpaSe auf die Anforderungen für die Anfragen an die Nexus-Anwendungen. Sendet die Sensorkomponente jedesmal, wenn sie einen neuen Sender empfängt eine Anfrage, so bestimmt sie die Anforderungen an den SpaSe. Die Anforderungen an den SpaSe sind dabei sehr hoch. Eine Reduzierung des Aufwands kann erreicht werden, wenn die Positionierungsanfrage nur im Zusammenhang mit räumlichen Anfragen durchgeführt wird.

Das folgende Kapitel beschreibt die Entwicklung und den Aufbau des Prototypen. Das darauffolgende Kapitel wird zeigen, in wieweit der Prototyp die Anforderungen erfüllen kann.

6 Der Prototyp

Dieses Kapitel beschreibt den Entwicklungsansatz, beschreibt den Ablauf der Entwicklung und stellt den Prototypen vor. Hierbei wird die Architektur des Prototypen, die verwendeten Komponenten und die Realisierung der Funktionen vorgestellt.

6.1 Ansätze

Spatial Model Server lassen sich mit Geographischen Informationssystemen (GIS) oder mit Datenbanksystemen erstellen, die um eine Verwaltung von räumlichen Daten erweitert wurden. AWQL und AWML werden von diesen Systemen nicht direkt unterstützt und müssen daher mit Hilfe von Wrappern in das Format der GIS oder DBS umgewandelt werden. DBS Systeme haben gegenüber GIS Systemen den Vorteil, dass sie für die Verwaltung sehr großer Datenmengen ausgelegt sind, die Umwandlung von AWQL in SQL leichter ist als eine Umwandlung in die proprietären Sprachen herkömmlicher GIS und manche GIS nur unzureichende Kommunikationsmöglichkeiten über ein Netzwerk besitzen [Nicklas et al 2001]. Ein Spatial Model Server auf Basis eines GIS wurde mit ArcView implementiert [Volz, Fritsch & Klinec 1999]. Ein Spatial Model Server auf Basis einer erweiterten DBS wurde im Rahmen eines Studienprojekts realisiert.

Ein dritter Ansatz ist die Erstellung eines Spatial Model Servers, der die Verwaltung der Daten selbst übernimmt. Dieser Ansatz wird in dieser Diplomarbeit verfolgt. Das Datenmodell ist auf nur wenige Klassen begrenzt und die Datenmenge kann im Hauptspeicher gehalten werden, da nur Daten zu Objekten innerhalb eines Gebäudes gespeichert werden. Dieser Ansatz hat gegenüber den anderen Ansätzen einige Vorteile. GIS und Datenbanksysteme besitzen Komponenten, die für SpaSes überflüssig oder überdimensioniert sind. So müssen die Komponenten für die Anfrageverarbeitung sehr viel komplexere Sprachen als AWQL verstehen oder besitzen Funktionen zur graphischen Aufbereitung, die SpaSes nicht benötigen. Ein Wegfallen oder Optimieren dieser Komponenten führt zu Geschwindigkeitssteigerungen und einem geringeren Speicherverbrauch. Die Umwandlung von AWQL in eine andere Sprache kann daher entfallen und ermöglicht weitere Leistungssteigerungen. Datenbanksysteme und GIS sind auf die Speicherung auf Sekundärspeicher optimiert. Wie bei Hauptspeicherdatenbanken führt dies dazu, dass auf Hauptspeicher optimierte Systeme Vorteile gegenüber ihnen haben. Die Verwaltung von räumlichen Daten lässt sich verändern, wodurch auch Konzepte zur Datenhaltung und Verarbeitung der Anfragen untersucht werden können. Ein weiterer Vorteil besteht darin, dass das Produkt auch in Umgebungen funktioniert, wenn keine räumlichen Datenbanksysteme oder GIS vorhanden sind und die Anschaffung solcher Systeme zu teuer wäre. Ein solcher Bereich sind private Haushalte, die in der Regel weder Datenbanksysteme noch GIS besitzen. Nachteile dieses Ansatzes sind vor allem der sehr viel höhere Entwicklungsaufwand und die höhere Fehleranfälligkeit des Produktes.

6.2 Der Entwicklungsprozess

Der Prototyp wurde iterativ entwickelt. Die erste Iteration sollte nur eine einfaches, lauffähiges und erweiterbares Grundgerüst des SpaSe liefern. Die zweite Iteration erweiterte die Funktionalität des Grundgerüsts aus der ersten Iteration.

6.3 Die erste Iteration

In der ersten Iteration wurde das Grundgerüst des Indoor SpaSe entwickelt. Der Indoor SpaSe sollte noch nicht als Serverprogramm funktionieren. Der Indoor SpaSe der ersten Iteration liest die zu verwaltenden Daten aus einer einfachen AWML-Datei. Die AWQL-Anfrage wird aus einer Datei eingelesen und das Ergebnis dieser Anfrage wird auf dem Bildschirm ausgegeben. Der Indoor SpaSe beendete sich danach. Er beherrscht keine räumlichen Anfragen. Es wird nur die Restriktion ausgewertet ohne „inside“ und „overlap“. Die Klassenhierarchie entspricht der Klassenhierarchie in der Spezifikation. Die Klassen enthalten alle dort aufgeführten Attribute. Allerdings wurde nicht jedes dieser Attribute geparst oder verarbeitet, da die Klassen für diese noch nicht implementiert wurden. Beispielsweise wurden Attribute von der Klasse Polygon (Extent, FieldOfVision) oder Punkt (Position) nur als Zeichenketten implementiert und wurden nicht geparst.

Da der Indoor SpaSe als Basiskomponente für andere Klassen verwendbar sein sollte bietet er Schnittstellen nach außen an. Die Schnittstellen zum Indoor SpaSe erlauben es die Dateinamen der AWQL und AWML Dateien zu bestimmen. Eine weitere Methode startete die Anfrage.

6.4 Die zweite Iteration

In der zweiten Iteration sollte die Funktionalität des Grundgerüsts aus der ersten Iteration erweitert werden und aus dem einfachen Programm einen Server entstehen. Zum Testen des Indoor SpaSe wurde ein Programm zur Erstellung von AWML-Dateien entwickelt und hiermit die Daten für das Fakultätsgebäude erstellt. Ein anderes Programm stellt Anfragen an den Indoor SpaSe. Der Indoor SpaSe bietet nun seine Dienste mit Hilfe eines Applikationsservers an. AWQL wird nun vollständig unterstützt. Es sind nun auch räumliche Anfragen und Änderungsoperationen möglich. Zusätzlich können einfache Events auf den Änderungsoperationen definiert werden. Fast alle Attribute können nun eingelesen werden. Als Koordinatensystem wird ein einfaches rechtwinkliges Koordinatesystem verwendet. Die Einheit ist Meter. Der Null-Punkt ist die linke obere Gebäudeecke des Gebäudes auf den Karten im Internet.

6.5 Verwendete Komponenten

Das Programm benötigt einen Applikationsserver mit dem über das SOAP Protokoll auf den Indoor SpaSe zugegriffen werden kann. Zur Verarbeitung von AWQL und AWML wird ein XML-Parser verwendet.

6.5.1 Der Applikationsserver

Tomcat 4.0 wird als Applikationsserver verwendet. Er dient dazu Servlets und Java Server Pages anzubieten. Eine Erweiterung ermöglicht es ihm das SOAP Protokoll zu verstehen und den Zugriff auf bei ihm registrierte Objekte bereitzustellen. Die Objekte können hierbei einfache Javaklassen sein.

6.5.2 SOAP

Das Simple Object Access Protocoll (SOAP) ist ein Übertragungsprotokoll, das als einfaches RPC-Protokoll dient. Es setzt auf anderen Transportprotokollen auf. Es sollte ursprünglich nur in Verbindung mit HTTP verwendet werden, ist aber mittlerweile unabhängig von einem bestimmten Transportprotokoll. Die SOAP Implementierungen ermöglichen es auch andere Protokolle wie SMTP oder FTP zu verwenden. Die Möglichkeit unterschiedlicher Transportprotokolle zu verwenden, ermöglicht es flexibel auf Zugriffsbeschränkungen reagieren zu können. Beispielsweise beschränken Firewalls den Zugriff auf nur wenige Ports und erlauben dadurch nur wenige Protokolle, wie HTTP oder SMTP. Technologien, die eigene Transportprotokolle verwenden, können deshalb nicht verwendet werden. SOAP kodiert die Methodenparameter als Text im XML-Format. Da die Daten in Textform dargestellt sind, ist eine sehr hohe Portabilität gewährleistet und die Verwendung von XML ermöglicht eine Typüberprüfung mit Hilfe von XML-Parsern. Da SOAP einfach sein sollte, existiert keine direkte Unterstützung von Sicherheitskonzepten oder Transaktionsmechanismen. Es existiert auch keine Verwaltung der Objekte, weshalb z.B. kein verteiltes Garbage Collection oder Object Discovery existiert. Die Server müssen daher selbst bestimmen, wann ein Objekt gelöscht werden kann und die Clients müssen die Adressen der Objekte selbst herausfinden. Die Verwendung von XML führt zu einem größere Aufwand für die Aufbereitung und den Transport der Daten. Außerdem ist ein Einsatz von XML-Parsern bei den Endpunkten nötig. Es können so Geschwindigkeitsprobleme durch die XML-Verarbeitung oder durch eine längere Kommunikation entstehen.

Der InSpaSe muss beim SoapService registriert werden. Der Name ist „urn:InSpaSe” kann aber prinzipiell beliebig gewählt werden, solange die Clients den richtigen Namen kennen. Die in SOAP zu registrierende Klasse ist inspase.SoapInSpaSeWrapper und bietet folgende Methoden an:

Für Anfragen:

- String delete(String awql) : Löscht die Elemente die das Ergebnis der AWQL Anweisung sind und liefert einen Change Report zurück.
- String insert(String awm): Fügt Objekte in den SpaSe ein. Objekte ohne NOL werden eine neue NOL bekommen. Liefert einen Change Report zurück
- String query (String awql): Führt eine normale Suchanfrage aus und liefert die Elemente als awm zurück
- String update (String awql): Führt eine AWQL-update query durch und liefert einen Change Report.

Für die Verwaltung des SpaSe

- loadAWM(String filename): lädt eine AWML-Datei
- saveAWM(String filename): speichert die verwalteten Objekte in eine AWML-Datei

Für Events:

- String registerOnDelete(String nol): registriert ein Delete Event auf dem Objekt mit der angegebenen NOL. Man erhält eine PredicateID zurück
- String registerOnDeleteType(String type)
- String registerOnInsert(String type)
- String registerOnUpdate(String nol)
- String registerOnUpdateAttribute(String nol, String attribute)

6.5.3 XML-Parser

XML-Parser ermöglichen eine einfache Verarbeitung von XML-Dokumenten und können die Korrektheit des XML-Dokuments überprüfen. Je nach Parser lassen sich die Dokumente nach ihrer XML-Konformität überprüfen und ob die enthaltenen XML-Daten einer bestimmten Form entsprechen. Diese Form lässt sich mit Hilfe von XML-DTDs oder XML-Schemas definieren.

Neben proprietären Datenstrukturen und Schnittstellen existieren standardisierte Schnittstellen für die XML-Verarbeitung. Die wichtigsten sind DOM (Document Object Model) und SAX (Simple Api for XML). Es existieren weitere Schnittstellen, die aber entweder sehr neu sind oder auf bestimmte Programmiersprachen beschränkt sind. Die beiden Schnittstellen unterscheiden sich in ihren Zielsetzungen und deshalb auch in ihrer Form und Art der Nutzung.

DOM erzeugt aus einem XML Dokument einen Baum mit dem Elementen dieses XML-Dokuments. Dieser Baum wird im folgenden als DOM Dokument bezeichnet und besitzt Methoden für den Zugriff auf die einzelnen Elemente. SAX hingegen erzeugt kein Dokument. Der Parser liest ein XML-Dokument sequentiell durch und erzeugt dabei „Parse-Events”, die durch einen vom Anwender übergebenen Handler verarbeitet werden.

Da der SAX Parser kein Dokument erzeugen muss, kann ein XML-Dokument sehr viel schneller als bei DOM geparst werden und auch der notwendige Speicher ist geringer. Ein weiterer Vorteil besteht darin, dass eine Verarbeitung eines XML-Dokuments sehr viel früher begonnen werden kann. Bei SAX muss man nur auf das erste Event warten und beginnt dann mit der Verarbeitung des XML-Dokuments. Bei DOM muss das XML-Dokument zu Ende geparst und das DOM Dokument erzeugt werden, bevor auf das DOM Dokument verarbeitet werden kann. Mit SAX kann man den Inhalt des XML-Dokuments bereits verarbeiten, während man bei DOM noch auf das DOM Dokument wartet. SAX ermöglicht hierdurch

eine Parallelisierung von Parsen und Verarbeiten eines XML-Dokuments. Bei DOM sind Parsen und Verarbeiten zwei sequentielle Tätigkeiten.

SAX hat aber auch Nachteile: Die Travesierung des XML-Dokuments ist vorgegeben. Man kann sich deshalb nicht auf dem Dokument beliebig bewegen, wie z.B. bei einem DOM Dokument. Man muss dadurch auch uninteressante Elemente verarbeiten und kann keine Informationen über vorherige Elemente erhalten. Der Anwender muss sich in solchen Fällen selbst, um die Speicherung kümmern. Wird hierfür auch Information über die Struktur des Dokuments benötigt, so entwickelt man eine Datenstruktur, die einem DOM Dokument ähnelt. Dies gilt besonders, wenn man als Endergebnis eine Datenstruktur benötigt, die den Inhalt und die Struktur eines XML-Dokuments wiederspiegeln soll. Da DOM Parser darauf spezialisiert sind, DOM Dokumente zu erzeugen, kann man annehmen, dass mit ihnen DOM Dokumente schneller und mit niedrigerem Speicherverbrauch erzeugt werden als bei einer vergleichbaren selbstgeschriebenen Struktur. DOM Dokumente lassen sich erzeugen, verändern und, da DOM Dokumente standardisiert sind, gibt es Klassen mit denen man sie auf dem Bildschirm oder in eine Datei ausgeben kann. Bei SAX muss man die Methoden selbst erstellen, um die selbstentwickelte Datenstruktur erstellen, verändern und ausgeben zu können.

Der verwendete XML-Parser ist der Xerces-J XML Parser. Es werden sowohl SAX als auch DOM als Schnittstelle zum XML-Parser verwendet. Hierdurch kann man Erfahrungen mit beiden Schnittstellen sammeln und vergleichen.

6.6 Aufbau des Systems

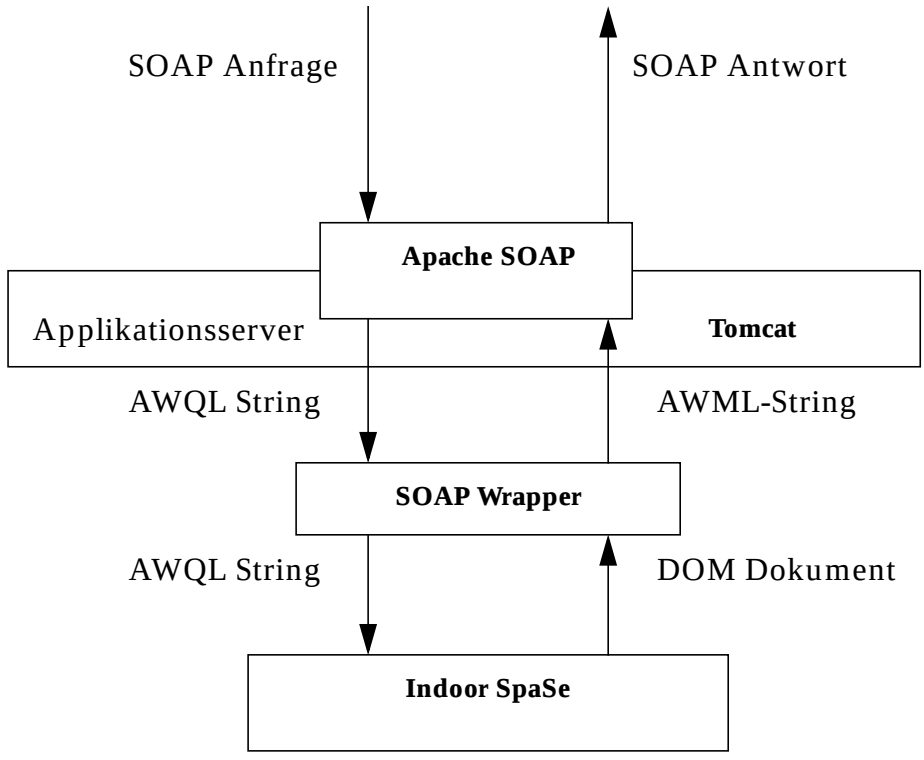


Abbildung 6-1: Aufbau des Systems

Abbildung 6-1 zeigt den Aufbau des Systems. Der Applikationsserver Tomcat bietet eine SOAP Schnittstelle für die von ihm verwalteten Objekte an. Der Indoor SpaSe ist ein von ihm verwaltetes Objekt. Allerdings greift Tomcat nicht direkt auf die Schnittstelle des Indoor SpaSe zu, sondern benutzt hierfür einen Wrapper. Der SOAP Wrapper bietet nach außen hin die Methoden an, die an der SOAP Schnittstelle angeboten werden sollen. Wird eine solche Methode aufgerufen, so ruft er die dazugehörige Methode des Indoor SpaSe auf. Das Ergebnis eines Methodenaufrufs muss vom SOAP Wrapper eventuell umgewandelt werden und wird dann an den Applikationsserver zurückgegeben. Apache SOAP erzeugt daraus eine SOAP Antwort.

6.7 Designentscheidungen

Der Spatial Model Server ist in Java geschrieben. Hierdurch ist eine hohe Portabilität möglich. Es sollte allerdings klar sein, dass eine solche Interpretersprache Probleme mit sich bringt. Die Geschwindigkeit ist geringer als bei einer Compiler Sprache wie C oder C++. Ein Vergleich mit Spatial Model Servern in solchen Sprachen ist daher nur sehr eingeschränkt möglich. So dürfte der Spatial Model Server des Studienprojekts schneller sein als der hier vorgestellte, da sie auf eine Datenbank zugreifen. Ein weiteres Problem ist das Java zur Speicherverwaltung einen Garbage Collector verwendet, wodurch eine Aussage über den tatsächlichen Speicherverbrauch der Anwendung erschwert wird.

Der Aufbau und die verwendeten Algorithmen enthalten kaum Optimierungen, da mehr Wert auf Vollständigkeit als wertgelegt wurde. Es sollen so die Problembereiche bei der Datenverwaltung leichter identifiziert werden können und ermöglicht so auch Aussagen zu möglichen Problemen bei anderen Systemen, die ähnliche Funktionen benutzen. Bei bestimmten Algorithmen wurde eine leichte Erweiterbarkeit des Programms einer effizienten Ausführung vorgezogen. Betroffen ist hiervon beispielsweise die Verarbeitung von AWML-Dateien oder die Suche nach Objekten mit bestimmten Attributwerren. Es werden keine Konzepte zur Parallelisierbarkeit der Anfragen betrachtet, da das Entwicklungssystem nur einen Prozessor besaß und davon ausgegangen wurde, dass wie im Kapitel 4 „Hauptspeicherdatenbanken” die CPU-Auslastung bei 100% liegen wird. Eine Parallelisierung hätte dadurch den SpaSe nur verlangsamt. Konzepte für die Persistenz wurden nicht implementiert. Allerdings kann man den SpaSe dazu veranlassen, die verwalteten Objekte in eine Datei abzuspeichern oder die Objekte aus einer Datei zu lesen. Mit einer der Gründe für die höhere Geschwindigkeit einer Hauptspeicherdatenbank gegenüber herkömmlichen Datenbanken ist die Verwendung von Referenzen auf Daten statt von Kopien der Daten. Dies wird beim Indoor SpaSe ebenfalls durchgeführt. Neben einer hohen Geschwindigkeit erhält man so auch einen geringen Speicherverbrauch. Der Nachteil hiervon sind Seiteneffekte, die nicht immer gewollt sein müssen.

6.8 Die Architektur des Indoor SpaSe

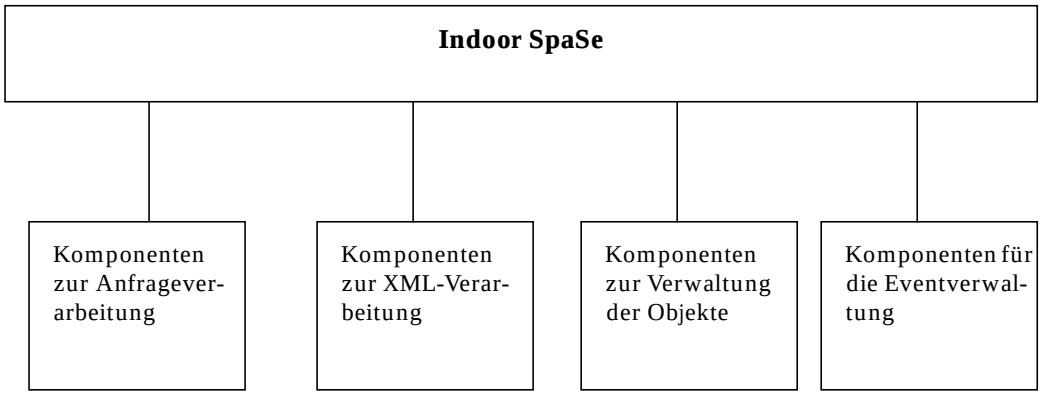


Abbildung 6-2: Grobarchitektur des Indoor SpaSe

Der grobe Aufbau des Indoor SpaSe wird in Abbildung 6-2 dargestellt. Seine Komponenten lassen sich grob in 4 Kategorien einteilen. Er besitzt Komponenten für die Anfrageverarbeitung, für die XML-Verarbeitung, der Verwaltung der Objekte und der Event Verwaltung. Abbildung 6-3 zeigt welche Komponenten zu welchen Kategorien gehören.

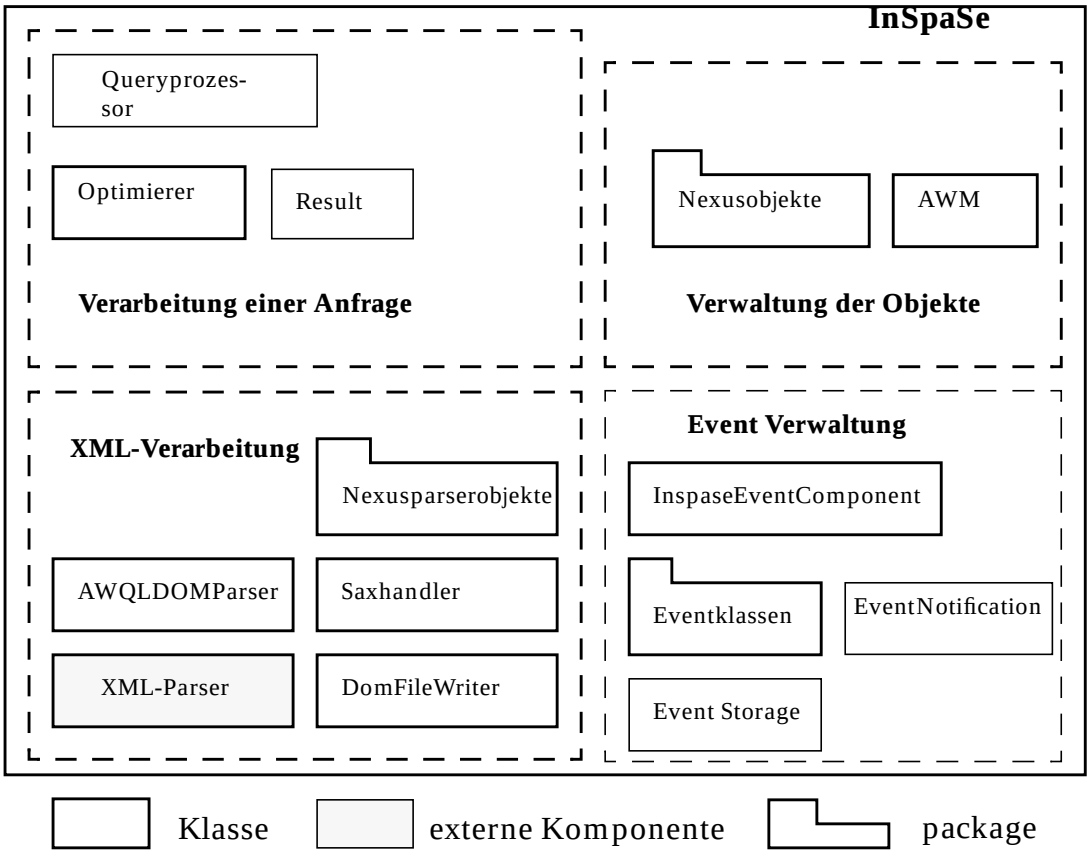


Abbildung 6-3: Komponenten des Indoor SpaSe

6.9 Die Komponenten des Programms

6.9.1 InSpaSe

InSpaSe enthält alle Komponenten des Indoor SpaSe. Es besitzt darüber hinaus Methoden, mit denen andere Programme den SpaSe nutzen können und wird beispielsweise vom SOAP-Wrapper für SOAP verwendet.

6.9.2 Verwaltung der Objekte

AWM

AWM (Augmented World Model) enthält alle Objekte der gespeicherten Augmented World. Die Objekte werden in einer ArrayList abgespeichert. Eine ArrayList ist eine Standard Java Datenstruktur, bei der ähnlich wie bei einem Array direkt auf ein Element zugegriffen werden kann, aber im Gegensatz zu Arrays zur Laufzeit wachsen kann. Für den Zugriff werden keine Indexe benutzt.

Nexusobjekte

Nexusobjekte sind die Objekte in der Form, wie sie in der AWM gespeichert werden. Sie besitzen Methoden mit denen man sie auf bestimmte Bedingungen überprüfen kann z.B. ob sie innerhalb eines Polygons sind oder ob ein bestimmtes Attribut einen bestimmten Wert hat.

6.9.3 Anfrageverarbeitung

Der Queryprozessor

Der Queryprozessor verarbeitet die Anfragen. Er erhält die Anfragen als DOM Dokument. Das Dokument ist baumförmig aufgebaut und seine drei Teile werden getrennt verarbeitet. Es besteht aus einer Restriktion mit den Bedingungen für die Anfrage („Restriction”), einem Closest-Prädikat mit dem das Ergebnis der Anfrage auf die n räumlich nächsten beschränkt werden kann und dem Teil, der den eigentlichen Befehl festlegt.

Der Optimierer

Der Optimierer soll die Anfrage in eine Form umwandeln, die besonders effizient ist. Der Optimierer soll nur sehr wenig machen. Hierdurch sollen Geschwindigkeitsprobleme offen zu Tage treten und Benchmarks leichter vorhersagbar sein. Ein weiterer Grund für einen einfachen Optimierer ist die Verschiebung des Aufwandsverteilung unter den Komponenten. Es besteht wie in Kapitel 4 „Hauptspeicherdatenbanken” beschrieben, die Gefahr, das eine Optimierung die Ausführungszeit verlängert statt verkürzt. Der Optimierer wandelt die Anfrage in eine äquivalente Anfrage um, in der es kein NOT gibt, sondern nur AND, OR, EQUAL, IN, INSIDE, OVERLAP, NOTEQUAL, NOTIN, NOTINSIDE und NOTOVERLAP.

Das NOT wird entfernt, da seine Realisierung sehr rechenintensiv ist. Das Ergebnis einer NOT Operation sind alle Elemente, die nicht in der bisherigen Ergebnismenge enthalten sind. Bei einer großen Augmented World führt das zu sehr vielen Kopieroperationen und Vergleichen. Die Inversen Funktionen zu EQUAL, IN, INSIDE und OVERLAP sind sehr viel effizien-

ter zu realisieren, da sie auf den nicht-inversen Funktionen aufbauen können. Es werden folgende Regeln benutzt:

- De Morgan:

  $\neg(A \cap B) = \neg A \cup \neg B \qquad \neg(A \cup B) = \neg A \cap \neg B$

- Doppelnegationsregel

  $\neg\neg A = A$

- Erstellen der inversen Funktion

not (A rel B) = (A notrel B),

rel = equal, in, inside oder overlap

Result

Result enthält das Ergebnis einer Anfrage. Es lassen sich Objekte einfügen, eine Schnitt oder eine Vereinigung mit einem anderen Result-Objekt durchführen. Es besteht nur aus einer ArrayList und enthält keine doppelten Objekte. Es enthält Referenzen auf Objekte und nicht Kopien der Objekte.

6.9.4 XML-Verarbeitung

Der XML-Parser

Der verwendete XML-Parser ist der Xerces-J XML Parser. Es werden sowohl SAX als auch DOM als Schnittstelle zum XML-Parser verwendet. Hierdurch kann man Erfahrungen mit beiden Schnittstellen sammeln und vergleichen. AWQL Anfragen werden mit Hilfe von DOM und AWML Dateien mit SAX geparst.

AWQLDOMParser

AWQLDOMParser parst AWQL-Anfragen mit Hilfe der DOM Schnittstelle und erzeugt ein DOM Dokument.

Saxhandler

Der Saxhandler wird bei der Verwendung der SAX-Schnittstelle benötigt. Er verarbeitet die Parse-Events, die beim Parsen einer AWML-Datei erzeugt werden

DOMFileWriter

DOMFileWriter erzeugt aus einem DOM Dokument eine XML Datei. Es kann für die Ausgabe verwendet werden. Es wird benutzt um AWML-Dateien zu erstellen oder wie bei der Rückgabe eines Ergebnisses AWML in einer Zeichenkette zu speichern.

Nexusparserobjekte

Die Nexusparserobjekte werden benutzt, wenn Nexusobjekte aus XML erzeugt werden sollen oder Nexusobjekte in ihre XML-Darstellung konvertiert

werden sollen. Sie haben die gleiche Klassenhierarchie wie die Nexusobjekte.

6.9.5 Event Verwaltung

InspaseEventComponent

InSpaSeEventComponent kapselt die Komponenten der Event Verwaltung und bietet eine Schnittstelle für das Erzeugen, Löschen, Auslösen von Events an.

Event Storage

Event Storage speichert die Events. Hierbei wird jede Event Art getrennt gespeichert.

Eventtypen

Dieses Paket enthält die unterschiedlichen Eventarten. Es existieren Events die beim Einfügen, Löschen und Verändern eines Objekts ausgelöst werden.

EventNotification

Die Benachrichtung wird mit Hilfe der Klasse EventNotification durchgeführt. Da es den Event Service noch nicht gibt, wird einfach Information über den ausgelösten Event auf dem Bildschirm ausgegeben.

6.10 Einlesen der AWML

Die AWML-Dateien werden mit der SAX-Schnittstelle eingelesen. AWML-Dateien können sequentiell eingelesen werden. Man benötigt die Struktur der XML-Datei nicht, wodurch die Objekte relativ einfach von XML in eine interne Darstellung konvertiert werden können. Da das DOM Dokument erstellt und selbst noch einmal verarbeitet werden muss, um die Objekte in die interne Darstellung zu überführen, wird bei großen Modellen sehr viel Rechenzeit verbraucht. Da AWML-Dateien prinzipiell beliebig groß sein dürfen, kann das DOM Dokument sehr viel Speicher verbrauchen. Reicht der Hauptspeicher nicht aus, so muss dann virtueller Speicher benutzt werden. Hierdurch wird die Geschwindigkeit sehr stark reduziert.

Dem Parser wird hierfür Saxhandler als Handler übergeben. Er verarbeitet die Parse-Events für Startelemente, Endelemente und für Zeichenketten. Sobald ein Objekt geparst wird erzeugt er ein dazu passendes Nexusparserobjekt und leitet die Parse-Events an das Nexusparseobjekt weiter. Dies verarbeitet die Events. Kennt es ein Element nicht, so ruft es die Methode für diesen Event bei seiner Oberklasse auf. Dieses kennt das Element oder ruft die Methode seiner Oberklasse auf. Wenn keine Oberklasse das Element kennt, so wird es einfach ignoriert. Das Nexusparserobjekt versucht nur einfache Elemente zu verarbeiten. Komplexe Elemente wie Polygone werden verarbeitet, indem es ein Nexusparserobjekt für Polygone erzeugt und dann die Events an dieses Objekt weiterleitet, bis das dazugehörige Endelement gefunden wird. Wird das Endelement des Objekts gefunden, so ruft der Saxhandler die Methode des Nexusparserobjekts auf, um ein Nexusobjekt zu erzeugen. Er ruft dann eine Methode des Nexusobjekts

auf, damit es sich selbst in die AWM einträgt. Der Grund weshalb nicht der Saxhandler das Objekt in die AWM einträgt liegt darin, dass ein Nexusobjekt seine dynamische Klasse kennt und so auch Methoden aufrufen könnte, die speziell zum Eintragen solcher Objekte benutzt werden.

```
<awml> <-- Erzeugt Startelement, Verarbeitung durch Saxhandler-->

 <--Beim folgenden Startelement wird ein Nexusparserobjekt von der
    Klasse PRoom erzeugt-->
  <nexusobject type="room" nol="nexus://..." kind="virtual">
    <number> <--Startelement, ab hier verarbeitet PRoom-->
      2.008  <--Character Event, Es wird dir Raumnummer eingetragen-->
    </number><--Endelement-->

    <extent> <--Startelement Extent, PRoom kennt kein extent. Es ruft
       die Methode für Startelemente von PBuildingElement auf, das
       kennt es aber auch nicht und ruft Oberklasse PStaticObject auf.
       Das ruft Klasse PSpatialObject auf, das ein Nexusparserobjekt
       PExtent erzeugt. -->
      <gml:polygon> <-- PExtent erzeugt PPolygon -->
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coordinates>
              40.0,40.0 40.0,35.0 35.0,35.0 35.0,40.0 40.0,40.0
            </gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:polygon> <--PPolygon erzeugt ein Polygon -->

    </extent> <--PExtent erzeugt ein Objekt Extent  -->

  </nexusobject> <-- PRoom erzeugt ein Objekt Room  -->
                 <-- Room trägt sich in die AWM ein -->
</awml> <-- Endelement wird von Saxhandler verarbeitet -->
```

Abbildung 6-4: Beispiel für das Parsen einer AWML

6.11 Verarbeitung einer AWQL Anfrage

Es gibt zwei sinnvolle Wege eine Anfrage auszuführen.

1. Der Queryprozessor wertet erst das Closest-Prädikat und dann das Restriction-Prädikat aus: Die Objekte werden der Entfernung nach eingelesen und jedes Objekt wird auf die Bedingungen in der Restriktion getestet. Hat man n Objekte, welche die Bedingungen erfüllen, kann man abbrechen.
2. Zuerst wird das Restriction-Prädikat ausgewertet und dann das Closest-Prädikat. Man sucht zuerst alle Objekte, welche die Bedingungen erfüllen. Diese werden dann der Entfernung nach sortiert und die n nächsten Objekte ausgegeben.

Welcher Weg sinnvoller ist, hängt von mehreren Faktoren ab:

- Die Selektivität der Restriktion und Closest-Prädikate. Je höher die Selektivität des einen gegenüber dem anderen Prädikats ist, desto eher sollte man dieses zuerst auswerten.
- Der Aufwand für die Berechnung, ob ein Objekt die Restriktion erfüllt im Vergleich zu einer räumlichen Sortierung. Dies wird durch die Anzahl der Objekte, die vorhandenen Indexe und den notwendigen räumlichen Funktionen beeinflusst.

Beim Indoor SpaSe die Restriktion ausgewertet, da man bei der Auswertung des Closest-Prädikats nicht auf Indexe zurückgreifen kann und so eine Sortierung nach Entfernung durchführen müsste, was relativ aufwendig sein kann. Eine Sortierung mit Quicksort hat einen Aufwand von ca. O(nlogn) und die Entfernung muss zur Laufzeit berechnet werden.

Abbildung 6-5 zeigt den Ablauf bei der Verarbeitung einer AWQL Anfrage. Der Soap Wrapper erhält eine Anfrage (1) und ruft die dazugehörige Methode beim Indoor SpaSe auf (2). Der parst sie mit Hilfe des AWQL DOM Parsers (3) und erhält ein DOM Dokument (4). Er ruft danach den Queryprozessor auf (5), der das übergebene DOM Dokument dem Optimierer gibt (6). Der liefert ein optimiertes DOM Dokument (7). Der Queryprozessor interpretiert das DOM Dokument und stellt Anfragen an die AWM (8). Er erhält die Ergebnisse dieser Anfragen (9). Falls die Anfrage ein Closest Element hat, wird ein Closest Objekt erzeugt und das Gesamtergebnis übergeben (10). Dieses erzeugt ein nach Entfernung sortiertes und auf eine bestimmte Elementzahl reduziertes Ergebnis. Existiert ein Filter, so wird ein Filterobjekt erzeugt (12). Das Objekt und das Gesamtergebnis werden vom Indoor SpaSe übernommen. Er erzeugt nun ein DOM Dokument, das durch Filter auf die notwendigen Attribute reduziert wird (14, 15). Dieses DOM Dokument wird zurückgeliefert und mit Hilfe eines DOMFileWriter Objekts vom Soap Wrapper in eine Zeichenkette umgewandelt (17, 18) und als Ergebnis zurückgegeben(19). In den folgenden Abschnitten wird der Ablauf genauer betrachtet.

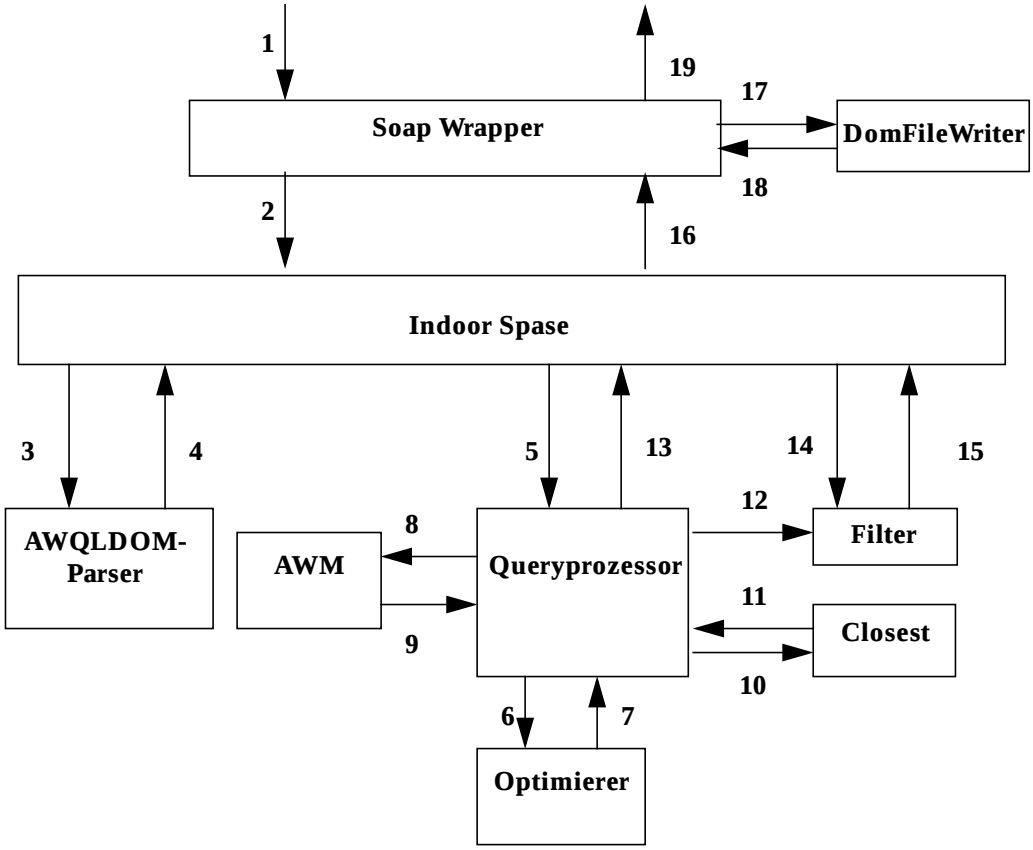


Abbildung 6-5: Der Ablauf einer Anfrage

6.11.1 Parsen der Anfrage

DOM wird für das Parsen von AWQL-Dateien benutzt. Die Anfragen sind recht klein (wenige KB), weshalb der Speicherverbrauch und die Zeit für die Erstellung eines DOM Dokuments gering sind. Das DOM Dokument enthält die Anfrage und wird vom Optimierer und Queryprozessor verarbeitet. Der Optimierer verändert das DOM Dokument, während er die Anfrage optimiert. Wenn man SAX benutzen würde, müsste man eine Datenstruktur ähnlich dem DOM Dokument entwickeln. Abbildung 6-6 zeigt eine Beispielanfrage. Abbildung 6-7 zeigt das daraus resultierende DOM Dokument. Der Übersichtlichkeit wegen werden keine für die Anfrageverarbeitung überflüssigen Elemente angezeigt, wie beispielsweise Knoten für die Leerzeichen zwischen den einzelnen Elementen.

```
<awql>
  <restriction>
    <not>
      <and>
        <equal>
          <attr name="type"/>
          <nexusdata>room</nexusdata>
        </equal>
        <equal>
          <attr name="emittedID"/>
```

```
          <nexusdata>1</nexusdata>
        </equal>
      </and>
    </not>
  </restriction>
  <closest num="5" acc="10">
    <nexusdata>..my position..</nexusdata>
  </closest>
  <filter>
    <includes>
      <attr name="number"/>
      <attr name=”fieldofvision”/>
    </includes>
    <excludeallother/>
  </filter>
</awql>
```

Abbildung 6-6: AWQL Anfrage

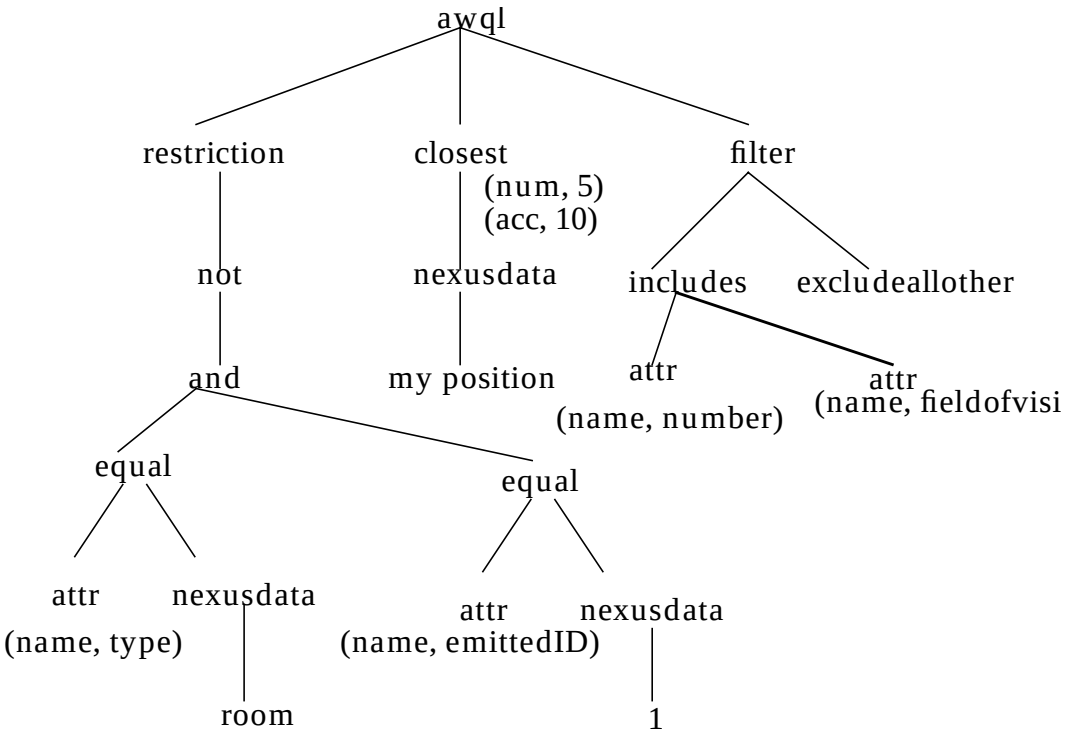


Abbildung 6-7: resultierendes DOM-Dokument

6.11.2 Optimieren der Anfrage

Der Optimierer betrachtet nur den Teilbaum für das Restriction-Prädikat. Falls die Anfrage wie in unserem Beispiel ein oder mehrere Not-Prädikate enthält, so wird die Anfrage in eine äquivalente Anfrage ohne Not-Prädikate umgeformt. Abbildung 6-8 zeigt den Teilbaum nach der Umformung. Zuerst wurde die De Morgan’sche Regel angewendet. Hierdurch wurde not(and(equal, equal)) zu or (not(equal),not (equal)). Im zweiten Schritt wurden beide not(equal) zu notequal() umgewandelt.

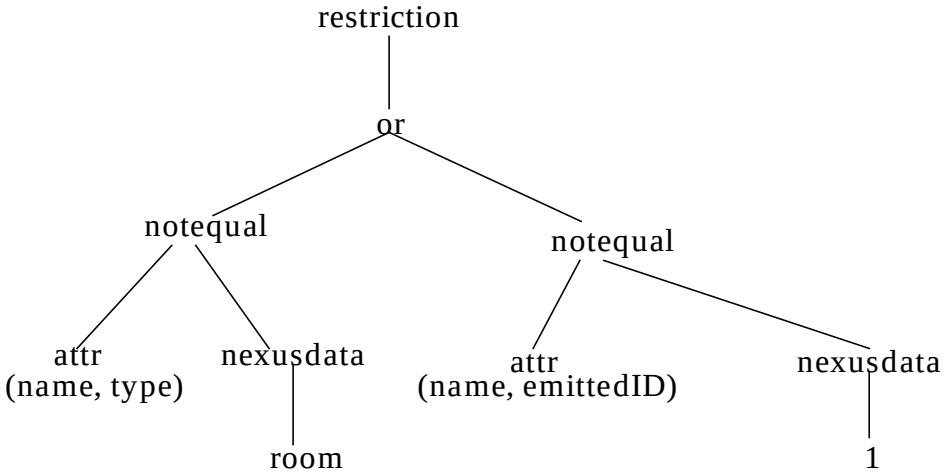


Abbildung 6-8: Der umgeformte Restriction-Teilbaum

6.11.3 Verarbeiten der Restriktion

Die Verarbeitung der Restriktion erfolgt, indem der Baum top down traversiert wird. Dabei wird je nach Kindknoten die dazugehörige Funktion aufgerufen z.B. für einen OR-Knoten die OR-Funktion. Der Baum wird ähnlich wie bei einer Tiefensuche durchlaufen. Solange man kein Blatt erreicht, wird beim ersten Kindknoten abgestiegen. Erreicht man ein Knoten mit equal, notequal, in, notin, inside, notinside, overlap und notoverlap, so muss man nur noch die Kindknoten mit den Parametern für die Operation holen und führt die Operation auf der AWM aus. Die AWM führt diese Operation auf allen Elementen aus und liefert alle Elemente, welche die Bedingung erfüllen.

Das Ergebnis wird der Funktion des übergeordneten Knotens zurückgeliefert und mit den Ergebnissen der anderen Kinder des Knotens verknüpft. Soll ein AND realisiert werden, so muss die Schnittmenge aller Ergebnisse gebildet werden. Bei einem OR die Vereinigungsmenge Danach wird das Ergebnis dieser Funktion zurückgeliefert.

In unserem Beispiel wird zuerst ein OR-Knoten gefunden und deshalb eine OR-Funktion aufgerufen. Diese liest nun den linken notequal Knoten und ruft die NOTEQUAL-Funktion auf. Da unter diesem Knoten nur noch Parameter für die Funktion stehen, werden diese gelesen. Der Knoten „attr” enthält das Attribut name, das den Namen des zu betrachtenden Attributs enthält (Im Beispiel „type”). Der Knoten „nexusdata” enthält den Wert des Attributs, den die Objekte nicht haben dürfen. In unserem Fall „room”. Nun wird die NOTEQUAL-Funktion der AWM aufgerufen. Die Parameter sind dabei „type” für den Name des Attributs und „room” für dessen Wert. Die AWM ruft nun die NOTEQUAL-Funktion jedes Objekts mit diesen Parametern auf.

Alle Objekte, welche die Bedingung erfüllen werden in das Ergebnis eingetragen. Nachdem alle Objekte betrachtet wurden, wird das Ergebnis der AWM an die NOTEQUAL-Funktion zurückgeliefert. Diese gibt das Ergebnis an die OR-Funktion zurück. Die OR-Funktion liest den nächsten Kindknoten ein und

ruft die NOTEQUAL-Funktion für den rechten Teilbaum auf. Der Ablauf entspricht dem beim linken Teilbaum. Die OR-Funktion muss nun die Ergebnisse von E1 und E2 vereinigen. Da keine weiteren Kindknoten existieren wird das Endergebnis zurückgeliefert.

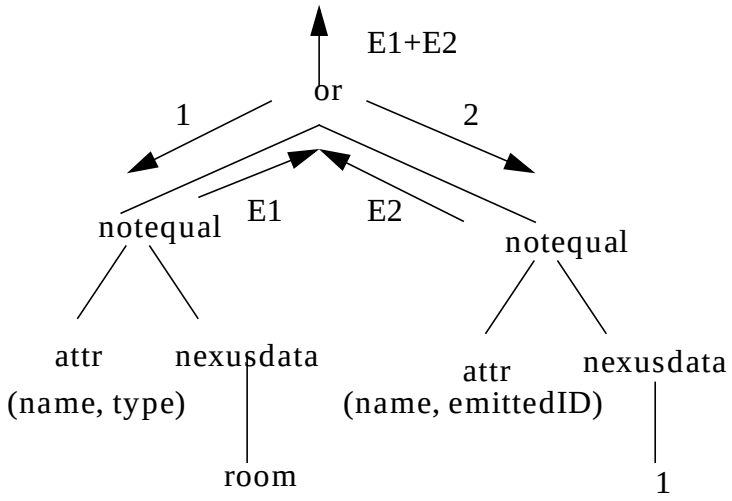


Abbildung 6-9: Verarbeitungsreihenfolge der Anweisungen

6.11.4 Beschreibung der Operationen

EqualTo/In

Bei einem EqualTo erhält ein Objekt zwei Parameter ein Parameter ist das Attribut und der andere der zu vergleichende Wert. Kennt das Objekt das Attribut, so vergleicht es die Werte vom Attribut mit dem übergebenen Wert. Die Funktion vergleicht dabei nur die nicht-geerbten Attribute der Klasse. Die geerbten Attribute werden durch den Aufruf der Methode der Oberklassen verglichen. „In“ funktioniert wie EqualTo nur wird stattdessen ein Attribut mit mehreren Werten verglichen.

Dies ist eine relativ langsame Methode, da jedes Objekt zuerst das Attribut finden muss, dessen Wert verglichen werden soll. EqualTo ließe sich beschleunigen, wenn die AWM stattdessen das Attribut herausfindet und bei den Objekten direkt eine Methode für den Vergleich dieses Attributs aufrufen würde. Der Nachteil wäre allerdings, dass die AWM die Attribute der gespeicherten Objekte kennen müsste. Außerdem besitzt nicht jedes Objekt dieses Attribut. Die AWM müsste dieses erkennen können und sinnvoll bearbeiten. Die AWM-Klasse würde so sehr viel komplexer werden.Ein weiterer Nachteil ist die schlechtere Erweiterbarkeit des Programms. Bei einer Änderung oder Erweiterung der Klassen müssten, dann nicht nur die Nexusparserobjekte und die Nexusobjekte verändert werden, sondern auch die AWM Klasse. Da bei Geschwindigkeitsproblemen wahrscheinlich ohnehin bei beiden Methoden Indexe erstellt werden müssten, wurde die leicht erweiterbare Methode benutzt.

Inside/Overlaps

Die beiden räumlichen Prädikate testen nur, ob ein Punkt innerhalb ihres Polygons liegt. Der verwendete Algorithmus wird in [Sedgewick 1992] beschrieben. Im Gegensatz zu EqualTo benötigt die von der AWM aufgerufene Methode keine Methoden der Oberklassen.

6.11.5 Verarbeitung von Closest

Closest soll die Ergebnismenge auf die n räumlich nächsten Objekte beschränken. Die Closest Funktion sortiert deshalb die Ergebnismenge der Restriction-Funktion nach der Entfernung und kopiert dann die ersten n Elemente in die endgültige Ergebnismenge. Bei räumlichen Objekten mit einer Position wird eine Entfernungsberechnung auf Position oder Extent gemacht. Bei nichträumlichen Objekten ist die Entfernung = 0 d.h getDistance(pos) liefert immer 0 zuruück.

In Gebäuden ist die Entfernungsberechnung über eine Pfadsuche sinnvoller als mit Hilfe der Luftlinie, da es oft keinen direkten Weg zwischen zwei Punkten gibt. Die Berechnung ist allerdings sehr aufwendig. Ein zweites Problem betrifft die räumlichen Indexe: nur wenige sind bei Entfernungsberechnungen mit Hilfe von Pfadsuche hilfreich, da sie sich an der Fläche und nicht an der Entfernung zwischen den Punkten orientieren. Abbildung 6-10 zeigt dies für einen Quad Tree-Index bei dem die Zellen nur ein Objekt enthalten dürfen. Benutzt man Luftlinie, so ist der Quad Tree nützlich, da so eine Abhängigkeit zwischen der Quad Tree Zelle und der Entfernung des enthaltenen Objekts vorhanden ist. Benutzt man dagegen den Pfad zur Entfernungsberechnung, so besteht keine Abhängigkeit zwischen Quad Tree Zelle und Entfernung, wodurch der Quad Tree für die Entfernungsberechnung praktisch nutzlos ist.

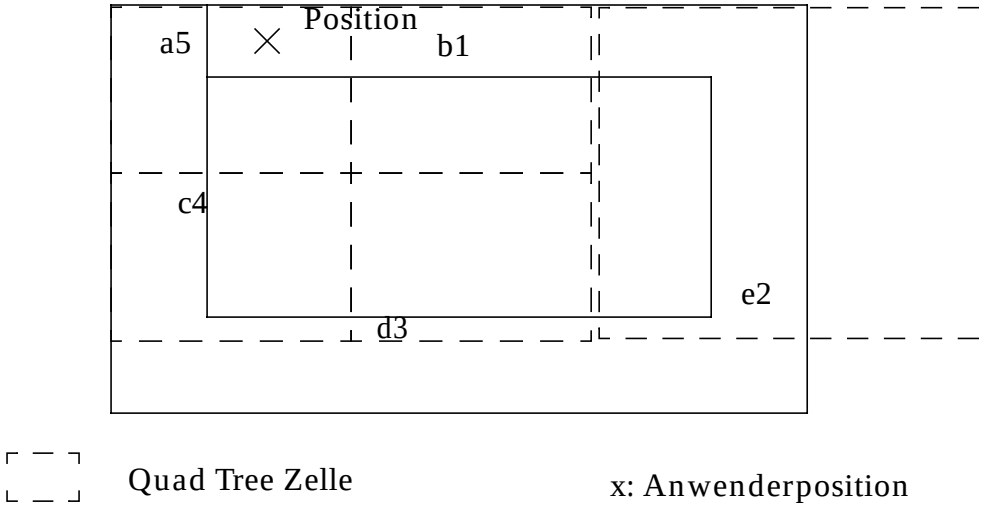


Abbildung 6-10: Quad Tree bei Gebäude

Man sieht auch, dass die Reihenfolge der Objekte zwischen den beiden Berechnungsarten unterschiedlich ist. Da Closest kein unterschiedliches Verhalten bei Anfragen an Objekte außerhalb und innerhalb des Gebäudes haben sollte, berechnet Closest auch beim indoor SpaSe die Luftlinie. Möchte der Anwender tatsächlich eine Reihenfolge der Objekte anhand der Pfadentfernung, so kann man diese mit Hilfe des Navigation Service erhalten, der auch optimale Wege berechnet und weitere Funktionalität anbietet. Hierfür wird ein Navigationsgraph benötigt. Da sich die Wege innerhalb eines Gebäudes nicht schnell ändern, können SpaSes Informationen hierüber speichern. Je nach Aufgabenteilung wird der ganze Graph oder nur Teile davon im SpaSe gespeichert.

6.11.6 Erzeugen eines DOM Dokuments

Aus dem Ergebnis wird ein DOM Dokument mit den Objekten des Ergebnisses erstellt.Hierfür wird zuerst ein leeres DOM Dokument erzeugt und das Result-Objekt ruft bei jedem Objekt die Methode auf mit der es sich in ein DOM-Dokument einträgt.

Die XML-Verarbeitung erzeugt mit ihnen Nexusobjekte. Da die Nexusobjekte als einzige ihre dynamische Klasse kennen, tragen sie sich selbst mit Hilfe der Nexusparserobjekte in ein DOM Dokument ein. Neben der Ausgabe der AWM wird diese Funktion von der Anfrageverarbeitung benötigt, da sie das Ergebnis in ein DOM Dokument umwandeln muss.

6.11.7 Filter

Nachdem die Restriktion und das Closest Element verarbeitet wurden, wird ein Filter Objekt erzeugt, indem in der AWQL Anfrage die filter Anweisung analysiert wird. Es wird damit eine Liste mit den einzufügenden Elementen und den auszulassenden Elementen erstellt.

Es gibt zwei Möglichkeiten wie das Ergebnis gefiltert werden kann. Man kann Filter bei der Erzeugung des DOM Dokuments verwenden, um festzustellen ob ein Attribut eingetragen werden soll oder nicht. Bei der zweiten Möglichkeit wird das DOM Dokument erzeugt und danach die überflüssigen Attribute entfernt. Die zweite Möglichkeit ist in mehrfacher Hinsicht die schlechtere. Sie ist langsamer und verbraucht mehr Speicher, da die Attribute erst eingetragen werden und durch das Filterobjekt wieder gelöscht werden. Sie hat außerdem den Nachteil, dass sie für die Verwendung bei Methoden nicht praktikabel ist, da die Objekte wissen müssen dass sie die Methode eintragen sollen und benötigen die Methodenparameter. Sie wird im Prototypen verwendet, da es so messbar ist wie aufwendig die Erzeugung eines DOM Dokuments ist und wie aufwendig eine Operation wie das Löschen von Elementen auf einem DOM Dokument ist.

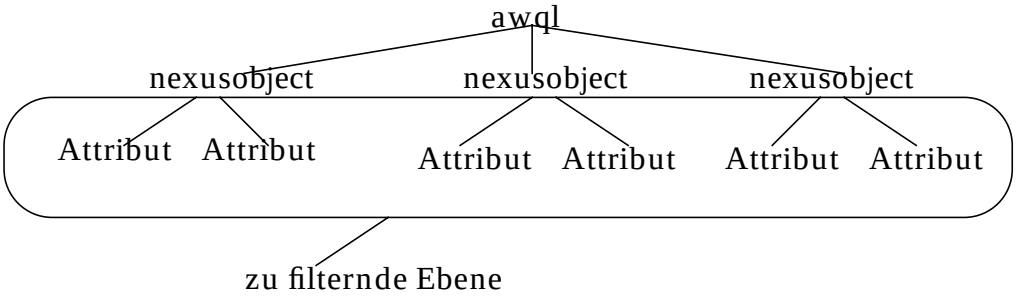


Abbildung 6-11: Die zu filternden Attribute

Attribute befinden sich direkt unterhalb der Nexusobject-Knoten (Abbildung 6-11). Ein Filter muss daher nur diese Ebene betrachten. Ein Attribut wird nicht gelöscht, wenn folgendes gilt:

- Sowohl die includes als auch die excludes sind leer. (Es wurde kein Filter angegeben)
- Es steht in den includes.
- Es steht nicht bei den excludes und in den includes steht ein includeallother oder die includes sind leer.
- Es steht nicht bei den includes und steht nicht bei den excludes und es gibt kein excludeallother.

6.11.8 Die Ausgabe des Ergebnisses

Das DOM Dokument wird an den SOAP Wrapper übergeben, der es mit einem DOMFileWriter Objekt in eine Zeichenkette umwandelt. Das DOM Dokument wird dabei travesiert und in ein XML-Format überführt.

6.12 Einfügen von Objekten

Das Einfügen von Objekten funktioniert prinzipiell wie das parsen einer AWML-Datei. Allerdings werden beim Einfügen eines Objekts in die AWM erst getestet, ob es ein Event gibt, der dadurch ausgelöst wird. Es wird außerdem ein Change Report erstellt, der angibt welche Objekte erfolgreich eingefügt wurden. Ein Change Report wird nicht über den Umweg eines DOM Dokuments erstellt. Stattdessen exisitiert eine Klasse, die einen String im XML-Format verwaltet.

6.13 Ändern oder Löschen von Objekten

Objekte werden geändert oder gelöscht, indem eine AWQL-Anweisung an die entsprechende Methode für das Ändern oder Löschen von Objekten übergeben wird. Hierbei kann allerdings nicht Filter angegeben werden. Stattdessen steht dort bei einer Änderungsoperation ein Update Element mit den Elementen, die geändert werden sollen. Bei einer Löschoperation wird weder Filter noch eine Operation angegeben. Zurückgegeben wird ein Change Report.

Beide Operationen werden zuerst, wie AWQL-Anfragen verarbeitet. Das Ergebnis dieser Anfrage wird nun aber nicht in ein DOM Dokument verwandelt und ausgegeben. Sollen die Objekte gelöscht werden, so wird eine Methode der AWM aufgerufen, die diese Objekte entfernen soll. Es entsteht dabei ein Change Report, der angibt welche Objekte gelöscht wurden. Eine Änderungsoperation ist komplizierter. Es muss nun zuerst das Update Element ausgewertet werden. Da das Ergebnis aus Referenzen auf die Objekte in der AWM besteht, wirken sich Änderungen auf ihnen sofort auf die AWM aus. Jedes dieser Objekte besitzt eine Update Methode, die als Parameter eine Liste mit den Attributen und ihren neuen Werten enthält. Die Objekte setzen die Attribute nun auf die neuen Werte. Ist ein Attribut nicht bekannt oder ist sonst ein Update auf ein Attribut nicht möglich, so erhält das Objekt wieder seine alten Attributwerte vor der Update-Operation. Im Change Report wird dann der Update Versuch als gescheitert markiert.

Wie beim Einfügen können bei erfolgreichen Änderungen oder Löschfunktionen Events ausgelöst werden.

6.14 Events

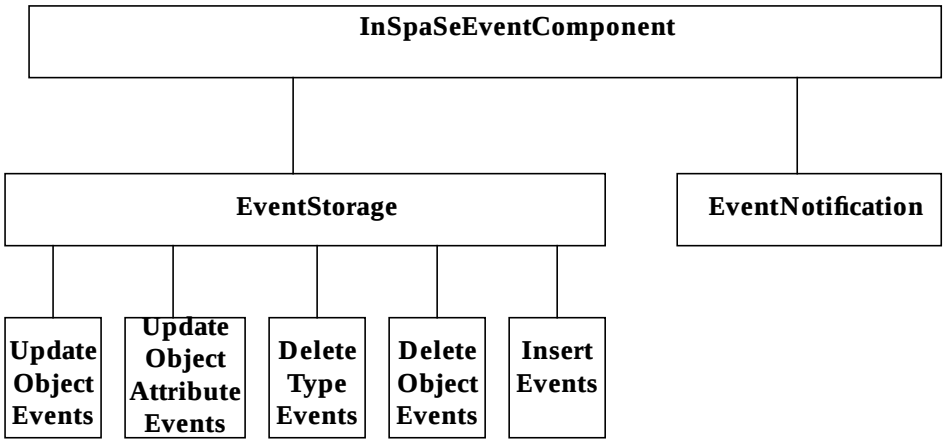


Abbildung 6-12: Aufbau der Eventkomponente

Abbildung 6-12 zeigt den Aufbau der Eventkomponenten des Indoor SpaSe. Er besitzt zwei Komponenten. Event Storage verwaltet die Events und EventNotification dient zum Versenden von Event Notifications. Das Event Storage ist nicht die einzige Komponente, in der man Events speichern könnte. Es ist auch denkbar sie in der AWM als Nexusobjekte zu speichern. Hierdurch könnte man Daten zu den Events mit Hilfe von AWQL ermitten. Da die AWM allerdings nicht darauf optimiert ist, wäre die Geschwindigkeit der Eventverarbeitung geringer. Ein weiterer Grund gegen eine Speicherung in der AWM liegt darin, dass die meisten Anwendungen nicht an diesen Events interessiert sind. Möchte man die Events trotzdem in einem SpaSe speichern, so kann man dies auch bei einer

getrennten Speicherung von Events und Nexusobjekten erreichen. Hierzu muss man eine Nexusklassen für die Events erstellten und registriert Events beim SpaSe. Sobald man eine Event Notification erhält fügt man ein entsprechendes Objekt von der Klasse Event in den SpaSe ein.

6.14.1 Eventarten

Es werden die folgenden Eventarten unterstützt:

- **Delete Event:** Ein Delete Event wird auf ein Objekt mit einer angegebenen NOL gelegt und triggert, sobald dieses gelöscht wird.
- **Delete-Type Event:** Ein Delete Type Event triggert, wenn ein Objekt von einem bestimmten Typ gelöscht wird. Werden bei einem Delete mehrere Objekte dieses Typs gelöscht, so wird für jedes Objekt der Event ausgelöst. Bei 5 Objekten wird der Event 5 mal ausgelöst und benachrichtigt
- **Insert Event:** Insert Events werden ausgelöst, wenn Objekte des angegebenen Typs eingefügt werden.
- **Update Event:** Update Events werden ausgelöst, wenn ihr beobachtetes Objekt ein update erhält. Es ist dabei egal auf welches Attribut und bei meinem SpaSe wird momentan nicht überprüft, ob das Update tatsächlich etwas ändert oder nicht.
- **Update-Attribute Event**: Ähnlich wie Update Events, allerdings wird nur getriggert, wenn ein bestimmtes Attribut von einem Update betroffen ist.

6.14.2 Die IDs

Template IDs dienen dazu, um Eventarten voneinander zu unterscheiden. Ein Delete Event hat daher eine andere Template ID als ein Insert Event.

Eine Predicate ID wird generiert, wenn der SpaSe ein Event registrieren soll. InSpaSeEventComponent erzeugt eine eindeutige Predicate ID, indem es eine Zahl verwendet, die bei der Registrierung eines Events hochgezählt wird. Die Predicate ID enthält die TemplateID.

Eine Notification ID wird generiert, wenn ein Event eintritt, enthält die Predicate ID und wird dann versendet. Der Indoor SpaSe speichert hierbei eine Zahl, die bei jedem auftreten eines Events hochgezählt wird.

Jedes Event speichert eine Template ID, eine Predicate ID und die als nächstes auszugebende Notification ID.

6.14.3 Auslösen eines Events

War eine Änderungsoperation erfolgreich, so wird die Eventkomponente darüber informiert. Für jede Änderungsoperation bietet die Eventkomponente eine Methode an, die von der dazugehörigen Änderungsoperation aufgerufen wird. Als Parameter werden bei Insert und Delete nur die Objekte übergeben. Bei Update müssen auch die geänderten Attribute angegeben werden. Die Eventkomponente holt sich vom EventStorage alle getriggerten Events. Hierzu muss das Event Storage, alle Events der in Frage kommenden Eventarten betrachten und feststellen welche Triggern. Bei einem Update Event etwa müssen alle UpdateObject und UpdateObjectAttribut Events betrachtet wer-

den. Die Eventkomponente versendet Event Notifications für die getriggerten Events mit Hilfe eines EventNotification Objekts.

6.15 Zusammenfassung

SpaSes werden in der Regel mit Hilfe von GIS oder um räumliche Funktionen erweiterte Datenbanken realisiert. Der Indoor SpaSe wurde durch einen anderen Ansatz entwickelt. Der SpaSe übernimmt die Verwaltung selbst und speichert die Objekte im Hauptspeicher. Die Entwicklung erfolgte in zwei Iterationen. In der ersten Iteration wurde ein Grundgerüst erstellt, das in der zweiten Iteration erweitert wurde. Der Indoor SpaSe bietet seine Dienste durch einen Applikationsserver an, über den mit Hilfe von SOAP auf die Methoden des SpaSe zugegriffen werden kann. Die Komponenten des Indoor SpaSe lassen sich in Komponenten für die Anfrageverarbeitung, Komponenten für die Verwaltung der Objekte, Komponenten zur XML-Verarbeitung und in Komponenten für die Eventverarbeitung aufteilen. Der Komponenten sind einfach aufgebaut und besitzen kaum Geschwindigkeitsoptimierungen. Es sollen hierdurch mögliche Probleme leichter erkannt werden können. Das folgende Kapitel zeigt, ob die Geschwindigkeit ausreicht und wo Probleme entstehen können.

7 Evaluation

In diesem Kapitel wird der Indoor SpaSe evaluiert. Hierbei wird die Geschwindigkeit betrachtet, Optimierungen vorgeschlagen. Danach wird der Einsatz von SOAP betrachtet.Zwei weitere wichtige Bereiche die betrachtet wurden, sind die Erstellung der Daten und der Einsatz des Standard Class Schemas. Zum Abschluss wird der verwendete Ansatz betrachtet.

7.1 Leistungsfähigkeit

Es wurden einige Geschwindigkeitstest durchgeführt. Die Daten der verwendeten Umgebung zeigt Tabelle 7-1.

Tabelle 7-1: die verwendete Umgebung

Rechnerbezeichnung	IBM Thinkpad A22m
Betriebssystem	Windows 2000
Prozessor	Mobile Pentium III 850 MHz
Enwicklungsumgebung	IBMs Visual Age 4.0 Enterprise Edition
Speicher	320 MB RAM
freier Speicher	135 MB

Die Daten wurden mit Hilfe eines Programms generiert. Das Programm erzeugte eine bestimmte Anzahl von Objekten und variierte dabei diverse Attribute. Tabelle 7-2 zeigt die Attribute und ihre Verwendung an. Nicht alle

Tabelle 7-2: Attribute der Sender

id	einmalig
name	einmalig
partOf	nicht verwendet
attributes	nicht verwendet
kind	überall gleich („real“)
pos	einmalig
extent	nicht verwendet
floor	überall gleich („0“)
belongsTo	nicht verwendet
emittedID	einmalig
fieldOfVision	einmalig

Attribute wurden verwendet. Die verwendeten Attribute konnten bei allen Objekten gleich oder unterschiedlich sein. Bei den Attributen, die alle gleich sind steht in Klammern ihr Wert. Tabelle 7-3 zeigt die Attribute bei Räumen an.

Tabelle 7-3: Attribute der Räume

id	einmalig
name	einmalig
partOf	nicht verwendet
attributes	nicht verwendet
kind	überall gleich („real“)
pos	einmalig
extent	nicht verwendet
floor	überall gleich („0“)
belongsTo	nicht verwendet
yearOfConstruction	überall gleich („1980”)
navigationNode	nicht verwendet
representation	einmalig
infor	einmalig
number	einmalig
illuminationState	on

Beim ersten Test wurde die Zeit gemessen, in der eine AWML-Datei vom Indoor SpaSe eingelesen wird. Der Test wurde für Sender und Räume durchgeführt. Tabelle 7-4 zeigt die Zeit für den Import von Sendern. Es werden dabei ca. 2ms pro Objekt benötigt.

Tabelle 7-4: Import von Sendern

Anzahl Sender	Dauer	Dateigröße
500	1.042 ms	244 KB
2.000	4.026 ms	987 KB
10.000	19.498 ms	5.010 KB
40.000	85.233 ms	20.313 KB

Räume besitzen im Vergleich zu Sendern mehr Daten. Ihr Import dauert daher länger. Die Zeit pro Objekt beträgt im Durchschnitt 2.5 - 3ms. Tabelle 7-5 zeigt die Import von Räumen. Hierbei wird auch die Dauer des

Tabelle 7-5: Import von Räumen

Anzahl Räume	Dauer	Dateigröße
200 (Fakultät)	480 ms	125 KB
400	1.152 ms	280 KB

Tabelle 7-5: Import von Räumen

1.000	2.373 ms	703 KB
5.000	14.162 ms	3.558 KB
10.000	24.185 ms	7.147 KB
60.000	184.805 ms	43.514 KB

Imports der 200 Räume des Fakultätsgebäudes dargestellt. Die Geschwindigkeit ist mehr als ausreichend, da wahrscheinlich nur bis zu 1000 Objekte tatsächlich importiert werden. Ein Import von größeren Datenmengen ist auch möglich und, da er selten vorkommt, auch kein Problem. AWML-Dateien entsprechen von der Form her den Antworten einer AWML- Anfrage. Anhand der Dateigröße lässt sich erkennen, wie groß die Datenmenge bei der Übertragung einer solchen Antwort ist. Die Übertragungszeit größerer Datenmengen kann zu einem Problem werden, da bereits die Übertragung von 1.000 Räumen mit einem WLAN über eine Sekunde dauert. Die Übertragungszeit der 60.000 Räume würde sogar über eine Minute benötigen.

In weiteren Tests wurde die Ausführungsgeschwindigkeit von AWQL Anfragen gemessen. Zuerst wurden die Positionierungsanfragen der Sensorkomponente getestet. In weiteren Tests wurden dann auch allgemeine Anfragen gestellt. Die Test erfolgten, indem der SOAP Wrapper als Programm gestartet wurde. Die Anfragen wurden über Methodenaufrufe direkt an den SOAP Wrapper gestellt und wurden nicht über SOAP verschickt, da nur die Geschwindigkeit des SpaSe bewertet werden sollte. Die erste Positionierungsanfrage wird für Hoarding benutzt und verlangt alle Sender. Die zweite wird bei eine laufenden Positionierung benutzt und verlangt den Sender zu der dazugehörigen ID. Tabelle 7-6 zeigt die Ausführungsdauer der beiden Positionierungsanfragen bei unterschiedlichen Objektmengen.

Tabelle 7-6: Ausführungszeit der Positionierungsanfragen

	Hoarding	Einzelner Sender
500 Emitter	711ms	60 ms
2.000 Emitter	5.438 ms	60 ms
10.000 Emitter	25.727 ms	130 ms
40.000 Emitter	174.120 ms	351 ms

Die Geschwindigkeit der Anfrage für einzelne Sender ist sehr hoch. Die Anforderungen durch eine laufende Positionierung, wie sie in Kapitel 5 „Anforderungen an den Indoor Spatial Model Server" vorgestellt wurden, können zwar nicht ganz erreicht werden, allerdings stellt diese Positionierungsmethode sehr hohe Anforderungen. Wie zu erwarten war die Anfrage für Hoarding langsamer als die Anfrage für einen einzelnen Sender. Vergleicht man beide Anfragen so stellt man fest, dass die großen Geschwindigkeitsunterschiede nicht durch das Ermitteln des Ergebnisses entstanden sein können. Beide Anfragen machen einen kompletten Scan über die gespeicherten Objekte. Der einzige Unterschied ist, dass beim Hoarding sehr viel mehr

Objekte im Ergebnis enthalten sind. Analysiert man die Ergebnisse so zeigt es sich, dass die Zeit für das Durchsuchen der Objekte in der AWM bei beiden Anfragen in etwa gleich lange dauert. Die großen Geschwindigkeitsunterschiede müssen sich daher an anderer Stelle ergeben. Tabelle 7-7 zeigt

Tabelle 7-7: aufgeschlüsselte Ausführungszeit bei 10.000 Sendern

	Hoarding	Positionierung
Parsen der Anfrage	30 ms	30 ms
Verarbeiten von EqualTo	80 ms	70 ms
Erzeugen des DOM Dokuments	10.800 ms	0 ms
Erstellen des Ergebnisstrings	10.000 ms	30 ms

die Zeit die Teilfunktionen bei 10.000 Sendern benötigen. Die Probleme des Hoardings entstehen offensichtlich nicht durch das Verarbeiten einer EqualTo Anweisung oder durch das Parsen. Die Erstellung eines DOM Dokuments und des Ergebnisstrings kosten die meiste Zeit.

Die zwei wichtigen Teilfunktionen Filter und Closest wurden von den Positionierungsanfragen nicht getestet. Verknüpfungen mehrerer Teilbedingungen sind hierdurch ebenfalls nicht getestet worden. Auch werden weder inside noch overlap von diesen Funktionen benötigt. Da solche Funktionen nur von Nexusanwendungen benutzt werden und diese auf Räumen arbeiten wurde in einem zweiten Benchmark, der diese Funktionen testen soll, Räume als Objekte verwendet. Auch hier zeigte sich, dass die Geschwindigkeit der Anfragen maßgeblich durch die Anzahl der Objekte im Ergebnis bestimmt werden. Es wurden daher weitere Anfragen erstellt. Die Anfragen zum Test von Filter und von Closest ermitteln alle Räume und wenden danach Filter beziehungsweise Closest an. Bei Filter werden alle Attribute bis auf den Namen des Objekts herausgeschnitten. Bei Closest wird das Ergebnis auf die 5 nächsten Räume beschränkt. Tabelle 7-8 zeigt die Ausführungszeit und vergleicht sie mit der Anfrage, die alle Räume als Ergebnis hat und keine weiteren Operationen darauf ausführt.

Tabelle 7-8: Allgemeine Anfragen bei 10000 Räumen

	Alle Räume	Filter	Closest
Parsen der Anfrage	40 ms	20 ms	20 ms
Filter	0	751 ms	0
Closest	0	0	120 ms
Erzeugen des DOM Dokuments	14.000 ms	12.000 ms	10 ms
Erstellen des Ergebnisstrings	10.000 ms	10.000 ms	0 ms

Bei der getesteten räumlichen Anfrage wurden die Objekte mit einer Position innerhalb eines kleinen Bereichs gesucht. Es zeigte sich, dass der Algorithmus für Overlaps mit 60 ms so schnell wie ein EqualTo (50 ms) ist. Es zeigte sich auch, dass die Verknüpfung mehrerer Bedingungen ebenfalls sehr aufwendig werden kann, so erhöht eine Verknüpfung zweier Bedingungen die Ausführungszeit um vier Sekunden, wenn beide Bedingungen 10.000 Elemente als Teilergebnis liefern. Dies war unabhängig von der Verknüpfungsmethode. Die letzten betrachteten Funktionen waren die Änderungsoperationen zum Löschen und Verändern der Daten. Das Löschen von 10.000 Räumen dauerte 21 Sekunden während das Ändern der Daten aller Räume 9 Sekunden benötigte. Da beide Operationen sehr selten sind und auch etwas länger dauern dürfen, ist die Geschwindigkeit ausreichend.

Bei Experimenten mit 40.000 und 60.000 Räumen zeigte sich ein weiteres Problem wenn viele Objekte zurückgegeben werden. Der Speicherverbrauch ist so hoch, dass der Rechner Daten auslagern muss oder sogar der Speicher nicht mehr ausreicht. Werden nicht alle Objekte, sondern nur ein kleiner Teil als Ergebnis zurückgegeben, so konnten deutlich mehr (>100.000) Objekte vom Prototypen verarbeitet werden.

Die Experimente zeigen, dass die XML-Verarbeitung zu einem Problem werden kann. Dieses Problem betrifft den Indoor SpaSe wahrscheinlich nicht, da er auf Objekte innerhalb eines Gebäudes beschränkt ist. Er wird nur sehr viel kleinere Objektmengen verwalten und die Anzahl der Objekte in einem Anfrageergebnis wird nur sehr klein (<50) sein. Bei einem Test mit einer Objektmenge von 200 Räumen und 500 Sendern lag die Ausführungszeit der Anfragen zwischen 100 und 400 ms. Dies ist zwar zu langsam für eine laufende Positionierung der Benutzer allerdings ausreichend, wenn die Position nur bei Bedarf einer Nexus-Anwendung bestimmt werden muss. Möglicherweise werden Nexusknoten Probleme mit der XML-Verarbeitung bekommen, da sie die Ergebnisse aus mehreren Systemen beziehen, verarbeiten und zu einem größeren Ergebnis verknüpfen. Am Beispiel von Filter sieht man, dass dies länger dauert als ein EqualTo.

7.2 Optimierungen

Möchte man eine laufende Positionierung machen oder übersteigen die Anforderungen die Anforderungen an den Indoor SpaSe, so wird man Optimierungen vornehmen müssen. Anhand der Daten aus dem Benchmark kann man gut erkennen, dass nun nicht nur der Zugriff auf die Daten, die Ausführungszeit bestimmt. Wie bei Hauptspeicherdatenbanksystemen, haben nun auch andere Komponenten einen großen Anteil an der Ausführungsgeschwindigkeit. Möchte man die Ausführungsgeschwindigkeit erhöhen, so ist es wahrscheinlich am wichtigsten die Umwandlung der Objekte in ein XML-Format zu beschleunigen. Man könnte beispielsweise die Geschwindigkeit dadurch erhöhen, dass die Nexusobjekte nicht jedesmal aufs neue die Daten erzeugen, die in das DOM Dokument eingetragen werden sollen, sondern diese Daten nur einmal erzeugen und speichern, um bei Bedarf nur einen Knoten einhängen zu müssen. Dies ist schneller, kostet dafür aber auch Hauptspeicher. Reduziert man den Aufwand für die XML-Verarbeitung auf

ein Minimum, so wird man wahrscheinlich den Zugriff auf die Daten beschleunigen wollen. Interessanterweise ist es praktisch bedeutungslos, welchen der in Kapitel 4 „Hauptspeicherdatenbanken" vorgestellten Indexe man verwendet. Geht man davon aus, dass man nicht mehr als 4 Milliarden Objekte verwalten wird, so lässt sich die maximale Anzahl Vergleiche bei einem AVL-Baum mit 33 abschätzen, da er fast einem vollständig ausbalanciertem binären Baum entspricht. Geht man davon aus, dass T-Bäume und B-Bäume nicht mehr als doppelt so viele Vergleiche benötigen, so werden diese Datenstrukturen nie mehr als 66 Vergleiche durchführen. Bei allen diesen Indexen dürfte die Zeit zum Finden des gesuchten Elements weniger als 1 ms betragen. Die Ausführungsgeschwindigkeit wird nun von der Verarbeitung der AWQL-Anfrage bestimmt. Da allein das Parsen 20 ms oder mehr verbraucht, wäre dies die bestimmende Komponente für die Ausführungsszeit. Möchte man mehr als 500 Anfragen pro Sekunde verarbeiten, so muss man entweder einen schnelleren Parser verwenden oder statt DOM eine effizientere XML-Schnittstelle verwenden.

7.3 Evaluation von SOAP

Die Anbindung des Indoor SpaSe an die SOAP Schnittstelle ließ sich sehr leicht bewerkstellen, da keinerlei Anforderungen an den Wrapper gestellt wurden und er dadurch relativ einfach aufgebaut sein konnte. Die Methodenparameter und Rückgabedaten sind einfache Zeichenketten. Da die Übertragung von Zeichenketten direkt von SOAP unterstützt wird, ist kein Programm notwendig, um die Daten in das XML-Format umzuwandeln. Aus Sicht des Indoor SpaSe ist die Anbindung an SOAP vollkommen transparent und erfordert keinerlei Anpassungen. Völlig anders sieht dies auf der Client Seite aus. Hier ist die Anbindung an SOAP nicht transparent. Der Client greift auf Methoden entfernter Objekte anders zu als auf lokale Objekte. Hierbei werden keinerlei Typüberprüfungen zur Kompilationszeit durchgeführt. Es ist daher leicht möglich, dass ein Client für einen Methodenaufruf fehlerhafte Parameter angibt. Dies unterscheidet sich sehr stark von den Mechanismen, wie etwa bei CORBA, bei denen der Zugriff auf entfernte und lokale Objekte identisch ist und aus Sicht des Clients vollkommen transparent erfolgt. In CORBA werden Methodenköpfe (sogenannte Stubs) erzeugt mit denen Compiler die Parameter bei Methodenaufrufen bereits zur Kompilationszeit überprüfen können.

Die Geschwindigkeit von SOAP wurde gemessen, indem die Zeit zwischen der Rückgabe und dem Empfang des Ergebnisses bestimmt wurde. Hierbei sendete ein Testprogramm eine Anfrage. Der InSpaSe gab bevor er das Ergebnis zurückgab die interne Uhrzeit aus. Sobald das Testprogramm die Antwort empfangen hat, gab auch er die Uhrzeit aus. Da sich das Testprogramm und der SpaSe auf dem selben Rechner befanden, konnte so die Zeit gestoppt werden.Der SpaSe enthielt die Daten des Fakultätsgebäude und damit 200 Räume und null Sender. Da die Dauer wahrscheinlich von der Größe der übertragenen Daten abhängt wurde einmal ein Anfrage gestellt die eine leere Ergebnismenge zurückliefert und einmal eine, die eine große Ergebnismenge zurückliefert. Der SpaSe sollte daher zuerst alle Sender zurückliefern und in einer anderen Anfrage alle Räume.Tabelle 7-9

zeigt das Ergebnis. Die Geschwindigkeit ist, wie man erkennen kann, nicht sehr gut.

Tabelle 7-9: Dauer der Datenübertragung

Zeit bei leerer Ergebnismenge	50ms
Bei Ergebnis mit 200 Räumen	21.220 ms

7.4 Erstellung der Daten

Prinzipiell kann man AWML Dateien mit jedem beliebigen Editor erstellen. Die Dateien werden allerdings sehr schnell sehr groß, da man bei XML sehr viele zusätzliche Daten erzeugt. Die Dateneingabe ist auch sehr fehleranfällig, da man sehr schnell ein Element vergisst und Tippfehler auch die Elemente betreffen. Sollen die Daten von anderen Menschen gelesen werden, so ist es daher sinnvoll einen XML-Editor zu benutzen. Ein XML-Editor erleichtert die Dateneingabe, indem er XML-Elemente einrückt und einfache Überprüfungen des XML-Dokuments durchführt, indem er etwa überprüft, ob die Regeln von XML eingehalten werden. Allerdings dürfte auch mit der Verwendung eines XML-Editors die Eingabe großer Datenmengen zu aufwendig sein. Beispielsweise ist die im Leistungstest verwendete Datei mit 10.000 Räumen 7 MB groß. Dies entspricht in etwa 700 Byte pro Raum. Die tatsächliche Datenmenge eines Objekts dürfte bei weniger als der Hälfte liegen. Der Rest entsteht durch die XML-Darstellung. Es ist daher sinnvoll, für die Eingabe großer Datenmengen ein Programm zu entwickeln, bei dem man nur die Daten zu den Objekt eingeben muss und das daraus eine XML-Datei erstellt.

Neben dem Aufwand für die Dateneingabe muss auch mit einer Aufbereitung der bereits vorhandenen Daten gerechnet werden. Die Daten sind oft für die Darstellung und Verarbeitung auf Desktoprechnern optimiert. Die Nexusanwendungen sind auf mobilen Geräten, die in der Regel andere Anforderungen an die Daten stellen. So wurde in [Grzan 2000] eine mobile Datenbankanwendung entwickelt, die Karten des Gebäudes anzeigte. Diese Karten wurden speziell für die Darstellungsmöglichkeiten eines PDAs entwickelt. Ein weiterer Aufwand entsteht durch die Ermittlung der Positionen der Objekte. Bei der Erstellung des Fakultätsszenarios mussten die Positionen der Räume aus Karten ermittelt werden. Hierfür musste zuerst für jedes Stockwerk Karten aus den Karten des im Internet verfügbaren Gebäudeplans erstellt werden.

7.5 Evaluation des Standard Class Schema

Das Standard Class Schema, wie es in [Meßmer 2001] beschrieben ist, musste für das Einsatzszenario erweitert werden. Es wurde die Klasse Sender hinzugefügt und Standard Klassen mussten um zusätzliche Attribute wie etwa „info“ ergänzt werden. Ein besonderes Problem ist die Angabe der Höhe innerhalb eines Gebäudes. Im Gegensatz zu vielen Outdoor-Anwendungen bei denen die Höhe keine Rolle spielt, ist dies bei eine Anwendung innerhalb eines Gebäudes sehr wichtig, da es in der Regel mehrere Stockwerke besitzt. Da GML nur zwei Dimensionen kennt, muss die Höhe selbst definiert werden.

Leider verlangen unterschiedliche Anwendungen unterschiedliche Höhenangaben. So gibt es Anwendungen, die als Höhenangaben eine Angabe relativ zum Meeresspiegel benötigen. Beispielsweise wird bei Wetterangaben die Schneefallgrenze als Höhe relativ zum Meeresspiegel angegeben. Ein Bergwanderer braucht daher eine Höhenangabe relativ zum Meeresspiegel. Andere Anwendungen verlangen eine Höhe relativ zum Erdboden, um etwa die Höhe eines Turms anzugeben. Beide Varianten sind für Anwendungen innerhalb eines Gebäudes oft eher ungeschickt, da Anwender aus solchen Höhenangaben nicht auf eine Position im Gebäude schließen können. Hier ist eine Angabe wie das Stockwerk sehr viel sinnvoller und wird daher im Datenmodell des Indoor SpaSe verwendet.

Die Verwendung von optionalen Attributen ist sehr sinnvoll, da es Anwendungen gibt, bei denen diese Attribute überflüssig sind. So ist etwa das Attribut „attributes“ in dem Einsatzszenario überflüssig. Es sollten allerdings noch weitere Attribute als optional gekennzeichnet werden. So ist etwa das Attribut „NavigationNode“ nicht optional und müsste daher von jedem SpaSe unterstützt werden, selbst wenn keine Unterstützung für Navigationsknoten geplant ist.

7.6 Bewertung des Ansatzes

Die Sensorkomponente benötigt zur Positionsbestimmung nur die Umwandlung einer ausgesendeten ID in einen Sender und dessen Position. Die Funktionalität des Indoor SpaSe war für die Sensorkomponente bereits nach der ersten Iteration fast ausreichend. Benötigt wurde nur noch die Anbindung an das Netz und das Einlesen von GMLPolygonen und Positionen. Es sind keine räumlichen Funktionen notwendig und auch Closest oder Filter sind nicht nötig.

Nexusanwendungen benötigen allerdings eine komplette Implementierung von AWQL. Der Prototyp zeigt, dass ein funktionierender und ausreichend schneller Spatial Model Server in Java schnell realisierbar ist. Ein großes Problem ist dabei die Entwicklung von Klassen, die geometrische Funktionen realisieren müssen. Da momentan Klassen für diese Aufgaben realisiert werden, sollte sich der Aufwand auch hierbei reduzieren.

Da die verwalteten Objekte eines SpaSe nicht sehr groß sind, lassen sich viele Objekte im Hauptspeicher verwalten. Da die Hauptspeicherausstattung heutiger Computer mehrere hundert Megabyte umfasst, ist es wahrscheinlich, dass der hauptspeicher SpaSe sogar für Gebiete, die größer als ein Gebäude sind, verwendet werden kann.

7.7 Zusammenfassung

Der Prototyp wurde mehreren Geschwindigkeitstests unterzogen. Hierfür wurden AWML-Dateien mit unterschiedlich vielen Objekten generiert. Als erstes wurde der Import der Daten getestet. Es dauert durchschnittlich zwischen 2-3 ms pro Objekt und ist damit mehr als ausreichend schnell. Als nächstes wurden zwei Positionierungsanfragen getestet. Hierbei wur-

den eine typische Hoarding Anfrage und eine normale Positionierungsanfrage verwendet. Wie zu erwarten benötigt eine Hoarding Anfrage länger als eine normale Positionierungsanfrage. Die Geschwindigkeitsunterschiede entstehen durch die langsame Erstellung des DOM Dokuments und der sehr viel größeren Datenmenge, die in eine Zeichenkette umgewandelt werden muss. Diese beiden Teilfunktionen haben bei Ergebnissen mit vielen Objekten den größten Anteil an der Ausführungszeit und verursachen einen hohen Speicherverbrauch. Da der Indoor SpaSe allerdings nur relativ wenige Daten verwaltet und die Ergebnisse der Anfragen in der Regel nur wenige Objekte enthalten, ist die langsame Erstellung des XML-Ergebnisses kein Problem. Werden Aufgrund höherer Anforderungen Optimierungen benötigt, so muss wahrscheinlich zuerst die Erzeugung des XML-Ergebnisses beschleunigt werden. Danach kann man Indexe verwenden, um die Geschwindigkeit zu erhöhen, die Unterschiede zwischen AVL-, B- und T-Bäumen sind dabei gering.

Neben der Geschwindigkeit wurde auch die SOAP-Anbindung betrachtet. Die Anbindung eines Servers ist einfach. Allerdings ist der Aufwand auf der Clientseite sehr viel größer und der Zugriff erfolgt nicht transparent. Ein weiterer betrachteter Bereich ist die Erstellung der Daten. Die Datenmenge kann sehr schnell groß werden und muss manuell erstellt werden, um die Daten in eine Form zu bringen, die für die mobilen Geräte benötigt wird. Das Standard Class Schema enthielt nicht alle benötigten Klassen und Attribute und wird wahrscheinlich um Sender erweitert. Die Verwendung von optionalen Attributen ist sinnvoll, da sonst einige für den Anwendungsfall überflüssige Attribute implementiert werden müssten. SpaSes zur Unterstützung von Sensorkomponenten lassen sich sehr schnell realisieren, da keine räumlichen Anfragen benötigt werden. Die Anzahl der verwaltbaren Objekte ermöglicht es wahrscheinlich sogar größere Gebiete als Gebäude abzudecken.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Das Nexus-Projekt entwickelt eine Plattform, die als Infrastruktur für ortsbasierte Anwendungen dienen soll. Ein wichtiger Teil dieser Plattform sind Spatial Model Server. Sie verwalten statische Objekte und erhalten Anfragen von anderen Systemen wie etwa Nexusknoten. Sie können Events erzeugen und sind daher eine Event Source für den Event Service. Der entwickelte Spatial Model Server soll Objekte eines Gebäudes verwalten und die Sensorkomponente bei der Positionierung unterstützen. Die Sensorkomponente benötigt den Spatial Model Server nicht für Outdoor Positionierungssysteme, sondern nur für bestimmte Indoor Positionierungssysteme. Die wichtigsten dieser Positionierungssysteme bestehen aus Sendern die IDs ausstrahlen, die vom Spatial Model Server in eine Position umgewandelt werden müssen.

Spatial Model Server sollen wie Datenbanksysteme Daten verwalten. Hauptspeicherdatenbanksysteme unterscheiden sich von herkömmlichen Datenbanksystemen wegen der unterschiedlichen Eigenschaften von Hauptspeicher und Festplattenspeicher. Werden Daten komplett im Hauptspeicher gehalten, so beschleunigt sich der Zugriff auf die Daten. Dies führt dazu, dass nun nicht nur die Verwaltung der Daten die Gesamtgeschwindigkeit beeinflusst, sondern auch die anderen Komponenten eines Datenbanksystems. Nicht nur die Architektur, sondern auch viele Implementierungsdetails müssen an die Eigenschaften von Hauptspeicher angepasst sein. So werden andere Indexe benötigt und die Transaktionsverwaltung muss auf sehr hohe Datendurchsätze ausgelegt sein.

Betrachtet man die Anforderungen an den Spatial Model Server, so zeigt es sich, dass er die zu speichernden Objekte weitgehend im Hauptspeicher verwalten kann, aber bestimmte Formen der Positionierung sehr hohe Anforderungen an die Anfrageverarbeitung stellen können.

Der Aufbau des entwickelten Prototypen ist sehr einfach gehalten. Er besitzt keine Indexe oder besondere Optimierungen, da eher auf eine große Funktionalität wertgelegt wurde. Hierdurch zeigen sich Probleme bei bestimmten Aufgaben sehr schnell, wodurch auch Aussagen für andere Systeme ermöglicht werden. Die Evaluation zeigte, dass die Geschwindigkeit für das gewünscht Einsatzszenario (Fakultätsgebäude) vollkommen ausreicht. Es zeigten sich bei der Verwendung größerer Objektmengen Probleme beim Speicherverbrauch und der Geschwindigkeit. Die Probleme wurden allerdings nicht durch den Zugriff auf die Objekte verursacht, sondern durch die XML-Verarbeitung, die bei großen Objektmengen sehr speicherintensiv und langsam war. Die Umwandlung solch großer Objektmengen sollte allerdings in der Realität nicht vorkommen, da allein die Übertragung der Daten auf Funknetzen lange dauern würde und die Datenmenge für viele mobile Computer zu groß ist. Außerdem stellt das Erstellen vieler Objekte einen großen Auf-

wand dar, weshalb nicht viele Spatial Model Server eine so große Anzahl von Objekte verwalten müssen.

Es zeigte sich auch, dass der Aufwand, um einen Spatial Model Server zur Unterstützung eines Positionierungssystems zu erstellen relativ gering ist, da hierfür nur wenig Funktionalität benötigt wird. Die Bereitstellung der Dienste ist bei der Bereitstellung mit SOAP sehr einfach. Das verwendete Datenmodell hätte das Standard Class Schema erweitern müssen, jedoch ist es sehr wahrscheinlich, dass das Standard Class Schema um die notwendigen Klassen erweitert wird.

8.2 Ausblick

Die modellierten Daten würden ausreichen, um Anwendungen zu unterstützen, mit denen die Benutzer sich im Fakultätsgebäude orientieren können oder Informationen zu bestimmten Räumen zu erhalten. Eine solche Anwendung ist die Anwendung die in [Grzan 2000] beschrieben wurde. Man kann sich hierdurch auch andere Anwendungen überlegen. Beispielsweise könnte man sich Spiele ausdenken bei denen virtuelle Räume verwendet werden.

Für eine abschließende Evaluation des Indoor SpaSe müssen die anderen Komponenten der Nexus-Architektur implementiert werden, um so einen Test unter realen Bedingungen zu ermöglichen. Hierbei werden dann auch die Anforderungen der Komponenten präzisiert. So wird dann das Verhalten der Sensorkomponente bestimmt werden und die Anforderungen an den Spatial Model Server werden genauer definiert werden. Nachdem Nexusanwendungen erstellt worden sind, ist es auch möglich die Art der Anfragen genauer zu bestimmen. So ist es dann auch möglich festzustellen, wie kompliziert AWQL-Anfragen sind. Die beschriebenen Geschwindigkeitsprobleme bei der Erstellung von großen XML-Dokumenten sind wahrscheinlich nicht wirklich ein Problem für Spatial Model Server. Nexusknoten haben, damit aber möglicherweise Probleme. Sie müssen die Ergebnisse mehrere Spatial Model Server und Location Server zusammenfassen. Dies bedeutet, dass sie die ankommenden AWML-Daten einlesen und miteinander verknüpfen müssen und danach wieder in AWML umwandeln. Gibt eine Nexus-Anwendung keine Begrenzung der Objektmenge an, so werden sehr leicht große Objektmengen übertragen.

Eine große Unbekannte in der Nexus-Architektur ist der Event Service. Da sich zum Zeitpunkt der Diplomarbeit der Event Service in einem frühen Stadium befand, waren keine Aussagen über die Anforderungen an die Event Verwaltung zu machen. Seine Anwendungsgebiete werden sich wahrscheinlich erst mit der Zeit ausprägen. Seine Primäraufgabe sollte die Überwachung und Erkennung von räumlichen Events sein. Man kann mit seiner Hilfe aber auch die Konsistenzprobleme innerhalb der Augmented World reduzieren, da Events auf Objekten registriert werden können und so bei Änderungen dieser Objekte andere Systeme benachrichtigt werden.

Ein weiterer wichtiger Bereich für Nexus sind die Kommunikationsmechanismen zwischen den Systemen. Hierbei sind nicht nur die verwendeten Kommunikationsnetze interessant. Die Kommunikation zwischen den einzelnen Diensten benötigt Standards bei den Übertragunsprotokollen und der Verteilung von Objekten oder Diensten. Eine Möglichkeit ist das in der Diplomarbeit verwendete SOAP. Es gibt aber auch andere Möglichkeiten wie die Verwendung von CORBA oder das Remote Method Invocation (RMI) von Java und selbst die Erstellung eigener Protokolle ist denkbar. Welche sich durchsetzen werden und von der Nexus-Plattform verwendet werden, ist noch nicht klar.

Interessanterweise können sehr viele Objekte im Hauptspeicher verwaltet werden. Die Speicherprobleme entstanden in erster Linie durch Ausgabe der Daten und dürften bei Spatial Model Servern nicht vorkommen, da so große Objektmengen nicht übertragen werden sollen. Heutzutage haben Rechner mehrere hundert Megabyte Hauptspeicher. Der Nexus-Server besitzt 4 GB RAM. Würde man diesen Rechner als Spatial Model Server einsetzen, so belegt das Programm und notwendige Software zusammen ca. 200 Megabyte. Wenn nur kleine Objektmengen zurückgeliefert werden sollen, so hat man dann etwa 3,8 GB nur für die Speicherung der Daten zur Verfügung. Geht man davon aus das ein Objekt 500 Byte an Speicher belegt so ließen sich etwas mehr als 7 Millionen Objekte speichern. Bei 100 Byte großen Objekten wären es sogar über 38 Millionen Objekte. Es ließen sich dadurch sogar große Gebiete abdecken. Betrachtet man dies so scheint es wahrscheinlich, dass in Zukunft immer mehr Anwendungsbereiche existieren, deren Daten früher nur auf Sekundärspeicher verwaltet werden konnten und heutzutage komplett in den Hauptspeicher passen. Hierdurch werden Hauptspeicherdatenbanksysteme mehr Anwendungsgebiete besitzen und herkömmliche Datenbanksysteme werden in ihrer Architektur Konzepte und Methoden von Hauptspeicherdatenbanksystemen übernehmen. Auch andere Bereiche bei Datenbanksystemen können durch den großen Hauptspeicher beeinflusst werden. Beispielsweise ist es aus Geschwindigkeitsgründen oft notwendig gewesen Denormaliserungen durchzuführen und so Redundanzen in die Daten einzubauen. Es kann allerdings bei einer großen Hauptspeicherausstattung nun auch wieder sinnvoll sein in eine höhere Normalform zu wechseln, um die Redundanzen zu verlieren und so die ganzen Daten oder zumindest große Teile davon im Hauptspeicher halten zu können. Da die Geschwindigkeit beim Zugriff auf Hauptspeicher sehr viel höher ist, kann die Geschwindigkeit trotz der notwendigen Verknüpfungen der Tabellen sehr viel höher sein als bei einer niedrigeren Normalform.

A Literaturverzeichnis

[AHLERS & ZIEGLER 2001]
Ahlers, Ernst; Ziegler, Peter-Michael
Luftbrücken: USB-Adapter und Basisstationen für die Funkvernetzung
Artikel in c't 18/2001, Verlag Heinz Heise GmbH & Co KG

[BARTELME 2000]
Bartelme, Norbert
Geoinformatik: Modelle, Strukturen, Funktionen
Springer, Berlin, Heidelberg, New York, 3. Auflage, 2000

[BAUER 2001]
Bauer, Martin
INFOBOX: Spatial Events
In: GeoBIT/GIS 9/2001 Oktober

[BONCZ & KERSTEN 1994]
Boncz, Peter; Kersten, Martin
Monet: An Impressionist Sketch of an Advanced Database System
Proc. Basque International Workshop on Information Technology, San Sebastian, Spain, July 1995

[BONCZ, RÜHL & KWAKKEL 1998]
Boncz, Peter; Rühl Tim; Kwakkel Fred
The Drill Down Benchmark
Proceedings of the international Conference on Very Large Data Bases, New York 1998

[CERI, COCHRANE & WIDOM 2000]
Ceri, Stefano; Cochrane, Roberta J.; Widom Jennifer
Practical Applications of Triggers and Constraints: Success and Lingering Issues
In: Proceedings pf the 26th VLDB Conference, Cairo, Egypt, 2000

[CORMEN ET AL 1990]
Thomas H. Cormen, Charles F. Leiserson, Ronald L. Rivest
Introduction to Algorithms
MIT Press, Cambridge, Massachussets, London, 1990/1994

[DE WITT ET AL 1984]
DeWitt, David J.; Katz, Randy H.; Olken Frank; Shapiro, Leonard D.; Stonebraker, Michael R., Wood D.
Implementation Techniques for Main Memory Database Systems
In: Proceedings of the ACM SIGMOD Conference June 1984

[DITTRICH & GATZIU 2000]
Dittrich, Klaus.R; Gatziu, Stella
Aktive Datenbanksysteme: Konzepte und Mechanismen
dpunkt.verlag, Heidelberg, 2. Auflage, 2000

[DJUKNIC & RICHTON 2001]
Djuknic, Goran M.; Richton, Robert E.
Geolocation and Assisted GPS
In: IEEE Computer, Volume 34, Number 2, 2001

[FCC E911]
Homepage der FCC über die erweiterte 911 Notrufnummer
FCC- Enhanced 911 (E911) Home Page
“http;//www.fcc.gov/e911/”, Letzter Besuch 14.1.2002

[FRITSCH, KLINEC & VOLZ 2001]
Fritsch, Dieter; Klinec, Darko; Volz, Steffen
Nexus-Positioning and Data Management Concepts for Location Aware Applications
In: Proceedings of the 2nd international Symposium on Telegeoprocessing, Nice-Sophia-Antipolis, France, 2000

[GARCIA-MOLINA & SALEM 1992]
Garcia-Molina,Hector; Salem, Kenneth
Main Memory Database Systems: An Overview
In: IEEE Transactions on Knowledge and Data Engeneering, Vol.4 No6, December 1992

[GML 2.0]
Cox, Simon; Cuthbert, Adrian; Lake, Ron; Martell, Richard
Geography Markup Language (GML) 2.0: OpenGIS Implementation Specification, 20 February 2001
OGC Document Number: 01-029;
zu finden unter : “http://opengis.net/gml/01-029/GML2.html”

[GESPRÄCH MIT BAUER 2002]
Bauer, Martin
Gespräch mit Martin Bauer am 10.1. 2002

[GROßMANN ET AL 2001]
Großmann, Matthias; Nicklas, Daniela; Schwarz, Thomas; Volz, Steffen
Information Management and Exchange in Nexus
Technical Report, Research Group Nexus, Universität Stuttgart, 2001

[GRZAN 2000]
Grzan, Stjepan
Konzeption und Entwicklung einer mobilen Datenbankanwendung
Studienarbeit Nr. 1794, Fakultät Informatik, Universität Stuttgart, 2000

[HÄRDER & RAHM 1999]
Härder, Theo; Rahm, Erhard
Datenbanksysteme Konzepte und Techniken der Implementierung
Springer, Berlin; Heidelberg; New York; Barcelona, 1999, 1.Auflage

[HOHL ET AL 1999]
Hohl, Fritz; Kubach, Uwe; Leonhardi, Alexander; Rothermel, Kurt; Schwehm, Markus
Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial Aware Applications
In: Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, Washington, USA

[KEE ET AL 2001]
Kee, Changdon; Jun, Haeyoung; Yun, Doohee; Kim, Byunyeon; Kim Youngbaek; Parkinson, Bradford W., Langenstein, Thomas; Pullen, Sam; Shim, Sang-hyun; Shim, Suh; Shim Woo-seong]
Laboratory and Field Test of Indoor Navigation System using Asynchronous Pseudolites
In: International Symposium on Indoor Navigation - INDOORNAV 2001, München, Deutschland, 2001

[KLINEC &VOLZ 2000]
Klinex, Darko; Volz, Steffen
Nexus-Positioning and Communication Environment for Spatially Aware Applications
In: ISPRS Congress 2000, IAPRS, Amsterdam, 2000

[KONISHI & SHIBASAKI 2001]
Konishi Yusuke; Shibasaki, Ryosuke
Development of an Autonomous Personla Positioning System
Symposium on Asia GIS 2001, Tokio, June, 2001

[LEHMAN & CAREY 1986A]
Lehman, Tobin J.; Carey, Michael J.
A Study of Index Structures for Main Memory Database Management Systems
Proceedings of the Twelth International Conference on Very Large Databases, 1986

[LEHMAN & CAREY 1986B]
Lehman, Tobin J.; Carey Michael J.
Query Processing in Main Memory Database Management Systems
ACM SIGMOD Conf, May 1986

[LEHMAN, SHEKITA & CABRERA 1992]
Lehman, Tobin J.; Shekita, J. Eugene; Cabrera, Luis-Felipe
An Evaluation of Starburst's Memory Resident Storage Component
IEEE Transactions on Knowledge and Data Engeneering, Vol.4 No6, December 1992

[LEONHARDI & ROTHERMEL 2001]
Leonhardi, Alexander; Rothermel, Kurt
A Comparison Of Protocols for Updating Location Information
In: Baltzer Cluster Computing Jounal 4, 4

[LEONHARDI, NICU & ROTHERMEL 2002]
Leonhardi, Alexander; Nicu, Christian; Rothermel, Kurt
A Map-based Dead-reckoning Protocol for Updating Location Information
akzeptiert für 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing (IPDPSWPIM 2002), Ft. Lauderdale, FL, USA

[MANEGOLD, BONCZ & KERSTEN 1999]
Manegold, Stefan; Boncz, Peter; Kersten, Martin
Optimizing Main-Memory Join on Modern Hardware
CWI Technical Report INS-R9912, 2001

[MENGEL & HENKEL 2001]
Mengel, Stefan; Henkel, Joachim
Einer Speichert alles: MRAM- der lange Weg zum Universalspeicher
Artikel in c't 18/2001, Verlag Heinz Heise GmbH & Co KG

[MEßMER 2001]
Meßmer, Jens
Modellierung der Augmented World in NExus
Diplomarbeit Nr 1870, Fakultät Informatik, Universität Stuttgart, 2001

[NICKLAS & MITSCHANG 2001]
Nicklas, Daniela; Mitschang, Bernhard
The Nexus Augmented World Model: An Extensible Approach for Mobile, Spatially Aware Applications
In: Proceedings of the 7th International Conference on Object-Oriented Information Systems

[NICKLAS ET AL 2001]
Nicklas, Daniela; Großmann, Matthias; Schwarz, Thomas; Volz, Steffen; Mitschang, Bernhard
A Model-Based, Open Architecture for Mobile, Spatially Aware Applications
In: Proceedings ot the 7th International Symposium on Spatial and Temporal Databases, Juli 200

[REKIMOTO & AYATSUKA 2000]
Rekimoto, Jun; Ayatsuka Yuji
CyberCode: Designing Augmented reality Environments with Visual Tags
In: Proceedings of Designing Augmented Reaility Environments 2000 (DARE 2000), Elsinore, Denmark, 2000

[SEDGEWICK 1992]
Sedgewick, Robert
Algorithmen in C++
Addison Wesley, 1992

[SCHIELE 1999]
Schiele, Gregor
Positionierung von Benutzern innerhalb eines Gebäudes
Studienarbeit Nummer 1739, Fakultät Informatik, Universität Stuttgart, 1999

[TIMESTEN 1999A]
TimesTen Performance Software
TimesTen In-Memory Database Technology A performance Brief
http://www.TimesTen.com/downloads/
TimesTen_Performance_Brief.pdf

[TIMESTEN 1999B]
TimesTen Performance Software
Architected for Real-Time Data Management TimesTen's Core In-Memory Database Technology
http://www.TimesTen.con/downloads/tt_wp.pdf

[VOLZ, FRITSCH & KLINEC 1999]
Volz, Steffen; Fritsch, Dieter; Klinec, Darko
Nexus: Spatial Model Servers for location aware applications on the basis of ArcView
In: Proceedings of the 14th ESRI European User Conference, 1999

[WAHLSTER ET AL 2001]
Wahlster, Wolfgang; Baus, Jörg; Kray, Christian; Krüger, Antonio
REAL: Ein ressourcenadaptierendes mobiles Navigationssystem
In: Informatik Forschung und Entwicklung, November 2001, Vol 16, 4

[YAO & DE JONG 1978]
Yao, S. B., DeJong D.
Evaluation of Database Access Paths
In: Proceedings of the 1978 SIGMOD Conference on the Management of Data, May 1978

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

_____

(Stjepan Grzan)