Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Tobias Hutzler

Pervasive Gaming Framework

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tobias Hutzler

Pervasive Gaming Framework

Projektebericht im Rahmen des Masterprojektes
im Studiengang Informatik (Master of Science)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Prüfer : Prof. Dr. Olaf Zukunft

Abgegeben am 8. März 2009

Inhaltsverzeichnis

1 Einleitung

Dieser Projektbericht dient der Dokumentation des Masterprojetkes „Pervasive Gaming Framework“ an der HAW Hamburg im Masterstudiengang Informatik im Wintersemester 2008/2009.

Kaum eine Technologie ausser der des Mobiletelfons hat die Arbeitswelt und den Alltag in den letzten Jahren so stark geprägt. Sich kurz von unterwegs aus per Telefonat oder Textnachricht zu verabreden wird heute als selbstverständlich angesehen. Auffallend ist die zunehmende mobile Nutzung von Internet-Diensten oder Email. Neue Geräte wie das Apple iPhone [1] oder das Google Phone [2], sowie bezahlbare Tarife machen diesen Trend zur allgegenwärtigen und mobilen Kommunikation immer attraktiver.
Die enorme Nachfrage nach Spielen erzeugt einen nicht abreisenden Bedarf nach neuen Trends in der Informatik. Durch interessante und herausforderne Spielkonzepte und Technologien hat das Computerspielen, ehemals nur in Kinderzimmern angesiedelt, die Wohnzimmer bereits erobert. Die Eroberung andere Bereiche unseres Lebens ist nur noch eine Frage der Zeit.
Durch Kombination dieser beiden Disziplinen können ganz neue Formen des Spielens geschaffen werden. Diese Spielformate werden als „Pervasive Games“ bezeichnet, welche „*die reale Umgebung mit virtuellen Elementen verbindet und dabei andere Aktivitäten zeitlich, räumlich und sozial durchdringt*"[Pri06].

Diesen sehr interssanten und zukunftträchtigen Bereich untersuchten acht Studenten der HAW Hamburg im Masterprojekt „Pervasive Gaming Framework“. Intention dabei ist es nicht ein produktivfähiges kommerzielles Produkt zu entwickeln, sondern vielmehr die technologische Herausforderung anzunehmen, ein Framework für pervasive Spielen zu entwickeln. Damit sollen Erfahrungen auf dem noch nicht tief genug durchdrungenen Feld des *Pervasive Gaming* gewonnen werden. Beides, die Entwicklung einer Anwendung und das dazugehörigen Framework zu entwerfen und implementieren, dienen dem Erfahrungsgewinn diese Feldes und ermöglichen es den Studenten praxis- und teamorientiert zu arbeiten.

[1] iPhone http://www.apple.com/de/iphone/ referenziert am 8.3.2009
[2] G1 http://www.t-mobile.de/g1/ referenziert am 8.3.2009

Die vorliegende Arbeit dient als Abschlussbericht des Projektes. Es werden das Projekt, Projektumfeld, Vorgehen und wichtige Artefarkte des Softwareentwicklungsprozesses festgehalten. Das Projekt organisierte sich in einer Gruppe und der hier resultierende Bericht gibt keine vollständige Dokumentation der gewonnen Erkenntnisse und erarbeiteten Ergebnisse wieder, sondern legt klar den Schwerpunkt auf die Beiträge des Autors zu diesem Projekt.[3]

[3] Die Projektberichte der anderen Teilnehmern können auf HAW http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master08-09-proj/berichte.html abgerufen werden.

2 Projekt

2.1 Projektziel

Ziel des Projektes ist die Entwicklung eines Frameworks für mobile Spieleanwendungen. Das Framework soll die daraus resultierenden Anfordungen pervasiver Spiele, Mehrbenutzerfähigkeit, mobile Persistenz, ortsabhängige Dienste und die Kommunikaktion zwischen den mobilen Teilnehmern über ein zentrales Backend unterstützen. Die daraus resultierende Architektur lässt sich grob in eine Serverkomponente und eine Clientkomponenten aufteilen, wobei die Clientkompontene auf den mobilen Endgeräten ausgeführt wird. Als Plattform für die mobilen Clients wird Android [1]eingesetzt.

Die technischen Aspekte des Frameworks beinhalten den Kommunikationslayer, den Positionierungslayer und einen Persistenzlayer auf dem mobilen Gerät. An letzeres wurde der Anspurch erhoben, eine möglichst komfortable Abstraktionsschicht für die Anwendungsentwickler mobiler Spieleanwendungen zu entwicklen.

Am Ende des Projektes soll ein lauffähiges Produkt entstehen und die gewonnenen Ergebnisse in Form eines Projektberichtes dokumentiert werden.

2.2 Projektumfeld

Das Projekt fand im Rahmen des Masterstudiengangs Informatik an der Hochschule für Angewandte Wissenschaften Hamburg statt. Die zeitlichen und finanziellen Rahmenbedingungen sind hierfür vorgegeben. Das Projektteam setzt sich aus acht Studierenden zusammen. Hierbei wird eine hoher Wert auf die kooperative Zielerfüllung des Projektes gelegt. Wichtige Entscheidungen wurden in der Gruppe getroffen und aktuelle Ergebnisse an diese berichtet. Die Entwicklung wird in einzelne Aufgabenpakte aufgeteilt und von einzelnen Studenten oder kleineren Teams bearbeitet.

[1] Android http://code.google.com/android/ referenziert am 8.3.2009

2.3 Vorgehen

Ein frühes Treffen, welches bereits am Ende des letzen Semsters stattfand, ermöglichte es bereits den Projektteilnehmern Interessensgebiete und Schwerpunkte mit den anderen Teamteilnehmern abzusprechen. Dies führte zu einer möglichst frühen Konkretisierung der Version und festigte das gemeinsame Projektziel. Die Aufteilung und Definition in zwei Anwendungsgruppen und eine Frameworkgruppe der Projektteilnehmer ermöglichtete frühzeitig eine klare Aufgabenverteilung.
Durch eine erste Anforderungsanalyse der Anwendungsgruppen war es möglich die technischen Anforderungen an das Framework abzuleiten. Parallel dazu erfolgte seitens der Frameworkgruppe die Einarbeitung in Android. Der ausgezeichnete Umgang mit Wissen untereinander ermöglichtete es den anderen Teamteilnehmern die gewonnen Erkenntnisse sofort und effektiv einzusetzen. Nach der Anforderungsanalyse konnte ein Entwurf der Komponenten festgelegt werden. Abschließend wurde unter Berücksichtigung der Erkenntnisse der Einarbeitungszeit, der Analyse- und Entwurfsphase die Implementierung angegangen.
Die Planung und Einhaltung der Fristen sowie die Qualität der Ergebnisse wurden seitens der Projektleitung gesteuert und kontrolliert.

3 Analyse

Da die Rolle des Autors klar in der Frameworkerstellung definiert ist, wird sich die vorliegende Analyse auf diesen Teilbereich des Projektes beziehen. Daneben war der Autor auch maßgeblich an der Analyse und Entwurf der Gesamtarchitektur des Frameworks und der Kommunikationskomponenten beteiligt [1]. Es wird allgemein auf die technischen Anforderungen an das Framework eingegangen und konkret die mobile Persistenzschicht vorgestellt.

3.1 Anfoderungsanalyse

Die Anfoderungsanalyse der Anwendungsprojektgruppen definiert die hier aufgeführten technischen Anforderungen an das zu entwickelnde Framework.

3.1.1 Kommunikation

Die Kommunikationskomponente wurde als klar benötigte Instanz für mobile Spiele erkannt. Aufgrund der fachlichen Anforderungen ist es nötig, dass die Spielteilnehmer für verschiedenste Szenarien und Abläufe Daten untereinander austauschen können.

3.1.2 Positionierung

Eines der wichtigsten Attribute pervasiver Spiele sind Aufgaben, die ortsabhängig durchgeführt werden. Hierzu ist es entscheidet sowohl die eigene Position als auch die Position der anderen Spielteilnehmer in zeitlichen Abständen zu erfahren.

[1] Einen Überblick über die Framworkarchitektur und die Kommunikationskomponenten gibt Peter Salchow http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master08-09-proj/berichte.html

3.1.3 Persistenz

Daten zu speichern ist eine ganz grundsätzliche Anforderung. Spielstände müssen geladen werden und nach dem Beenden der Anwendung in einen nicht flüchtigen Speicher zurückgeschrieben werden. Da mobile Anwendungen nicht immer über ständige Konnektivität zu einer zentralen Instanz verfügen, müssen die Daten auf dem mobilen Endgerät festgehalten werden. Auch andere Szenarien sind denkbar, wie beispielsweise eine Peer-to-Peer Architektur in der die einzelnen Teilnehmer zugleich Client als auch Server sind.

Konkret wurden folgenden Anforderungen an das Frameworkteam für die Persistenzschicht auf den Clients gestellt:

Datenbank erstellen Die Persistenzschicht soll die Erstellung einer Datenbank aus den fachlichen zu persistierenden Klassen erzeugen. Über eine definierte Beschreibung, der zu persistierenden Klassen, soll das nötige Datenbankschema generiert werden.

Datenbank zurücksetzen/löschen Besonders in der Entwicklungsphase, aber auch für einen späteren möglichen Einsatz der Anwendung ist es gewünscht, die Datenbank in ihren Ursprungszustand zurückzusetzen. Dabei sollen alle bisherigen Tabellen inklusive ihrer Daten gelöscht werden.

Objekte speichern Es muss eine Schnittstelle definiert sein, um Objekte in die Datenbank zu speichern.

Objekte laden Eine weitere Schnittstelle ist festzulegen, um Objekte aus der Datenbank in die Anwendung zu laden. Dieser Anwendungsfall ist nach Absprache mit dem Anwendungsteam sehr einfach gehalten. Für die Umsetzung reichte das Laden einer Instanz durch Suchen nach einer eindeutigen „Id“. Mächtige Datenbankabfragen waren nicht gefordert und hätten den zeitlichen Rahmen gesprengt.

Objekte ändern Die Änderung der Objekte, die bereits in der Datenbank existieren, muss ebenfalls erfüllt sein. Es werden die eindeutige „Id“ und das geänderte Objekte übergeben.

Objekte löschen Das Löschen eines Objektes aus der Datenbank ist ebenfalls gewünscht. HIerbei wird die eindeutige „Id“ übergeben.

4 Entwurf

Dieses Kapitel beinhaltet den Entwurf und die Entscheidungen die zu der Architektur der Persistenzschicht auf den mobilen Clients geführt hat. Im Folgenden wird auf die allgemeine Architekturentscheidung und bestehenden Alternativen eingegangen. Anschließend wird konkret das zugrunde liegende Design der Komponente angegeben.

4.1 Architekurentscheidung

Ziel war es eine komfortable Abstraktionsschicht für die Datenbank den Spieleentwicklern seitens des Frameworks zur Verfügung zu stellen. Diese Abstraktionsschicht sollte die in der Analyse erkannten Anforderungen der Anwendungsentwickler abdecken.

Die Rahmenbedingungen waren durch die verwendetete mobile Plattform „Android“ [1]und der dort eingesetzten Datenbank SQLLite vorgegeben.

Android bietet die Möglichkeit in objektorientierter Manier javabasierte Anwendungen zu entwickeln. Die Entscheidung Android als Technologie für das Projekt einzusetzen wurde zu einem sehr frühen Zeitpunkt in der Projektphase beschlossen.
Aufgrund des „Impedance Mismatch“ zwischen der objektorientierten Welt und relationalen Datenbanken [Rus08], als auch aus gewonnen Erfahrungen komponentenbasierter Architekturen ist die Designentscheidung auf die Umsetzung eines OR-Mappers als Abstraktionsschicht gefallen. Für die Anwendungsentwickler hat dies im Allgemeinen den Vorteil die objektorientierte Welt bei der Implementierung der Anwendungslogik nicht verlassen zu müssen. Tiefes Wissen über datenbankspezifischen Sachverhalten bleibt so für den Entwickler transparent. Konkret auf das Projekt und Android bezogen soll es möglich sein die SQLLite-Komponente für die Entwickler zu kapseln. Anstelle von SQL-Abfragen und SQL-Ausdrücken sollen einfache Servicemethoden die Handhabung mit der Persitenzschicht anbieten.

[1] Android http://code.google.com/android/ referenziert am 8.3.2009

4.1.1 Alternativen

Als Alternative könnte man auch frei verfügbare Frameworks wie Hibernate [2] in Android integrieren. Von dieser Entwurfalternative hat der Autor aber aus folgenden Gründen abgesehen:

Da der Android-Stack keine vollständige JRE zur Verfügung stellt, war die Integration von Hibernate in Android ein zu hohes Risiko. Faktoren waren der zeitliche Rahmen und Umfang des Quellcodes von Hibernate.
Hibernate ist ein sehr umfangreicher und mächtiger OR-Mapper, die komplette Funktionalität und Kompatibilität zu verschiedenen Datenbanktypen und SQL-Dialekten wird nicht benötigt. Weiterhin verfolgt der Autor eine schlanke Lösung prototypisch umzusetzen.

4.1.2 Entscheidungsergebnis

Die Entscheidung zwischen Eigenimplementation eines schlanken OR-Mappers und Integration eines freiverfügbaren OR-Mappers ist klar auf die eigene Umsetzung gefallen.

4.2 Entwurf der Komponente

Beim Entwurf eines OR-Mappers gibt es einige Sachverhalte zu betrachten. Eine gute theoretische Zusammenfassung und Überblick gibt Russell in [Rus08]. Im Folgenden werden die verwendeten Techniken und Strategien und deren Alternativen vorgestellt.

4.2.1 Mapping

In objektorientieren Programmiersprachen wird Vererbung eingesetzt. Vererbung ist eine Beziehung zwischen zwei Klassen, in der die eine Klasse die andere Klasse spezifiziert. Um Vererbung der fachlichen Klassen abzubilden, gibt es verschiedene Strategien. Hier werden die drei Gängigsten kurz vorgestellt:

single-table strategy Bei der *single-table strategy* werden alle Klassen einer Vererbungshierarchie in einer einzigen Datenbanktabelle abgebildet. Die Datenbanktabelle beinhaltet für jedes Feld der Klasse eine eigene Spalte. Nachteil neben dem Umstand, dass bestimmte Felder auch den Wert „null“ annehmen können. Eine redundanzfreie

[2] Hibernate http://www.hibernate.org/ referenziert am 8.3.2009

Datenhaltung kann nicht gewährleistet werden. Vorteil ist die einfache Implementierung.

table-per-class inheritance strategy Bei der *table-per-class inheritance strategy* wird für jede Klasse in der Vererbungshierarchie eine eigene Tabelle mit den Feldern angelegt. Über einen eindeutigen Primärschlüssel wird die Klasse auf verschiedene Tabellen verteilt abgespeichert. Fremdschlüsselreferenzierungen sorgen bei der Abfrage für den Zusammenbau einer Instanz der Klasse.

table-per-concrete-class inheritance strategy Bei dieser Strategie erfolgt eine Trennung nach abstrakter Klasse und konkreten Klassen. Dabei werden alle nichtabstrakten Klassen in einer eigenen Tabelle angelegt. Dies reduziert gegenüber der *table-per-class inheritance strategy* die Anzahl der resultierenden Tabellen.

Für die Architektur wurde die single-table strategy gewählt. Grund für diese Entscheidung war die einfache Implementierung dieser Strategie (vgl. [Rus08]).

4.2.2 Relationship

In relationalen Datenbanken werden Beziehungen über Fremdschlüsselreferenzen ausgedrückt. Man unterscheidet hierbei folgende Beziehungstypen: one-to-one, one-to-many / many-to-one, many-to-many.
Die Beziehungen müssen bei der Implementierung eines OR-Mappers berücksichtigt werden. So gibt es beispielsweise in Java Containerobjekte, wie beispielsweise Listen oder Maps, die durch solche Beziehungen ausgedrückt werden können. Darüber hinaus werden auch die Modellierungsmittel Aggregation und Komposition durch Beziehung realsiert (vgl. [Rus08]).

4.2.3 Mapping-Description

Das Mapping zwischen Objekten und Tabellen muss durch eine Beschreibung erfolgen. Gängige OR-Mapper wie Hibernate bieten hier zwei Alternativen an [3]. Man unterscheidet hierbei zwischen interner und externer Beschreibung. Die externe Beschreibung wird meist durch eine XML-Datei pro Mapping festgehalten. In einem Schema werden die möglichen Konfigurations- und Deklarationsoptionen festgehalten.
Alternativ dazu bietet Java die Möglichkeit Klassen mit Hilfe von Annotations zu beschreiben. Annotations sind nichts anderes als ein spezieller Interfacetyp, der je nach Definition in der Klasse selbst, an deren Feldern oder Methoden gebunden werden kann.

[3] vgl. hierzu Hibernate Core und Hibernate Annotations http://hibernate.org/ referenziert am 8.3.2009

Für die Mapping-Description wurden Annotations gewählt. Für die Umsetzung der Mapping-Description waren folgende Annotations notwendig:

BeanAnnotation Eine `BeanAnnotation` besteht aus den Einträgen `table`, `class` und dem optionalen Eintrag `primary-key`.

PersitentFieldAnnotation Die `PersitantFieldAnnotation` wird an die Attribute einer Klasse gebunden, die in die Datenbank gespeichert werden sollen. Eine `PersitantFieldAnnotation` beinhaltet die Einträge `SQLLiteType` und `Relationship`. Der `SQLLiteType` definiert den Typ des Tabelleneintrags in der SQLLite-Datenbank. Der Eintrag `Relationship` definiert die Art des Mappings one-to-one, one-to-many oder many-to-many. Der Defaultwert ist ein leerer String. In diesem Fall wird das Feld auf den angegebenen `SQLLiteType` abgebildet.

4.3 Resultierendes Design

Abschließend wird in diesem Kapitel das resultierende Design auf Basis der Architektur- und Komponentenentscheidungen vorgestellt. Das Design der Datenbankabstraktionsschicht setzt die aus der Analyse geforderten Anforderungen um.
Im folgenden werden die Klassen und Interfaces beschrieben (siehe auch Abbildung 4.1):

IBeanDBAdapter Diese Schnittstelle definiert die angebotenen Dienste der Persistenzkomponenten. Folgenden Servicemethoden sind spezifiziert:

```
int storeBean(Object o)
Object loadBean(Class clazz, int id),
int updateBean(Object o, int id)
int deleteBean(int id)
```

BeanDBAdapter Diese Klasse implementiert die vereinbarten Dienste der Schnittstelle `IBeanDBAdapter`. Alle Datenbankzugriffsmethoden sind in der internen Klasse `DBHelper` gekapselt. Die Methode `open()` baut eine Verbindung zur Datenbank auf, die Methode `close()` wieder ab.

SQLLiteSkriptCreator Diese Klasse erzeugt die nötigen Data Definition Language (DDL) Skripte, um die Datenbank mit Tabellen und Beziehung zu bestücken, oder diese für einen Reset der Datenbank zu löschen.

SQLStatementHelper Diese Klasse erzeugt die nötigen insert, delete oder update - Statements und sorgt weiterhin für das Laden einer Instanz aus der Datenbank.

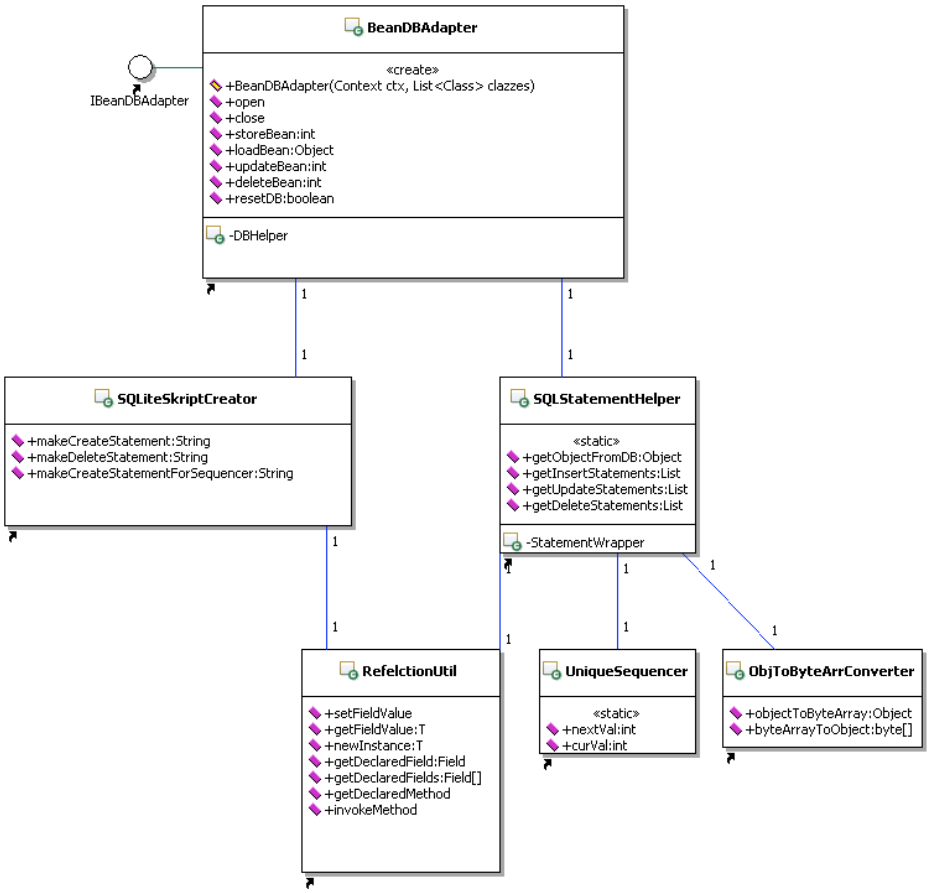

Abbildung 4.1: Klassendiagramm der Persistenzkomponente

Die Funktionalität der einzelne Aufgaben wurden in den dafür vorgesehenen Klassen `UniqueSeqencer`, `RefelctionUtil` und `ObjToByteArrConverter` implementiert.

UniqueSeqencer Der UniqueSeqencer sorgt für die Vergabe eindeutiger „Ids“. Dadurch kann die Eindeutigkeit der Primärschlüssel sichergestellt werden.

ReflectionUtil In dieser Klasse werden Methoden angeboten, um aus den Tabellen Instanzen zu erzeugen, oder für das Persistieren der Objekte die Annotations auszulesen.

ObjToByteArrConverter Um Objekte als Blobs in der Datenbank abzuspeichern oder zu Laden bietet diese Klasse die dafür nötige Funktionalität.

Zu dem resultierenden Entwurf hat während dieser Phase die Erstellung experimenteller Prototypen beigetragen. Besonders im Umgang mit Reflections im Zusammenspiel mit An-

notations konnte durch diese Vorgehen wichtige Erkenntnisse gewonnen werden. In der anschließenden Realisierung sind die dadurch gewonnen Ergebnisse mit eingeflossen.

Für die Anwendungsentwickler ist durch diesen Entwurf ein komfortables persistieren der fachlichen Klassen möglich. Allerdings müssen einige Randbedingungen erfüllt sein. Die Klassen müssen mit den definierten Annotations beschrieben sein, es muss ein privater Default-Konstruktur vorhanden sein und die Klassen müssen das Interface `Serializable` implementieren [4].

[4] vgl. Wikipedia JavaBeans http://de.wikipedia.org/wiki/JavaBeans referenziert am 8.3.2009

5 Realisierung

Die Implementierung der Persistenzkomponenten wurde in Android unter der Entwicklungsumgebung Eclipse 3.4 umgesetzt. Durch Android ist die mobile Datenbank SQLite vorgegeben.
Die Komponente wurde größtenteils nach dem Entwurf umgesetzt. Die für den Prototyp notwendigen Anwendungsfälle wurden vollständig implementiert und haben die Funktionalität und Robustheit auf einer abschließenden Präsentation über den Hamburger Weihnachtsmarkt unter Beweis gestellt.

Die Realisierung der Persistenzkomponente verlief weitestgehend problemlos. Allerdings führten einige Umstände zu zeitlichen Verzögerung der Umsetzung.

JUnit-Test Da Android eine abgesteckte Java-Runtime mitliefert, war es anfangs nicht möglich Test mit dem JUnit-Testframework auszuführen. Eine Recherche ergab prompt die Antwort auf das Problem. In Android werden nicht alle notwendigen Stubs von JUnit implementiert. Um das Problem zu umgehen müssen die Tests wie gewohnt in der JRE ausgeführt werden. Dies lässt sich durch eine angepasste Konfiguration der Testausführung beheben. In der Konfiguration wird die Android-Runtime durch die Java-Runtime ersetzt.

Annotations Um während der Laufzeit auf Annotations der Klassen zuzugreifen, was erforderlich die Annotations mit `@Retention(RetentionPolicy.RUNTIME)` zu definieren. Die unglückliche Fehlermeldung hatte anfangs nicht auf das Problem schließen lassen, konnte aber dann doch sehr schnell gelöst werden.

Activity-Lifecycle und Beenden der Datenbank Android besitzt ein sehr ausgeklügeltes Ressourcen- und Speichermanagement. Anhand bestimmter Faktoren entscheidet Android, wann welche Anwendung oder Anwendungskomponente in den Ruhezustand versetzt oder aus dem Speicher entfernt wird. Bei der Nutzung der Datenbank muss für eine ordnungsgemäße Terminierung über die Methode`close()` geachtet werden. Bei der Implementierung fiel auf, dass der dokumentierte Activity-Lifecycle [1] von der tatsächlichen Ausführung abweicht. Durch entsprechende Versuche konnten aber

[1] Android Guide http://developer.android.com/guide/topics/fundamentals.html referenziert am 8.3.2009

die Stellen gefunden werden, an denen für das Beenden der Datenbankkomponente gesorgt werden musste.

6 Fazit

Abschließend wird an dieser Stelle ein Fazit über das Projekt gegeben.
Die ursprüngliche Vision des Autors, ein Spiel nach dem Vorbild von Scotland Yard[1] zu entwickeln, welche nicht umgesetzt wurden, verringerte die Motivation des Autoren nicht. Ganz im Gegenteil. Der Schwerpunkt des Autors einen OR-Mapper in Android zu entwerfen und prototypisch umzusetzen ermöglichten es, viel Erfahrung im Umgang mit Android, Java Reflections und Annotations zu sammeln.
Auch die intensive Auseinandersetzung mit der Erstellung eines Frameworks hat gezeigt, das im Gegensatz zur Entwicklung einer Anwendung, die Komplexität in der viel generelleren Nutzung eines Framworks eine sehr große Herausforderung darstellt. Spezielle Anforderungen von Anwendungen so weit zu standardisieren, dass sie von mehren von einander unabhängigen Anwendungen benutzbar sind, steht in einem klaren Zielkonflikt von Effektivität / Benutzbarkeit und Einsatz für möglichst viele unterschiedliche Anwendungen. Dieser Zielkonflikt konnte leider im Rahmen des Projektes nicht untersucht werden. Der Erfolg des Frameworks im Zusammenspiel mit der Anwendung „The World within“ konnte erlebt werden.
Besonders angenehm empfand der Autor den partnerschaftlichen Umgang zwischen den Projektteilnehmern, welcher für eine perfekte Arbeitssituation sorgte. Leider ist anzumerken, dass während der Projekt- und Aufgabendefinition bestimmte Arbeitsschritte falsch priorisiert wurden. So wurden das Design und die Entwicklung der Kommunikationskomponente zu spät begonnen. Dies konnte aber durch die sehr kooperative und flexible Einstellung der Verantwortlichen kompensiert werden, was sich in einer erfolgreichen Präsentation der Anwendungen und des Framworks auf dem Hamburger Weihnachtsmarkt gezeigt hat.

[1] Scotland Yard http://www.ravensburger.de/web/Scotland-Yard__3245369-3246414-3246415-3246590.html referenziert am 8.3.2009

Abbildungsverzeichnis

Literaturverzeichnis

[Pri06] PRINZ, WOLFGANG: *Pervasive Games*. Fraunhofer FIT - IuK Wirtschaftssummit, 2006.

[Rus08] RUSSELL, CRAIG: *Bridging the Object-Relational Divide*. Queue, 6(3):18–28, 2008.