Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Leif Hartmann, Jan Schönherr

Pervasive Spine
Ein Framework für mobile Anwendungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mobile Geräte, wie Mobiltelefone oder PDAs werden immer leistungsfähiger. Dabei werden nicht nur die Prozessoren schneller, die Geräte stellen auch einen immer größeren Funktionsumfang bereit, wie z. B. die Verwendung von Standortinformationen, Bluetooth und WLAN. Diese vielfältigen Funktionen bieten interessante Möglichkeiten in den Bereichen Pervasive Computing und Pervasive Gaming.

Im Rahmen eines Projekts des Masterstudiengangs Informatik 2007/2008 an der HAW Hamburg beschäftigten sich die Projektmitglieder mit der Entwicklung eines Frameworks für pervasive Anwendungen auf mobilen Geräten. Das Projekt sah vor, die einzelnen Interessen der Projektmitglieder in einem Kontext zusammenzubringen. Location-Based-Services, Pervasive Gaming, die Frameworkentwicklung und ein generelles Interesse an mobilen Geräten waren ausschlaggebend bei der Definition des Projektziels.

1.1 Projektziel

Das Projektziel sah vor, Erfahrungen im Bereich von mobilen Anwendungen zu erlangen und Ideen für Themen von Masterarbeiten zu sammeln, die im Anschluss entstehen sollen. Hierbei stand nicht die Entwicklung einer konkreten Anwendung im Vordergrund, sondern die Erstellung eines Frameworks. Für Entwickler, die das Framework nutzen, soll der Prozess der Anwendungsentwicklung beschleunigt und erleichtert werden.

1.2 Projektablauf

Nachdem ein grundsätzlicher Konsens für das Projektziel gefunden war, folgte eine Evaluationsphase (siehe Kapitel 3), in der die technischen Möglichkeiten der vorhandenen Geräte untersucht sowie Ideen für Anwendungen gesammelt wurden. Bei den Ideen für Anwendungen wurde beschlossen, sich dem Thema spielerisch zu nähern und einen Fokus auf pervasive Spiele zu legen.

Danach beschäftigte sich ein Teil der Projektmitglieder mit der Entwicklung von zwei prototypischen Beispielanwendungen, die dem Prozess der Ideenfindung entsprangen (siehe Kapitel 4). Es entstand ein lokationsabhängiger Instantmessenger und ein „schnitzeljagdähnliches Spiel. Ein anderer Teil arbeitete an der Umsetzung des Frameworks (siehe Kapitel 5) selbst. Die Beispielanwendungen dienten der Validierung des Frameworks und lieferten Anregungen für Funktionen, die das Framework abdecken sollte.

Der Schwerpunkt dieses Berichts liegt – neben einer allgemeinen Projektbeschreibung – auf den Tätigkeiten der Autoren während des Projekts. Dazu wird im Kapitel 4 die von den Autoren realsisierte

Beispielanwendung betrachtet. Der Fokus in Kapitel 5 liegt auf den Komponenten Persistenz und Karte. Abschließend wird in Kapitel 6 der entwickelte Server beschrieben, dessen Anforderungen sich aus den Anforderungen der Beispielanwendungen ableiten.

Kapitel 2

Grundlagen

In diesem Kapitel werden grundlegende Techniken und Technologien beschrieben, die im Projekt Verwendung fanden. Außerdem erfolgt eine Eingrenzung des Frameworkbegriffs.

2.1 J2ME

Die Java 2 Micro Edition[1] (J2ME) wurde 1999 von Sun vorgestellt, um auf die Besonderheiten von mobilen Geräten und Embedded-Devices einzugehen. Sie bietet eine Java-Plattform für eine große Anzahl an heterogenen Geräten.

Um mit der Diversität der Geräte umzugehen, wurde ein Schichtenmodell für Spezifikationen erstellt. Dabei gehen die Spezifikationen der Konfigurationsschicht auf die Virtuelle Maschine und die generelle Java API ein, die auf einer großen Gruppe von Geräten laufen. Darüber befinden sich die Spezifikationen der Profilschicht. Diese definieren die besonderen Bedürfnisse an die Java API für eine spezielle Untermenge von Geräten, wie etwa den Mobiltelefonen. Eine Spezifikation der Profilschicht, zusammen mit einer Spezifikation der Konfigurationsschicht, bietet eine vollständige Umgebung für eine Java-Applikation auf einem mobilen Gerät bzw. einem Embedded-Device (Siehe Abbildung 2.1). Die Spezifikationen werden im „Java Community Process“ (JCP) von Sun in Form von „Java Specifikation Requests“ (JSR) organisiert und können in [Sun Microsystems (2008)] nachgelesen werden.

In diesem Projekt wurde mit Mobiltelefonen gearbeitet. Für die Gruppe der Mobiltelefone wurde die Profilspezifikation „Mobile Information Device Profile“ (MIDP) entworfen, die auf der Konfigurationsspezifikation „Connected Limited Device Configuration“ (CLDC) basiert. Vgl. [White (2001)]. Die im Projektbericht folgenden Ausführungen über J2ME basieren auf diesen beiden Spezifikationen.

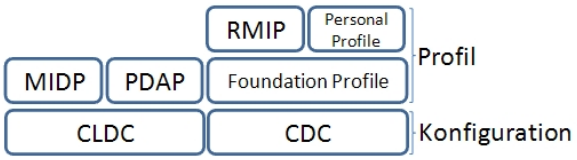

Abbildung 2.1: J2ME: Konfigurationen und Profile. Angelehnt an [White (2001)]

[1] http://java.sun.com/javame/index.jsp

2.2 Framework

Da das Ziel des Projekts die Entwicklung eines Frameworks war, ist die Definition des Begriffs Framework sowie dessen Abgrenzung zu einer Bibliothek wichtig. In [Gamma u. a. (2004)] wird der Begriff Framework wie folgt definiert:

> A framework is a set of coorperating classes that makes up a reusable design for a specific class of software. [...] You can customize a framework to a particular application by creating application-specific subclasses of abstract classes from the framework.
>
> The framework dictates the architecture of your application. It will define the overall structure of its partitioning into classes and objects, the key responsibilities thereof, how the classes and objects collaborate, and the thread of control.

Ein wichtiger Aspekt hierbei ist, dass ein Framework bei der Strukturierung der Anwendung hilft, indem es die Architektur vorgibt. Dies unterscheidet ein Framework im Wesentlichen von einer Bibliothek. Eine Bibliothek bietet Funktionalitäten für Entwickler an, gibt aber keine Struktur vor.

2.3 Lokalisierung

Das Global Positioning System (GPS) wurde ursprünglich vom US Militär entwickelt und diente Verteidigungszwecken. Es basiert auf einem System von 24 Satelliten, die sich im Orbit der Erde befinden. Ein GPS-Empfänger kann unter der Voraussetzung, dass mindestens drei Satelliten sichtbar sind, seine Position durch die Estimated-Time-Of-Arrival-Methode[2] bestimmen. Die Genauigkeit liegt bei etwa 15 Metern[3], es können aber auch höhere Genauigkeiten erreicht werden. Das Positionieren per GPS ist ein kostenloser Dienst, und es bedarf lediglich der Hardware, um die GPS-Signale zu empfangen. Solche GPS-Empfänger halten mehr und mehr Einzug in Mobiltelefone, und die Kosten dafür sinken.

GPS hat allerdings auch einige Nachteile, die eine Verwendung einschränken; denn der Benutzer muss eine freie Sichtlinie zu den Satelliten haben, damit GPS funktioniert. Das ist in urbanen, bebauten Gebieten und im Gebäudeinneren nicht der Fall.

Indoor-Lokalisierungssysteme sind Systeme, bei denen der Benutzer mit Hilfe seines mobilen Geräts mit Objekten oder Systemen interagiert. Diese Objekte und Systeme können eine bekannte Position relativ zur Mobilfunkinfrastruktur oder eine relative Position zu einem Point-Of-Interest haben. Aus diesem Wissen lässt sich die Position des Benutzers bestimmen, sobald eine Interaktion stattfindet.

Bei der Interaktion mit Objekten gibt es unter anderem zwei Möglichkeiten: zweidimensionale Barcodes und RFID-Tags. Die zweidimensionalen Barcodes gibt es in verschiedenen Ausführungen: Quick-Response-Codes[4], Sema-Codes[5] und Shot-Codes[6]. Allen gemein ist, dass sie Informationen wie zum Beispiel Internetadressen enthalten können. Auf den über das Mobilgerät abgerufenen Internetseiten können dann weiterführende Informationen angeboten werden. Eine Kamera muss im Mobilgerät vorhanden sein, um ein Bild des Barcodes zu machen und dieses dann auszuwerten.

Eine ausführlichere Diskussion über Lokalisierungsformen und deren Eignung für mobile Anwendungen bzw. Spiele kann [Schönherr (2007)] entnommen werden.

[2] http://www.kowoma.de/gps/Positionsbestimmung.htm
[3] http://www.kowoma.de/gps/Genauigkeit.htm
[4] http://www.denso-wave.com/qrcode/index-e.html
[5] http://semacode.com/
[6] http://www.shotcode.com/

Kapitel 3

Evaluationsphase

Am Anfang des Projekts stand eine Evaluationsphase, während der zum einen die Hard- und Softwareplattform ausgewählt wurde und zum anderen verschiedene Funktionen der Testgeräte evaluiert wurden. Die Ergebnisse der von den Autoren durchgeführten Evaluationen sowie die Wahl der Hard- und Softwareplattform, sind im Folgenden erläutert.

3.1 Wahl der Hardware- und Softwareplattform

Auf mobilen Geräten sind zwei Technologien relativ weit verbreitet: Zum Einen das von Microsoft entwickelte .NET-Framework[7] und zum Anderen Suns Java Micro Edition. Da alle Projektmitglieder bereits Erfahrung in Java mitbrachten, fiel die Entscheidung auf J2ME.

Bei dem verwendeten Testgerät handelte es sich um das Nokia E70[8] Smartphone. Da dieses Gerät keinen eigenen GPS-Empfänger integriert hat, wurde hierfür ein separater GPS-Empfänger (Holux GPSlim 236[9]) via Bluetooth angesprochen.

3.2 Persistenz

J2ME bietet ein RecordStore-System zum Speichern und Laden von Daten an. Auf einen RecordStore kann über einen eindeutigen Schlüssel (String) zugegriffen werden. Innerhalb eines RecordStores werden die Daten als Byte-Arrays unter ganzzahligen Indices abgelegt. Zwar ist der direkte Zugriff auf RecordStores relativ performant, jedoch bringt es Umstände für Entwickler mit sich:

- Um die Umwandlung der zu speichernden und zu ladenden Daten in Byte-Arrays, müssen sich die Entwickler selbst kümmern.
- Die Handhabung der RecordStores ist zum Teil umständlich. So gibt es beispielsweise keine Methode, die abfragt, ob ein RecordStore bereits existiert. Erst beim Zugriff auf einen nicht existierenden RecordStore, kann dies durch eine Exception herausgefunden werden.

[7] http://msdn.microsoft.com/netframework/

[8] http://web.nokia.de/de/mobiltelefone/modelluebersicht/e70/startseite/184914.html

[9] http://www.holux.com/JCore/en/products/products_content.jsp?pnoY340

Floggy

Um Entwicklern den Zugriff auf persistente Daten zu vereinfachen, wurden verschiedene bestehende Lösungen evaluiert. Ein viel versprechendes Projekt in diesem Bereich ist Floggy[10], ein Framework, das Entwicklern einen Datenbank-ähnlichen Zugriff auf gespeicherte Objekte ermöglicht. Es besteht aus zwei Modulen:

Framework Das Framework ist für den eigentlichen Zugriff auf die persistenten Objekte zuständig. Es bietet eine leichtgewichtige API, über die nicht nur einzelne Objekte gespeichert, geladen und gelöscht werden können, sondern auch Listen von Objekten sortiert und gefiltert werden können.

Weaver Der Weaver wird in den Build-Prozess der J2ME-Anwendungen eingebunden. Er analysiert den Java-Bytecode und reichert ihn mit Code zum Persistieren an.

Bewertung

Floggy macht grundsätzlich einen guten Eindruck und macht den Zugriff auf persistente Daten deutlich komfortabler. Bei einigen Tests stellte sich jedoch heraus, dass sich der Weaver – trotz eines von den Floggy-Entwicklern bereitgestellten Eclipse-Plugins – nicht zuverlässig in den Build-Prozess einbinden ließ. In Anbetracht der begrenzten Projektzeit wurde daher ein einfacherer Service entwickelt, der in 5.1.1 näher beschrieben wird.

3.3 GPS

J2ME bietet für den Zugriff auf Standortinformation die Location-API (JSR 179) an. Diese bietet unter anderem:

- Das Auslesen der aktuellen Position, Ausrichtung und Geschwindigkeit.
- Einen LocationListener, der periodisch Events erzeugt, die die Positionsdaten bzw. den Status des Location-Providers enthalten.
- Einen ProximityListener, der ein Event erzeugt, sobald sich der Empfänger in einem bestimmten Radius zu einem bestimmten Ziel befindet.

Am Anfang des Projekts wurde eine kleine Anwendung entwickelt, um diese drei Funktionen zu testen. Während das Auslesen der Positionsdaten und der LocationListener wie erwartet funktionierten, konnte der ProximityListener zwar registriert werden, generierte aber nur im Emulator Events.

3.4 Webservices

Der Standard JSR 172 beinhaltet die nötigen Funktionen, um mit J2ME auf Webservices zugreifen zu können. In der Evaluationsphase wurden einfache Test-Webservices (von einem Apache-Axis-Server[11]) aufgerufen. Wie bei der Verwendung von Webservices üblich, wird auf Clientseite dafür ein

[10] http://floggy.sourceforge.net/
[11] http://ws.apache.org/axis/

Stub generiert, der die Kommunikation und die Umwandlung der Daten übernimmt. Das Wireless Toolkit von Sun beinhaltet hierfür einen Stubgenerator, der eine WSDL[12]-Datei ausliest und Stubklassen und -methoden für die darin definierten Services erstellt.

Während der Evaluationsphase gab es keine Probleme beim Aufruf der Webservices. Später stellte sich jedoch heraus, dass komplexere Datentypen – wie z. B. Arrays innerhalb von Objekten – nur im Emulator korrekt funktionierten (siehe Abschnitt 6.2.2).

3.5 Spielideen

Zu Beginn der Projektphase wurden verschiedene Spielideen erörtert, anhand derer Beispielanwendungen entwickelt und Anforderungen an das Framework bestimmt worden sind. Neben dem später realisierten PS-Community-Manager (4.1) und dem OE-Spiel (4.2), gab es auch Ansätze für actionbetontere Spiele:

Fangen Hier wird von dem klassischen Fangen aus Schulzeiten ausgegangen. Gefangen ist man beispielsweise, wenn es zu einer Verbindung zweier Geräte z. B. mittels Ad-Hoc-WLAN kommt.

Capture the flag Auf einer Karte werden Positionen bestimmt, die „eingenommen“ werden müssen. Hat ein Team mehr als die Hälfte dieser Positionen besetzt, hat es gewonnen.

Aufgrund der begrenzten Projektzeit wurden diese Ansätze aber zugunsten der schneller realisierbaren, obenerwähnten verworfen.

3.6 Karte

Es wurden verschiedene Softwarelösungen für das Darstellen von Karten auf dem mobilen Gerät evaluiert. Dabei sollten diese einer Reihe von Anforderungen genügen:

Open Source Der Code musste frei verfügbar und erweiterbar sein.

J2ME Da in einer frühen Phase des Projekts die Entscheidung auf die Entwicklungssprache J2ME gefallen war, musste auch die Kartenkomponente in J2ME geschrieben sein.

Integration Eine einfache Integration ins Framework sollte möglich sein.

Offlinekarten Es musste möglich sein, Kartenmaterial, das auf dem Telefon abgelegt ist, anzuzeigen, um eine Offlinefunktionalität zu gewährleisten. In Abschnitt 4.2.4 wird dieses Thema genauer diskutiert.

Positionsanzeige Positionen sollten über grafische Symbole markiert werden.

Zu den evaluierten Systemen zählen unter anderem die Folgenden:

J2MEMap J2MEMap[13] ist eine Interface-Komponente die primär darauf ausgerichtet ist, eine Anbindung an Google Maps[14] zu ermöglichen. Nach anfänglich schnellen Erfolgen stellte sich schnell

[12] Web Services Description Language
[13] http://j2memap.landspurg.net/
[14] http://maps.google.com/

heraus, dass die Integration einer Offlinefunktionalität mit hohem Aufwand verbunden ist. Nach Rücksprachen mit dem Entwickler der Anwendung wurde von einer Integration ins Framework abgesehen.

MagicMap MagicMap[15] ist ein Projekt der Humboldt-Universität zu Berlin. Es bietet als Softwarelösung weit mehr als das reine Darstellen von Kartenmaterial. So beschäftigt sich das Projekt mit der Positionsbestimmung von mobilen Geräten, basierend auf der Auswertung von WLAN-Signalstärken.

MagicMap ist an sich bereits ein Framework mit großem Funktionsumfang. Eine Reduzierung von MagicMap auf die Kartendarstellung wurde wegen des entstehenden Overheads als nicht sinnvoll erachtet.

Navlet Navlet[16] ist ein Projekt, das an der FH Johanneum Kapfenberg entstand. Es erfüllt weitestgehend die oben beschriebenen Anforderungen.

Es stellte sich heraus, dass Navlet leicht erweiterbar ist, und so fiel die Entscheidung, es ins Framework zu integrieren. Eine genauere Betrachtung der Kartenkomponente ist in Abschnitt 5.1.2 zu finden.

[15] http://www2.informatik.hu-berlin.de/rok/MagicMap/index.htm
[16] http://www.navlet.org/

Kapitel 4

Anwendungen

Dieses Kapitel gibt einen Überblick über die im Projekt realisierten Beispielanwendungen (4.1) und (4.2). Die Entwicklung eines Frameworks für pervasive Anwendungen stand dabei im Vordergrund. Auf Basis der in Abschnitt 4.2 beschriebenen Beispielanwendung, werden dann in Abschnitt 4.2.3 fachliche und technische Anforderungen an das Framework identifiziert. Im Fokus dieses Kapitels steht das OE-Spiel, da die Autoren einen großen Teil der Projektzeit mit dessen Entwicklung zugebracht haben.

4.1 PS-Community-Manager

Hier erfolgt eine kurze Beschreibung des PS-Community-Managers, der hauptsächlich von Andreas Herglotz und Ralf Kruse realisiert wurde. Die Anwendung gehört zur Klasse der Finder-Applications, wie sie in [Schiller und Voisard (2004)] beschrieben sind. Die Anwendung unterstützt den Nutzer dahingehend, dass sie ihn auf Points-Of-Interest (POI) und Teilnehmer der Community – z. B. eingetragene Freunde – hinweist, die sich in der nahen Umgebung befinden. So wird auf die „beste Eisdiele der Stadt“ in Fußreichweite hingewiesen oder zwei Freunde, die sich zufällig gleichzeitig in einer fremden Stadt befinden, voneinander aber nichts wissen, können sich treffen.

Befinden sich andere Teilnehmer in der Nähe, so kann diesen eine Nachricht geschickt werden. Informationen zu den POIs sind individuell von den Benutzern anpassbar.

Der Fokus der Anwendung liegt im Bereich der Context-Awareness und hier speziell in der Location-Awareness. Technologien, die zum Einsatz kommen, sind einerseits Bluetooth sowie die Nutzung eines Positionsabgleiches über den Server. Die Lokationsbestimmung mittels GPS und der Abgleich der Daten über den Server haben sich bewährt. Eine stabile gegenseitige Dienstfindung über Bluetooth zum Aufspüren von Personen in unmittelbarer Umgebung konnte im Rahmen des Projektes nicht befriedigend realisiert werden.

Der PS-Community-Manager wird genauer in [Herglotz (2008)] und [Kruse (2008)] beschrieben.

4.2 OE-Spiel

Das OE-Spiel gehört in die Kategorie der „Schnitzeljagd“-ähnlichen Spiele, wie sie unter anderem in [Schiller und Voisard (2004)] beschrieben werden. Jedes Semester wird an der HAW traditionell während der Orientierungseinheit eine Schnitzeljagd veranstaltet. Bei dieser müssen die Teilnehmer verschiedene Orte auf dem Campus, wie etwa Mensa oder Studierendenwerk, aufsuchen. An diesen Orten können kleinere Aufgaben gestellt sein, die im Nachhinein belegen, dass die Teilnehmer auch

tatsächlich vor Ort waren. Ziel des Spiels ist es, sämtliche Punkte mit der schnellsten Gesamtzeit abzulaufen. Die Spielidee bleibt in der Beispielanwendung die gleiche, es kommen aber mobile Geräte zum Einsatz, die den Ablauf vorgeben und unterstützen.

4.2.1 Vorbereitung des Spiels

Auf dem Server müssen vor Spielbeginn die folgenden Vorbereitungen getroffen werden:

- Es müssen Wegpunkte eingetragen werden.
- Wegpunkte müssen zu einer Route zusammengefasst werden.

Die Vorbereitung auf dem Client läuft dann wie folgt ab:

- Der Spieler entscheidet zunächst, ob er ein vorhandenes Spiel fortsetzt oder ein neues Spiel beginnt.
- Ein vorhandenes Spiel kann nur fortgesetzt werden, falls bereits ein Spiel läuft.
- Wird ein neues Spiel begonnen, so muss zunächst eine Authentifizierung über ein clientseitiges Formular beim Server erfolgen. Hierzu gibt der Spieler in der entsprechenden Maske seine Benutzerdaten, also Benutzernamen und Passwort, ein.
- Soll ein neues Spiel erstellt werden, so muss der Spieler aus einer Liste von möglichen Spielrouten eine auswählen. Nach Eingabe von Spielname und Startzeit wird eine Spielinstanz auf dem Server angelegt. Im Anschluss tritt der Spieler dem Spiel bei und bezieht die Spieldaten vom Server.
- Soll einem vorhandenen Spiel beigetreten werden, wählt der Spieler aus einer Liste von möglichen Spielen eines aus. Im Anschluss tritt der Spieler dem Spiel bei und bezieht die Spieldaten vom Server.
- In beiden Fällen wird dann auf den Spielstart gewartet. Ist dieser erfolgt, kann das Spiel beginnen.

4.2.2 Ablauf des Spiels

Der Ablauf des OE-Spiels wird im Folgenden skizziert:

- Das Spiel wird auf dem Mobilfunktelefon der Benutzer gespielt, es beginnt, sobald der Spielstart erfolgt ist.
- Jedem Spieler werden nun drei zufällig aus einer Route ausgewählte Hinweise angezeigt. Die Hinweise in Rätselform führen den Spieler zu einem Wegpunkt.
- Aus diesen drei Hinweisen wählt der Spieler einen aus. Dabei können diese Hinweise von zwei verschiedenen Typen sein:
  - Hinweistyp GPS: Der Wegpunkt wird erreicht sobald sich der Spieler innerhalb eines bestimmten Gebiets aufhält. Dabei liefert der GPS-Empfänger des Mobiltelefons konstant die Position des Spielers.

  - Hinweistyp Barcode: Barcodes haben eine bestimmte Position. Diese ist durch ihren Aufenthaltsort vorgegeben. Erreicht ein Spieler einen Wegpunkt, muss er den Barcode fotografieren. Wird dieser erkannt, ist sichergestellt, dass sich der Spieler am gewünschten Ort befindet.

- Der Spieler muss anhand des Hinweises zum Wegpunkt finden.
- Ist der Spieler beim Wegpunkt angekommen, wird, abhängig von der Art des Hinweistyps, die nächste Aktion zum Erreichen des Ziels bestimmt.
- Ist das Ziel erreicht, wird der Wegpunkt entsprechend markiert, und ein neuer zufälliger Wegpunkt wird in die Auswahlliste hinzugefügt.

Zu jeder Zeit im Spiel können dabei die folgenden Aktionen ausgeführt werden:

- Auf einer Karte werden die aktuelle Position des Spielers angezeigt und ein Richtungspfeil. Letzterer zeigt in die Richtung, in der sich der nächste Wegpunkt befindet.
- Es kann ein anderer Hinweis aus der Liste der drei zufälligen Hinweise ausgewählt werden. So kann derjenige Hinweis unter den Dreien ausgewählt werden, der am schnellsten abgearbeitet werden kann oder der am dichtesten ist.
- Bei verfügbarer Internetverbindung, kann der Benutzer seinen eigenen Spielstand an den Server übertragen. Der Spielstand besteht dabei aus den abgeschlossenen Wegpunkten und der jeweiligen Zeit des Erreichens.
- Ist eine Internetverbindung verfügbar, kann der Spielstand vom Server abgefragt werden. Dies geschieht in Form einer Highscoreliste.
- Da sich sämtliche Spieldaten auf dem Client befinden, kann das Spiel zu jedem Zeitpunkt unterbrochen und wieder aufgenommen werden. Dazu bedarf es keiner Verbindung zum Server.

Beim Erreichen des letzten Wegpunktes:

- Ist das letzte Ziel erreicht, so wird der Spielstand an den Server übertragen.
- Die Platzierungen der einzelnen Spieler werden vom Server abgefragt und angezeigt. Dabei ergibt sich eine Sortierung nach der Anzahl der erreichten Hinweise und der Zeit des zuletzt erreichten Hinweises.

4.2.3 Anforderung an das Framework

Die Anforderungen an das Framework werden auf Basis der Projektziele und der beiden Beispielanwendungen – OE-Spiel und PS-Community-Manager – identifiziert. Weiterhin werden sie in fachliche und technische Anforderungen unterteilt.

Fachliche Anforderungen

Die Anforderungen sind bewusst unscharf bzw. generell gehalten, damit die grundlegenden Funktionalitäten erkennbar sind. Die fachlichen Anforderungen des PS-Community-Managers an das Framework können den Arbeiten [Herglotz (2008)] und [Kruse (2008)] entnommen werden, die für das OE-Spiel sind im Folgenden aufgeführt:

- Das Framework muss die Entwicklung von lokationsabhängigen Spielen im In- und Outdoorbereich ermöglichen.
- Benutzerverwaltung und Spieleverwaltung
- Validierung, ob ein gesuchter Ort erreicht wurde.
- Das Spiel soll zu jeder Zeit unterbrechbar sein und später fortgesetzt werden können.
- Der aktuelle Standort soll auf einer Karte visualisiert werden.

Technische Anforderungen

Aus den fachlichen Anforderungen lassen sich technische Anforderungen entwickeln. Diese wurden teilweise nicht im Framework umgesetzt, sondern lediglich in den Beispielanwendungen. Dies ist auf einen Mangel an Zeit im Projekt zurückzuführen.

- Zentrale Objektverwaltung/Spieleverwaltung auf einem Server über Serverkommunikation.
- Das eigentliche Spiel wird auf mobilen Clients gespielt.
- Daten müssen auf dem mobilen Gerät und auf dem Server persistiert werden können.
- Die Kommunikation zwischen Client und Server soll mittels Webservices erfolgen.
- Möglichkeiten zur Lokalisierung. Diese sollen entweder per GPS oder per Barcode realisiert werden. Aus der Anforderung des „Barcode Auslesens“ entsteht die Anforderung, integrierte Kamerafunktionen im Framework zu abstrahieren und verfügbar zu machen.
- Da verschiedene Arten von Anwendungen unterstützt werden sollen, wird eine einfache Konfigurationsmöglichkeit für das Framework benötigt.

4.2.4 Design und Realisierung

In diesem Abschnitt wird auf die Entwicklungs- und Designphase des OE-Spiels eingegangen. Der grundlegende Ablauf des OE-Spiels aus Benutzersicht ist in Abbildung 4.1 dargestellt. Abschließend wird in 4.2.4 die Umsetzung der Anwendung und deren Zusammenspiel mit dem Framework betrachtet.

Prototyp-Entwicklung

Die Entwicklung von Framework und Beispielanwendungen lief lange Zeit parallel und in einem Wechselspiel. Aus der Entwicklung des OE-Spiels entstanden neue Anforderungen für das Framework, und die im Framework umgesetzten Funktionen mussten wiederum ins OE-Spiel integriert werden.

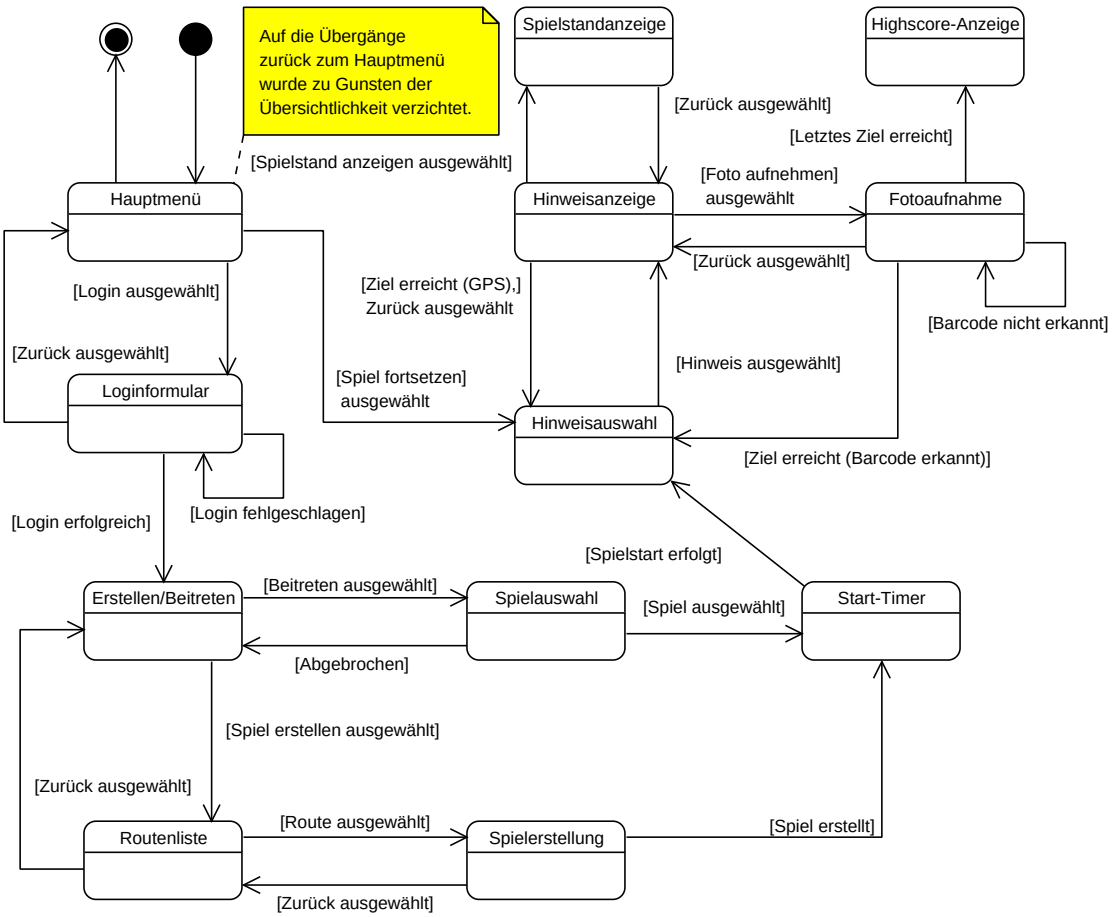

Abbildung 4.1: Zustandsdiagramm: OE-Spiel

Zunächst wurde für das OE-Spiel ein Prototyp entworfen. Dieser entsprach bereits dem Grundgedanken des Model-View-Controller-Patterns (MVC), um eine spätere Integration des Frameworks zu vereinfachen. Das Framework sah bereits zu einem frühen Zeitpunkt die Verwendung des MVC-Patterns vor. Siehe dazu Abschnitt 5.1.3.

Outdoor- und Indoor-Lokalisierung

Aus den Anforderungen für das OE-Spiel ist ersichtlich, dass es zwei verschiedene Formen der Lokalisierung geben muss. Das Spiel ist auf den Campus der HAW eingegrenzt, hier sollen Wegpunkte sowohl innerhalb der Gebäude als auch im Freien erkannt werden.

In Abschnitt 2.3 ist bereits die eingeschränkte Nutzbarkeit von GPS innerhalb von Gebäuden beschrieben. Es bedarf demnach einer zweiten Form der Lokalisierung. Dabei fiel die Entscheidung auf das Erkennen von Barcodes. Das erwies sich als verlässliche Form der Positionsbestimmung. Einzig bei eingeschränkten Lichtverhältnissen kam es zu Problemen. Dies gilt es bei der Anbringung der Barcodes zu beachten.

Ein großer Vorteil der Barcodes gegenüber GPS, im Hinblick auf mobile Geräte, ist der geringe

Stromverbrauch dieser Lokalisierungsmethode. Dies kann teilweise durch den Einsatz eines externen GPS-Empfängers umgangen werden.

Online- und Offline-Funktionalität

Eine wichtige Designentscheidung betraf die Netzverfügbarkeit auf den mobilen Geräten. Am Anfang des Projekts wurde davon ausgegangen, dass sich eine Netzverfügbarkeit auf den Aufenthalt in der Nähe eines WLAN-Hotspots beschränkt. Erst später eröffnete sich die Möglichkeit, mittels UMTS eine ständige Kommunikation mit dem Server zu erreichen. Natürlich kann es auch über UMTS zu Störungen im Empfang kommen. Des Weiteren muss der Faktor der Kosten für eine Netzverfügbarkeit berücksichtigt werden. Eine providerlose Kommunikation ist ohne laufende Kosten zu realisieren.

Aus dieser anfänglichen Einschränkung entstanden eine Reihe von Entwicklungsentscheidungen, die das Spiel auch im Offlinebetrieb spielbar machen:

- Daten müssen auf dem mobilen Gerät persistent sein können, da sie nicht, etwa nach einer Spielunterbrechung, erneut vom Server geholt werden können. Die Spieldaten liegen immer komplett auf dem Client.
- Der Spielablauf ist so ausgelegt, dass mehrere Teilnehmer das Spiel parallel, aber separat, spielen können. Lediglich zu Spielbeginn (Spieldaten empfangen) und am Spielende (Spielauswertung) ist eine Serververbindung zwingend erforderlich.
- Kartenmaterial für die Anzeige der Karte muss (zusätzlich) auf dem Client liegen.

Realisierung

Die Realisierung der Beispielanwendungen erfolgte auf Basis des Frameworks, wie es in Kapitel 5 beschrieben ist. Einige der Funktionalitäten wurden allerdings nur prototypisch für das OE-Spiel implementiert:

Model in MVC Da das Framework keine Vorgaben zur Abbildung des Modells vorgibt, ist es im OE-Spiel auf Basis von einfachen Datenobjekten realisiert, die lediglich über Attribute mit Getter- und Settermethoden verfügen.

Webservices Die Kommunikation zwischen dem in Kapitel 6 beschriebenen Server und den Clients findet mittels Webservices statt. Zum Austausch von Objekten werden Transport-Objekte benutzt. Da sich herausstellte, dass der Webservice mit komplexen Datentypen nicht korrekt arbeiten konnte, wurden diese einfach gehalten. Auf Clientseite gibt es eine Klasse, die Methoden bereitstellt, um aus den Transport-Objekten wieder Datenobjekte des Clientmodells zu erstellen. Siehe hierzu Abschnitt 6.2.2.

Timer Über einfache Timerthreads erfolgt das Warten auf den Spielstart.

Im Folgenden sind Komponenten beschrieben, die im Framework abstrahiert wurden und vom OE-Spiel genutzt werden:

CameraService/BarcodeService Die Kamera und die Barcodeerkennung werden für die Lokalisierung eingesetzt. Abbildung 4.2d zeigt die Anwendung dabei, wie sie einen Barcode abfotografiert.

Abbildung 4.2: OE-Spiel: Auswahl an Screenshots

LocationService Wird benutzt, um die Entfernung zu einem Wegpunkt sowie das Erreichen eines Wegpunktes zu ermitteln (Abbildung 4.2c). Außerdem wird die aktuelle Position auf einer Karte dargestellt. Siehe Abbildung 4.2e.

AudioService Spielt bei besonderen Ereignissen kurze Audiodateien ab.

Konfiguration und Ablaufsteuerung In einer XML-Datei wird das Framework konfiguriert und eine Ablaufsteuerung vorgegeben.

MapService Wird für das Anzeigen der Karte benutzt. Siehe Abbildung 4.2e.

Persistenzservice Das Spiel wird regelmäßig nach wichtigen Ereignissen gespeichert. Etwa dem Erreichen eines Wegpunktes oder einer neu generierten Auswahlliste für die Hinweise.

Event- und Exception-Handling Erwartete Fehler, wie etwa Fehler bei der Serverkommunikation, werden von der Anwendung abgefangen. Die Anwendung wird im Anschluss, nach dem Anzeigen einer Fehlermeldung, wieder in einen konsistenten Zustand überführt. Unerwartete Fehler werden vom Framework behandelt. Siehe auch Abschnitt 5.1.4 und 6.2.2.

Abbildung 4.2 zeigt einige ausgewählte Screenshots des implementierten OE-Spiels. Für die Realisierung der GUI wurden weitestgehend J2ME-Standardelemente benutzt – z. B. Listen und Choicegroups. Lediglich die Karte verwendet ein auf der Canvaskomponente selbst entworfenes bzw. gezeichnetes GUI-Element. Generell ist das Entwickeln von GUI-Elementen durch die beschränkten Eingabemethoden von Mobilfunktelefonen begrenzt.

Erfahrungen aus der Realisierungsphase

Die Entwicklung von mobilen Anwendungen stellt den Entwickler vor einige spezielle Herausforderungen:

- Das Debbuging von Applikationen ist nur während des Betriebs im Emulator möglich. Fehler, die in Anwendungen, die auf dem Telefon laufen, auftreten, können im Wesentlichen nur über das klassische Anzeigen von Text eingegrenzt werden.
- Das Verhalten von Anwendungen hängt davon ab, ob die Anwendung auf dem Telefon oder im Emulator läuft. Beispielsweise lief die Bluetooth-Kommunikation auf dem Telefon deutlich instabiler als im Emulator. Generell verursachten externe Events, die von außerhalb der Applikation kamen, wie etwa Proximity-Events (GPS) und eben Bluetooth, im Emulator weniger Probleme.
- Trotz eines relativ leistungsstarken Geräts, ist beim Entwickeln darauf zu achten, dass die Anwendung im Emulator auf die volle Rechenleistung eines PCs zurückgreift und sich so wesentlich performanter verhält.

Kapitel 5

Framework

Die in 4.2.3 vorgestellten Anforderungen wurden für die Erstellung der Framework-Komponenten hinzugezogen, die im Folgenden vorgestellt werden. Abbildung 5.1 zeigt eine Übersicht über die Architektur des Frameworks. In diesem Kapitel wird primär auf die Komponenten Persistenz und Karte eingegangen, da diese von den Autoren entwickelt wurden. Eine genauere Beschreibung der Funktionsweise des Frameworks ist den Ausarbeitungen [Dreyer (2008)], [Mas (2008)], [Tutzschke (2008)] und [Vollmer (2008)] zu entnehmen.

5.1 Komponenten

5.1.1 Persistenz

Wie bereits in Abschnitt 3.2 ausgeführt, kam der Einsatz von Floggy nicht in Frage. Stattdessen wurde ein PersistenceService entwickelt, der den Zugriff auf die RecordStores vereinfacht und bei der Umwandlung von Objekten in Byte-Arrays hilft.

Um nicht zu viel Zeit in die Entwicklung zu investieren, wurde der Zugriff auf persistente Daten durch einfache Key-Value-Paare realisiert. Entwickler greifen also nicht mehr über einen Schlüssel auf einen RecordStore zu, um danach über einen Index an die gewünschten Daten zu kommen, sondern erhalten durch den PersistenceService über den Schlüssel direkten Zugriff auf die Daten.

Die Persistenz wird von den Modulen PersistenceHelper und PersistenceService übernommen.

PersitenceHelper

Der PersistenceHelper besteht aus statischen Methoden, die die Umwandlung von Objekten in Byte-Arrays (und umgekehrt) übernehmen. Die Funktionsweise orientiert sich an dem in [J2ME Tech Tips (2006)] vorgeschlagenen Verfahren. Primitive Datentypen und Strings können direkt in Byte-Arrays umgewandelt werden. Entwickler müssen sich darum nicht selbst kümmern.

Die Umwandlung von Objekten verläuft nicht transparent für Entwickler. Der PersistenceService kann nur Objekte behandeln, die das Interface Persistent implementieren. Dies besteht aus zwei Methoden:

- *persist* bildet den Zustand des Objekts auf ein Byte-Array ab.
- *resurrect* liest ein Byte-Array aus und überführt das Objekt zurück in den gespeicherten Zustand.

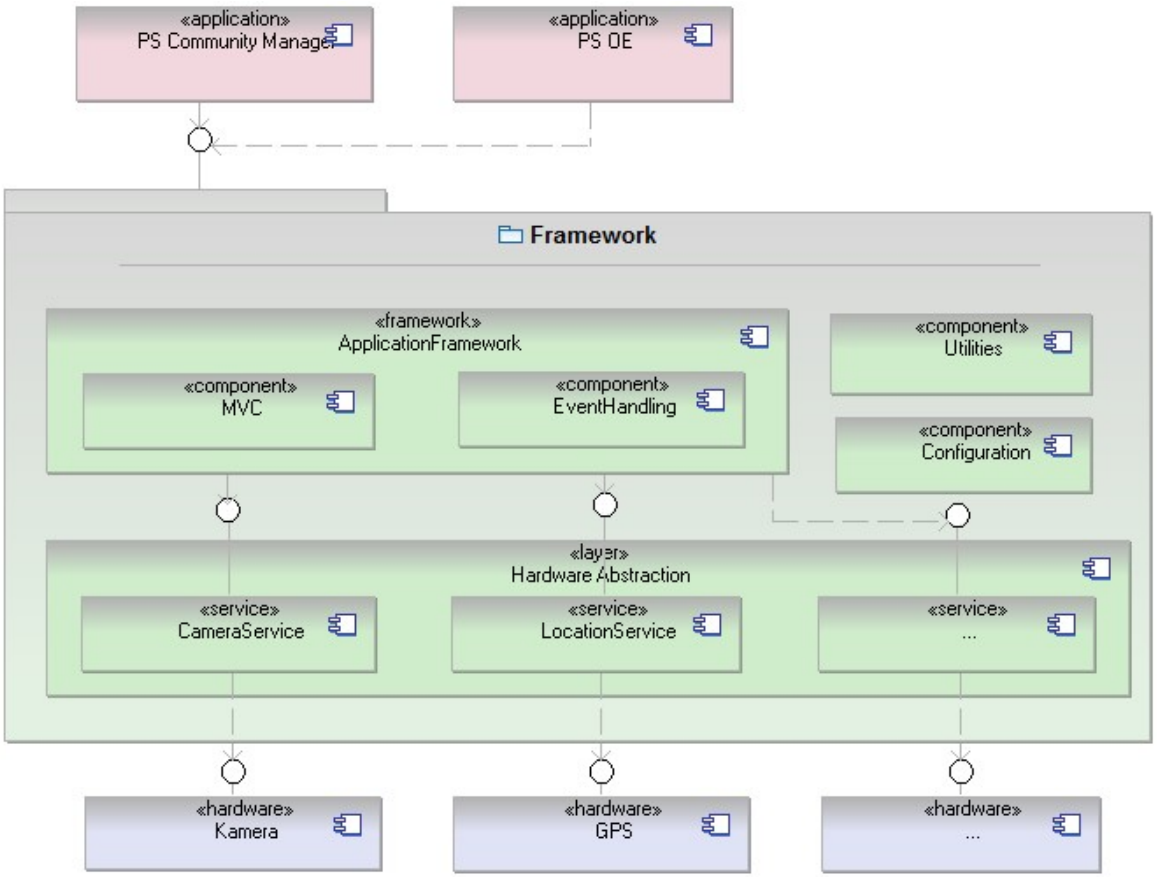

Abbildung 5.1: Architektur des Frameworks

Um die Umwandlung zu vereinfachen bietet das Framework die abstrakte Oberklasse AbstractPersistentObject an, die das Persistent-Interface implementiert. Diese übernimmt die Initialisierung der von J2ME bereitgestellten Klassen ByteArrayInputStream und ByteArrayOutputStream, die Methoden für die Konvertierung von primitiven Datentypen und Strings in Byte-Arrays anbieten. Anwendungsentwickler müssen so nur noch diese beiden Streams verwenden, um die Umwandlung der Objektvariablen vorzunehmen.

Die manuelle Erstellung der Umwandlungsmethoden bringen zwei wesentliche Nachteile mit sich: Zum Einen müssen sie von den Anwendungsentwicklern für jede Klasse manuell erstellt werden, zum anderen ist es schwer Fehler zu finden. Wenn eine Persist-Methode die Daten beispielsweise in einer anderen Reihenfolge speichert, als die dazugehörige Resurrect-Methode sie ausliest, schlägt der Resurrect-Vorgang fehl. Wenn derart fehlerhaft implementierte Klassen Teile anderer Klassen sind, funktioniert der gesamte Resurrect-Vorgang nicht mehr.

PersistenceService

Der PersistenceService bietet die API für das Speichern, Laden und Löschen von Daten an. Er verwendet den PersistenceHelper, um die Umwandlung der Daten vorzunehmen. Außerdem wird die Funktionalität des RecordStore-Systems um eine Methode ergänzt, die überprüft, ob bereits ein Wert zu einem gegebenen Schlüssel existiert.

5.1.2 Karte

In Abschnitt 3.6 wurde bereits erläutert, warum die Wahl für die Kartenkomponente auf die Softwarelösung Navlet fiel. Große Teile des Quellcodes wurden übernommen und später um Funktionalitäten erweitert. In diesem Abschnitt wird nun der Aufbau der Komponente beschrieben.

Die Kartenkomponente ist Teil des Frameworks. Über eine Datenverbindung können digitale Landkarten im Rasterformat von einem Map-Server bezogen werden, und die aktuelle Position kann in dieser Karte auf dem mobilen Gerät visualisiert werden. Dazu wird eine HTTP-Anfrage an den Map-Server gestellt, in der die entsprechenden Koordinaten übermittelt werden. Der Server reagiert auf diese Anfrage, in dem er, unter Berücksichtigung der Koordinaten, ein Bild generiert und zurücksendet.

Wie in Abschnitt 4.2.4 beschrieben, ist es aber wichtig, dass für den Fall, dass keine Datenverbindung verfügbar ist, auch lokale, im Dateisystem des mobilen Geräts abgelegte Kartenmaterialien verwendet werden können. Diese Karten müssen im PNG-Grafikformat vorliegen. Neben der Grafikdatei bedarf es noch einer Datei mit Metadaten. In dieser liegen die geografischen Koordinaten der linken oberen Ecke und der rechten unteren Ecke der Karte. Aus diesen beiden Koordinaten, können die Koordinaten aller auf der Karte befindlichen Punkte berechnet werden. Der Breitengrad wird hierbei mit ddmm.mmmm, der Längengrad mit dddmm.mmmm angegeben[17].

Die Kartenkomponente rechnet intern mit geografischen Koordinaten im NMEA-Format [NMEA (2008)]. Da die GPS-Komponente aber Koordinaten für Längen- und Breitengrade als Dezimalwerte liefert, wurden hierfür Methoden zur Umrechnung implementiert.

Da die Bilddatei einer Offlinekarte viel Platz auf dem Telefon belegt, ist die Auflösung der Karte und die Größe der Fläche stark eingeschränkt. Bei dynamisch vom Server bezogenen Karten kommt dieser Nachteil nicht zum Tragen, da hierbei immer nur der aktuell angezeigte Kartenbereich auf dem Client vorgehalten werden muss.

[17] „d“ steht dabei für Grad, „m“ für Minuten

Die Karte kann die aktuelle Position – die Koordinaten, die vom GPS geliefert werden – anzeigen. Zusätzlich können auch weitere wichtige Orte mit Icons versehen werden. Die Location-API von J2ME stellt eine Funktion bereit[18], die die Richtung von einer Position zu einer anderen in Grad wiedergibt. Die im Framework abstrahierte Methode wird von der Kartenkomponente genutzt, um einen Richtungspfeil von der aktuellen Position zu einem bestimmten Ziel anzuzeigen.

5.1.3 Model-View-Controller

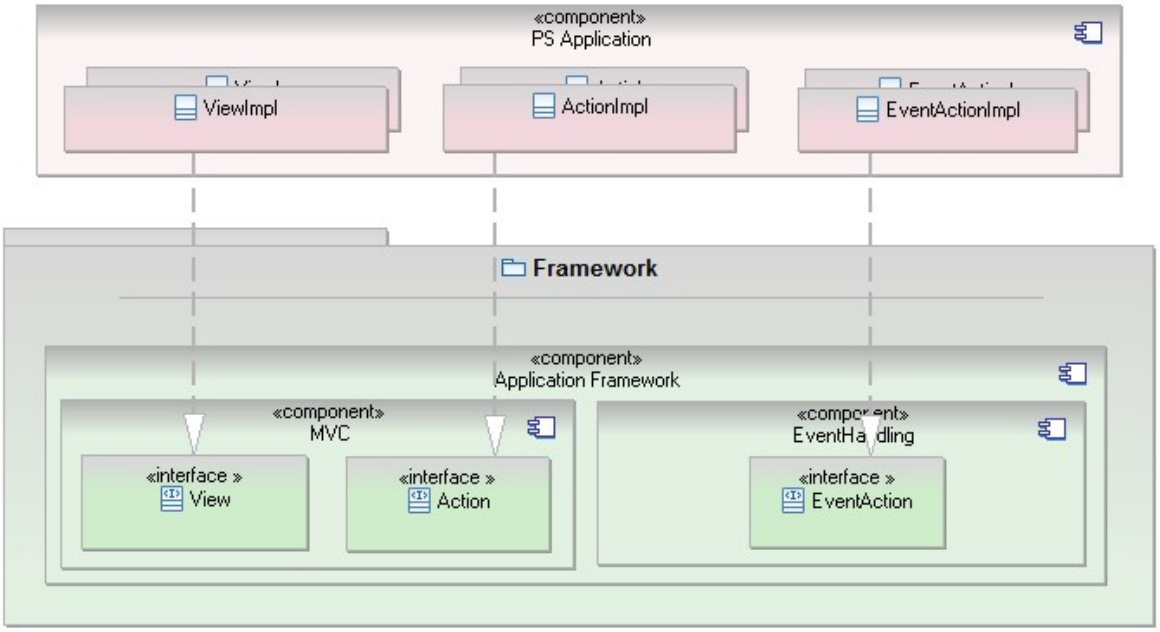

Abbildung 5.2: Model-View-Controller-Komponente

Um mit dem Framework entwickelte Anwendungen zu strukturieren wird das Model-View-Controller-Pattern verwendet. Das Grundprinzip des MVC-Pattern beruht auf der Aufteilung von Daten (Model), Präsentation (View) und Programmsteuerung (Controller) [Fowler (2006)].

Wie in Abbildung 5.2 zu sehen, sind View und Controller (hier als Action bezeichnet), im Framework als Java-Interfaces realisiert, die in den konkreten Anwendungen implementiert werden. Für das Modell stellt das Framework keine explizite Funktionalität zur Verfügung. Die Implementierung bleibt hier den Anwendungsentwicklern überlassen.

Um die Interaktion zwischen Actions und Views zu ermöglichen, wird der SessionContext verwendet, in dem beliebige Daten anwendungsweit hinterlegt werden können. Das Framework sorgt dafür, dass innerhalb von Views und Actions auf den SessionContext zugegriffen werden kann.

[18] Spezifiziert in der JSR 179

5.1.4 Event- und Exception-Handling

Innerhalb einer Konfigurationsdatei ist es möglich, Actions zu definieren, die beim Auftreten von Exceptions oder Events ausgeführt werden. Beispiele für unterstützte Events sind die in Abschnitt 3.3 vorgestellten Location-Events.

Ein in der Konfigurationsdatei definierter Exception-Handler kann dafür eingesetzt werden, nicht vorgesehene Exceptions zu behandeln. Er kann beispielsweise dazu verwendet werden, die Anwendung nach einem Fehler (falls möglich) wieder in einen konsistenten Zustand zu überführen und dem Benutzer eine aussagekräftige Fehlermeldung anzuzeigen.

5.1.5 Sonstige Komponenten

Das Framework enthält noch weitere Komponenten, die hier nicht weiter betrachtet werden, dazu zählen:

CameraService Erfassen von Fotos

BarcodeService Auslesen von Semacodes

AudioService Abspielen von Audiodateien

LocationService Erfassen und Verarbeiten von Positionsdaten

5.2 Konfiguration und Ablaufsteuerung

Um die Wiederverwendbarkeit einzelner Anwendungskomponenten zu erhöhen, wird das Framework in einer zentralen XML-Datei konfiguriert. Diese besteht im Wesentlichen aus den folgenden Bereichen:

Actions und Views Für jede Action und jede View wird die zuständige Java-Klasse und ein eindeutiger Name definiert.

FlowPoints Innerhalb eines FlowPoints wird die Ablaufsteuerung definiert. Dafür werden für jede Action ein oder mehrere Ergebniswerte definiert. Jedem Ergebniswert wird ein View zugeordnet, der angezeigt wird, wenn die Action den jeweiligen Ergebniswert zurückgibt.

FlowPoints können verwendet werden, um zwischen verschiedenen, vordefinierten Abläufen umzuschalten.

Kapitel 6

Server

Der Schwerpunkt des Projekts lag eindeutig auf Clientseite. Der Server wurde daher nur als Prototyp realisiert. Ein wichtiger Aspekt bei der Entwicklung des Servers war das einfache Bereitstellen des Datenspeichers und der Webservices.

6.1 Anforderungen durch das OE-Spiel

Im Rahmen des OE-Spiels sollte der Server folgenden Anforderungen genügen:

Webservices Für die Kommunikation mit dem Client soll der Server Webservices anbieten. Aufgrund der in 4.2.4 beschriebenen Daten-Problematik muss der Server die für den Client bestimmten Daten in ein einfacheres Format umwandeln.

Spielbeschreibung Der Server muss alle Daten, die zur Beschreibung des Spiels nötig sind, speichern. Teile dieser Daten werden via Webservice dem Client zur Verfügung gestellt. Außerdem übernimmt er den serverseitigen Teil der Spielelogik.

Authentifizierung Benutzer sollen sich, innerhalb des OE-Spiels, am Server mit Benutzername und Passwort authentifizieren können.

Session-Handling Der Server soll für die Anwendung eindeutige Session-IDs für die eingeloggten Benutzer vergeben. Außerdem soll er die Sessions nach einer festgelegten Zeitspanne automatisch löschen.

6.2 Design und Realisierung

6.2.1 Persistenz

Um sich nicht auf eine Datenbank festzulegen, wurde Hibernate Annotations[19] verwendet. Während der Entwicklung kam HSQL[20] als Datenbank zum Einsatz. Neben einer Datenbankabstraktion bietet Hibernate die Abbildung der relationalen Datenbank auf Objekte (Object-relational Mapping, kurz ORM).

[19] http://annotations.hibernate.org/
[20] http://www.hsqldb.org/

Ein Vorteil von Hibernate Annotations (gegenüber dem klassischen Hibernate) ist die einfache Beschreibung des Datenmodells auf Basis von Java-Klassen. Java-Annotations werden hierbei verwendet, um Metadaten hinzuzufügen, die für die Abbildung der Klassen auf die Datenbank (und umgekehrt) benötigt werden. Die Annotations folgen dabei dem EJB3-Standard[21] (Enterprise Java Beans) und können bereits zur Compile-Zeit auf Fehler überprüft werden.

Peer-Generator

Um den Zugriff auf das Datenmodell zu kapseln, wurde ein Generator entwickelt, der Peer-Klassen erzeugt, die Zugriffsmethoden für das Auffinden, Speichern und Löschen von persistenten Objekten bereitstellen. Um diese Zugriffsmethoden zu generieren, werden die als Annotations definierten Metadaten, wie z. B. die Primär- und Alternativschlüssel, verwendet.

Durch die Verwendung von Hibernate Annotations und den Peer-Klassen ist es verhältnismäßig einfach möglich, nicht nur eine andere Datenbank, sondern auch eine andere OR-Mapping-Implementation, die dem EJB3-Standard folgt, einzusetzen.

Datenmodell

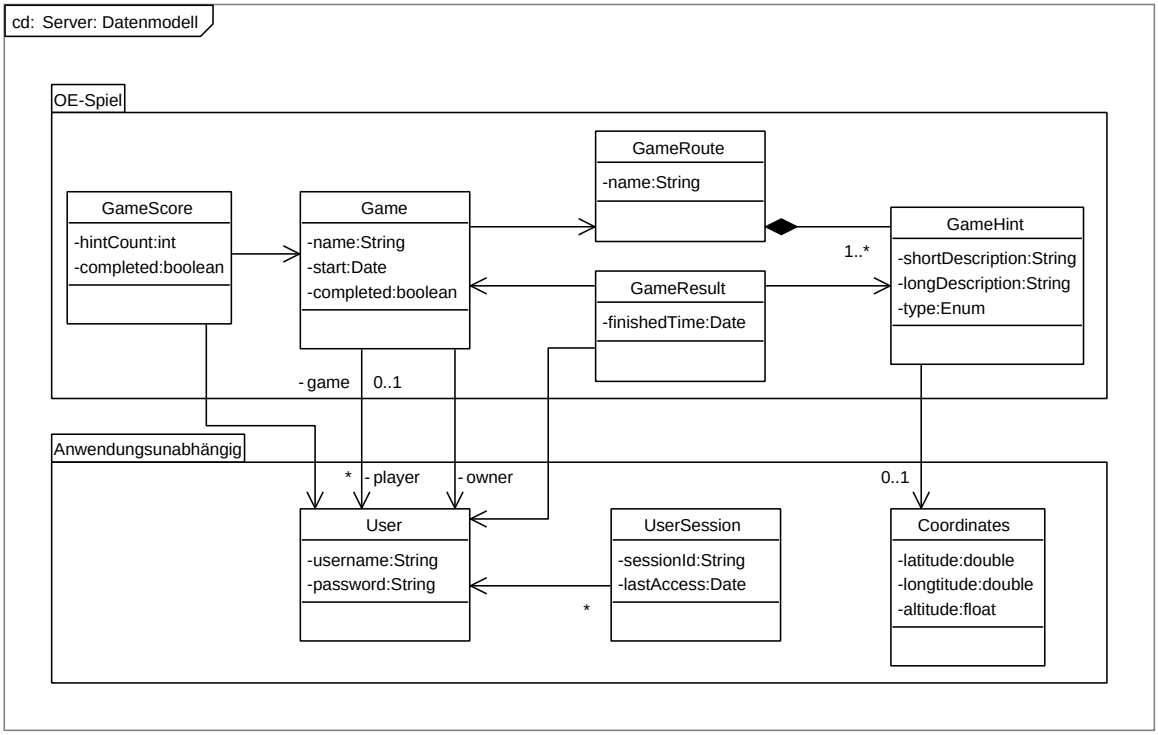

Abbildung 6.1: Allgemeines Datenmodell und OE-Spiel-Datenmodell

Der Server stellt ein anwendungsunabhängiges Basis-Datenmodell zur Verfügung. Hierbei handelt es sich um wenige Klassen, die häufig in mobilen Anwendungen nützlich sind. Primär dienen diese der Benutzerauthentifizierung. Abbildung 6.1 zeigt die anwendungsunabhängige Datenschicht sowie

[21] http://java.sun.com/products/ejb/

die Datenschicht des OE-Spiels. Die anwendungsunabhängige Schicht hat dabei keinen Zugriff auf die höherliegende Schicht des OE-Spiels.

6.2.2 Kommunikation

Für die Client/Server-Kommunikation verwendet der Server Webservices, die von der Apache-Axis-Engine bereitgestellt werden.

Um die Entwicklung von Services zu vereinfachen, die über den mobilen Client von Benutzern aufgerufen werden, stellt der Server einen Basis-Service zur Verfügung. Dieser übernimmt die Authentifizierung der Benutzer sowie das Session-Handling.

Transport-Objekte

Wie bereits in Abschnitt 3.4 erwähnt, gab es Schwierigkeiten bei der Verwendung einiger Datentypen mit den Webservices. Während die Stubs korrekt generiert wurden und auch der Emulator korrekt mit den generierten Stubs arbeitete, kam es beim Unmarshalling[22] der SOAP-Nachrichten auf dem Mobiltelefon zu Fehlern. Um dieses Problem zu umgehen, werden für die Client/Server-Kommunikation Transport-Objekte verwendet, die so einfach strukturiert sind, dass sie auch auf dem Client problemlos verarbeitet werden können.

Fehlerbehandlung

Für das Übertragen von Fehlern werden ebenfalls die Transport-Objekte verwendet. Hierbei werden alle Exceptions abgefangen, bevor die Service-Methoden abgearbeitet sind. Wird eine erwartete Exception abgefangen, werden dem Transport-Objekt unter anderem ein Errorcode und eine für den Benutzer verständliche Fehlermeldung hinzugefügt. Für die nicht erwarteten Exceptions kann für den Benutzer allerdings keine verständliche Fehlermeldung hinzugefügt werden. Dennoch kann auf Clientseite eine allgemeine Meldung angezeigt werden, die besagt, dass ein Fehler während der Client/Server-Kommunikation aufgetreten ist.

[22] Das Parsen der XML-Nachrichten schlug fehl. Daher konnten keine Java-Objekte daraus erzeugt werden

Kapitel 7

Fazit und Ausblick

Im Laufe des Projekts wurden einige Erfahrungen mit der Entwicklung von (J2ME-basierten) Anwendungen für mobile Geräte gesammelt. Die in Kapitel 3 beschriebene Evaluationsphase gab einen guten Einblick in die technischen Möglichkeiten der verwendeten Testgeräte. Dennoch traten viele Probleme erst auf, als die getesteten Funktionen, in Zusammenarbeit mit anderen Komponenten, in das Framework übernommen wurden.

Der Fokus der Entwicklung lag auf dem Framework selbst und den Anwendungen zur Validierung der Framework-Funktionen. Das Wechselspiel zwischen der Entwicklung der Anwendungen (Kapitel 4) auf der einen Seite und der Entwicklung des Frameworks (Kapitel 5) auf der anderen Seite erwies sich als zweckmäßig. Der Fokus der Server-Entwicklung (Kapitel 6) lag auf der für den Entwickler vereinfachten Bereitstellung der Service-Methoden und des Datenspeichers.

Das Ergebnis des Projekts ist ein Framework für J2ME-Anwendungen, das Entwicklern viel Arbeit mit alltäglichen Problemen abnimmt. Dazu zählen – neben den strukturgebenden Elementen, wie den MVC-Komponenten und der Ablaufsteuerung – vor allem kleinere Komponenten, die den Zugriff auf die Hardware erleichtern, wie z. B. der PersitenceService oder der LocationService.

7.1 Ausblick

Obwohl das Framework dem Anwedungsentwickler viel Arbeit abnimmt, gibt es viele Stellen, an denen angeknüpft werden kann:

Module Das Framework bietet zwar clientseitige Komponenten an, es fehlt aber an vorgefertigten, höherschichtigen Modulen, die typische Aufgaben auf Client- und Serverseite übernehmen.

Beispiele für solche Module sind aus den beiden Anwendungen ableitbar:

- Ein Chatmodul, das eine anpassbare Clientoberfläche, die dazugehörige Serverlogik und Webservices (inklusive Clientstubs) für die Kommunikation zur Verfügung stellt. Eine ähnliche Funktionalität ist im PS-Community-Manager enthalten.
- Ein Modul, das die Entwicklung von Routen-basierten Spielen, wie dem OE-Spiel erleichtert. Hierbei ist es wünschenswert, die GPS-Koordination der Routenpunkte nicht per Hand einzugeben. Einfacher ist es, eine geplante Route abzulaufen, diese mit Hilfe der Anwendung auf dem Gerät aufzuzeichnen und auf den Server zu übertragen.

Server Da der Server nicht im Fokus der Entwicklung lag, gibt es hier einige Funktionen, die noch hinzugefügt werden sollten. So fehlt es dem Server an Administrationsoberflächen für anwendungsübergreifende Module, wie z. B. einer Benutzerverwaltung. Eine engere Kopplung mit dem Client-Framework ist denkbar, was beispielsweise die Kommunikation weiter vereinfachen könnte. Der Nachteil einer engeren Kopplung liegt allerdings darin, dass der Server dann unter Umständen nicht mehr so leicht ausgetauscht werden kann.

Andere Plattformen Das Framework wurde ausschließlich für J2ME-basierte Anwendungen entwickelt. Möglich ist eine Neuentwicklung oder Portierung auf andere Plattformen. Einige Projektmitglieder planen beispielsweise ähnliche Entwicklungen für die in der Entstehung befindliche Android-Plattform[23] im Rahmen ihrer Masterarbeit durchzuführen.

Sicherheit Weder das Framework noch die beiden Anwendungen, verwirklichen Sicherheitskonzepte. Es bleibt also zu untersuchen, inwiefern sich altbekannte Konzepte wie Verschlüsselung via HTTPS für mobile Anwendungen eignen. Außerdem wurden keine expliziten Vorkehrungen getroffen, um das Schummeln bei Spielen einzuschränken. Da im OE-Spiel die gesamte Route und der eigene Spielstand auf dem Client (unverschlüsselt) gespeichert wird, ist es theoretisch möglich, alle Hinweise ohne Erreichen der Ziele als „erreicht“ zu markieren.

Serverlose Anwendungen Die beiden entwickelten Anwendungen basieren unterschiedlich stark auf der Kommunikation mit einem Server. Anders als beim PS-Community-Manager ist Nachrichtenübermittlung und Chat direkt von Gerät zu Gerät denkbar [Schönherr (2008)]. In seiner Masterarbeit wird sich Jan Schönherr voraussichtlich mit der Entwicklung von Peer-To-Peer-basierten Anwendungen auf mobilen Geräten beschäftigen.

[23] http://code.google.com/android/

Literaturverzeichnis

[Dreyer 2008] DREYER, Markus: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[Fowler 2006] FOWLER, Martin: *GUI Architectures*. Verifiziert am 21.02.2008. 2006. – URL `http://www.martinfowler.com/eaaDev/uiArchs.html`

[Gamma u. a. 2004] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2004. – ISBN 0-201-63361-2

[Herglotz 2008] HERGLOTZ, Andreas: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[J2ME Tech Tips 2006] MICROSYSTEMS, Sun: *Wireless Tech Tips – J2ME Tech Tips: February 26, 2002*. Verifiziert am 18.02.2008. 2006. – URL `http://java.sun.com/developer/J2METechTips/2002/tt0226.html`

[Kruse 2008] KRUSE, Ralf: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[Mas 2008] MAS, Alexander: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[NMEA 2008] KÖHNE, Dr. A. ; WÖSSNER, Dr. M.: *NMEA-0183 Daten*. Verifiziert am 18.02.2008. 2008. – URL `http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm`

[Schiller und Voisard 2004] SCHILLER, Jochen ; VOISARD, Agnes: *Location Based Services*. Morgan Kaufmann, 2004. – ISBN 1-55860-929-6

[Schönherr 2007] SCHÖNHERR, Jan: *Location-based Services For Pervasive Gaming*, HAW Hamburg, Seminarausarbeitung für Anwendungen 1, 2007

[Schönherr 2008] SCHÖNHERR, Jan: *Peer-To-Peer In Mobile Information Environments*, HAW Hamburg, Seminarausarbeitung, 2008

[Sun Microsystems 2008] MICROSYSTEMS, Sun: *JSRs: Java Specification Requests*. Verifiziert am 20.02.2008. 2008. – URL `http://www.jcp.org/en/jsr/overview`

[Tutzschke 2008] TUTZSCHKE, Jan-Peter: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[Vollmer 2008] VOLLMER, Sven: *Pervasive Spine – Ein Framework für mobile Anwendungen*, HAW Hamburg, Projektbericht, 2008

[White 2001] WHITE, James: An introduction to Java 2 micro edition (J2ME); Java in small things. In: *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2001, S. 724–725. – ISBN 0-7695-1050-7