Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterprojekt - Bericht

Alexandra Revout

Pervasive Gaming Framework

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexandra Revout

Masterprojekt - Bericht
Pervasive Gaming Framework

Stichworte
Framework, Pervasive Gaming, mobile Systeme, Orts-basierte Spiele, XML-Synchronisierung, Daten-Persistenz

Kurzzusammenfassung
In letzter Zeit gewinnt in der Informatik der Bereich des Pervasive Computing mehr und mehr an Bedeutung, nicht zuletzt wegen wachsenden Mobilitäts-Ansprüchen der Benutzer und steigenden Möglichkeiten von mobilen Geräten und drahtlosen Netzwerken. Pervasive Gaming ist eine Form des Pervasive Computing und ist auf Technologien und Konzepte für Spiele spezifiziert, die versuchen, reale und virtuelle Welten zu verschmelzen. Aus dieser Motivation und allgemeinem Interesse für mobile Anwendungen und solchen Themen wie z.B. Location Based Services entstand das in dieser Ausarbeitung beschriebene Projekt. Als Ziel des Projektes wurde der Entwurf und die Realisierung eines Frameworks für die Entwicklung von pervasiven Spielen (*Pervasive Gaming Framework*) gesetzt, die sich durch das mobile Spielen und die Integration zwischen physikalischen und virtuellen Welten auszeichnen. Außerdem sollte ein konkretes Spiel als Proof-of-Concept-Prototyp entwickelt werden. In dieser Ausarbeitung werden tragende Ideen und Konzepte des Projektes beschrieben, besonders wird dabei auf den Entwurf der Spielabläufe und auf das Server-Design eingegangen, an deren Entwicklung die Autorin dieses Berichtes maßgeblich beteiligt war.

Inhaltsverzeichnis

1 Einleitung **4**

2 Entwurf und Realisierung **4**
2.1 Spielidee 4
2.2 Allgemeiner System-Aufbau 5
2.3 Server 8
2.3.1 Data-Manager 10
2.3.2 Communication Layer 13

3 Fazit und Ausblick **13**

Literaturverzeichnis **14**

A Anhang **15**
A.1 Definition der Server-seitigen Schnittstellen 15
A.2 PGF-Klassendiagramm 17

1 Einleitung

In letzter Zeit gewinnt in der Informatik der Bereich des Pervasive Computing [Mattern] mehr und mehr an Bedeutung, nicht zuletzt wegen wachsenden Mobilitäts-Ansprüchen der Benutzer und steigenden Möglichkeiten von mobilen Geräten und drahtlosen Netzwerken. Pervasive Gaming ist eine Form des Pervasive Computing und ist auf Technologien und Konzepte für Spiele spezifiziert, die versuchen, reale und virtuelle Welten zu verschmelzen. Es werden zahlreiche Projekte in dieser Richtung geführt, z.B. MobileChase an Fraunhofer Institut [MobileChase] oder das Projekt Cron an der TU Berlin [Cron]. Darüber hinaus existiert eine europäische Initiative zum Auftrieb des Pervasive Gaming [IPerG]. Aus dieser Motivation und allgemeinem Interesse für mobile Anwendungen und solchen Themen wie Location Based Services, Kollaboratives Editieren von Dokumenten, Autonomic Computing und mobile Prozesse entstand das in dieser Ausarbeitung beschriebene Projekt.

Pervasive Spiele können durch drei wesentliche Aspekte ausgezeichnet werden (nach [Jegers]):

- Das mobile Spielen
- Die Integration zwischen physikalischen und virtuellen Welten
- Soziale Interaktionen zwischen den Spielern

Als Ziel des Projektes wurde der Entwurf und die Realisierung eines Frameworks für die Entwicklung von pervasiven Spielen (*Pervasive Gaming Framework*) gesetzt, die besonders zwei ersten oben genannten Aspekten entsprechen. Außerdem sollte ein konkretes Spiel als Proof-of-Concept-Prototyp entwickelt werden. In dieser Ausarbeitung werden tragende Ideen und Konzepte des Projektes beschrieben, besonders wird dabei auf den Entwurf der Spielabläufe und auf das Server-Design eingegangen, an deren Entwicklung die Autorin dieses Berichtes maßgeblich beteiligt war.

2 Entwurf und Realisierung

2.1 Spielidee

Das Ziel des Projektes war die Entwicklung eines Frameworks für mobile, orts-basierte Spiele. Das Framework sollte den Kontrollfluss eines Spiels vorgeben und typische Abläufe abbilden. Aus diesen Gründen wurde für mögliche von dem Framework unterstützte pervasive Spiele das bekannte Geläu-

despiel *Schnitzeljagd* als eine Metaphor definiert, um die mögliche Funktionalität des Frameworks abzugrenzen und es nicht zu abstrakt zu machen. Unter diese Metapher fallen solche Spielausprägungen wie Rallyes (z.B. Stadt-Rallye), Städte- und Museen-Erkundungstouren, Krimis in der Art von Cluedo und natürlich das Spiel „Schnitzeljagd“ selbst. Alle diese Spiele haben ähnliche Abläufe, die im Folgenden beschrieben werden.

Das Spiel soll in Teams gespielt werden, die aus einem oder mehreren Spielern bestehen können. Die Grundlage für Spiele bildet die Lösung einer Kette von mehreren orts-gebundenen Aufgaben in Form von voneinander unabhängigen Fragen oder Teilen eines großen Gesamt-Rätsels. Der allgemeine Spielablauf soll auf folgenden Prinzipien basieren:

- Jedes Team bekommt eine Aufgaben-Beschreibung (Ortskoordinaten, Fragen etc.) - eine Spiel-Route.

- Jedes Team hat ein Journal, in dem gelöste Aufgaben und Lösungen protokolliert werden - eine Ergebnis-Datei.

- Zu jedem Zeitpunkt ist einem Spieler nur eine Aufgabe zugänglich (sichtbar).

- Jede Aufgabe ist orts-basiert. Das bedeutet, sie kann nur an einem bestimmten in der Aufgabe vorgegeben Ort gelöst werden.

- Für jede gelöste Aufgabe werden, abhängig von der Korrektheit der Antworten, Leistungspunkte vergeben.

- Wenn eine Aufgabe gelöst wurde, wird dem Spieler eine neue Aufgabe gestellt. Die Richtigkeit der Lösung kann dabei keine Rolle spielen.

- Jede Aufgabe besteht aus mehreren Teil-Abschnitten: 1. Zu einem bestimmten in der Aufgabe vorgegebenen Ort ankommen; 2. Eine „Location“-basierte Frage beantworten, die nur an diesem Ort für den Spieler sichtbar wird; 3. Die Lösung in die Result-Datei eintragen; 4. Neue Aufgabe bekommen, für die jetzt eine Freigabe erteilt wird.

Wenn ein Team aus mehreren Mitgliedern besteht, können die Aufgaben unter den Teammitgliedern aufgeteilt werden. Darüber hinaus können zusätzliche orts-abhängige Ad-hoc-Aufgaben (Quests) an die Spieler gestellt werden, die die Spieler nicht gezwungen sind, zu beantworten, aber die Bonus-Punkte bringen. Das Ziel des Spiels ist, die meisten Aufgaben zu lösen und / oder schneller als die anderen Teams werden.

2.2 Allgemeiner System-Aufbau

Das Framework wurde für pervasive Spiele entwickelt, die auf einer Client-Server-Architektur basieren. Als Clients agieren dabei mobile Geräte wie PDA oder SmartPhones und als Server ein oder mehrere stationäre Rechner. Der allgemeine System-Aufbau ist auf der Abbildung 1 schematisch dar-

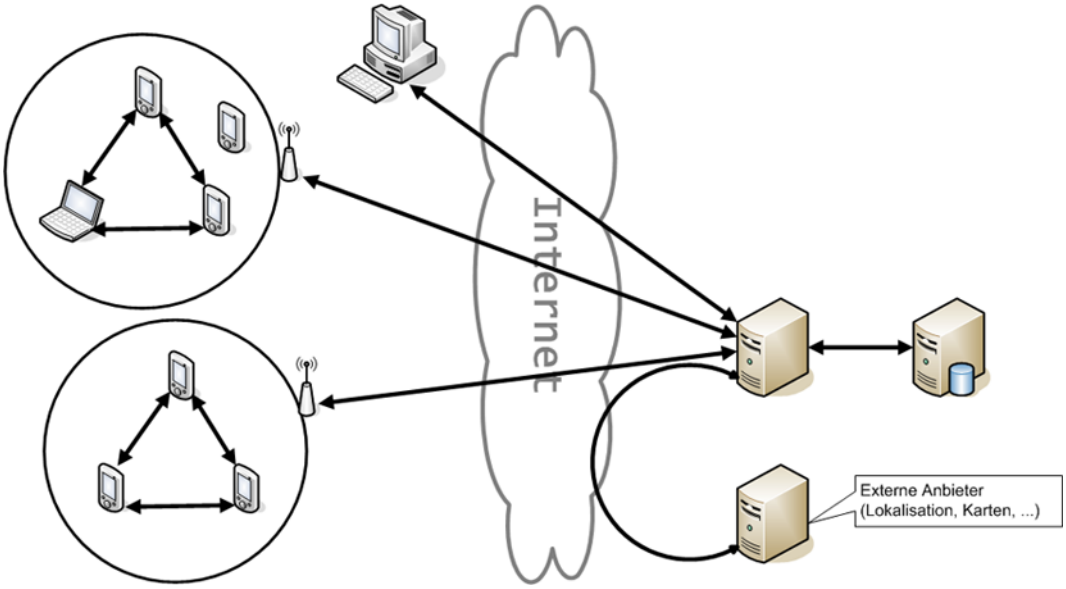

Abbildung 1: System-Aufbau

gestellt. Das Framework besteht folglich aus zwei Teilen, die miteinander agieren: einem serverseitigen und einem Client-seitigen Teil.

Mobile Clients haben die Möglichkeit zu einer drahtlosen Netzwerk-Verbindung zum Server. Nach dem aktuellen Stand der mobilen Kommunikation kann aber nicht von einer permanenten Verbindung zwischen dem Server und dem Client ausgegangen werden. Es kann zu Unterbrechungen kommen, da nicht überall ein frei zugänglicher Access-Point, so genannter Hotspot, zur Verfügung steht. Darüber hinaus ist ein Hand-held-Device wegen des hohen Energie-Verbrauchs nicht im Stande, auf längere Zeit Online zu sein. Aus diesen Gründen soll auf dem Client sowohl ein Online als auch ein Offline-Spielmodus möglich sein. Der Client soll fähig sein, das Spiel komplett ohne einer Netzwerk-Verbindung zu spielen. Die Kommunikation mit dem Server kann in diesem Fall auf das Laden der Route am Anfang des Spiels und das Laden der Ergebnisse am Ende des Spiels begrenzt werden, wenn zwischendurch keine weitere Möglichkeiten zu einer Verbindung gegeben werden. Diese Überlegungen wurden als Grundlage für die Bestimmung der Funktionalität des Servers und des Clients genommen, die im Folgenden beschrieben sind.
Funktionalität des Servers:

- Verwaltung von Spielen, Spielern und Teams (Registrierung, Highscore-Listen etc.)

- Verwaltung von Routen und Ergebnissen

- Kommunikation mit dem Client: An- und Abmelden des Spielers (Login, Logout), Runterladen der Route, Hochladen der Ergebnisse, Runterladen der Ad-hoc-Aufgaben

Funktionalität des Clients:

- Kontrolle des Spielablaufs

- Spielzustand-Sicherung (Fortsetzen des Spiels nach dem Neuanschalten des Gerätes)

- Positionsbestimmung

- Selbstkonfiguration (z.B. Energie-Sparmodus)

- Kommunikation mit dem Server

- Kommunikation mit anderen Clients: Spielstand-Synchronisation, zusätzliche Aktionen wie Chat etc.

Für die Kommunikation zwischen dem Client und dem Server wurden Web Service-Technologien gewählt. Für Web Services sprechen unter anderem solche Eigenschaften wie Interoperabilität, lose Kopplung der Dienste, etablierte Standards und die Möglichkeit zur dynamischen Lokalisation und Einbindung eines Dienstes [Hauser]. Die Verbindungs-Initiierung erfolgt dementsprechend immer von der Seite des Clients.

Allgemeiner Spielablauf

Der allgemeine Spielablauf kann in zwei wesentliche Abschnitte gegliedert werden: Vorbereitung des Spiels und das eigentliche Spielen. Für das Verständnis des Gesamt-Spielkonzeptes und der Funktionsweise des Frameworks werden diese Abschnitte im Folgenden ausführlich erläutert.

Die Spiel-Vorbereitung

Ein Spiel kann durch einen Benutzer angelegt werden. Dafür meldet sich der Benutzer über eine Web-Oberfläche im System an. Er kann aus einer Liste von Routen eine auswählen, auf deren Basis ein neues Spiel angelegt wird. Beim Erstellen des Spiels soll festgelegt werden, wann das Spiel starten soll, wie viele Personen maximal am Spiel teilnehmen können, ob das Spiel in Teams gespielt wird oder im Einzelspieler-Modus und wie viele Teams maximal gebildet werden können. Darüber hinaus wird der Teilnahme-Anmeldeschluss und der Zeitpunkt definiert, bis wann alle Ergebnisse von den Clients hochgeladen werden müssen (Spielende). Außerdem kann zusätzlich bestimmt werden, ob es ein öffentliches oder ein privates Spiel werden soll. Nach dem Anlegen des Spiels können andere Benutzer sich an diesem Spiel anmelden, z.B. über eine allgemeine Suche nach geöffneten Spielen oder über eine explizite Benachrichtigung durch den Spiel-Ersteller (z.B. Versenden des entsprechenden Web-Links). Die Anmeldung geschieht auch über die

Web-Oberfläche. Nach dem Anmeldeschluss können angemeldete Teilnehmer das Spiel auf ihre mobile Geräte herunterladen. Im Spiel-Objekt ist dabei die gesamte Route enthalten. Zudem wird bei einem Team-Spiel für jeden Team-Mitglied seine „eigene“ Routenpunkte vermerkt, die er ablaufen soll.

Das Spielen

Das Spiel beginnt auf dem Client zum vordefinierten Start-Zeitpunkt (Dann wird die erste Aufgabe für die Spieler sichtbar). Jeder Spieler bekommt einen für ihn festgelegten ersten Routenpunkt angezeigt. Er muss diesen Punkt aufsuchen und eine Aufgabe lösen, die erst beim Erreichen des Punktes angezeigt wird. Nach dem Lösen der Aufgabe wird der nächste Routenpunkt angezeigt usw. Während des Spiels kann der Spieler seine Zwischenergebnisse auf den Server hochladen oder zusätzliche Daten vom Server bekommen, z.B. Ergebnisse anderer Team-Mitglieder oder Bonus-Aufgaben, vorausgesetzt, er hat eine Internet-Verbindung. Wenn der Spieler die letzte Aufgabe gelöst hat, muss er sein Endergebnis auf den Server hochladen. Wenn alle Teilnehmer ihre Endergebnisse hochgeladen haben oder der vordefinierte Zeitpunkt für das Spielende erreicht wurde, ist das Spiel beendet. Der Server wertet dann die Spiel-Ergebnisse aus und aktualisiert die Highscore-Listen.

In weiteren Abschnitten wird das Design des Servers beschrieben und auf die Module ausführlich eingegangen, an deren Entwicklung die Autorin beteiligt war.

2.3 Server

Der Server besteht aus fünf wesentlichen Modulen: Context Manager, Game Manager, Data Manager, Administration Manager und Communication Layer (siehe Abbildung 2 auf Seite 9). Jedes Modul ist ein Bestandteil des Frameworks und an der Steuerung und / oder Umsetzung des Anwendungsablaufs beteiligt. In der Abbildung sind die Framework-seitige Teile des Servers als Rechtecke dargestellt. Darüber hinaus besitzen vier der Module Komponenten, die spezifisch für jede Anwendung (Spielart) implementiert werden müssen und dementsprechend nicht zum Framework gehören. Diese Teile sind im Bild als Ovale zu sehen.

Den Kern der serverseitigen Anwendung stellen der Context-Manager und der Game-Manager dar. Der Context-Manager ist für den Spielablauf maßgeblich verantwortlich. Er verwaltet Zustände für geöffnete und laufende Spiele und daran beteiligte Spieler. Darüber hinaus ist er für die Koordinierung des Verbindungsstatus zu den Clients (Sessions) zuständig. Der Game-Manager verwaltet fachliche Aspekte der Spiele, z.B. Anwendungs-spezifische Auswertung von Lösungen und Ergebnissen, die Punktevergabe etc. Da jede

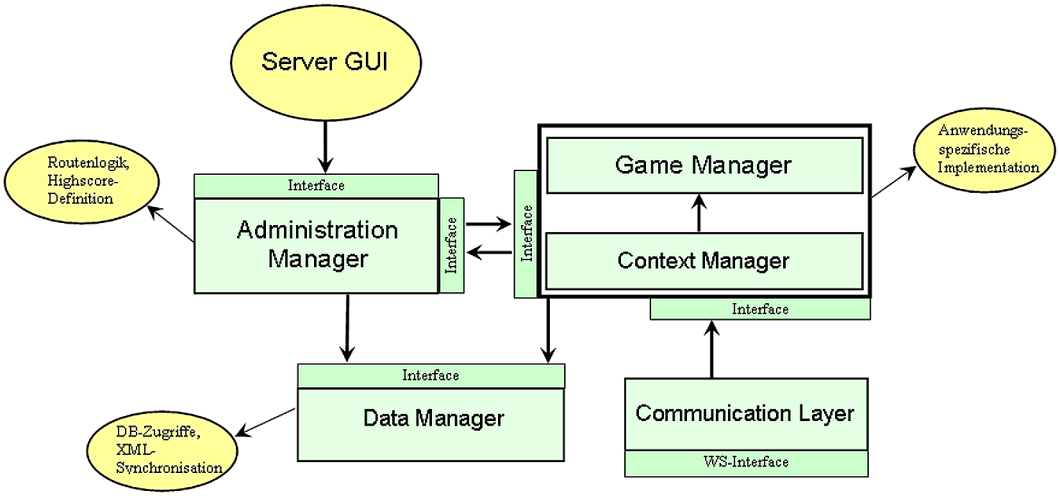

Abbildung 2: Server-Architektur

Spiel-Anwendung andere Ansprüche an diese fachlichen Aspekte haben kann, gehören sie nicht zum Framework, sondern sollen Anwendungs-spezifisch implementiert werden. Das Framework definiert nur eine Schnittstelle dafür.

Der Communication-Layer stellt eine Web Service-Schnittstelle für die Clients dar. Er arbeitet mit dem Context-Manager eng zusammen und ist ein Teil des Frameworks.

Die Verwaltung von Routen, Spielen und Spieler ist die Aufgabe des Administration-Managers. Darüber hinaus ist dieses Modul für die Rechte-Verwaltung und die Erstellung von Highscore-Listen zuständig. Es stellt eine Schnittstelle für eine serverseitige Web-Anwendung zur Verfügung (Server GUI in der Abbildung 2), über die sich unter anderem neue Spieler registrieren können, neue Routen ins System eingetragen werden können oder ein neues Spiel erstellt werden kann. Sowohl die Server GUI als auch die Interpretation der Routenlogik und die Highscore-Definition können sich von Spielart zu Spielart unterscheiden. Aus diesem Grund gehören diese Funktionalitäten nicht zum Framework und müssen für jede Anwendung extra umgesetzt werden.

Das letzte Modul des Frameworks ist der Data-Manager. Er beinhaltet unter anderem einen Datenbank-Abstraktionslayer und ist für die Synchronisation von Spielständen (vorläufigen und Endergebnissen) verantwortlich.

Eine ausführliche Schnittstellen-Beschreibung der einzelnen Module ist im Anhang auf Seite 15 zu finden.

2.3.1 Data-Manager

Zwischen dem Server und dem Client werden Daten ausgetauscht. Zu diesen Daten gehören in erster Linie Spiel-Routen und Ergebnisse der einzelnen Spieler. Dazu kommen zusätzlich Session-Daten, Spiel-bezogene Infos (Start- und End-Zeitpunkt, Spiel-Bezeichnung und Id etc.) und Client-spezifische Kontext-Informationen. Darüber hinaus müssen im System unter anderem Benutzerdaten, Spiele und Routen persistent gehalten werden.

Beim Entwurf des Frameworks wurde beschlossen, dass für Routen und Ergebnisse ein XML-basiertes Format benutzt wird. Dafür wurden eine Route Description Language und eine Result Description Language ausgearbeitet. XML ist ein etabliertes Format und erlaubt, Daten plattformunabhängig auszutauschen. Außerdem existieren für XML zahlreiche Werkzeuge, die die Transformation von Dokumenten aus einem XML-Format in ein anderes ermöglichen [XSLT, Geeb]. Das erhöht die Flexibilität und Anpassbarkeit des Frameworks im Bezug auf verschiedene Formate für Routen und Resultate. Z.B. könnten Geocaching-Spiele [Geo], die das GPX-Format [GPX] benutzen, mit dem Framework auch entwickelt werden.

Da die im System beinhalteten Daten sich nicht auf XML-Routen und -Ergebnisse begrenzen, wurde eine Datenbank-basierte Lösung für die Daten-Speicherung gewählt. Das im Rahmen des Projektes entwickelte Datenbank-Schema gehört zu den Bestandteilen des Frameworks und erlaubt das Speichern von solchen standardmäßigen Daten wie Benutzer-Informationen, Spiele etc. (siehe DB-Schema in der Abbildung 3). Das Schema kann aber nach Bedarf der jeweiligen Spiel-Anwendung erweitert werden.

Routen und Ergebnisse werden in der Datenbank als ganze Dateien gespeichert. Das erspart das unnötige Mapping der XML-Dateien auf die DB-Entities und zurück, da diese Daten ohnehin in Form von XML-Dateien zwischen dem Server und den Clients ausgetauscht werden. Außerdem macht es das Framework flexibler: Die Datenbank-Schema braucht nicht bei jeder von dem vorgegebenen Konzept abweichenden Spiel-Anwendung angepasst werden.

Beim Spielen in Teams können Aufgaben innerhalb eines Teams unter den Mitgliedern aufgeteilt werden. Da aber jedes Team nur eine Ergebnis-Datei besitzt, müssen die einzeln erbrachten Resultate am Ende des Spiels oder auch zwischendurch auf dem Server zusammengeführt werden. Diese Aufgabe wird von dem Data-Manager übernommen. Darüber hinaus bildet der Data-Manager eine Abstraktionsschicht für die Datenbank-Zugriffe und ist für das Mapping zwischen den DB-Entities und den Modell-Objekten verantwortlich. Im Folgenden werden die für diese Aufgaben ausgewählten und im Framework eingesetzten Strategien beschrieben.

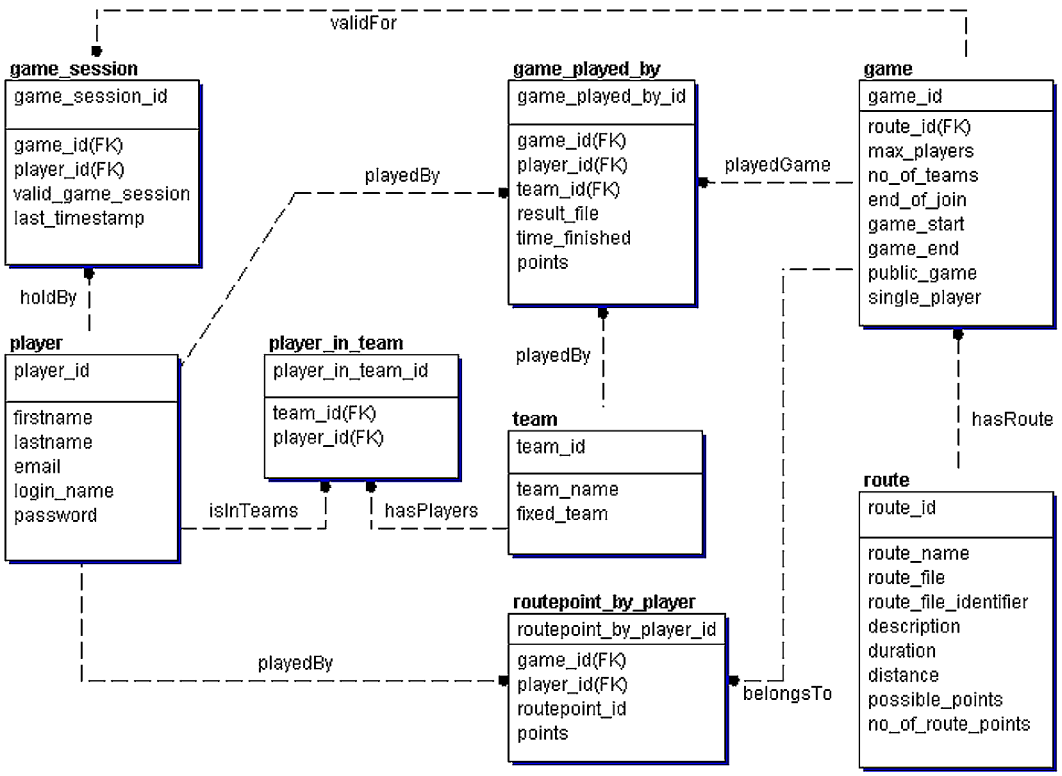

Abbildung 3: Datenbank-Schema

DB-Zugriffe

Für die Lösung der Datenbank-Zugriffe und des Mappings standen am Anfang der Entwicklung zwei Möglichkeiten zur Verfügung: Nutzung eines fertigen Persistenz-Tools oder eigene Umsetzung. Als ein fertiges Produkt war fürs Projekt das Objekt-relationale Mapping-Tool OpenAccess der Firma Vanatec [OpenAccess] verfügbar. Dieses Produkt verspricht unter anderem transparente Persistenz und Datenbanken-Flexibilität. Außerdem ermöglicht OpenAccess die Integrierung von schon bestehenden Datenbank-Schemas in das Objekt-Modell und das Arbeiten in einem verbindungslosen Modus (durch den Einsatz des so genannten ObjectContainers). Das Tool bietet also viele Möglichkeiten. Durch den Einsatz von XML-Dateien für Routen und Ergebnisse ist aber das Datenbank-Schema des Frameworks überschaubar geblieben (siehe Abbildung 3). Dazu kommt noch, dass nicht alle DB-Entities eins-zu-eins auf Objekte abgebildet werden müssen. Bei einigen Modell-Klassen werden nur ihre bestimmte Attribute gebraucht, so z.B. bei der Suche nach existierenden Routen (bei der Spiel-Erstellung) werden

nicht alle Route-Daten benötigt. Es werden solche Attribute wie Id, Name und Beschreibung der Route übertragen, aber keine Route-Datei, sonst kann der Daten-Volumen der Suchergebnisse zu groß werden. Darüber hinaus ist die Einarbeitung in ein neues unbekanntes Persistenz-Tool immer mit großem Zeitaufwand verbunden. Aus diesen Gründen wurde die Entscheidung getroffen, den Tool nicht einzusetzen, sondern eine eigene Lösung zu entwickeln.

Die Datenbank-Zugriffe und das Objekt-Relationale Mapping wurden in die Klasse *DBConnector* ausgelagert (siehe Klassendiagramm im Anhang, Seite 18). Diese Klasse ist ein Bestandteil des Frameworks und bietet standardmäßige, dem allgemeinen Spielablauf entsprechende Funktionalitäten an. Für die Erweiterung oder Anpassung dieser Funktionalität steht beim eventuellen Bedarf (z.B. bei zusätzlichen Ansprüchen einer Spiel-Anwendung) eine Schnittstelle zur Verfügung.

Synchronisation von Dokumenten

Zu den wichtigsten Problemen beim Editieren von XML-Dokumenten durch mehrere Personen gehören Konflikte, die bei der Zusammenführung von unterschiedlichen Versionen entstehen können. Es existieren zahlreiche Lösungen für diese Probleme (siehe [LaFontaine, Neyem]).

In Spielen, die mit Hilfe des Frameworks entwickelt wurden, können verschiedene Versionen der Ergebnis-Dateien beim Modifizieren durch mehrere Teammitglieder entstehen. Da aber am Anfang des Spiels die Aufgaben zwischen den Spielern disjunkt aufgeteilt werden und jede Aufgabe bzw. jeder Routenpunkt eine eindeutige Identifizierung besitzt, kann es zu keinen Konflikten kommen: Jeder Spieler bearbeitet eigene Aufgaben und hat keine Möglichkeit, auf die anderen zuzugreifen. Dadurch entfällt die Notwendigkeit, eine Strategie zur Konflikten-Behebung auszuarbeiten.

Die Merging-Funktionalität beinhaltet die Klasse *XmlSynchronizer* (siehe Anhang, Seite 18). Die Zusammenführung von verschiedenen Versionen geschieht durch die Auswertung der jeweiligen XML-Bäume. Beim ersten Hochladen des Ergebnisses durch einen Teammitglied, wenn noch keine Version auf dem Server vorhanden ist, wird seine Version als Haupt-Dokument in die Datenbank gespeichert. Bei allen nachfolgenden Versionen werden nur die Punkte in diese Datei geschrieben, die dort noch nicht enthalten sind. *XmlSynchronizer* stellt die Funktionalität zur Auswertung von Resultaten im Standard-Format, das für das Framework definiert wurde (Result Description Language). Außerdem wird eine einfache Berechnung von Leistungspunkten angeboten. Wenn aber eine Spiel-Anwendung ein eigenes Ergebnis-Format erfordert, muss die entsprechende Auswertungslogik extra von jeweiligen An-

wender implementiert werden. Dafür steht im Framework ein Interface zur Verfügung.

2.3.2 Communication Layer

Wie es schon beschrieben wurde, erfolgt die Kommunikation mit dem Client über Web Services. Der allgemeine Kommunikationsablauf kann folgendermaßen beschrieben werden (Die ausführliche Beschreibung der Schnittstelle steht im Anhang, Seite 15):

- Anmeldung beim Server.
- Holen aller Spiele, zu denen der Spieler sich angemeldet hat (über den Web-Interface) und bei denen der Anmeldefrist schon abgelaufen ist.
- Zwischenergebnisse auf den Server hochladen, wenn die Gelegenheit dazu da ist.
- Beim Beenden des Spiels das Endergebnis auf den Server hochladen.

Darüber hinaus kann der Spieler das Spiel abbrechen. Dabei wird das Spiel für ihn wie oben beendet, aber ohne der Auswertung der bisher erbrachten Ergebnisse.

3 Fazit und Ausblick

Das Projekt kann in zwei Phasen unterteilt werden. In der ersten Phase wurde das Konzept des Frameworks ausgearbeitet, im zweiten Teil wurde die Realisierung dieses Konzeptes durchgeführt. Dabei fand parallele Umsetzung sowohl des Frameworks wie auch des prototypischen Spiels statt. An dieser Stelle kann als ein Kritikpunkt genannt werden, dass die Konzept-Ausarbeitung zu lange gedauert hat, was als Folge Einbußen in der Realisierung hatte. So wurden z.B. der Administration-Manager und Server GUI aus zeitlichen Gründen nicht implementiert. Als Prototyp wurde das Spiel „Schnitzeljagd“ entwickelt. Obwohl das Spiel erfolgreich getestet wurde, konnte auch hier nicht alles so realisiert werden, wie es geplant war. Z.B. wurde die Funktionalität des Game-Managers nur teilweise umgesetzt.

Für weitere Entwicklung des Frameworks kann die Implementierung des Administration-Managers vorgeschlagen werden. Außerdem könnten prototypisch Server GUI und Anwendungs-spezifischer Teil des Game-Managers realisiert werden. Als weitere Ausbau des Data-Managers könnte ein Konzept für das Einbinden von „fremden“ Route- und Result-Formaten in das Framework entwickelt werden, z.B. über den Einsatz von XSLT.

Im Ganzen kann das Pervasive Gaming Projekt als erfolgreich bewertet werden.

Literatur

[Cron] Cron, ein Pervasive Gaming Projekt an TU Berlin, URL: http://www.oks.cs.tu-berlin.de/forschung/pervasive-gaming, Stand: Februar 2007

[LaFontaine] Robin La Fontaine, Nigel Whitaker, „A Generalized Grammar for Three-way XML“, XML Conference Proceeding by RenderX, 2005

[Geeb] Franziskus Geeb, „Das XML/XSLT-Seminar: Einführung für Studium und Beruf“, 2003, ISBN: 3-934424-13-9

[Geo] Geocaching-Portal für den deutschsprachigen Raum, URL: http://www.geocaching.de/, Stand: Februar 2007

[GPX] GPS Exchange Format, URL: http://www.topografix.com/gpx.asp, Stand: Februar 2007

[Hauser] Tobias Hauser, Ulrich M. Löwer, „Web Services: Die Standards“, 2004, ISBN: 3-89842-393-X

[IPerG] Integrated Project on Pervasive Gaming, URL: http://iperg.sics.se, Stand: Februar 2007

[Jegers] Kalle Jegers, Mikael Wiberg, „Pervasive gaming in the everyday world“, in: Pervasive Computing, IEEE, Volume: 5 , Issue: 1, Jan.-March 2006

[OpenAccess] Vanatec OpenAccess: Object-relational Mapping for .NET, URL: http://www.vanatec.com/en/product-information, Stand: Februar 2007

[Mattern] Friedemann Mattern, „Pervasive Computing/ Ubiquitous Computing“, Informatik-Spektrum, Vol. 24, No. 3, pp. 145-147, June 2001

[MobileChase] MobileChase, Pervasive Gaming Projekt an Fraunhofer Institut für Grafische Datenverarbeitung IGD, URL: http://www.igd.fhg.de/igd-a5/imga5/projects/mobilechase/MobileChase_flyer.pdf, Stand: Februar 2007

[Neyem] Andrés Neyem et al., „A Strategy to Share Documents in MANETs using Mobile Devices“, in the 8th International Conference Advanced Communication Technology (ICACT'06), 2006

[XSLT] XSL Transformations (XSLT) Version 2.0, URL: http://www.w3.org/TR/2006/PR-xslt20-20061121, Stand: Februar 2007

A Anhang

A.1 Definition der Server-seitigen Schnittstellen

Administration Manager

GUI Interface

sessionId login(username, password)

List<Route> getRouteList(searchCriteria)
Route-Objekt enthält Id, Name und Description

gameId createGame(routeId, maxPlayer, teamCount, endOfJoin, gameStart, gameEnd, isPublic)

List<Game> getGameList(sessionId)
Liste mit aktuell erstellten Multiplayer-Spielen, deren Anmeldeschluss noch nicht abgelaufen ist.

status joinGame(gameId, teamId, playerId)
Anmelden bei einem Multiplayer-Spiel, dessen Anmeldeschluss noch nicht abgelaufen ist.

Game & Context Manager Interface

sessionId loginPlayer(playerName, password)

Communication Layer

WS-Interface

sessionId login(username, password, context)

status logout(context, sessionId)

Game[] getJoinedGames(context, sessionId)
Geöffnete Spiele, für die der Anmeldeschluss schon abgelaufen ist.

status submitTempResult(resultXML, context, gameSessionId)
Hochladen der aktuellen Ergebnisse vom Client auf den Server

void submitEndResult(resultXML, context, gameSessionId)
Hochladen des Endergebnisses vom Client auf den Server

Data getData(context, gameSessionId)
Herunterladen der aktuellen Ergebnisse (von anderen Spielern) oder / und zusätzlichen Aufgaben vom Server.

void abortGame(sessionId)
Abbruch des Spiels

Data Manager

Player getPlayer(playerName, password)

List<Route> getRouteList(searchCriteria)
Route-Objekt enthält Id, Name und Description

gameId createGame(routeId, maxPlayer, teamCount, endOfJoin, gameStart, gameEnd, isPublic)

List<Game> getGameList(sessionId)
Liste mit aktuell erstellten Multiplayer-Spielen, deren Anmeldeschluss noch nicht abgelaufen ist.

status joinGame(gameId)
Anmelden bei einem Multiplayer-Spiel, dessen Anmeldeschluss noch nicht abgelaufen ist.

List<Game> getJoinedGames(sessionId)
Liste mit den geöffneten Spielen, für die der Player angemeldet ist und für die der Anmeldeschluss schon abgelaufen ist. Game-Objekt enthält Route-XML und zusätzliche Informationen, wie Beschreibung des Spiels und Route-Punkte des Spielers.

ResultXML getResultFile(gameId)
XML-Datei mit aktuellen Spielergebnissen.

RoutePointXML getQuestFile(gameId)

status synchroniseResult(ResultXML)
Synchronisierung der Ergebnisse mit den bereits auf dem Server vorhandenen Ergebnissen.

status persistGameResult(ResultXML)
Speicherung des synchronisierten Ergebnisses.

void endGameForPlayer(playerId)
Speicherung des Spielendes für den aktuellen Player.

void persistFinishedGame(gameId)
Speicherung der nötigen Informationen für das beendete Spiel.

Context Manager

Communication Layer Interface

sessionId loginPlayer(username, password)

status logoutPlayer(sessionId)

List<Game> getJoinedGames(sessionId)

Geöffnete Spiele, für die der Anmeldeschluss schon abgelaufen ist. Game = RouteXML, personalRoutePoints, ...

status submitTempResult(resultXML, gameSessionId)
Hochladen der aktuellen Ergebnisse vom Client auf den Server

Data getData(gameSessionId)
Herunterladen der aktuellen Ergebnisse (von anderen Spielern) vom Server.

status submitEndResult(resultXML, gameSessionId)
Hochladen des Endergebnisses vom Client auf den Server

status abortGame(gameSessionId)
Abbruch des Spiels

Administration Manager Interface

List<Route> getRouteList(searchCriteria)
Route-Objekt enthält Id, Name und Description

gameId createGame(routeId, maxPlayer, teamCount, endOfJoin, gameStart, gameEnd, isPublic)
Erzeugen (Anlegen) eines neuen Spiels.

List<Game> getGameList(sessionId)
Liste mit aktuell erstellten Multiplayer-Spielen, deren Anmeldeschluss noch nicht abgelaufen ist.

status joinGame(gameId, teamId, playerId)
Anmelden bei einem Multiplayer-Spiel, dessen Anmeldeschluss noch nicht abgelaufen ist.

Game Manager

List<Route> getRouteList(...)

gameId createGame(...)

List<RouteXML> getJoinedGames(...)

status submitTempResult(resultXML, gameSessionId)

status submitEndResult(resultXML, gameSessionId)

Data getData(gameSessionId)

void finishGame(gameId)

void abortGame(gameId)

A.2 PGF-Klassendiagramm

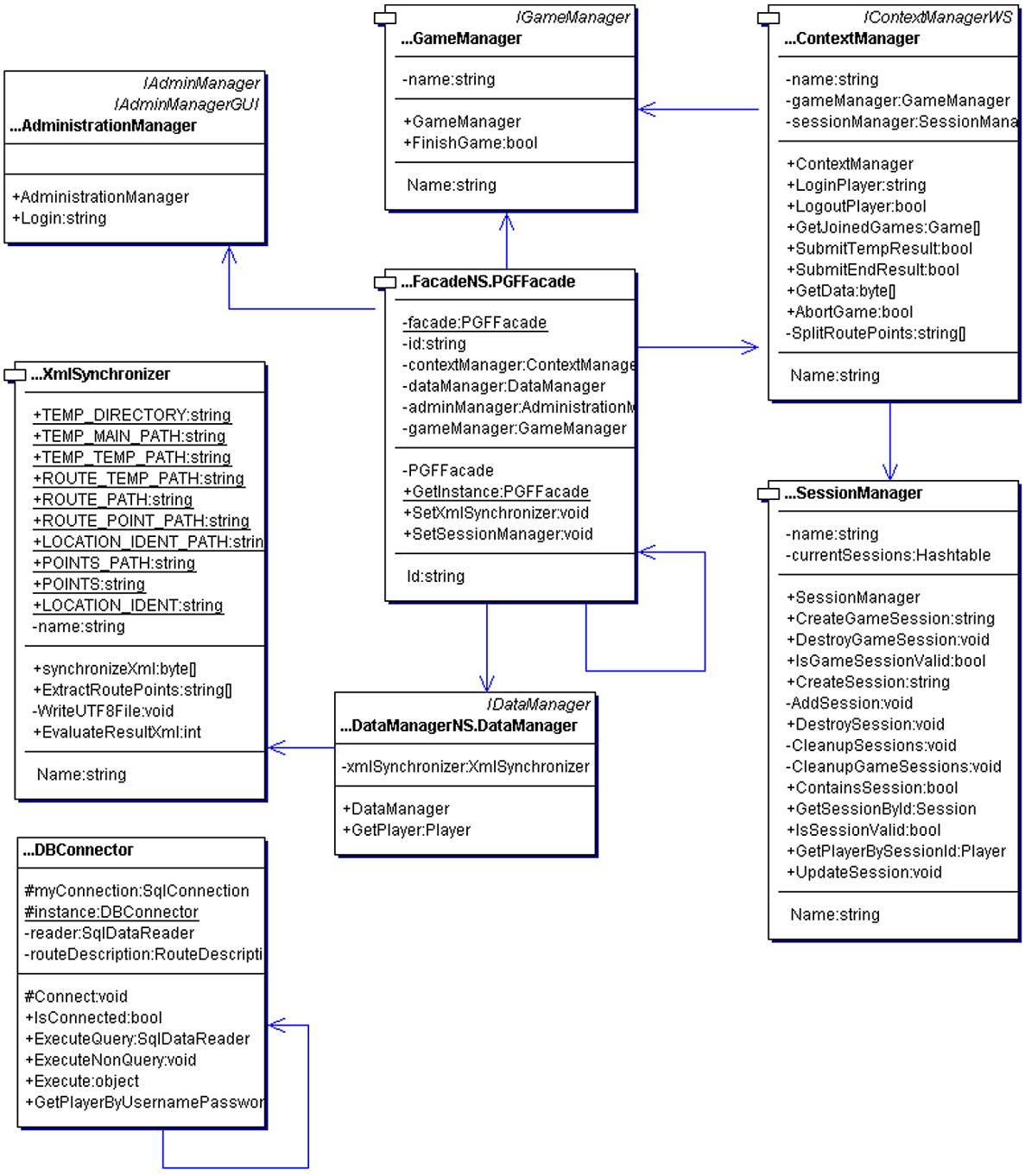

Abbildung 4: PGF: Klassendiagramm