Projektbericht

Milen Koychev

Indoornavigation mit Nutzung von ortsabhängigen
und adhoc Diensten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Milen Koychev

Indoornavigation mit Nutzung von ortsabhängigen und adhoc Diensten

Ausarbeitung im Rahmen des Master-Projektes
im Studiengang Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer : Diplom. Inform. Birgit Wendholt
Zweitbetreuer: Prof. Dr. rer.nat. Kai von Luck

Abgegeben am 01. März 2007

Milen Koychev

Thema der Ausarbeitung

Indoornavigation mit Nutzung von ortsabhängigen und adhoc Diensten

Stichworte

Indoornavigation, ortsabhängige und adhoc Dienste, Softwareengineering, Softwarearchitektur, Microsoft .NET Framework, Microsoft .NET Compact Framework, C#

Kurzzusammenfassung

Dieser Bericht dokumentiert die Planung, die Durchführung sowie die gesammelten Erfahrungen während der Projektarbeit auf dem Gebiet der Indoornavigation mit Nutzung von ortsabhängigen und adhoc Diensten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung und Motivation

Durch die rasante Entwicklung der Bereiche Hardware und Software werden in den letzten Jahren zahlreiche neune Möglichkeiten für die Informationsverarbeitung und den Informationsaustausch erschaffen. Viele dieser Möglichkeiten finden schnell Anwendung in dem menschlichen Alltag, indem sie den Menschen bei seinen Aufgaben unterstützen und seine Arbeitsweise optimieren. Dabei, um dem Anwender den besten Service anzubieten, werden die neu entstandenen Anwendungsfälle ständig weiterentwickelt und verbessert. Um diese Weiterentwicklung voranzutreiben, wurde im Rahmen des Masterprojektes im Wintersemester 06/07 an der Hochschule für Angewandte Wissenschaften Hamburg die Zielsetzung definiert, auf dem Gebiet der Indoornavigation mit Nutzung von ortanhängigen und adhoc Diensten eigene Erfahrungen zu sammeln, sowie zur Entwicklung dieses Gebietes einen eigenen Beitrag zu leisten. Als Grundlage wurde ein Beispielszenario (s. Kapitel Projektplanung) definiert.

Dieser Projektbericht dokumentiert die Planung und die Durchführung des Projektes sowie die gesammelten Erfahrungen während der Projektarbeit. Die Ergebnisse der Projektdurchführung können für weitere Arbeiten auf dem oben definierten Gebiet benutzt werden.

2 Projektplanung

In diesem Kapitel wird das Beispielszenario des Projektes „Flughafen“ vorgestellt. Aufgrund des Szenarios wird die Systemvision definiert, die Aufgabenverteilung aufgestellt sowie das Risikomanagement beschrieben.

2.1 Szenario

Ein Geschäftsmann aus Hamburg kommt mit dem Taxi zum Flughafen und meldet sich mit seinem PDA[1] beim Flughafen-System an. PDA teilt ihm mit, dass aufgrund einer Reparaturmaßnahme sein Flug nach Genf einige Stunden Verspätung hat. Mit der Hilfe des PDAs setzt der Geschäftsmann seine Geschäftspartner in Genf über die Verspätung in Kenntnis und benachrichtigt den Abholservice. Weiterhin versucht der Hamburger, die Zeit optimal zu nutzen, und sucht auf der Flughafenmesse nach einer für ihn interessanten Veranstaltung. Dabei hat er noch Hunger. Und tatsächlich läuft eine interessante Veranstaltung auf der Flughafenmesse, die er besuchen kann. Außerdem schlägt der PDA mehrere Restaurants, die auf dem Weg zur Messe liegen und dem Profil (dem Geschmack) des Geschäftsmannes entsprechen, vor. Er entscheidet sich für italienische Küche dabei reserviert der PDA einen Tisch. Auf dem Weg zum Restaurant, bekommt der Fluggast über seinen PDA profilbezogene Werbung. Ein Rollex-Angebot ist sehr verlockend und er will sich die Uhr ansehen. So weicht er von der Route ab. Der PDA errechnet umgehend die neue Route, storniert die alte Tischreservierung und bucht einen neuen Tisch für spätere Uhrzeit.

Auf der Messe, wo er glücklich angekommen ist, und schon einige Stände besucht hat, bekommt der Geschäftsmann rechtzeitig eine Benachrichtigung über die neuen Flugdaten (genaue Abflugzeit sowie Abflugterminal und Gate). Von der Messe geht er zu seinem Terminal. Da die Abflugzeit und damit auch die Ankunftszeit bekannt sind, teilt der PDA den Geschäftspartnern in Genf das mit, und bestellt gleich ein Taxi in Genf, da der Abholservice nicht mehr zur Verfügung steht.

Im Flugzeug will der Geschäftsmann seine Verträge und Präsentation zum letzten Mal überprüfen. Er überträgt die Dokumente von seinem PDA auf einen im Sitzt eingebauten Bildschirm, um eine bessere Ansicht auf die Dateien zu bekommen. Weiterhin verläuft die Reise ohne unvorhergesehene Planabweichungen.

Am Ende der Reise ist der Geschäftsmann mit dem Service des Flughafen-Systems sehr zufrieden, da trotzt seines engen Terminplans und überraschender Änderungen seines Reiseverlaufs er seine Ziele ereicht hat und dabei noch weitere interessante Kontakte auf der Messe in Hamburg geknüpft hat.

2.2 Systemvision

Aufgrund des Beispielszenarios wurde die Vision für das Flughafen-System entwickelt. Im Wesentlichen handelt es sich beim Flughafen-System um ein verteiltes Zusammenspielt von mehreren Software- bzw. Hardwarekomponenten.

[1] Mobile Rechner wie Personal Digital Assistants (PDA) oder Mobilfunktelefone

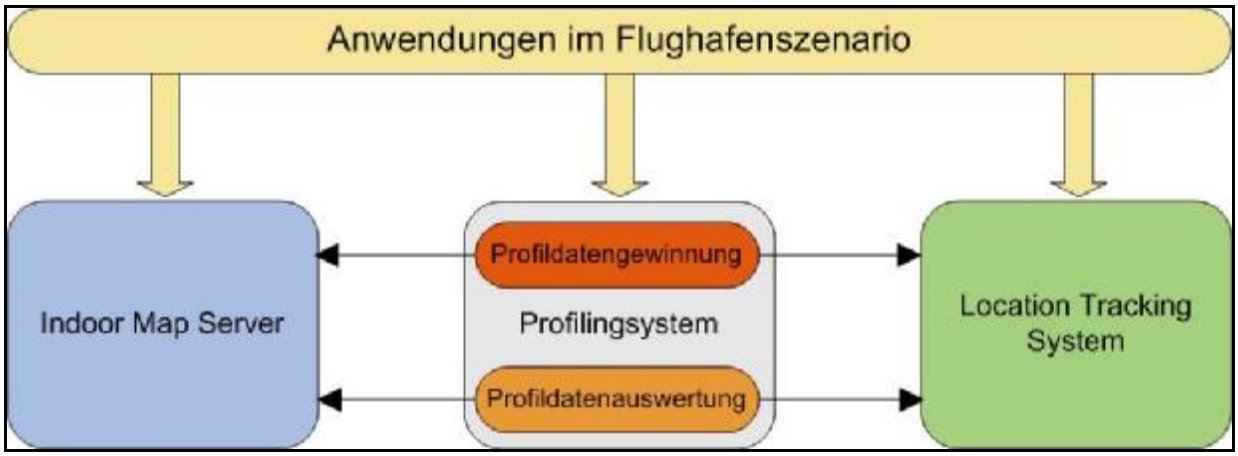

Abbildung 2-1 Grobe Systemarchitektur

Eine grobe Architektur des Flughafen-Systems ist in Abbildung 2-1 dargestellt. Das System soll folgendes ermöglichen:

- **Indoorlokalisierung und Indoornavigation:** Um die beschriebenen Routen zu berechnen sowie die Routeabweichungen zu registrieren, muss das Flughafensystem über eine Location-Tracking-Komponente (s. Abbildung 2-1) verfügen. Diese kann zu jeder Zeit die Position des Benutzers auf dem Flughafengelände feststellen und weiteren Komponenten zur Verfügung stellen.
- **Verwalten und Auffinden von Diensten:** Um das Dienstangebot auf dem Flughafen zu verwalten sowie ortsabhängig Dienste dem Fluggast anzubieten, beinhaltet das System eine Indoor-Map-Komponente (s. Abbildung 2-1). Diese Komponente überträgt die Informationen über vorhandene Dienste auf das Gebäudemodell des Flughafens. Weiterhin bietet diese Komponente Schnittstellen, das Dienstangebot aufgrund der aktuellen Fluggastposition abzufragen.
- **Personalisieren von Diensten:** Für Personalisieren des Dienstangebotes ist die Profiling-Komponente (s. Abbildung 2-1) zuständig. Diese ermöglicht das Personalisieren des Dienstangebotes eines Flughafens sowie die dazu gehörige Verwaltung von personalisierten Daten. Mehr zum Thema Personalisieren von Diensten ist in [Koychev06] zu finden.
- **Interaktion über mobile Geräte:** Die Interaktion mit dem Fluggast, dem Anwender des Systems, erfolgt ausschließlich auf mobilen Geräten (z.B. ultra mobile PC, PDA oder Smartphone).
- **Anbindung von weiteren Anwendungen:** Das Flughafen-System muss die Anbindung von weiteren Anwendungen vorsehen. Diese können spezielle Dienste für das Flughafenszenario zur Verfügung stellen sowie Dienste des Flughafen-Systems in Anspruch nehmen. Eine Anbindung soll mit geringem Aufwand möglich sein.

Aufgrund der definierten Systemvision werden im weiteren Verlauf der Arbeit, die genauen Anforderungen an das System definiert und bewertet.

2.3 Aufgabenverteilung

Nachdem im Projekt eine gemeinsame Systemvision entwickelt wurde, erfolgte die Aufgabenverteilung. Dabei wurde Wert draufgelegt, dass jeder der Projektteilnehmer (Alewtina Schumann, Edyta Kutak, Milen Koychev und Jan Napitupulu) einen spezifischen

Aspekt der Systemvision selbständig bearbeitet, dennoch musste am Ende der Entwicklungsphase das Zusammenspiel der Systemkomponenten sichergestellt werden.

2.3.1 Spezifische Aufgaben

Die spezifischen Aufgaben sind aufgrund der groben Systemarchitektur wie folgt definiert worden:

- **Indor-Map-Komponente:** Die Entwicklung dieser Komponente hat Jan Napitupulu übernommen (s. [Napitupulu07]).
- **Profiling-Komponente:** Die Entwicklung dieser Komponente haben Alewtina Schumann (s. [Schumann07]) und Milen Koychev übernommen.
- **Location-Tracking-Komponente:** Die Entwicklung dieser Komponente hat Edyta Kutak übernommen (s. [Kutak07]).

Da zu der erfolgreichen Durchführung eines Softwareprojektes nicht nur die Entwicklung der einzelnen Komponenten des Systems gehört, sind weitere projektübergreifende Aufgaben definiert worden. Eine Beschreibung dieser folgt im nächsten Unterkapitel.

2.3.2 Projektübergreifende Aufgaben

Um das Flughafen-System realisieren zu können, müssten noch folgende Punkte berücksichtigt werden und als Aufgaben in der Projektplanungsphase den Projektteilnehmern zugewiesen werden:

- **Aufbau der Testumgebung:** Um die Entwicklung und den Test des Flughafen-Systems zu ermöglichen, musste im Rahmen des Projektes eine Testumgebung aufgebaut werden. Diese beinhaltete nicht nur die IT-Landschaft (benötigte Software bzw. Hardware) sondern auch Raumlayout sowie Installation der Indoorpositionssystems (mehr diesem Thema s. [Kutak07]). Diese Aufgabe ist von allen Projektteilnehmern übernommen worden.
- **Auswahl der Technologie:** Bei der Technologieauswahl haben wir uns auf eine gemeinsame Technologie - Microsoft .NET 2.0 (mehr Details zu dieser Teamentscheidung sind im weiteren Verlauf des Berichtes vorgestellt) zur Entwicklung der Systemkomponenten geeinigt, damit das optimale Zusammenspiel zwischen den Komponenten einfach erreicht werden konnte.
- **Technologie-Research:** Um die Entwicklung der fachlichen Aspekte des Systems zu sichern, musste ständige Forschung auf dem Gebiet der .NET-Technologie betrieben werden. Diese wurde von mir während des ganzen Projektes durchgeführt.
- **Definition der genauen technischen Architektur:** Um die Basis für die System-Architektur zu gewährleisten, musste eine genaue technische Architektur entwickelt werden. Dabei ist eine strikte Einleitung zum Einsetzen der .NET-Technologien (z.B. .NET-Webservices, .NET-Webservice-Session-Management usw.) entstanden worden (s. Kapitel Projektrealisierung). Die Entwicklung dieser Architektur wurde eng mit der Forschung auf dem .NET-Gebiet von mir durchgeführt worden.
- **Teamtrainings:** Damit wir alle auf dem gleichen Wissensstand bezüglich der eingesetzten Technologie sind, hat das Team an einem von mir durchgeführten mehrtägigen Workshop zu Themen aus dem Bereich .NET teilgenommen.

Nachdem die Aufgaben im Projekt vollständig definiert wurden, müssten wir Maßnahmen zum Minimieren der schon erkannten Risiken treffen.

2.4 Risikomanagement

Schon in der Planungsphase waren einige der Projektrisiken absehbar. Im Folgenden wird beschrieben, gegen welche Risiken, wir was genau gemacht haben, um den erfolgreichen Projektverlauf zu gewährleisten:

- **Aufbau der Testumgebung:** Die Aufbau der Testumgebung hat sich in so weit als Risiko herausgestellt, weil wir nicht nur die Testumgebung sondern auch ein komplettes Softwarelabor einrichten müssten. Als Grundlage dafür haben wir einen komplett leeren Raum bekommen. Dies hatte als Folge, dass wir zusätzliche Zeit im engen Projektplan vorsehen müssten. Dagegen haben wir mit einem sehr hohem Teameinsatz in den ersten Projekttagen gesteuert und das Risiko erfolgreich auf das Minimum begrenzt.
- **Fehlende Hardware:** Die Serverkomponenten des Systems sowie die Source-Verwaltung sollten auf einem leistungsfähigen BladeCenter (ein Multi-Server-System s. [ibm]) ausgeführt werden. Zum Projektstart wurde nicht bekannt, wann die benötigte Hardware geliefert werden kann. Um dennoch diese Anforderung auch bei der Softwareentwicklung berücksichtigen zu können, haben wir einen provisorischen Server aufgebaut. Die fehlende Hardware wurde als Risiko eingestuft, weil wir schwer abschätzen konnten, welche Auswirkungen eine Migration der Software auf ein Multi-Server-System (wie das BladeCenter) haben kann. Dennoch haben wir uns dazu entschlossen, diese Tatsache bei der Softwarekonzeption zu berücksichtigen, um das Risiko einer Migration zu minimieren.
- **.NET-Erfahrungen:** Da für einige aus dem Team die .NET-Technologie neu war, müssten wir dafür sorgen, dass unser Wissensstand auf einem ähnlichen Niveau ist. Um dieses Risiko zu minimieren, haben wir die regelmäßige Teilnahme an einem Workshop zu Themen aus den Bereichen „Entwicklung mit .NET“ sowie „Entwicklung mit .NET für mobile Geräte“ indem Projektplan vorgesehen.

Durch das noch am Anfang durchgeführte Risikomanagement könnten wir die ersten Projektrisiken erkennen und eingrenzen.

2.5 Fazit

Mit der Definition des Projektszenarios, der Aufstellung der Systemvision, der Aufgabenverteilung sowie dem Risikomanagement haben wir schon in der Anfangsphase eine gute Basis für die erfolgreiche Durchführung des Projektes geschaffen. Die Ziele wurden klar definiert und der Start der Realisierung wurde somit gegeben.

3 Projektrealisierung

In diesem Kapitel wird die Projektrealisierung beschrieben. Die Entscheidungen werden dargestellt und begründet. Die wesentlichen Projekt-Erfahrungen werden ausführlich dokumentiert.

3.1 Technologieauswahl - .NET vs. Java

Zur Realisierung des Flughafen-Systems kamen zweit Technologien in Frage: Java von Sun und .NET 2.0 von Microsoft. Aus folgenden Gründen haben wir uns für die .NET-Technologie von Microsoft entscheiden:

- **.NET Entwicklungswerkzeuge:** Neben der .NET-Technologie bietet Microsoft eine sehr umfangreiche und gute Entwicklungsumgebungen (z.B. Microsoft Visual Studio 2005). Damit lässt sich Software schnell und einfach nicht nur für den Desktoprechner sondern auch für mobile Geräte entwickeln.
- **Softwarelizenzen:** Über Microsoft Developer Network - Academic Alliance (MSDNAA) (s. [msdnaa]) hatten wir die Möglichkeit, die weiteren Systemkomponenten wie z.B. Betriebsysteme und Datenbanken kostenlos von Microsoft zu erhalten.
- **Herausforderung:** Da einige aus unserem Team sich noch nicht mit der .NET-Technologie auseinandergesetzt haben, war der Wunsch da, sich einer neuen Herausforderung zu stellen und etwas Neues auszuprobieren.

Durch die Auswahl der .NET-Technologie für die Realisierung des Systems haben wir auch die komplette IT-Landschaft definiert. Wir haben uns nur auf Produkte der Firma Microsoft begrenzt, was den Vorteil brachte, dass alle Komponenten (z.B. Betriebsysteme, Datenbanken, Webserver usw.) möglichst optimal aufeinander abgestimmt sein sollten.

3.2 Technische Architektur

Bei der Definition der technischen Architektur musste die Frage geklärt werden, welche konkreten Techniken der .NET-Technologie und wie genau diese eingesetzt werden können, um die Systemanforderungen zu erfüllen.

Bei dem Flughafen-System handelt es sich um ein verteiltes System (s. Abbildung 3-1). Gewisse Systemkomponenten werden für die Interaktion mit dem Fluggast auf seinem mobilen Gerät eingesetzt, andere Komponenten müssen in einer zentralen Stelle (auf einem Application-Server oder einem Cluster von Application-Servern) die Planungs- und Routing-Aufgaben des Flughafen-Systems übernehmen. Weiterhin werden die Anwenderdaten nicht auf den mobilen Geräten sondern in einer zentralen Datenbank bzw. Datenbank-Cluster dauerhaft gespeichert.

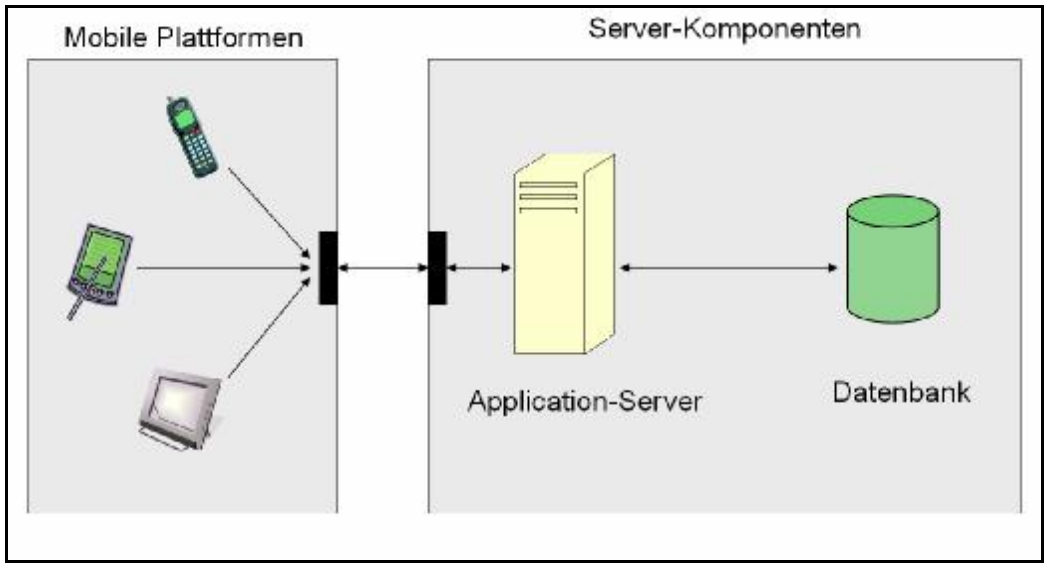

Abbildung 3-1 Systemverteilung

3.2.1 Interprozess-Kommunikation

Um eine Verteilung des Flughafen-Systems zu ermöglichen, musste eine Technologie zum Datenaustausch zwischen den Komponenten einsetzen werden. Die Microsoft .NET-Plattform bietet zwei Technologien zur Interprozess-Kommunikation -.NET-Webservices (s. [microsoft.webservices]) und .NET-Remoting (s. [microsoft.remoting]).

.NET-Remoting bietet Objektidentität ähnlich wie CORBA (Common Object Request Broker Architecture) (s. [omg]) und benötigt keine weitere Software, um eingesetzt werden zu können. Dennoch ist diese Technologie kein Standard und kann nur innerhalb von Microsoft .NET eingesetzt werden, was die Anbindung von weiteren Systemen an das Flughafen-System erschweren würde.

.NET-Webservices implementieren den Webservice-Standard (s. [w3c]) plus spezielle Erweiterungen. Dadurch können .NET-Webservices als interoperable Kommunikationsplattform eingesetzt werden.

Da unser System Schnittstellen für weitere Systeme im Flughafen-Szenario anbieten muss, haben wir uns für .NET-Webservices als Interprozess-Kommunikation entschieden. Der Einsatz von Webservices hat den Nachteil, dass unsere Server-Komponenten, die Schnittstellen anbieten, zustandslos sind. Das bedeutet, dass eine Server-Komponente im Normalfall keine Daten zwischen zwei Aufrufen der Webservice-Schnittstelle speichern kann. Weiterhin benötigen die .NET-Webservices den Microsoft Internet Information Server (IIS) als Webservice-Ablaufsumgebung.

Um die Zustandslosigkeit zu umgehen, bieten die .NET-Webservices Session-Management an. Dadurch wird für jeden Konsumenten eines Webservice-Dienstes im IIS ein Session-Objekt erstellt und verwaltet. Somit hat die Server-Komponente die Möglichkeit, wichtige Daten zwischen den Service-Aufrufen in das Session-Objekt zu speichern bzw. nach einem Service-Aufruf aus dem Session-Objekt auszulesen. Das Session-Management wird über

einen Schalter in der Webservice-Definition aktiviert. Es benötigt keinen weiteren Programmieraufwand.

Nachdem das Session-Management bei .NET-Webservices aktiviert worden ist, muss die Frage gestellt werden, wie sich das Session-Management bei einem Multi-Server-Scenario verhalten wird.

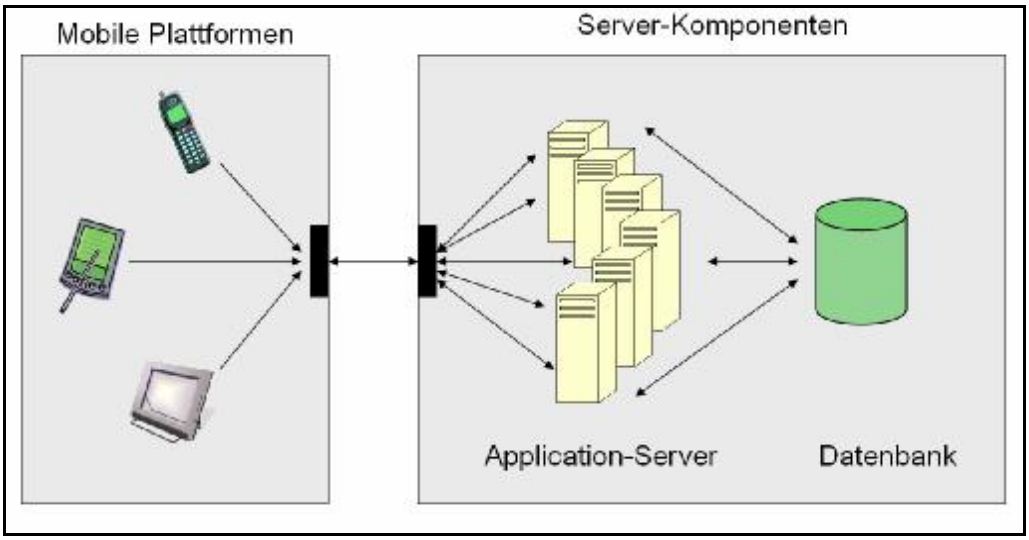

Abbildung 3-2 Multi-Server-Verteilung

Ziel eines Multi-Server-Szenarios ist, die Last der einzelnen Application-Server, auf denen die Geschäftslogik verteilt ist, zu balancieren. Dies hat den Nachteil, dass es nicht sicher gestellt ist, dass ein Service-Konsument innerhalb seiner Transaktion immer die Logik auf demselben Application-Server anspricht. Somit wären das Session-Management in dieser Form unbrauchbar und das Multi-Server-Szenario ein unbeherrschbares Risiko. Um dagegen zu steuern bietet der IIS folgende zwei Technologien an:

- **Session-Management durch zusätzlichen Server:** Alle Application-Server bilden einen Cluster und haben somit ein serverübergreifendes Session-Management. Dabei übernimmt ein Application-Server das Session-Management des Clusters. Das kann ein zusätzlicher Server oder ein der schon im Cluster vorhandenen Server sein.
- **Session-Management durch Datenbank:** Bei dieser Lösung wird die Verwaltung der Session-Objekte des Clusters von einer Datenbank oder einem Datenbank-Cluster übernommen, indem Session-Objekte in die Datenbank speichert bzw. aus der Datenbank ausgelesen werden. Im Vergleich zu Session-Management durch zusätzlichen Server ist dieses Verfahren durch die Lese- bzw. Schreiboperationen langsamer, dennoch bleiben die Session-Objekte auch nach einem Neustart des System erhalten und der Anwender kann weiter ohne Unterbrechungen mit dem System arbeiten.

Durch die oben beschriebenen Technologien kann der Multi-Server-Einsatz des Flughafen-System im Vorfeld vorbereitet werden. Dabei müssen keine besonderen Vorkehrungen bei der Softwareentwicklung getroffen werden. Das Multi-Server-Szenario bleibt für die Anwendung durch den IIS-Server verborgen. Die einzelne Bedingung, die erfüllt werden muss, ist dass das Session-Management bei den .NET-Webservices aktiviert ist. Der IIS selbst muss für das Multi-Server-Szenario konfiguriert werden. Dabei hält sich der Konfigurationsaufwand in Grenzen von einigen Stunden.

3.2.2 Datenbank und Datenbankzugriffsschicht

Als Datenbank haben wir für das Flughafen-System den Microsoft SQL Server 2005 vorgesehen. Der Microsoft SQL Server 2005 kann auch im Multi-Server-Szenario eingesetzt werden, indem aus mehreren SQL-Servern ein Cluster gebildet wird. Dabei wird dieser Cluster von der Anwendung, die den Datenbankdienst konsumiert, als einzelner Server wahrgenommen.

Bei der Konzeption der Datenbankzugriffschicht wurde die Möglichkeit, den OR-Mapper OpenAccess von der Firma Vanatec (s.[vanatec]) einzusetzen, in Betracht gezogen. Dabei bietet dieser OR-Mapper Unabhängigkeit von der eingesetzten Datenbanksoftware und verbirgt die relationale Welt einer Datenbank, indem das Objekt-Modell der Anwendung automatisch in ein relationales Modell umgewandelt wird. Ein solcher OR-Mapper spart bei komplizierten Objekt-Modellen Entwicklungszeit, dennoch beanspruch er auch einige Ressourcen und muss im Vorfeld konfiguriert werden. Da unser Objekt- sowie Datenbank-Modell sehr einfach sind, haben wir auf den Einsatz eines OR-Mapper verzichtet und eine schlanke Datenbankzugriffschicht selbst konzipiert.

3.2.3 Zusammenfassung der notwendigen Softwarekomponenten

Durch die Definition der technischen Architektur wurde die folgende Liste der notwendigen Softwarekomponenten aufgestellt

- **Betriebsysteme:**
  - **Server:**
    - Microsoft Windows Server 2003 Service Release 2
  - **Mobile Geräte:**
    - Microsoft Windows 2000;
    - Microsoft Windows XP;
    - Microsoft Windows Mobile 2005
- **Datenbank:**
  - Microsoft SQL Server 2005
- **Application-Server (Web-Server):**
  - Microsoft Internet Information Server 6.0
- **NET.Framework:**
  - **Desktop und Server:**
    - Microsoft .NET Framework Version 2.0
  - **Mobile Geräte**:
    - Microsoft .NET Compact Framework Version 2.0
- **Entwicklungsumgebung**:
  - Microsoft Visual Studio 2005 / C#
  - Windows Mobile 5.0 Pocket PC SDK
  - Microsoft Device Emulator Version1.0
  - Microsoft ActiveSync 4.0

3.3 System-Architektur

Nachdem die technische Architektur definiert worden war, haben wir die System-Architektur aufgestellt (s. Abbildung 3-3 und Abbildung 3-4).

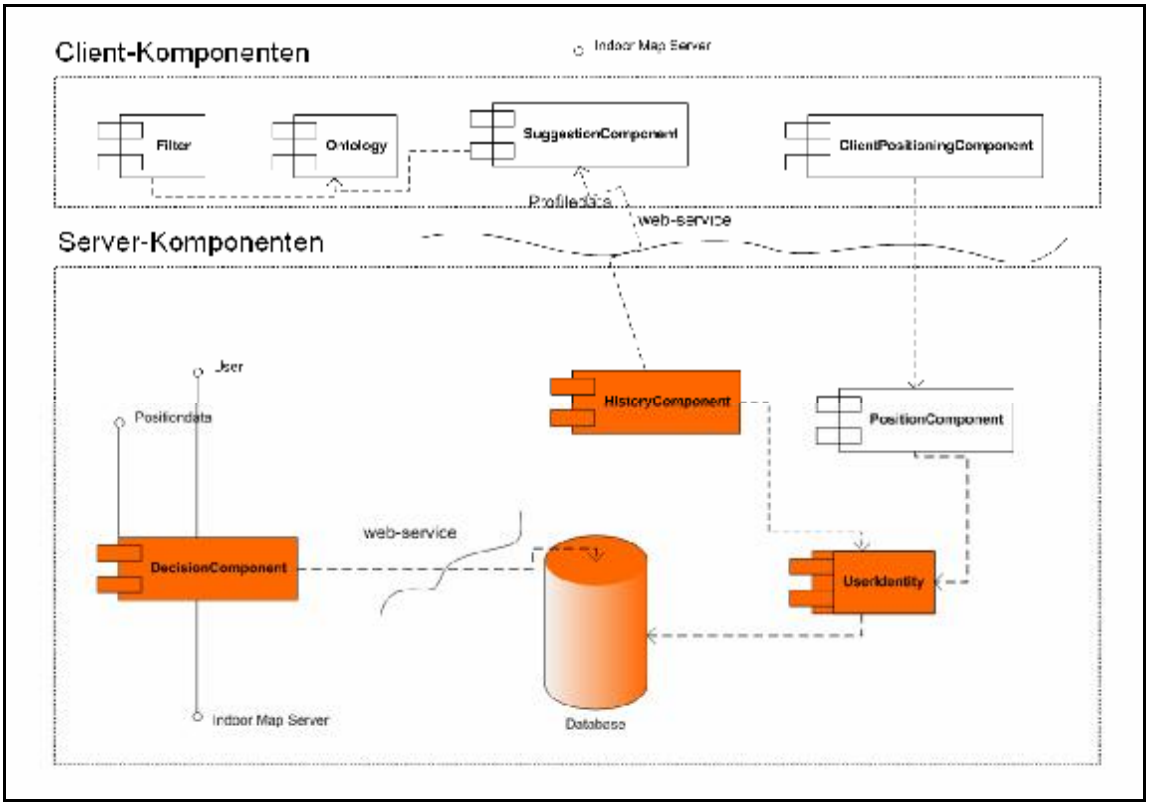

Abbildung 3-3 System-Architektur

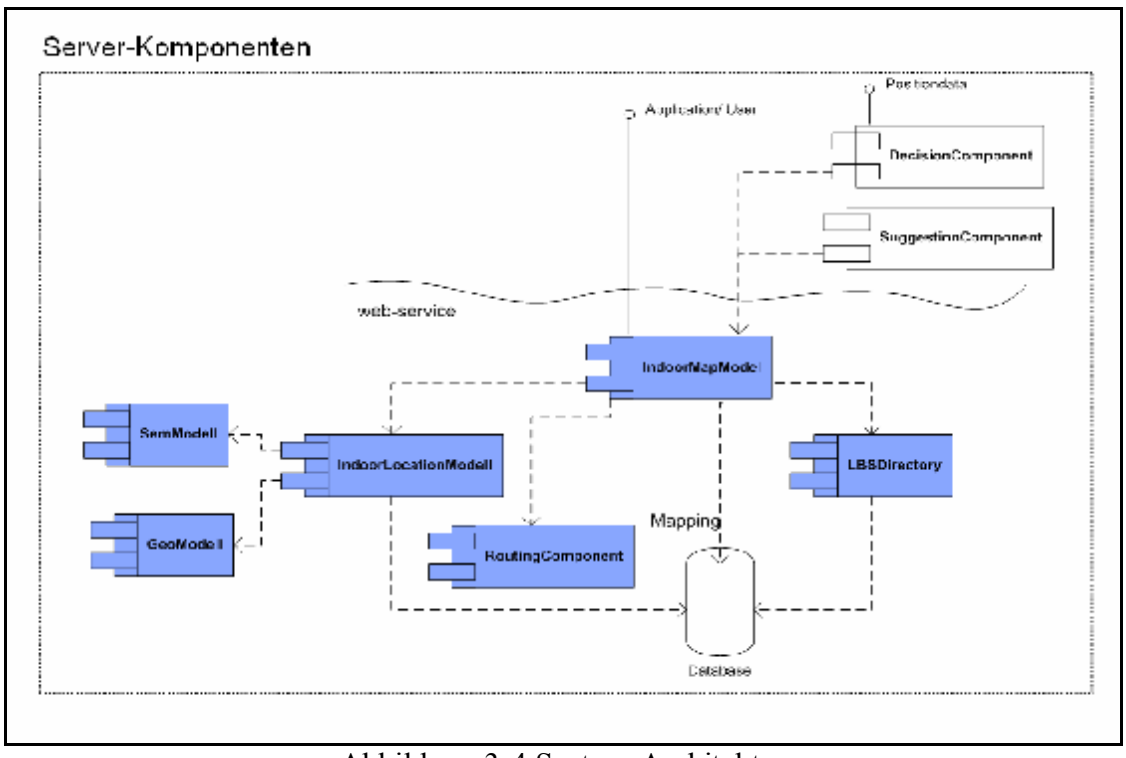

Abbildung 3-4 System-Architektur

Bei der Architektur gibt es zwei Komponentengruppen – die Client-Komponenten und die Server-Komponenten. Die Client-Komponenten sind solche, die auf den mobilen Clients (PDA, Smartphone, ultra mobile PCs usw.) zur Ausführung kommen. Die Server-Komponenten werden auf dem Application-Server bzw. im Cluster ausgeführt. Die Interprozess-Kommunikation findet über .NET-Webservices statt.

Folgende System-Komponenten wurden von mir konzipiert und entwickelt:

- **UserIdentity-Komponente**: Da unser System mit mehreren Benutzern agiert, musste im System Benutzeridentität vorsehen werden. Der Kontext jedes aktiven Benutzers muss ständig bekannt sein. Die erste Version dieser Komponente implementierte die Benutzeridentität im Flughafen-System aufgrund des Session-Managements der .NET-Webservices. Dabei wurde der Kontext der aktiven Benutzer in Session-Objekte vom IIS gespeichert. Diese Lösung war sehr schnell realisiert und hat bei den ersten System-Tests zuverlässig funktioniert. Im weiteren Verlauf des Projektes war unser mobiler Client unter Microsoft .NET Compact Framework für Microsoft Windows Mobile 2005 fertig gestellt. Wir haben festgestellt, dass das Session-Management bei System-Anfragen vom mobilen Client nicht funktionierte. Nach einer Reihe von Untersuchungen wurde der Grund bestimmt. Das Session-Management bei .NET-Webservices ist eine .NET spezifische Erweiterung und gehört nicht zum Webservice-Standard. Weiterhin implementieren die .NET-Webservices unter .NET Compact Framework für Microsoft Windows Mobile nur den Webservice-Standard und baten keine .NET spezifische Erweiterungen an. Dabei wurde uns klar, dass wir relativ spät im Projekt einen Fehler bei der technischen Architektur festgestellt haben. Es gab zwei Lösungsansätze, um diesen zu beheben:
    - **OpenNETCF Framework:** Das OpenNETCF Framework (s. [opennetcf]) für Microsoft Windows Mobile bietet Erweiterungen der .NET-Webservices für mobile Geräte. Eine diese Erweiterungen ist auch das Session-Management.
    - **Eigene Implementierung:** Die andere Möglichkeit war, auf das eingebaute Session-Management zu verzichten und ein eigenes zu implementieren. Um innerhalb des Webservice-Standards zu bleiben, müssten die Daten einer Session in die Datenbank manuell gespeichert und aus der Datenbank manuell ausgelesen werden, da in diesem Fall die .NET-Webservices zustandslos sind.

  Nach einigen fehlgeschlagenen Tests mit dem OpenNETCF Framework haben wir uns für eine eigene Implementierung des Session-Managements entschieden und diese als Teil der UserIdentity-Komponente realisiert. Dadurch wurde auch gewährleistet, dass die Schnittstellen des Flughafen-Systems ausschließlich nur den Webservice-Standard implementieren.
- **History-Komponente:** Diese Komponente ist dafür zuständig, historisierte Benutzerdaten (wie z.B. Dienste, die der Benutzer in Anspruch genommen hat) den weiteren Komponenten über eine Webservice-Schnittstelle zur Verfügung zu stellen. Die Daten werden aus der Datenbank der Anwendung ausgelesen.
- **Decision-Komponente:** Um festzustellen, welcher Dienst von welchem Benutzer in Anspruch genommen wurde, mussten wir eine Entscheidungskomponente vorsehen, die genau diese Aufgabe übernimmt. Es gibt mehrere Wege, um festzustellen, dass ein Dienst konsumiert wurde (z.B. durch Überwachung des Zahlungsverkehrs auf dem Flughafengelände). Die Decision-Komponente in unserem System wertet die Bewegungsprofile der Benutzer, die von der Location-Tracking-Komponente in die Datenbank gespeichert werden, aus. Wenn sich ein Benutzer in der Nähe eines Dienstes über eine bestimmte Zeit aufgehalten hat, nimmt die Komponente an, dass der Benutzer diesen Dienst konsumiert hat und speichert diese Information in die

Datenbank der Anwendung. Die Informationen, in welchem Umkreis der Benutzer sich befinden muss und wie lange er sich in der nähe eines Dienstes befinden muss, um den Dienst als konsumiert zu bezeichnen, werden von den Indoor-Map-Komponenten festgelegt. Da diese Decision-Komponente keine Benutzerinteraktionen anbieten, ist sie als ein Batch-Prozess konzipiert worden, der in bestimmten Zeitabständen ausgeführt wird. Um eine bessere Lastverteilung zu gewährleisten, ist die Decision-Komponente von den anderen Server-Komponenten durch einen .NET-Webservice entkoppelt. Somit kann diese Komponente auf einem anderen Application-Server ausgeführt werden.

- **Datenbank-Komponente:** Bei dieser Komponenten handelt es sich um eine schlanke Datenbankzugriffsschicht. Diese Kapselt die relationale Welt der Datenbank in sich und bietet allen anderen Komponenten Objekt-Schnittstellen für den Datenbankzugriff.

Während der Entwicklung der eigenen Komponenten musste man sich oft auch mit der Entwicklung oder Optimierung der Komponenten anderer Projektteilnehmer auseinandersetzen, somit konnte das Zusammenspiel aller Systemteile sichergestellt werden. Am Ende der Entwicklung hatte jeder das Fachwissen über seine Komponenten und den Einblick in alle weiteren System-Teile.

4 Zusammenfassung und Ausblick

Im Rahmen des Master-Projektes „Flughafen“ im Wintersemester 06/07 an der Hochschule für Angewandte Wissenschaften Hamburg haben wir ein Prototypsystem für Indoornavigation mit Nutzung von ortsgebundenen und adhoc Diensten entwickelt. Das Prototypsystem konnte am 01.02.2007 vor Publikum erfolgreich vorgeführt werden.

Durch die Prototypentwicklung wurden Technologien und Verfahren auf die Probe gestellt. Dabei ist festgestellt worden, dass unsere technische Architektur mit einigen Änderungen für ein solches System im reellen Einsatz tragfähig ist. Weiterhin muss die Location-Tracking-Komponente des Flughafen-System noch weiter entwickelt werden, da wir diese Komponente aus Zeitgründen nur für Microsoft Windows XP bereitgestellt haben und nicht auf Microsoft Windows Mobile portieren konnten. Bei dieser Portierung sehen wir nach unseren Untersuchungen keine technischen Schwierigkeiten. Die Hardware zur Positionsbestimmung muss weiter entwickelt werden, um bessere Positionsdaten liefern zu können.

Die Ergebnisse des Projektes „Flughafen“ werden als Grundlage für eine geplante Master-Arbeit mit dem vorläufigen Titel „Simulationsumgebung für Indoornavigation mit Nutzung von ortsgebundenen und adhoc Diensten“ im Sommersemester 2007 an der Hochschule für Angewandte Wissenschaften Hamburg dienen.

Literaturverzeichnis

[Koychev06]
Milen Koychev, „Ausarbeitung: Personalisieren von Diensten“, SS2006, Haw-Hamburg

[Kutak07]
Edyta Kutak, „Projektbericht“, WS06-07, Haw-Hamburg

[Napitupulu07]
Jan Napitupulu, „Projektbericht“, WS06-07, Haw-Hamburg

[Schumann07]
Alewtina Schumann, „Projektbericht“, WS06-07, Haw-Hamburg

[ibm]
BladeCenter-Specifikation, http://www-03.ibm.com/systems/de/bladecenter/ , (Stand 02.2007)

[msdnaa]
Microsoft Developer Network - Academic Alliance (MSDNAA), http://www.msdnaa.de/, (Stand 02.2007)

[microsoft.webservices]
.NET-Webservices, http://www.microsoft.com/germany/msdn/library/xmlwebservices/AllgemeinDasSindWebServices.mspx , (Stand 02.2007)

[microsoft.remoting]
.NET-Remoting, http://www.microsoft.com/germany/msdn/library/xmlwebservices/AllgemeinDasSindWebServices.mspx , (Stand 02.2007)

[omg]
Common Object Request Broker Architecture (CORBA), http://www.omg.org/gettingstarted/corbafaq.htm , (Stand 02.2007)

[opennetcf]
The Premier .NET Compact Framework Shared Source Site, http://www.opennetcf.org/home.ocf , (Stand 02.2007)

[vanatec]
Object-relational Mapping for .NET, http://www.vanatec.com/en , (Stand 02.2007)