Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Eike Falkenberg

Master Projekt
Pervasive Gaming Framework

Fakultät Technik und Informatik
Studiendepartment Informatik

Eike Falkenberg

Thema der Projektbericht
Pervasive Gaming Framework
Stichworte
Pervasive Gaming Mobile Computing
Kurzzusammenfassung
In unserem Master Projekt haben wir ein Spiele Framework für pervasive Spiele entwickelt. Dieser Projektbericht soll einen Überblick über meine Aufgaben und Erfahrungen geben.

Inhaltsverzeichnis

1 PGF - Das Pervasive Gaming Framework

1.1 Überblick

Wir trafen uns zum Zwecke der Themenfindung bereits vor Ende des 2. Semesters. Schnell stellten wir fest, dass unsere Erwartungen an das Projekt ähnlich waren und stellten uns die Frage, welche Themen Gebiete wohl für uns interessant sein könnten. Gemeinsam mit Herrn Zukunft besprachen wir die möglichen Themen und eines der Themen weckte sofort unsere Aufmerksamkeit: Gaming. Fatih berichtete von seinen Erfahrungen in einem früheren Projekt, das sich ebenfalls mit dem Thema Gaming beschäftigt hatte und wie motivierend der spielerische Aspekt für die Teilnehmer gewesen war.
Weiterhin bestand Interesse daran, sich mit Themen aus dem Bereich Mobile Computing auseinanderzusetzen. Einige von uns hatten bereits erste Erfahrungen in dem Bereich gesammelt und alle hatten Lust, die Themen Gaming und Mobile Computing interessant miteinander zu vereinen.
Jan Napitupulu hat 2005 an der HAW eine Bachelor Arbeit mit dem Titel *Pervasive Gaming. Entwicklung von ortsabhängiger Spielesoftware* geschrieben, die unser Interesse geweckt hat. Er beschreibt in dieser Arbeit, wie Spiele auf mobilen Endgeräten unter Einbezug des realen Kontextes entwickelt werden können. Jans Schilderung der Idee, dass die Position des Spieles Auswirkungen auf das Geschehen im Spiel haben könnte hat uns fasziniert. Den räumlichen Kontext in mobile Spiele einzubeziehen bedeutet ein wesentlich interessanteres Spielerlebnis und ist eine große Weiterentwicklung zu den derzeit verfügbaren, räumlich statischen Handyspielen.
Um dem Anspruch eines Master Projektes gerecht zu werden, haben wir nicht nur ein pervasives Spiel, sondern ein Framework für die Entwicklung pervasiver Spiele als Thema definiert. Der Entwickler von pervasiven Spielen sollte mit Hilfe unseres Frameworks, sich nicht um die Details von Kommunikation, Synchronisation oder Positionierung kümmern müssen. „Out-of-the-Box“ sollten die gängigen Technologien transparent nutzbar sein, aber auch eigene Erweiterungen und Anpassungen des Frameworks sollten dem Entwickler ermöglicht werden.

Wir haben bei der Entwicklung das .NET Framework 2.0 mit der Sprache C# programmiert. Auf den mobilen Geräten kam der kleine Bruder des .NET Framework, das .NET Compact Framework zum tragen. Es ist nahezu identisch zu programmieren, jedoch sind einige Namensräume nicht verfügbar. Der Vorteil gegenüber der Java Variante mit der Java Micro Edition war die wesentlich simplere Programmierung der mobilen Geräte. Aus eigenen Erfahrungen und von Kommilitonen wussten wir, dass bei Java ME ziemlich viel Arbeit im Detail liegt, die uns mit dem CFW abgenommen wurden. Weiterhin wollten wir unter Windows entwickeln und Microsoft liefert mit dem Visual Studio eine sehr gut integrierte Möglichkeit, mobile Geräte zu programmieren. Es existieren diverse Studien die uns gezeigt

haben, dass die beiden Alternativen in ihrem Leistungsumfang wenig Unterschied bieten.

Die Teambildung geschah eher zufällig und war trotzdem gut. Wir waren sechs Studierende, Professor Olaf Zukunft als Betreuer und Oliver Neumann als unser Projekt Assistent. Die Größe des Teams war optimal für die Aufgabe, die wir uns selber gestellt hatten und es herschte von Anfang an ein interessiertes und faires Arbeitsklima.
Wir haben schon während der Sommerferien damit begonnen, uns in das Projekt einzuarbeiten und diverse Papers von IEEE und ACM gelesen. Weiterhin haben wir uns auch bereits vor dem eigentlichen Semester mit der Auswahl der richtigen Hardware für die Clients beschäftigt und konnten so zum Beginn des Semesters voll in das Projekt starten.

1.2 PGF Architektur

Wir haben uns für eine recht klassiche Client-Server Architektur entschieden. Für ein Spiel soll ein Server existieren, der n-Clients bedienen kann. Beide sollen eine kompatible Architektur haben, jedoch soll vor allem der Client, zumindest theoretisch, austauschbar sein. Sinnvoll könnte dies werden, wenn ein Java Client benötigt würde. Um die Flexibilität zu sichern, setzt die Client Server Kommunikation auf Plain Text Web Services als Kommunikationstechnologie.
Wichtig war uns bei der Architektur, sauber getrennte Komponenten zu haben, die über klar definierte Schnittstellen miteinander kommunizierten.
Damit das Framework seinen eigentlichen Zweck erfüllen kann, als Basis für weitere Anwendungen zu dienen, legten wir viel Wert auf die Erweiterbarkeit des Frameworks.

1.2.1 Server

Der Server bietet nach außen seine Dienste auf zwei Wegen an, zum einen über Web Services für die Clients und zum anderen durch ein Web Interface. Die Client Kommunikation mit den Web Services wird im Communication Manager zusammengefasst. Dieser stellt die Web Services nach außen zur Verfügung und leitet die Anfragen weiter, besitzt selber also keine Logik. Das Admin Frontend wird mit dem Administration Manager zusammengefasst und bietet seine Dienste durch dynamische Webseiten an. Der Administration Manager ist eher ein zusätzliches Werkzeug, die eigentliche Serverfunktionalität wird durch den Communication Manager repräsentiert.
Der Communication Manager selber besitzt keine Logik und leitet die Anfragen an den Context Manager weiter. Der Context Manager besitzt Informationen darüber, welche Spiele gerade laufen, welche Spieler angemeldet sind und in welchem Zustand sie sich gerade befinden. Der Context Manager wiederum kennt einen Game Manager, in dem die eigentliche Logik des Spieles steckt. der Game Manager bewertet spielspezifisch die Informationen, die er vom Context Manager bekommt und reagiert entsprecht der implementierten Spielelogik.

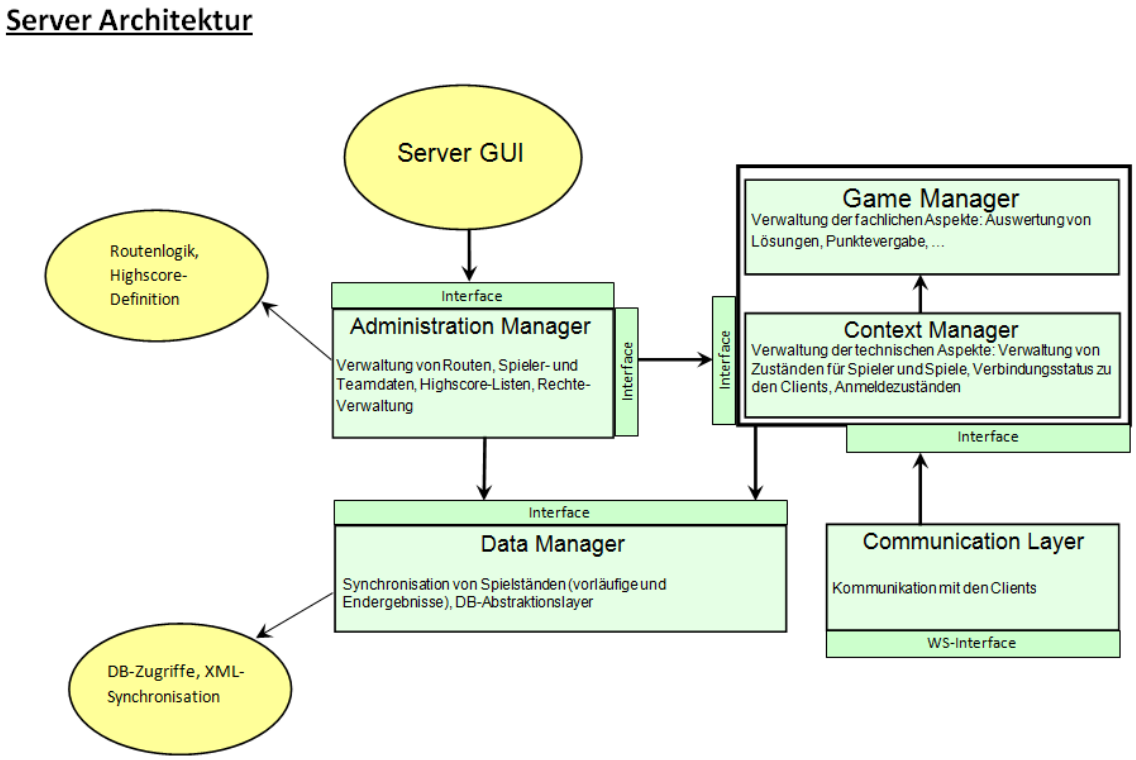

Abbildung 1: Server Architektur

Der Game- und der Context Manager gemeinsam sind die zentralen Stellen der Server Logik, wobei der Context Manager eine fertig implementierte, erweiterbare Komponente und der Game Manager Teil eines konkreten Spieles ist.
Der Context Manager selber greift die nicht auf die Datenstrukturen des Frameworks zu, sondern bedient sich hierfür den von dem Data Manager angebotenen Schnittstellen. Das Laden und Speichern von Daten wird in dem Data Manager gekapselt, wie dieser die Daten speichert und läd, ist für die anderen Manager nicht relevant.
In der Abbildung 1 ist sehr deutlich zu sehen, dass der Communication Manager losgelöst von der Logik ist und Context- und Game Manager logisch zu einer Einheit zusammengefasst werden können.

1.2.2 Client

Was bei dem Server als primäre Schnittstelle der Web Service ist, ist bei dem Client das GUI. Es besitzt keine Logik und dient nur der Repräsentation der Dienste nach außen, hier ist der Spieler quasi der Client, also der Konsument der Dienste. Das GUI wird mit dem Framework nur in Form einer prototypischen Anwendung ausgeliefert. Der Entwickler kann dies

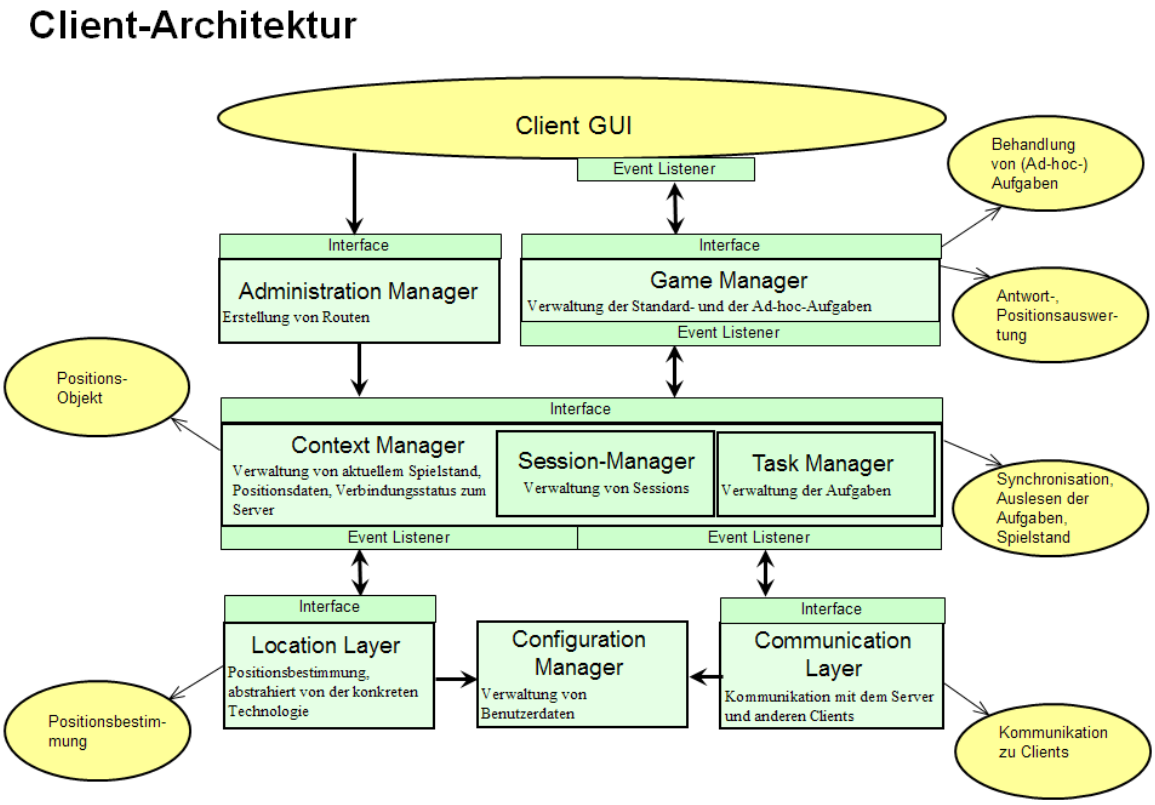

Abbildung 2: Client Architektur

für seine Bedürfnisse anpassen oder ein komplett eigenes User Interface entwickeln. Das GUI sollte nicht als Teil des Frameworks verstanden werden, da es zu sehr der konkreten Ausprägung des Spieles unterliegt.
Auch der Client hat wie der Server einen Administration Manager. Dieser ist ebenfalls eher als ein Werkzeug zu betrachten. Seine Aufgabe ist es, bei der Erstellung von Routen oder ähnlichen Daten zur spielvorbereitenden Sammlung von Lokationen zu unterstützen. Parallel hierzu existiert das Client Äquivalent zu dem Game Manager auf dem Server. Dieser bewertet Spielzustände und birgt die Spielelogik.
In der darunter liegenden Schicht befindet sich mit dem Context Manager die eigentliche Framework Logik des Clients. Dieser kapselt die darunter liegende Positionierungstechnologie und händelt die Verbindung zum Server sowie die Synchronisation. Der Task Manager verwaltet die anstehenden Aufgaben und der Session Manager die Sessions.
Eine weitere Schicht tiefer kommen dann die konkreten Technologie Implementierungen, die die Hardware kapseln. Der Location Manager kümmert sich um die Positionierungstechnolo-

gie, der Communication Layer kapselt die konkrete Verbindungstechnologien. Der Communication Layer bietet auch Dienste zur Kommunikation mit anderen Clients an, sofern dies möglich und aktiviert ist. Der Configuration Manager ermöglicht dem Entwickler das deklarative konfigurieren von Technologien wie Beispielsweise GPS oder Bluetooth. Es muss hierzu lediglich in der Konfigurationsdatei ein entsprechender Eintrag vorgenommen werden und Location Manager und Communication Layer verwenden die entsprechenden Technologien. Es ist möglich, dass der Entwickler eigene Communication Layer oder Location Manager schreibt und diese einbindet. So ist das Framework anpassbar und flexibel. Für die meisten Spiele wird aber vorerst GPS genügen und für die Kommunikation Bluetooth oder W-Lan genutzt werden. Aus diesem Grunde sollen konkrete Implementierungen hierfür mit dem Framework bereit gestellt werden.

1.3 Framework Definition

Zu Beginn des Projektes stellten wir fest, dass wir alle überzeugt waren, ein Framework sei eine gute Sache, die Definition viel uns jedoch schwer. Da wir im Laufe des Projekts merkten, dass es vielen so ging, wurde die Frage nach der Abgrenzung von Frameworks und Klassenbibliotheken zum Running Gag. Ich möchte daher hier versuchen eine gute Definition für Frameworks zu geben.
Die *Gang of four* bezeichnete Gruppe um Erich Gamme definiert in ihrem inzwischen schon legendären Buch „Design Patterns“ ein Framework wie folgt:

> A framework is a set of classes that makes up a reusable design for a specific class of software. A frameworks provides architectural guidance by partioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes

Aus dieser Defintion lässt sich schon offensichtlich die Abgrenzung zu einer Klassenbibliothek erkennen. Eine Klassenbibliothek wird in ein Projekt eingebunden und benutzt, bei einem Framework ist der Ansatz genau anders herum. In einem Framework wird das Projekt durch Vererbung selber Teil des Frameworks.
Die eigentliche Flusskontrolle liegt bei dem Framework, dies wird leicht ironisch auch als das Hollywood Prinzip bezeichnet („Dont’ call the framework, the framework calls you!“). Das Framework definiert die Abläufe, die Anwendung implementiert das spezifische Verhalten.

Frameworks werden hauptsächlich in zwei Arten unterteilt: *Domain (Specific) Frameworks* und *Application (Specific) Frameworks*.
Domain Frameworks dienen der Lösung von Problemen aus bestimmten Aufgabenbereichen, beispielsweise dem Bereich e-Commerce und werden auch als vertikale Frameworks bezeichnet.

Application Frameworks werden auch als horizontale Frameworks bezeichnet und dienen der Entwicklung von Lösungen mit bestimmten Technologien. Beispielweise ist ein Web Services Framework ein Application Framework. Application Frameworks sind grundsätzlich unabhängig von einer bestimmten Fachrichtung.
Weiterhin werden die Frameworks nach ihrem Grad der Transparenz unterschieden. Als *White Box Framework* bezeichnet man ein Framework, von dem der Entwickler die internen Strukturen kennen muss und seine Entwicklung durch direkte Vererbung integriert. *Bei Black Box Frameworks* soll die interne Struktur des Frameworks für den Entwickler nicht von Interesse sein, das Framework liefert fertige Komponenten mit, die der Entwickler instantiiert und kombiniert.
Das Pervasive Gaming Framework ist also ein *Domain Specific White Box Framework*, es dient einer bestimmten Art von Problemen (der Entwicklung von pervasiven Spielen) und der Entwickler integriert sich durch Vererbung.

2 Meine Aufgaben

Unser Projekt war ein Team Projekt, daher haben wir alle viele verschiedene Teilbereiche bearbeitet. Ich möchte hier beschreiben was ich gemacht habe und ein paar der Erfahrungen schildern.

2.1 Recherche

Zu Beginn des Projekts haben wir sehr viel Zeit damit verbracht uns einen Überblick zu verschaffen, was die derzeit wichtigsten Papers für unser Projekt sind. Wir wollten wissen, welche Arbeiten bedeutend sind und was es an aktuellen Forschungsprojekten und Forschungsergebnissen gibt. Wir recherchierten bei IEEE und ACM und fanden viele Papers, die aber oftmals unser Projekt nur am Rande betrafen. Am Ende sind nur eine handvoll wirklich relevanter Arbeiten übrig geblieben und eine Menge Anregungen. Die Recherche hat uns eine gute Wissens Basis gegeben und auch ein besseres Verständniss dafür, was von einem wissenschaftlichen Projekt erwartet wird.

2.2 .NET Schulung

Da wir zu beginn des Projekt sehr unterschiedliche Kenntnissstände in der C# Programmierung und dem Umgang mit Microsoft Visual Studiu 2005 hatten, habe ich zu Beginn des Semesters eine Einführung in die .NET Programmierung mit C# und dem .NET Compact Framework gegeben. Da alle aus dem Team bereits erste Erfahrungen mit .NET gesammelt hatten oder zumindest fundierte Java Kenntnisse besaßen, konnten wir sehr zügig in die Programmierung einsteigen. Bis zum Schluss versuchte ich bei Problemen mit C# und

Visual Studio zu helfen, dies beschränkte sich aber nach kurzer Zeit nur noch auf kleinere Problemchen, da der Einstieg in .NET für erfahrene Entwickler relativ leicht ist.

2.3 Architektur

Da ein großes Interesse an der Entwicklung der Architektur bei allen Team Mitgliedern bestand, wurde diese sehr kontrovers diskutiert. Professor Zukunft regte an, mehrere Teams sollten einzelne Architektur Vorschläge entwickeln, diese dann gemeinsam diskutieren um dann abschließend eine gemeinsame Architektur zu finden. Klar war, dass die Architektur der Framework Definition genügen sollte und nicht bloß eine Klassenbibliothek.
Ich entwickelte meine Architektur gemeinsam mit Maik Weindorf. Unser Ansatz entsprach einer klassichen Schichtenarchitektur mit definierten Schnittstellen, an die sich der Spieleentwickler mittels Vererbung einhängen kann. Als wir uns gegenseitig unsere Architekturen vorstellten, ähnelten sich diese zum Glück sehr, lediglich die Betrachtungswinkel in der Darstellung wichen voneinander ab. Wir entschieden, dass die Architektur Beschreibung von Alexandra und Stephanie am besten geeeignet war, um diese als Projekt Grundlage zu nutzen.

2.4 Serverseitige Datenbankadministration und Deployment

In der Standardinstallation von Visual Studio 2005 wird der SQL Server Express mitgeliefert. Wir haben deshalb bei der Entwicklung die Express Version verwendet. Wir hätten auf dem Server ebenfalls die Express Edition des SQL Servers verwenden können, da dieser aber einfach nur eine abgespeckte Version ist und wir den SQL Server in der Enterprise Version kostenlos durch das MSDNAA Programm zu Verfügung gestellt bekommen haben, verwendeten wir die Enterprise Version. Als Betriebssystem kam auf dem Server Windows Server 2003 zum Einsatz.
Ich hatte bis dahin wenig Erfahrung mit Windows Server 2003 und auch bezüglich der SQL Server Administration hatte ich nur Basiskenntnisse. Da wir jedoch über entsprechende Literatur verfügten, gelang die Installation des Betriebssystems und der Datenbank auf anhieb. Die Konfiguration des SQL Servers gestaltete sich etwas schwierig, da diese doch wesentlich komplexer war als erwartet. Durch ein wenig Hilfe und Recherche gelang es dann aber doch. Wir bereiteten die Datenbanken lokal auf den Entwicklungsrechnern vor und speicherten die Daten in einem Master Database File (MDF). Das MDF wurde dann auf den Server kopiert und dort eingebunden. Im MDF sind alle Tabellen, Daten, Schlüssel und Berechtigungen enthalten. So konnten wir die Datenbank durch ein simples x-Copy Deployment auf den aktuellen Stand bringen. Die Daten der Web Services wurden durch das in Visual Studio 2005 integrierte Deployment Werkzeug aktualisiert. Einmal eingerichtet wird es so möglich, direkt aus der Entwicklungsumgebung heraus auf einen entfernten IIS zu deployen.

2.5 Persistenzierung

Die Auswahl der richtigen Technologie zur Persistenzierung der Daten kann für den späteren Erfolg des Projektes ausschlaggebend sein. Ich beschränke mich auf die serverseitige Peristenzierung, da ich nur in diesem Bereich tätig war.
Uns stand für Speicherung der Daten auf dem Server der Microsoft SQL Server 2005 in der Enterprise Edition zur Verfügung. Da wir den Server selber installiert und konfiguriert hatten, standen uns dort alle Möglichkeiten zur Verfügung. Da die Anwendung auf Basis von .NET entwickelt wurde, beschränkte sich die Suche ausschließlich auf .NET kompatible Persistenzierungswerkzeuge. In .NET wird der Zugriff auf Datenquellen in einem einheitlichen API gekapselt, das als ADO.NET bezeichnet wird. ADO.NET ist jedoch nur als Basis zu verstehen, komplexere Lösungen setzen meist auf ADO.NET auf. Es gibt einige Werkzeuge, die dem Entwickler die Persistenzierung von Daten erleichtern sollen. Wir hatten den Anspruch, eine relativ schmale und flexible Zwischenschicht für die Datenhaltung zu integrieren.
Grundsätzlich gibt es drei verschieden Ansätze, wie Werkzeuge dieser Art funktionieren. Der erste Ansatz generiert aus bestehenden Relationen Datenobjekte, der entgegengesetzt Ansatz generiert Relationen aus Datenobjekten. Weiterhin besteht die Möglichkeit, bestehende Relationen und Objekte zueinander in Beziehung zu setzen, dies wird als *mapping* bezeichnet. Da wir noch zu Beginn der Entwicklung standen, waren weder Relationen noch Objekte vorhanden, wodurch wir theoretisch alle drei Sorten von Werkzeugen einsetzten konnten.
Ich begann mit der Recherche über die Verfügbaren Tools. Da es für die meisten kostenpflichtigen Produkte auch kostenlose Versionen für Forschungsprojekte gibt, war die Suche nicht auf Open Source Optionen beschränkt.
Schnell traf ich auf einen bekannten aus der Java Welt: NHibernate. Es handelt sich hierbei um die .NET Portierung des populären Persistenzierungswerkzeug Hibernate. Bei NHibernate werden die Objekte und Relationen unabhängig erstellt und anschließend die Beziehungen durch XML Dateien definiert. NHibernate lässt dem Entwickler sehr viel Freiheit, bietet dafür aber nicht den kompfort wie andere Werkzeuge.
Das nächste sehr interessante Werkzeug war Versant Open Access .NET. Hier überzeugt die saubere Integration in die Entwicklungsumgebung Visual Studio 2005. Zunächst richtet man in der Entwicklungsumgebung eine Datenquelle ein und kann dann Klassen und Relationen visuell mappen. Dieses Werkzeug machte insgesamt einen sehr soliden Eindruck und hat durch seine funktionalität überzeugt.
Es existieren noch viele weitere Werkzeuge, doch mir ist bei der Recherche schnell klar geworden, dass diese Form der Persistenzierung für unser Projekt eher ungeeignet ist. Zum Einen haben wir nur wenige Klassen, die Persistenziert werden müssen. Die Verwendung von kostenpflichtigen Werkzeugen ist ebenfalls keine gute Option. Wir könnten zwar im Projekt eine Forschungs-Lizenz verwenden, wenn jedoch ein dritter unser Framework verwenden wollte, benötigte er zunächst ein Lizenz für das Persitenzierungs-Werkzeug. Ein weiteres Problem ist die eingeschränkte Flexibilität, die durch den Einsatz eines Persistenzierungs-

werkzeugs eines Drittanbieters entsteht.
Wir haben uns aus den genannten Gründen dazu entschlossen, eine leichtgewichtige Persistenzschicht selber zu entwickeln. Stephanie, Alexandra und ich haben gemeinsam die entsprechenden Methode implementiert, so dass die Datenbankzugriffe in eine erweiterbaren Manager gekapselt wurden.

2.6 Sicherheit

Der Punkt der Sicherheit ist ein wichtiger in einem Projekt wie dem unseren. Ein Kernelement des PGF ist das Kapseln von Kommunikations Technologien. Da diese Technologien für den Benutzer transparent sein sollen, will er sich nicht um die Details kümmern müssen. Dies führt jedoch auch dazu, dass er der Frameworkentwicklung vertrauen muss und sich über Details wie der Sicherheit wenig bis gar keine Gedanken macht. Somit müssen Sicherheitskonzepte so in das PGF integriert werden, dass es dem Entwickler eine zuverlässige und transparente Nutzung ermöglicht.
Neben vielen kleineren Evaluationen habe ich mich im Rahmen des Projektes sehr Ausführlich mit der Sicherung der Synchronisation beschäftigt. Die Synchronisation ist ein eigenständiger Prozess, die den Datenaustausch zwischen dem Client und dem Server beschreibt. Die beiden primären Prozesse sind dabei das Laden des Spiels auf den Client vom Server und das übertragen der Ergebnisse vom Client auf den Server. Die Clients initiieren den Synchronisations Prozess, indem sie den Server über dessen Web Services aufrufen. Die dafür notwendige Internetverbindung baut der Client selber auf, entweder durch eine Einwahl ins mobile Internet oder durch eine Verbindung an einen PC.
Im ersten Schritt macht es Sinn, den Clientaufruf via SSL zu verschlüsseln. Da der Server Web Service direkt aufgerufen wird, entfällt die Gefahr der Intermediaries. Web Services können als Router für Web Service Nachrichten fungieren und werden dann als Intermediaries bezeichnet. Die Gefahr bei diesem Ansatz in Verbindung mit SSL ist, dass die Nachrichten nur Punkt-zu-Punkt verschlüsselt werden und auf den Intermediaries unverschlüsselt vorliegen. In unserem einfachen Fall langt jedoch eine SSL-Absicherung mittels HTTPS, da wir die Routing Funktion der Web Services nicht nutzen.
Das nächste Problem, das es zu bewältigen gab, war die Authentifizierung der Clients. Wir haben uns hier für die erste Version des PGF sehr pragmatisch entschieden und haben die Credentials in die Methodenparameter des Web Services gelegt.
In der weiteren Entwicklungen muss die Integration von WS-Security ein Ziel sein. WS-Security beschreibt SOAP-Header, die die Integration von Security Credentials in die XML Nachrichten ermöglichen. Durch diese Erweiterung wäre es möglich sowohl Name und Passwort (UserNameToken), als auch Zertifikate und sogar Kerberos Tokens zu verwenden. Die weiteren Möglichkeiten der WS-* Spezifikationen machen in dem Projektumfeld nur wenig Sinn. WS-Policy, WS-Trust, WS-SecureConversation oder WS-Federation setzen eine Flexibilität voraus, die in dem Einsatzbereich des PGF vorerst nicht anzufinden sein wird. Der

Einsatz von WS-Policy würde sinnvoll werden, wenn sich die geforderten Credentials des Web Services ändern würden, zu diesem Zeitpunkt ist dies jedoch nicht vorgesehen.
Für den Moment ist eine WS-Security integration ein logischer Schritt, weitere Sicherheitsstandards sollten erst bei Bedarf integriert werden.

3 Projekt Fazit

Im vergleich zu einigen anderen Projekten, war unseres von vornherein sehr stark als Team Leistung konzipiert. Wir haben uns von Beginn an mehrmals wöchentlich getroffen und haben auch einen Großteil der Zeit gemeinsam in der HAW entwickelt. In der ersten Hälfte des Semesters haben wir jeweils zwei ganze Tage an der HAW an unserem Projekt gearbeitet, später war dies durch den Vorlesungsplan nicht mehr möglich und reduzierte sich auf zwei halbe Tage. Zusätzlich wurde auch noch in der Freizeit und am Wochenende am PGF gearbeitet.
Die regelmässigen Treffen haben zu einem regen Ideen- und Informationaustausch geführt. Es war eine bewusste und wie ich finde gute Entscheidung, so eng gemeinsam zu arbeiten. Wir haben in kleineren, aufgabenorientiert zusammengestellten, Teilteams Lösungen für konkrete Probleme erarbeitet und diese dann dem Team vorgestellt. Durch diese Form des gemeinsamen Arbeitens hatte jeder den Überblick über das Gesamtprojekt und konnte sich dort einbringen, wo seine persönlichen Interessenschwerpunkte lagen. Dies führte dazu, dass ich mich während des Projektes mit Themen wie Datenbankdesign, Softwarearchitektur, Deployment, Infrastruktur, Programmierung und vielen weiteren interessanten Bereichen einbringen konnte. Das Arbeiten in den Teilgruppen war sehr interessant und abwechslungsreich, führte jedoch auch immer wieder zu Kommunikationsproblemen, da die einzelnen Entwicklungen integriert werden mussten.
Man könnte zu dem Schluss kommen, dass die Vielfallt der bearbeiteten Themen zu mangelnder Tiefe im Verständniss geführt haben. Ich bin jedoch der Meinung, dass die Themenvielfallt eine Chance war. Gerade auch in Hinblick auf die Master Arbeit konnte man so auf Themen stoßen, die für diese interessant sein könnte und mit der man sich sonst nicht beschäftigt hätte. Weiterhin war das Projekt so flexibel, dass stärkere Interessenschwerpunkte auch gut in das PGF einfließen konnten.
Bei der Entwicklung des eigentlichen Frameworks haben wir uns für die relativ unabhängige Entwicklung von Client und Server entschieden. Es gab zwei Gruppen mit je drei Teammitgliedern, von denen die eine den Server und die andere den Client entwickelt hat. Wir hatten zuvor die Schnittstellen ausführlich diskutiert und klar definiert. Im nachhinein würde ich die Entwicklung dieser beiden Komponenten enger verzahnen, da sich die beiden Teilgruppen doch stark voneinander entfernten.
Als wir uns vor den Semesterferien das erste mal trafen, war schnell klar, dass wir bei unserem Projekt auch Spaß haben wollten. Das Thema Gaming ist ein Thema, dass uns alle

sehr interessiert hat. Die Entwicklung von interessanter Software ist für die meisten von uns eine Passion und wir wollten diese Energie auch für unser Projekt nutzen. Ich finde, dass uns das gelungen ist. Die Entwicklung einer kleinen Beispielapplikation hat uns ermöglicht, das Framework zu benutzen und hat starkt Motivations fördernd gewirkt. Durch sie hatten wir stets eine Vision, wie wir als Spielentwickler ein Framework der von uns entwickelten Art nutzen wollen würden und das Definieren der Anforderungen an das PGF viel wesentlich leichter.
Rückblickend empfinde ich das PGF Projekt als ein sehr interessantes, lehrreiches Projekt und denke, wir alle haben sehr viel dazugelernt.