Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Thomas Schmidt & Fatih Keles

Pervasive Gaming Framework

Fakultät Technik und Informatik
Studiendepartment Informatik

Hamburg, 1. März 2007

Inhaltsverzeichnis

1 Einleitung & Motivation

Diese Ausarbeitung ist im Rahmen des Masterstudiengangs der HAW-Hamburg entstanden. Der Name Pervasive Gaming Framework bezeichnet ein Framework für mobile Spiele. Das Projekt wurde im Wintersemester 06/07 durchgeführt und bestand aus folgenden Teilnehmern: Alexandra Revout, Stefanie Gamm, Eike Falkenberg, Fatih Keles, Maik Weindorf und Thomas Schmidt.

Im Vordergrund des Gruppenprojekts stand das Sammeln von praktischen Erfahrungen und das Erlangen von Kenntnissen in mobilen Technologien. Das Projekt wurde in zwei Gruppen (Client und Server) aufgeteilt. Die Servergruppe bestand aus Alexandra Revout, Stefanie Gamm und Eike Falkenberg und die Clientgruppe aus Fatih Keles, Maik Weindorf und Thomas Schmidt. Während des gesamten Projektes haben Fatih Keles und Thomas Schmidt in der Clientgruppe eng zusammengearbeitet und gemeinsam entwickelt. Es gab große Überschneidungen und die wesentlichen Entscheidungen wurden zusammen getroffen. Aus diesem Grund ist diese Ausarbeitung eine gemeinsame Arbeit geworden und handelt hauptsächlich über den Client-Teil des Frameworks.

Kapitel 2 gibt einen Überblick über die, für das Projekt wichtigen, Grundlagen.
Im Kapitel 3 werden die Projektziele erläutert. Zunächst werden die Vision des Projektes betrachtet und die Projektziele aufgezeigt. Es wird ein Szenario dargestellt, das als Grundlage für die Entwicklung des Frameworks dient. Anschließend werden die technischen und fachlichen Anforderungen dargelegt.
Kapitel 4 beschreibt die Vorgehensweise für die Konzeption des zu entwickelnden Frameworks. Hier werden die einzelnen relevanten Entscheidungen diskutiert. Dies geschieht vom Vorgehen über die Architektur bis hin zur praktischen Umsetzung des Frameworks.
Abschließend wird das Projekt im Kapitel 5 bewertet. Es werden Erweiterungsmöglichkeiten aufgezeigt und eine Zusammenfassung gegeben.

2 Grundlagen

In diesem Kapitel werden einige Grundlagen erläutert, die zum Verständnis des entwickelten Frameworks wichtig sind. Die Abschnitte zu den Themen „Framework“, „GPS“, „Pervasive Gaming“, „Web Services“ und „Context Awareness“ sind jeweils kurz gehalten. Für detailliertere Informationen wird an den entsprechenden Stellen auf weiterführende Literatur verwiesen.

2.1 Framework

In diesem Projekt wurde ein Framework für pervasive Spiele entwickelt. Aus diesem Grund ist es von entscheidender Bedeutung zu verstehen, worum es sich bei einem Framework handelt.

Laut [2] ist ein Framework ein partiell vollständiges Software-System, das noch instanziiert werden muss. Es definiert die Architektur für eine Familie von Systemen und stellt die Grundbausteine für ihre Erzeugung dar.

Ein Framework besteht aus statischen Bereichen (*frozen spots*) und veränderbaren Bereichen (*hot spots*) [9]. Die statischen Bereiche beschreiben die Gesamtarchitektur des Softwaresystems und bleiben in jeder Instanziierung des Frameworks unverändert. Die veränderbaren Bereiche dagegen werden für jede Instanziierung des Frameworks neu implementiert und den speziellen Bedürfnissen der konkreten Implementierung angepasst. In der objektorientierten Softwareentwicklung wird dies meist über abstrakte Klassen realisiert, die in jeder Instanziierung des Frameworks von konkreten Klassen geerbt und implementiert werden müssen.

Eine ähnliche Beschreibung findet man auch in [5]. Demnach bestimmt ein Framework die Architektur einer Anwendung sowie den Kontrollfluss. Es enthält die Entwurfsentscheidungen, die in dem Anwendungsbereich allgemein anzufinden sind.

Besonders wichtig ist die Unterscheidung und Abgrenzung eines Frameworks zu einer Klassenbibliothek. Im zweiten Fall schreibt der Anwendungsentwickler den Hauptteil der Anwendung selbst und ruft an geeigneten Stellen Methoden der Klassenbibliothek auf. Auch der Kontrollfluss wird vom Anwendungsentwickler definiert. Bei der Nutzung eines Frameworks definiert das Framework den Kontrollfluss und stellt den Hauptteil der Anwendung dar. Das Framework ruft den Code des Anwendungsentwicklers auf. Dies führt zu einer Umkehrung der Steuerung der Anwendungssteuerung [5].

In diesem Projekt wurde keine Klassenbibliothek, sondern ein Frameworks für pervasive Spiele entwickelt. Wie sich in Kapitel 4 zeigen wird ist deshalb die Architektur so ausgelegt, dass das Framework die Hauptbestandteile der Anwendung implementiert und den Kontrollfluss vorgibt. Entwickler von pervasiven Spielen können das Framework nutzen. Sie müssen lediglich an den vorgesehenen Stellen ihren Code implementieren. Das Framework sorgt dafür, dass dieser Code an der richtigen Stelle im Programmablauf aufgerufen wird.

2.2 GPS (Global Positioning System)

Im Kapitel 4 wird sich zeigen, dass das Thema „Positionierung“ in diesem Projekt eine wichtige Rolle spielt. Die Architektur des Frameworks ist so ausgelegt, dass prinzipiell verschiedene Arten von Positionierungstechniken integrierbar sind. Beispielhaft implementiert wurde aus zeitlichen Gründen jedoch nur eine Positionierungstechnik. Aufgrund der relativ einfachen Realisierbarkeit haben sich die Autoren für die GPS-Technik entschieden.

GPS ermöglicht die Positionierung von Objekten auf der Erde mit einer Genauigkeit von bis zu 25 Metern in der Horizontalen und 45 Metern in der Vertikalen [11]. Um eine Positionierung an jedem Ort der Erde zu ermöglichen, befinden sich insgesamt 24 GPS-Satelliten auf sechs Bahnen mit jeweils vier Satelliten pro Bahn in einer Höhe von 20.200 km im Umlauf um die Erde.

Für den Empfang der Satellitensignale ist keine Anmeldung bei den Satelliten notwendig. Die Kommunikation erfolgt nur in eine Richtung, nämlich von den Satelliten zu den GPS-Empfangsgeräten auf der Erde [11]. Das bedeutet, dass die Position nicht von den Satelliten, sondern von den Empfangsgeräten ermittelt wird. Hierzu müssen die Signale von mindestens drei Satelliten empfangen werden können. Anhand der Signallaufzeiten kann dann die Position des Empfängers auf der Erde ermittelt werden. Detaillierte Informationen zum Verfahren der Positionsbestimmung können in [8] nachgelesen werden.

Neben der Position kann man mit GPS auch andere Informationen, wie z. B. die aktuelle Uhrzeit oder die Geschwindigkeit des Empfängers ermitteln [11].

Für den Empfang der Satellitensignale ist ein Sichtkontakt zu den Satelliten notwendig. Aus diesem Grund funktioniert die Ortung mittels GPS nicht oder nur eingeschränkt innerhalb von Gebäuden oder in Tunneln.

2.3 Pervasive Gaming

Pervasive Gaming ist eine Teildisziplin des Pervasive Computing. Pervasive Computing wiederum bedeutet soviel wie durchdringendes bzw. überall vorhandenes Computing. Es kann nach [12] als eine Weiterentwicklung der Bereiche Verteilte Systeme und Mobile Computing verstanden werden. Demnach stehen bei verteilten Systeme Anforderungen wie die Kommunikation zwischen Rechnern, die Fehlertoleranz, eine hohe Verfügbarkeit, der entfernte Zugriff auf Informationen und die Sicherheit des verteilten Systems im Vordergrund. Beim Mobile Computing kommen weitere Anforderungen hinzu. Diese sind z. B. der mobile Zugriff auf Informationen, Energiesparsamkeit und die Einbeziehung des Ortes.

Pervasive Computing ergänzt und erweitert die Funktionen und Anforderungen von verteilten und mobilen Systemen. Intelligente Räume sind eine der neuen Ideen. Durch Integration der Technologie in die Infrastruktur von Gebäuden können sich diese den Bedürfnissen der Nutzer anpassen. Die Regulierung der Temperatur und der Lichtverhältnisse nach den Vorlieben der Personen, die sich im Gebäude aufhalten, sind damit möglich. Eine weitere Anforderung ist die geforderte Unsichtbarkeit der Technik. Sie soll möglichst im Hintergrund arbeiten und den Benutzer nicht stören. Weitere Aspekte sind die automatische Anpassung an unterschiedliche technische Gegebenheiten an verschiedenen Orten und das Verstecken dieser Unterschiede vor dem Benutzer.

Pervasive Games sind Computerspiele, die unter Einbeziehung der Realität und der Aspekte des Pervasive Computings entwickelt und gespielt werden [7]. Dies wird meist dadurch realisiert, dass die Spiele in real existierenden Räumen gespielt werden und die Lokation der Spieler in das Spielkonzept integriert wird.

2.4 Web Services

Das in diesem Projekt entwickelte Framework besteht aus einer Server- und einer Clientkomponente (vgl. 4.2). Die Kommunikation dieser Komponenten wird über Web Services realisiert. Aus diesem Grund werden in diesem Abschnitt die Grundlagen von Web Services kurz erläutert.

Als Web Services bezeichnet man eine Reihe von Standards, über die eine Kommunikation zwischen Maschinen möglich ist [6]. Zu den wichtigsten Standards werden SOAP, WSDL und UDDI gezählt. Dabei spezifiziert SOAP einen Standard zur Nachrichtenübermittlung. Er definiert den notwendigen Aufbau einer Nachricht. Mittels WSDL (*Web Service Description Language*) werden die Web Services beschrieben. Ein WSDL-Dokument beschreibt die Schnittstelle, über die ein Web Service ansprechbar ist. UDDI ist die Abkürzung für *Universal Description, Discovery and Integration of Web Services* und ist ein Standard für Web Service-Verzeichnisse. Anbieter von Web Services können ihre Dienste in solchen Verzeichnissen publizieren. Konsumenten wiederum können die Web Services über die Verzeichnisse finden.

Weitergehende Informationen zum Thema Web Services können beispielsweise in [1] nachgelesen werden.

2.5 Context Awareness

Das Thema „Context Awareness“ spielt in dem entwickelten Framework eine wichtige Rolle und wird deshalb nachfolgen erläutert. Eine oft zitierte Definition findet man in [3]:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

Als Kontext einer Person kann man z. B. seinen Aufenthaltsort betrachten. Führt eine Applikation Aktionen aus, die abhängig sind von der Lokation, so kann man die Applikation als „Context Aware“ bezeichnen. Neben der Lokation gibt es weitere Ausprägungen für Kontext. [13] nennt hierzu beispielsweise die Zeit, die soziale Umgebung oder die aktuellen Aufgaben einer Person.

In dem entwickelten Framework soll der Client Informationen über seinen aktuellen Kontext an den Server übermitteln. Dies können z. B. Informationen über den aktuellen Akkustand oder die aktuelle Geschwindigkeit der Internetverbindung sein. Der Server kann diese Informationen nutzen, in dem er beispielsweise keine aufwändigen Videos an Clients mit schlechter Internetverbindung schickt.

3 Projektziele

Dieses Kapitel gibt eine Übersicht über die Ziele des Projekts. In Abschnitt 3.1 wird zunächst die Vision der Projektteilnehmer erläutert. Es wird dargelegt, was das eigentliche Ziel des Projekts ist. Dabei wird sich zeigen, dass die Hauptaufgabe die Entwicklung eines Frameworks darstellt. Für ein besseres Verständnis der Projektziele wird anschließend in Abschnitt 3.2 ein Beispielszenario erörtert. Zum Schluss folgt in Abschnitt 3.3 eine Auflistung der technischen und fachlichen Anforderungen an das zu entwickelnde Framework.

3.1 Vision

Das Ziel des Projekts ist die Entwicklung eines Frameworks für pervasive Spiele. Es geht nicht darum, ein konkretes Spiel zu implementieren. Der Frameworkgedanke steht im Vordergrund. Auf Basis des entwickelten Frameworks soll anschließend ein Spiel beispielhaft realisiert werden. Dieses dient für Testzwecke und als Beispielimplementierung für zukünftige Nutzer des Frameworks. In Kapitel 4, in dem es um die Architektur und die Umsetzung der Applikation geht, wird die Unterscheidung zwischen Framework und Beispielanwendung deutlich werden.

Wie bereits in Kapitel 2.1 beschrieben, gibt ein Software-Framework, anders als eine Klassenbibliothek, den grundsätzlichen Ablauf einer Applikation vor. Aus diesem Grund wurde das Framework für eine spezielle Ausprägung pervasiver Spiele entwickelt. Denn je größer die Gemeinsamkeiten der zu unterstützenden Spiele sind, desto mehr Funktionen können in das Framework integriert werden. Entwickler, die das Framework für die Spielentwicklung nutzen, haben dadurch einen minimalen Aufwand. Sie müssen im Wesentlichen die eigene Spiellogik programmieren und brauchen sich nicht um technische Details, die Verwaltung von Spielen und den Spielablauf kümmern. Diese Überlegungen waren ein wesentlicher Faktor für die Entscheidung lediglich eine bestimmte Ausprägung pervasiver Spiele zu unterstützen.

Die Projektmitglieder haben sich für die Entwicklung eines Frameworks entschieden, das für die Klasse der „Schnitzeljagd“-ähnlichen Spiele bestimmt ist. Das Hauptprinzip dieser Spiele besteht darin, dass die Spieler alleine oder in Teams eine Reihe von Orten nacheinander aufsuchen müssen. An den erreichten Orten können beispielsweise Aufgaben gestellt werden, die gelöst werden müssen. Vorstellbar ist auch, dass an den Orten lediglich Informationen angezeigt werden. Eine konkrete Ausprägung solch eines Spiels ist z. B. eine Stadtrallye. Aber auch eine Stadt- oder Museumsführung könnte mit dem Framework realisieren werden.

3.2 Szenario

Zu Beginn des Projekts wurde ein Szenario entwickelt. Dieses diente als Grundlage für die Entwicklung des Frameworks und der Beispielapplikation. Es wurde in den späteren Phasen der Architekturentwicklung und Umsetzung als Leitfaden verwendet.

Das Szenario gliedert sich in zwei Abschnitte. Der erste Abschnitt beschreibt die Vorbereitungsphase, d. h. die Schritte bis zum Beginn eines Spiels. In der zweiten Phase geht es um den eigentlichen Spielablauf. Es folgt eine detaillierte Beschreibung der beiden Szenario-Abschnitte.

3.2.1 Vorbereitung des Spiels (Szenario-Abschnitt 1)

- Ein Spieler meldet sich mit seinen Benutzerdaten über eine Weboberfläche beim Spiele-Server an.
- Er kann aus einer Liste von verfügbaren Spielerouten eine Route auswählen und auf deren Basis eine neue Spielinstanz anlegen.
- Er wählt die maximale Anzahl von Teams und Spielern, die mitspielen dürfen. Bei Teamspielen werden die Routen in Teilrouten gegliedert, so dass die Teammitglieder unterschiedliche Teilstrecken zugewiesen bekommen.
- Neben einem Start- und einem Endzeitpunkt wird eine Anmeldefrist für das Spiel definiert. Bis zum Ablauf dieser Frist können sich andere Spieler für die Teilnahme anmelden. Bis zum Endzeitpunkt müssen alle Spieler das Spiel beendet und ihre Spielergebnisse auf den Spiele-Server hochgeladen haben.
- Der Spieler kann sich entscheiden, ob er ein öffentliches oder privates Spiel erstellen möchte. Bei privaten Spielen schränkt er die Gruppe der möglichen Teilnehmer ein.
- Nach dem Einrichten des Spiels werden alle potenziellen Teilnehmer per E-Mail benachrichtigt. Diese können sich über einen Link in der E-Mail oder direkt über die Weboberfläche für die Teilnahme am Spiel registrieren.
- Nach Ablauf der Anmeldefrist können sich die Teilnehmer die Spieldaten inklusive der Spielroute auf ihre PDAs herunterladen. Bei Teamspielen ist in den Spieldaten vermerkt, welche Teilrouten die einzelnen Teammitglieder laufen müssen.

3.2.2 Spielablauf (Szenario-Abschnitt 2)

- Das Spiel wird auf den PDAs der Teilnehmer gespielt und beginnt zum vorher definierten Startzeitpunkt.
- Jeder Spieler bekommt den nächsten Routenpunkt als Zwischenziel angezeigt.
- Er muss diesen Routenpunkt aufsuchen. Dem Spieler wird die Entfernung zum nächsten Routenpunkt (z. B. 115 Meter) und die Himmelsrichtung (z. B. Norden) angezeigt. Zusätzlich kann er sich seine eigene Position und die Position des Zwischenziels auf einer Karte anzeigen lassen[1].

[1] Die Idee für die Kartenanzeige wurde nachträglich in den Spielablauf integriert.

- Nach dem Erreichen eines Routenpunktes muss der Spieler eine Aufgabe lösen, z. B. eine Frage beantworten und erhält im Erfolgsfall Punkte. Anschließend wird dem Spieler der nächste Routenpunkt angezeigt.
- Dieser Ablauf wiederholt sich bis zum Ende des Spiels.
- Da alle benötigten Spieldaten zu Beginn des Spiels auf den PDA geladen wurden, ist während der Spielphase keine Verbindung zum Spiele-Server notwendig. Sollte jedoch zwischendurch eine Internetverbindung bestehen, können Zwischenergebnisse an den Server übertragen oder zusätzliche Bonusaufgaben vom Server heruntergeladen werden.
- Hat der Spieler alle Aufgaben gelöst, überträgt er seine Spieldaten an den Spiele-Server. Dieser kontrolliert das Spielergebnis und aktualisiert die zugehörige Highscore.
- Das Spiel endet, wenn alle Spieler ihre Ergebnisse hochgeladen haben oder der zuvor bestimmte Endzeitpunkt abläuft.

3.3 Anforderungen

Die Anforderungen an das Framework ergeben sich unmittelbar aus der Vision und dem oben vorgestellten Szenario. Sie werden nachfolgend, unterteilt in technische und fachliche Anforderungen, vorgestellt.

3.3.1 Technische Anforderungen

- Es soll einen Server geben, der für die Verwaltung von Spielen und für die Koordination zwischen Spielern zuständig ist. Auf einem mobilen Client (PDA) soll das eigentliche Spiel gespielt werden können.
- Die Kommunikation zwischen Server und Client soll über Web Services realisiert werden.
- Das Framework soll die Kommunikation zwischen Server und Client übernehmen. Es soll von der konkreten Kommunikationstechnologie abstrahiert werden, so dass sich der Anwendungsprogrammierer damit nicht auseinandersetzen muss.
- Die Positionierung soll ebenfalls vom Framework realisiert werden. Auch hier soll von der eigentlichen Technologie, die für die Positionierung genutzt wird, abstrahiert werden.
- Die vom Server an den Client übertragenen Daten sollen abhängig vom Client-Kontext sein (Context Awareness). Das bedeutet, dass beispielsweise die übertragenen Daten und Datenmengen von der Verbindungskapazität und Akkulaufzeit abhängig sein sollen.
- Da nicht davon ausgegangen werden kann, dass die mobilen Geräte jederzeit Zugriff auf das Internet haben, sollen die Spiele auch offline gespielt werden können.

3.3.2 Fachliche Anforderungen

- Das Framework soll mobile pervasive Spiele der Klasse „Schnitzeljagd“ ermöglichen. Die Spiele sollen in realen Umgebungen gespielt werden, so dass die Position der Spieler integraler Bestandteil des Spiels ist.
- Der Spielablauf soll prinzipiell so aussehen, dass Punkte entlang einer Route abgelaufen werden können. An den erreichen Routenpunkten sollen Fragen gestellt werden können.
- Das Framework soll sowohl einen Einzelspieler- als auch Mehrspieler-Modus ermöglichen.
- Bei Teamspielen sollen die Spieler Spielinformationen ad-hoc untereinander austauschen können. Die Aufgaben sollen an verschiedene Teammitglieder verteilt werden können, so dass jedes Teammitglied eine Teilaufgabe erhält.
- Ein Spieler soll zu jedem Zeitpunkt nur eine Aufgabe sehen können. Erst nach dem Lösen einer Aufgabe soll ein Spieler die nächste Aufgabe erhalten.
- Unabhängig von den Routen soll es Ad-hoc-Aufgaben geben, die die Spieler zusätzlich lösen können.
- Es soll eine zentral verwaltete Highscore geben.

4 Konzept

In Kapitel 4.1 werden die Vorgehensweise für die Konzeption des zu entwickelnden Frameworks beschrieben und die einzelnen Entscheidungen, die hierbei eine Rolle spielen, diskutiert.
Wie in Kapitel 1 beschrieben, handelt diese Ausarbeitung über den Clientteil des Frameworks. Abschnitt 4.2 beschreibt die Client-Architektur mit den zugehörigen Komponenten und Modulen und deren Zusammenspiel. Anschließend wird in Kapitel 4.3 beschreiben, wie das Framework praktisch umgesetzt worden ist.

4.1 Vorgehen

Um einen Überblick über die Funktionalität zu bekommen, wurde ein allgemeiner Spielablauf von der Spielvorbereitung bis hin zum Spielablauf erstellt (siehe Kapitel 3.2). Auf Basis dieses Dokuments wurde ein Prototyp entwickelt, der die allgemeinen Abläufe eines, auf dem Framework laufenden Spiels illustriert.

Um eine gemeinsame Vorstellung über das Gesamt-System zu bekommen, wurden insgesamt drei Architekturentwürfe in Zweierteams erstellt. Die Entwürfe umfassten sowohl die Client als auch die Serverseite und die dazugehörigen Ablaufdiagramme. In einem anschließenden Gruppenmeeting haben sich alle Gruppenmitglieder auf einen Entwurf geeinigt, der als

Grundlage für das Projekt dienen soll. Dieser Entwurf wurde während des Projektverlaufs weiter verfeinert.[2]

Nachdem eine Einigung über den Architekturentwurf erfolgt war, wurde eine erste Trennung durch eine Gruppenaufteilung in Client- und Servergruppe vorgenommen. Die Servergruppe besteht aus Alexandra Revout, Stefanie Gamm und Eike Falkenberg und die Clientgruppe aus Fatih Keles, Maik Weindorf und Thomas Schmidt.

Nach der Gruppenaufteilung haben die Gruppen einen „Vertrag“ über die Schnittstellen des Systems ausgehandelt.[3] Beide Gruppen haben ab diesem Zeitpunkt getrennt an der Umsetzung des Projekts gearbeitet und gegen die definierten Schnittstellen programmiert.

4.2 Client-Architektur

In diesem Kapitel wird die Client-Architektur des Pervasive Gaming Frameworks beschrieben. Für eine Beschreibung der Server-Architektur wird auf [10] verwiesen. Abschnitt 4.2.1 erläu-

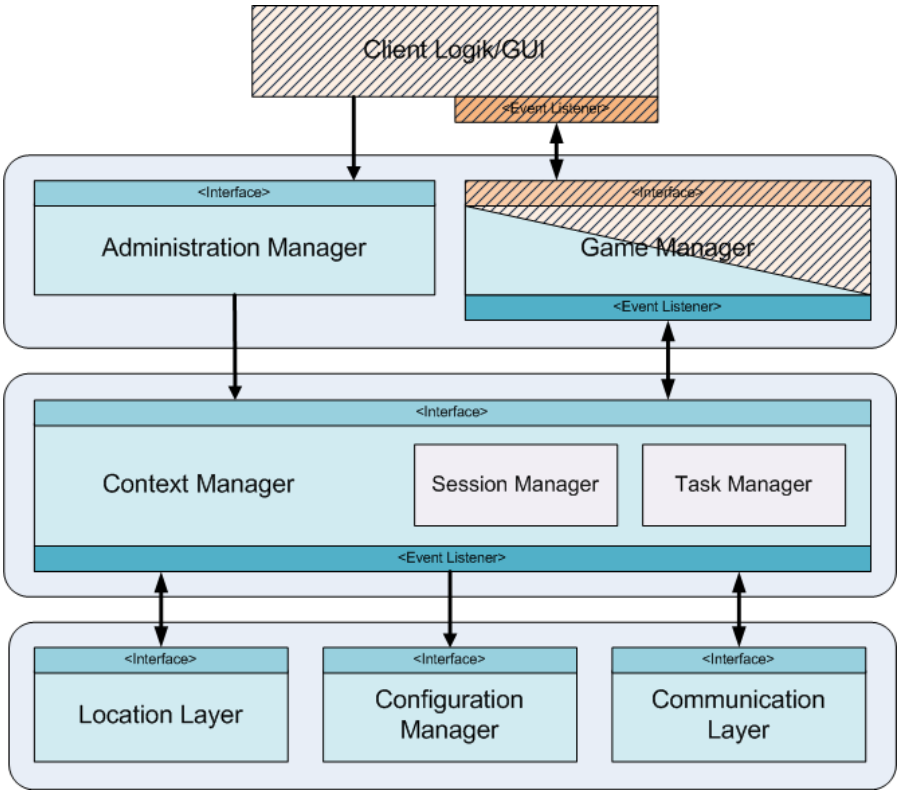


Abbildung 1: Pervasive Gaming Framework - Client Architektur

tert die einzelnen Komponenten der, in Abbildung 1 dargestellten, Client-Architektur. In Kapitel

[2] Abschnitt 4.2 geht näher auf die Framework-Architektur ein
[3] Siehe Anhang A.1

4.2.2 wird anschließend das Zusammenspiel der Client-Komponenten untereinander und das der Client-Komponenten mit den Server-Komponenten gezeigt.

4.2.1 Komponenten

Die Architektur des Clients besteht im Wesentlichen aus drei Schichten. Die oberste Schicht ist die „Management-Schicht“, die aus den Komponenten „Administration Manager“ und „Game Manager“ besteht. Sie ist die Fassade für den Client und somit der einzige Zugangspunkt zum Framework.
Die mittlere Schicht ist die „Context-Schicht“ und hat die Verwaltung des Spiel- und Umgebungsstatus als Hauptaufgabe. Sie besteht aus dem „Context Manager“, der die Komponenten „Session Manager“ und „Task Manager“ beinhaltet.
Die untere Schicht ist eine „Environment-Schicht“, die für die Verwaltung und Interaktion der konkreten Umgebungsdienste zuständig ist. Zu ihr gehören folgende Komponenten: „Location Layer“, „Communication Layer“ und „Configuration Manager“.

An dieser Stelle wird eine Aufteilung zwischen technischen und fachlichen Komponenten vorgenommen. Allgemein kann gesagt werden, dass die Manager-Komponenten die fachlichen und die Layer-Komponenten die technischen Komponenten des Frameworks sind.

Fachliche Komponenten

- „Administration Manager“
  Der Administration Manager ist für die Erstellung von Spiel-Routen vorgesehen. Über diese Klasse ist es möglich, bestimmte Orte als Routenpunkt zu markieren und dort eine Aufgabe bzw. einen Task zu erstellen.
- „Game Manager“
  Der Game Manager dient als zentraler Zugriffspunkt des Spiele-Frameworks. Es kapselt die Aufgaben der darunter liegenden Schichten und gibt eine bestimmten Ablauf vor.[4]
- „Context Manager“
  Der Context Manager hat die Aufgabe den aktuellen Spielstand, die Positionsdaten sowie den Verbindungsstatus zu verwalten. Zusätzlich sorgt er für die Session und Aufgabenverwaltung für die er einen Session Manager und einen Task Manager besitzt.
- „Session Manager“
  Der Session Manager ist Bestandteil des Context Managers und verwaltet die Server-Session. Die Session kann hier auf Gültigkeit überprüft und ungültig gemacht werden.
- „Task Manager“
  Der Task Manager ist für die Spielstandverwaltung und insbesondere für die Aufgabenverwaltung des Spiels zuständig.

[4] Siehe Kapitel 2.1

- „Configuration Manager“
  Der Configuration Manager ist für die Konfiguration des Frameworks bzw. der Umgebungsinformationen des Benutzers zuständig. Über diesen Manager kann z.B. eingestellt werden, unter welcher Adresse der Server erreichbar ist.

Technische Komponenten

- „Location Layer“
  Der Location Layer stellt die Positionsbestimmungs-Komponente dar. Von der konkret verwendeten Technologie wie z.B. GPS[5] wird dabei abstrahiert.
- „Communication Layer“
  Der Communication Layer ist für die Kommunikation mit dem Server und mit anderen Clients zuständig. Von der verwendete Kommunikationsform wie z.B. WLAN wird hier abstrahiert.

4.2.2 Zusammenspiel der Komponenten

Um das Zusammenspiel der einzelnen Komponenten des Frameworks aufzuzeigen, wird eine Unterteilung zwischen Client-Komponenten untereinander (*Client*) und Client-Komponenten mit Server-Komponenten (*Client/Server*) gemacht.

Client-Komponenten
Das Zusammenspiel der einzelnen Komponenten wird in der Abbildung 1 durch Pfeile visualisiert.

- „Administration Manager“
  Der Administration Manager ist eine Administrationskomponente und greift direkt auf das Interface des Context Managers zu. Er bedient sich zur Aufgabenerfüllung bei Methoden des Location Layers und des Task Managers.
- „Game Manager“
  Der Game Manager ist der zentrale Zugriffspunkt für den Anwendungsprogrammier. Er greift über das Interface des Context Managers auf die Funktionen zu, die er nicht selbst bereitstellt und macht diese nach „außen“ über sein Interface verfügbar. Über einen Event Listener kann er Informationen vom Context Manager erhalten.
- „Context Manager“
  Der Context Manager bietet seine Funktionalität über ein Inferface an.[6] Er kann sowohl auf den Location Layer als auch auf den Communication Layer über deren Interfaces zugreifen. Der Context Manager kann über den Event Listener-Mechanismus die aktuellen Positionsdaten vom Location Layer und/oder den Verbindungsstatus vom Communication Layer erhalten und sendet dieses Event weiter.

[5] Siehe Kapitel 2.2
[6] Da der Session Manager und der Task Manager ein Teil des Context Managers sind, werden diese im Zusammenspiel der Komponenten nicht extra berücksichtigt.

- „Location Layer“
  Der Location Layer bietet seine Funktionalität über ein Interface an und sendet ein Event, sobald sich die Position geändert hat.
- „Communication Layer“
  Der Communication Layer bietet seine Funktionalität über ein Interface an und sendet ein Event, sobald sich der Status der Verbindung geändert hat.
- „Configuration Manager“
  Der Configuration Manager wird vom Context Manager für die interne Konfiguration verwendet.[7]

Client/Server
Die Kommunikation mit dem Server findet über WebServices statt.
Für diese Aufgabe ist der Commuication Layer des Clients gedacht. In seinen, durch sein Interface angebotenen Methoden werden diese Server-Dienste verwendet und somit gekapselt.[8]
Der clientseitige Ablauf der Kommunikation mit dem Server ist in den Abbildungen in Anhang A.2 dargestellt.

4.3 Umsetzung

In diesem Kapitel wird die Umsetzung der Client-Architektur beschrieben. Diese geschieht anhand der konkreten Realisierung in Kapitel 4.3.1, den durchgeführten Tests in Kapitel 4.3.2 und des Deployments & Betriebs in Kapitel 4.3.3.

4.3.1 Realisierung

Für die Realisierung des Frameworks und der Beispielanwendung wurde die Programmiersprache C# verwendet. Als Entwicklungsumgebung wurde Visual Studio 2005 eingesetzt. Desweiteren wurden die Klassenbibliotheken „OpenNETCF Smart Device Framework 2.0“[9] und „Windows Mobile 5.0 Pocket PC SDK“[10] benutzt.
In diesem Kapitel wird eine Übersicht über die Klassenstruktur des Frameworks gegeben und die Hauptfunktionalität anhand einzelner Klassen dargestellt.

Framework
Das UML-Diagramm des Frameworks in Abbildung 2 veranschaulicht die Klassenstruktur des Frameworks.

Es ist zu sehen, dass sich die Client-Architektur (Abbildung 1) im Klassendiagramm widerspiegelt.

[7] Für weitere Informationen siehe Kapitel 4.3.1
[8] Für eine Übersicht der angebotenen Dienste siehe A.1
[9] OpenNETCF Smart Device Framework 2.0 - http://opennetcf.org/home.ocf - Stand 02.2007
[10] Windows Mobile 5.0 Pocket PC SDK - http://www.microsoft.com/downloads/details.aspx?familyid=83A52AF2-F524-4EC5-9155-717CBE5D25ED&displaylang=en - Stand 02.2007

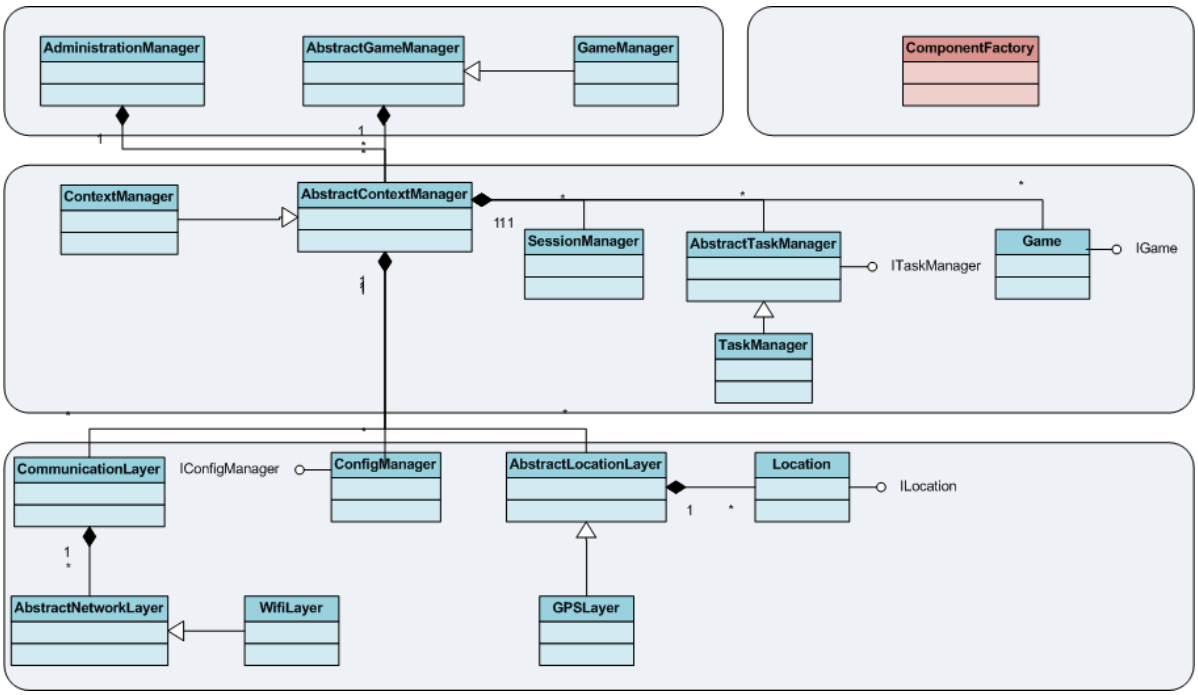


Abbildung 2: Pervasive Gaming Framework - Client UML-Diagramm

Eine zentrale Rolle im Framework spielt die abstrakte Klasse „AbstractGameManager“ und deren konkrete Implementierung „GameManager“. Der „AbstractGameManager“ gehört zum statischen Teil des Frameworks. Der „GameManager“ hingegen gehört zum dynamischen Teil und ist der Zugriffspunkt des Framework-Nutzers.

Der „AbstractGameManager“ besitzt die folgenden beiden abstrakten Methoden:

```
public abstract void LocationChangedImpl(ILocation loc);

public abstract void DestinationReachedImpl(ILocation loc);
```

Die Methode „LocationChangedImpl“ wird vom Framework immer dann aufgerufen, wenn sich die Lokation des Spielers ändert. Analog dazu wird die Methode „DestinationReachedImpl“ beim Erreichen eines Ziels aufgerufen.
Der „GameManager“ ist eine konkrete Implementierung des „AbstractGameManagers“ und besitzt dadurch die folgenden Methoden:

```
public override void LocationChangedImpl(
                        HawHamburg.Pgf.CommonNS.ILocation loc){}

public override void DestinationReachedImpl(
                        HawHamburg.Pgf.CommonNS.ILocation loc){}
```

Der Nutzer des Frameworks muss die beiden geerbten Methoden in der Klasse „GameManager“ implementieren und dort die eigene Spiellogik abbilden. Ihm wird hierdurch ein Ablauf

vorgegeben. Er muss entscheiden, was geschehen soll, wenn eine der Methoden ausgeführt wird. Um den restlichen Ablauf muss er sich nicht kümmern, da dieser vom Framework übernommen wird.
Technisch geschieht der Aufruf dieser beiden Methoden durch ein Zusammenspiel des oben dargestellten „Hook-Mechanismus“ mit dem „Event-Mechanismus“. Die Klasse „AbstractGameManager“ registriert sich bei den Events der „ContextManager“-Klasse. Wenn ein entsprechendes Event gefeuert wird, wird die entsprechend registrierte Methode aufgerufen. Ein Code-Beispiel für den Event-Mechanismus ist im Anhang A.4 zu finden.

Wie im Klassendiagramm in Abbildung 1 zu sehen existiert eine Klasse „ComponentFactory“. Diese Klasse dient der Objekterzeugung und sorgt zusätzlich dafür, dass es jeweils nur ein Exemplar der Basis-Klassen „ContextManager“, „ConfigurationManager“, „CommunicationLayer“, „SessionManager“, „AbstractTaskManager“, „AbstractLocationLayer“ und nur ein Exemplar des „PGFWebServices“ existiert.

Der „TaskManager“ liest eine XML-Datei ein, die vom Server heruntergeladen worden ist, und erhält hierdurch „Wissen“ über die aktuelle Route. Somit sind die Ziel-Position und der Ziel-Umkreis bekannt, die für den „AbstractContextManager“ von besonderer Bedeutung sind. Ein Beispiel für eine solche XML-Datei ist in Anhang A.5 zu finden.[11]

Der „AbstractContextManager“ ist das Bindeglied des Frameworks und sorgt dafür, dass die Events des „LocationLayers“ und des „CommunicationLayers“ verarbeitet und an den „GameManager“ weitergeleitet werden. In dieser Klasse wird das „DestinationReached“-Event geworfen, da hier der „TaskManager“ und somit die vom Benutzer eingestellte Route mit Zielen bekannt ist (siehe oben). Es wird überprüft, ob sich die aktuelle Position, die vom „AbstractLocationLayer“ bzw. vom konkreten „GpsLayer“ durch ein Event bekannt gemacht worden ist, im Bereich eines Zielpunktes befindet.

Der „ConfigurationManager“ liest die, für die Funktionalität des Frameworks wichtigen, Informationen aus einem XML-Config File ein. Der Benutzer des Frameworks muss diese Datei erzeugen bzw. die ausgelieferte Datei nach seinen Bedürfnissen anpassen. Die Restriktion hierfür ist, dass der Name dieser Datei „config.xml“ lauten und im Hauptverzeichnis des Spiels abgelegt werden muss. Aus dieser XML-Datei werden die Adresse des Servers, der Pfad des Web Services und die Klassennamen der konkreten Implementierungen des zu verwendenden „LocationLayers“, „NetworkLayers“ und „TaskManagers“ gelesen. Per Reflections wird mit diesen Klassennamen die Objekterzeugung der entsprechenden Klassen in der „ComponentFactory“ ermöglicht. Ein Beispiel für eine „config.xml“-Datei ist in Anhang A.6 zu finden.

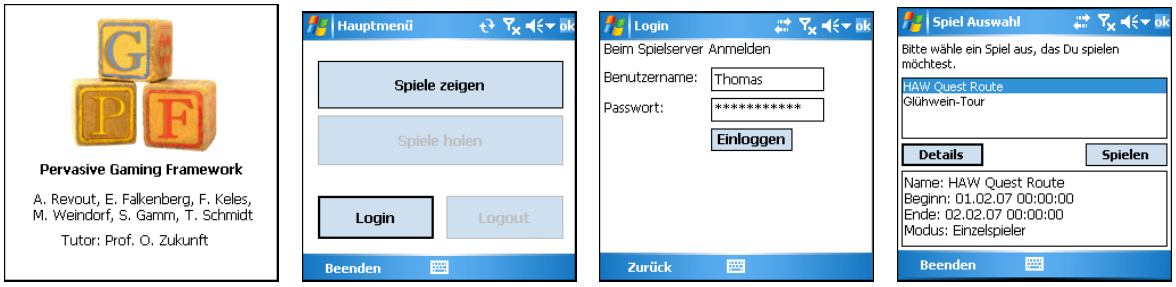
Der „CommunicationLayer“ kümmert sich um die Web-Referenzen und damit um die Kommunikation mit dem Server. Die entsprechend vorhandenen Web Services werden hier verwendet und mit entsprechendem Exceptionhandling behandelt. Hierfür wird der „AbstractNetworkLayer“ verwendet, der durch die Klasse „WifiLayer“ implementiert worden ist.

[11] Für nähere Informationen siehe [14]

Wie bereits erwähnt ist für den „NetworkLayer“ und den „LocationLayer“ jeweils eine konkrete Implementierung innerhalb des Frameworks vorhanden.
Die Wifi-Komponente ermöglicht die Nutzung von WLAN- und Kabel-Betrieb und stellt spezielle Methoden bereit. Durch diese Komponente ist es möglich auf das Internet zuzugreifen und über eine Ad-hoc-WLAN-Verbindung mit anderen Clients zu kommunizieren.
Die GPS-Komponente ermöglicht die Nutzung von GPS zur Positionsbestimmung. Mit dieser Komponente ist es möglich die aktuelle Position zu bestimmen, sofern eine Sichtverbindung zu den Satelliten vorhanden ist.[12]

Beispielanwendung
Es wurden insgesamt zwei Beispielanwendungen implementiert: „SampleClient“ und „SampleGame“. Die Implementierung der „SampleClient“ war eine frühe Entwicklung und diente der Funktionsüberprüfung der einzelnen Module.[13] Die Implementierung „SampleGame“ ist eine Beispielanwendung, die zusammen mit dem Framework ausgeliefert werden soll und nachfolgend näher beschrieben wird.

Die Beispielanwendung besteht aus einem spielunabhängigem Teil und dem eigentlichen Spiel.



(a) Intro und Extra (b) Haupt-/Auswahlfenster (c) Login (d) Spielauswahl

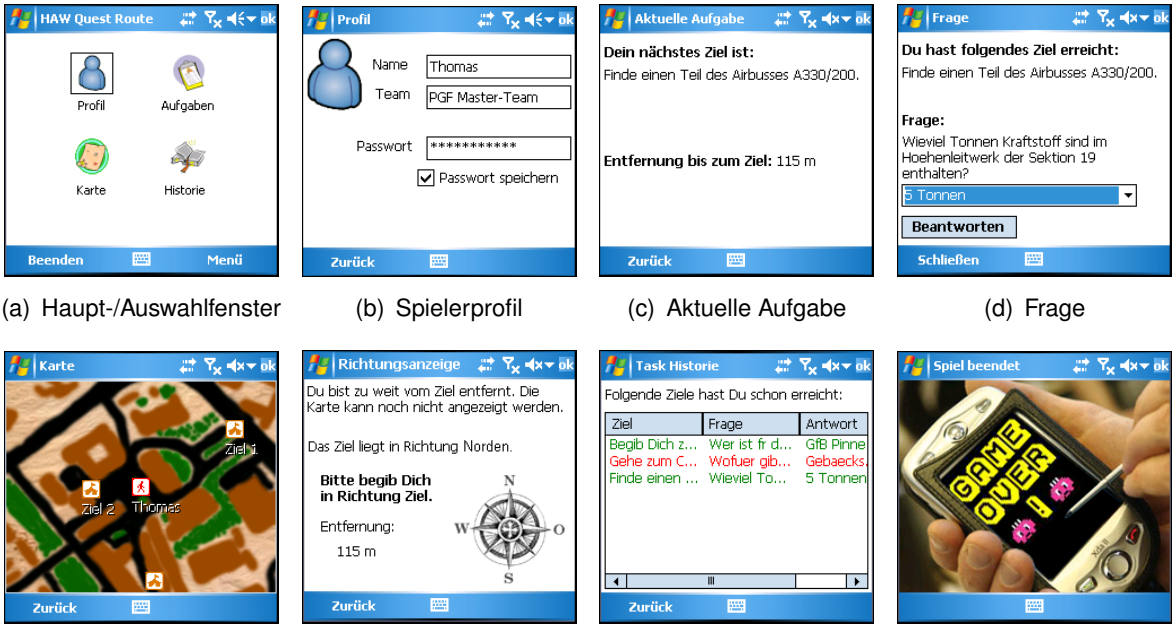
Abbildung 3: Beispielanwendung - spielunabhängige Screens

Spielunabhängiger Teil
Abbildung 3 zeigt die Screens, die nicht zu dem eigentlichen Spiel gehören. Bild 3(a) zeigt das Intro und Extra der Anwendung, auf der das Projektlogo, die Projekteilnehmer und der betreuende Professor zu sehen sind.
Bild 3(b) zeigt das Hauptfenster. Hier ist gibt es die Auswahlmöglichkeit zwischen „Spiele zeigen“, „Spiele holen“ und „Login“ bzw. „Logout“. Die Option „Spiele holen“ holt eine eine Liste von Spielen, für die sich ein Benutzer im Vorwege angemeldet hat, vom Server. Dies funktioniert nur, wenn der Benutzer eingeloggt ist. Wenn ein Benutzer nicht eingeloggt ist, wird er mit dem Screen 3(c) dazu aufgefordert. Sofern eine Liste von Spielen auf dem Gerät vorliegt, kann der Benutzer mit der Option „Spiele zeigen“ auf den Screen 3(d) gelangen. Die Spiele werden hier als Liste angezeigt. Es ist möglich Details zu einem Spiel anzuzeigen oder es zu starten.

[12] Siehe Kapitel 2.2
[13] In Kapitel 4.3.2 wird genauer auf „SampleClient“ eingegangen.

Spiel-Teil

Für die Beispielanwendung wurden zwei verschiedene Spiele entwickelt. Die „Glühwein-Tour“ und die „HAW-Tour“. Beide Spiele sind nach dem gleichen Prinzip entwickelt und unterscheiden sich nur durch Optik und Aufgabenstellung. In den Abbildungen 4 sind die Screens des



(a) Haupt-/Auswahlfenster (b) Spielerprofil (c) Aktuelle Aufgabe (d) Frage

(e) Karte (f) Ersatzkarte (g) Aufgaben-Historie (h) Spielende

Abbildung 4: Beispielanwendung - Screens

„Haw Quest Route“-Spiels zu sehen.

Im Haupt-/Auswahlfenster 4(a) hat der Spieler die Auswahl zwischen „Profil“, „Aufgaben“, „Karte“ und „Historie“. Dieser Screen ist als Hauptmenü gedacht und dient zu Navigation des Spiels. Durch Auswahl der jeweiligen Menüpunkte gelangt der Spieler zu den entsprechenden Screens.

Unter „Profil“ kann der Spieler Angaben zu seiner Person und seinem Team finden. Er hat hier zudem noch die Möglichkeit sein Passwort zu speichern, das für eine spätere Synchronisation oder einen Datei-Upload auf den Server benötigt wird.

Der Screen „Aufgaben“ 4(c) zeigt an, welche Aufgabe aktuell zu erfüllen ist und wie weit der jeweilige Erfüllungsort von der momentanen Position entfernt ist. Wenn der Spieler den Ort erreicht hat, erscheint ein Hinweis und die zum Ort gehörende Frage wird dem Benutzer wie in Screen 4(d) angezeigt. Nachdem die Frage beantwortet worden ist, bekommt man eine Rückmeldung ob die Antwort korrekt war und gelangt wieder auf den „Aufgaben“-Bildschirm.

Auf welchen Screen der Menüpunkt „Karte“ verweist, hängt von der Position des Benutzers ab. Wenn sich der Benutzer innerhalb der Karten-Koordinaten befindet, wird er zur eigentlichen Karte 4(e) und andernfalls auf die Ersatzkarte 4(f) geleitet. Die Karte dient der Orientierung und soll dem Spieler ein leichteres Auffinden des Ziels ermöglichen. An entsprechender Po-

sition wird ein Symbol des Spielers mit seinem Namen angezeigt. Zudem werden die schon erreichten und das nächste zu erreichende Ziel angezeigt und, sofern es nicht auf dem Kartenausschnitt liegt, eine Richtungsangabe mit Entfernung. Wenn das Ziel erreicht worden ist, wird wieder ein Hinweis und der Screen 4(d) mit der zugehörigen Frage angezeigt.
Unter dem Punkt Historie 4(g) kann sich der Spieler die bisher erledigten Aufgaben als Liste ansehen. Ein Eintrag enthält das Ziel, die Frage, die Antwort und das Ergebnis der Beantwortung. Durch die Farben „Grün“ für „Richtig“ und „Rot“ für „Falsch“ wird zusätzlich hervorgehoben, wie die Frage beantwortet worden ist.
Wenn alle Aufgaben erfüllt und alle Fragen beantwortet worden sind, erscheint der „Spielende“-Screen 4(h). Vom Haupt-/Auswahlfenster kann das Ergebnis auf den Server hochgeladen werden. Bei verlassen des Spiels wird das Extra 3(a) angezeigt.

4.3.2 Test

Für die Tests wird nachfolgend zwischen Framework und Beispielanwendung unterschieden, da hierbei zwei unterschiedliche Ansätze verfolgt worden sind.

Framework
Wie bereits erwähnt, wurde Visual Studio 2005 für die Entwicklung des Frameworks verwendet. Zum Testen der einzelnen Klassen wurde das Tool „TestDriven.NET“[14] verwendet. Mit diesem Plugin ist es möglich, sowohl Unit-Tests mit der NUnit-TestSuite[15] durchzuführen als auch einen Coverage-Report anzeigen zu lassen.
Die Abbildungen in Anhang A.3 geben eine Übersicht über die bisher durchgeführten Tests und die Testabdeckung. Es ist zu sehen, dass nicht alle Klassen des Frameworks vollständig getestet worden sind. Dies liegt daran, dass einige Klassen und Methoden ein mobiles Gerät voraussetzen. Dadurch ist für diese eine Durchführung von Unit-Tests nicht möglich gewesen.

Beispielanwendung
Da die Beispielanwendung für mobile Geräte konzipiert ist, konnte diese nicht durch die oben angeführten Unit-Tests auf Funktionstüchtigkeit überprüft werden. Aus diesem Grund wurden die Komponenten während der Entwicklungszyklen durch einfaches Ausprobieren getestet.

4.3.3 Deployment & Betrieb

Dieses Kapitel zeigt, welche Schritte notwendig sind, um das entwickelte Framework clientseitig in Betrieb nehmen zu können.

Das Framework wird als DLL-Datei zusammen mit einer Beispielanwendung ausgeliefert. Um das Framework zur Spieleprogrammierung verwenden zu können, muss diese DLL-Datei referenziert werden. Im Hauptverzeichnis des zu erstellenden Spiels wird eine, wie bereits in

[14] TestDriven.NET - http://www.testdriven.net/
[15] NUnit-Testsuite - http://www.nunit.org/

Kapitel 4.3.1 beschrieben, „config.xml“-Datei benötigt, die wie die in Anhang A.6 dargestellte Datei beschaffen sein muss. Der Spieleentwickler muss nun noch eine Klasse erzeugen, welche die „AbstractGameManager“-Klasse implementiert.

Nach diesen Einstellungen kann sich der Entwickler auf die Spiellogik konzentrieren und die Funktionalität des Frameworks nutzen. Durch den Code der ausgelieferten Beispielanwendung ist ersichtlich, auf welche Art das Framework genutzt werden kann und welche Funktionen bisher zur Verfügung stehen.

Erweiterungsmöglichkeit
Wenn eine andere Lokalisierungstechnik oder eine andere Kommunikationstechnik als die bisher im Framework integrierten GPS- und Wifi-Komponenten verwendet werden sollen, müssen diese Implementierungen lediglich von der entsprechenden Ober-Klasse („AbstractLocationLayer“ bzw. „AbstractNetworkLayer“) erben. Die Einstellungen in der „config.xml“ müssen entsprechend angepasst werden.

5 Fazit

In diesem Kapitel wird zunächst eine Bewertung des durchgeführten Projekts vorgenommen. Anschließend wird auf Erweiterungsmöglichkeiten des entwickelten Frameworks eingegangen.

5.1 Bewertung

In diesem Abschnitt findet eine Bewertung der Planungs- und Realisierungsphase des Projekts statt.

Planungsphase
Das Projektteam bestand aus sechs Personen, die unterschiedliche Interessenschwerpunkte hatten. Deshalb wurde versucht, die Interessen aller Teammitglieder in das Projekt zu integrieren. Dies gelang zwar weitestgehend, jedoch führte es auch im Laufe der Planungsphase zu intensiven Diskussionen über die Architektur des entwickelten Frameworks. Dies kann man als einen Vorteil sehen, da sich unterschiedliche Meinungen und Ideen ergänzten. Allerdings war das Finden von Konsensentscheidungen zeitintensiv. Außerdem wurden einige Aufgaben von verschiedenen Teammitgliedern parallel durchgeführt. Dies führte ebenfalls dazu, dass Zeit vergeudet wurde.

Da dieser Umstand den Teammitgliedern während der Planungsphase auffiel, wurde das Team in zwei Gruppen zu je drei Personen unterteilt. Eine Gruppe war für den Server-Teil zuständig, die andere für den Client-Teil. Es wurde eine klare Schnittstelle zwischen Server und Client definiert (vgl. A.1), so dass beide Teams gegen die Schnittstelle arbeiten konnten. In den kleineren Dreiergruppen konnte die Projektarbeit dann wesentlicher schneller durchgeführt werden.

Realisierungsphase
Am Ende des Projekts ist ein Prototyp entstanden, der den Anforderungen aus Kapitel 3.3 weitestgehend gerecht wurde. Wie vorgesehen, ist nicht ein konkretes pervasives Spiel entstanden, sondern ein Framework, das für die Entwicklung solcher Spiele geeignet ist. Mit der Entwicklung einer Beispielanwendung, die auf dem Framework basiert, wurde der Funktionsnachweis für das Framework erbracht.

Einige Teilaspekte sind derzeit aus zeitlichen Gründen nicht konkret ausimplementiert worden oder nicht Bestandteil des Frameworks. Beispielsweise sieht die Architektur des Frameworks die Unterstützung von Context Awareness vor. Die entsprechenden Methodensignaturen und Codebausteine hierfür existieren zwar, jedoch wird derzeit ein leeres Kontext-Objekt vom Client zum Server übertragen. Der Server hat deshalb momentan nicht die Möglichkeit, sich dem Client-Kontext anzupassen.

Die Kartenanzeige ist zu einem späten Zeitpunkt der Realisierungsphase in die Entwicklung aufgenommen worden. Dies führte dazu, dass sie nicht mehr in das Framework integriert werden konnte, weil sie Änderungen der Client/Server-Schnittstelle verursacht hätte. Diese hätten in der verbliebenen Zeit nicht durchgeführt werden können. Deshalb ist die Kartenanzeige in der aktuellen Version nicht Bestandteil des Frameworks, sondern Teil der Beispielanwendung.

5.2 Erweiterungsmöglichkeiten

Eine Möglichkeit der Erweiterung besteht darin, die noch nicht ausimplemetierten Funktionen (vgl. 5.1) in das Framework zu integrieren. Zu diesen gehört z. B. die Integration von Context Awareness.

Derzeit ist als Positionierungstechnologie lediglich GPS beispielhaft implementiert worden. Bei Bedarf könnten zusätzliche Positionierungstechnologien in das Framework eingebaut werden. Das Gleiche gilt auch für die Kommunikationstechnologie. Dort ist derzeit nur WLAN als mögliche Kommunikationstechnik vorhanden.

Die Kartenansicht ist derzeit Bestandteil der Beispielanwendung. Da sie wahrscheinlich für viele pervasive Spiele sinnvoll sein kann, sollte sie in einer späteren Version in das Framework integriert werden.

In der momentanen Version muss eine XML-Datei erstellt werden, die Routeninformationen enthält. In dieser Datei wird die Spielroute mit den zu erreichenden Zielen und dazugehörigen Fragen festgelegt[16]. Diese Datei muss mit einem Text-Editor von Hand erzeugt werden. In einem späteren Release sollte die Erzeugung der XML-Datei über eine Webanwendung mit Unterstützung des „AdministrationManagers“ möglich sein.

[16] Vgl. Anhang A.5

A Anhang

A.1 Schnittstellen zwischen Client und Server

Die Schnittstellen werden durch den Server als Webservices implementiert.

Einleitende Erklärungen:

IContext-Objekt
Das IContext-Objekt wird bei jedem Methodenaufruf an den Server übertragen. Es enthält Informationen über den aktuellen Kontext (z. B. Lokation, Energiezustand, Verbindungsqualität, etc.) des Clients. Der Server kann seine Antworten von dem Client-Kontext abhängig machen. Beispielsweise kann auf das Senden von Bildern verzichtet werden, wenn der Client eine langsame Verbindung hat.

sid (Session-Id)
Anhand der Session-Id können Server und Client Sessions eindeutig identifizieren. Die Session-Id wird verwendet, solange auf dem Client noch kein aktuelles Spiel gespielt wird. Erzeugt wird die Session-Id serverseitig beim Login-Aufruf. Anschließend wird sie dem Client zurückgeliefert. Die aktuelle Session-Id muss dann beim Herunterladen der Spiele, zu denen man angemeldet ist, beim Server-Aufruf mitgegeben werden. Die gültige Session-Id wird entweder beim anschließenden Logout oder nach Ablauf eines bestimmten Timeout gelöscht.

gameId (Game-Id)
Beim Erzeugen einer konkreten Spielinstanz erhält diese eine Game-Id, mittels derer jedes laufende bzw. abgeschlossene Spiel eindeutig identifiziert werden kann. Bei einem Multi-Player-Spiel erhält also jeder Mitspieler ein Spiel mit derselben Game-Id.

gsid (Game-Session-Id)
Die Game-Session-Id wird serverseitig beim Herunterladen der Spiele, zu denen man angemeldet ist, erzeugt. Dabei findet eine Zuordnung von einem über die mitgelieferte Session-Id identifizierten Benutzer zu einer konkreten Spiel-Instanz (siehe gameId) statt, so dass der Server bei späteren Webservice-Aufrufen anhand der mitgelieferten Game-Session-Id das entsprechende Spiel und den aktuellen Spieler identifizieren kann. Beim Hochladen der endgültigen Spielergebnisse bzw. nach Ablauf eines bestimmten Timeouts wird eine Game-Session-Id gelöscht.

Es folgen die Methodensignaturen:

Einloggen: Der User loggt sich beim Server ein. Bei erfolgreichem Login wird eine Session-Id (sid) als String zurückgegeben; sonst wird null zurückgegeben.

```
string Login(string username, string password, IContext context);
```

Ausloggen: Der User loggt sich vom Server aus. Bei erfolgreichem Logout wird true zurückgegeben, sonst wird false zurückgegeben.

```
bool Logout(IContext context, string sid);
```

Spiele anzeigen: Liefert alle Spiele als Array von IGame-Objekten zurück, für die sich der aktuelle User angemeldet hat und für die der Anmeldeschluss schon abgelaufen ist.

```
IGame[] GetJoinedGames(IContext context, string sid);
```

Das IGame-Interface beschreibt eine Spiel-Bean und hat folgende Instanzvariablen (mit Accessoren):

```
byte[] RouteXML                                  // Die Route des Spiels:
                                                 // XML-Datei
string[] PersonalRoutePoints; {set; get;}        // Die Routenpunkte des Spielers
IGameDescription GameDescription {set; get;}
String GameSessionId;                            // Die Game-Session-Id
```

Das IGameDescription-Interface hat folgenden Aufbau:

```
string GameId {set; get;}                        // Die Game-Id
string GameName {set; get;}                      // Bezeichnung des Spiels
bool IsSinglePlayerGame {set; get;}              // true, wenn es sich um ein
                                                 // Einzelspieler-Spiel handelt;
                                                 // false, sonst
Date GameStart {set; get;}                       // Start des Spiels
Date GameEnd {set; get;}                         // Datum, wann das Spiel
                                                 // spätestens beendet ist
```

Zwischenergebnis hochladen: Lädt ein Zwischen-Spielergebnis (XML-Datei) auf den Server hoch. Bei Erfolg wird true zurückgeben, sonst wird false zurückgegeben.

```
bool SubmitTempResult(byte[] resultXML, IContext context, string gsid)
```

Endergebnis hochladen: Lädt ein End-Spielergebnis (XML-Datei) auf den Server hoch. Bei Erfolg wird true zurückgeben, sonst wird false zurückgegeben.

```
bool SubmitEndResult(byte[] resultXML, IContext context, string gsid)
```

Daten vom Server holen: Holt sich vom Server Daten ab. Diese Daten können z. B. aktuelle Spielstände der Mitspieler sein. Sie können auch Quests enthalten, die der Spieler als zusätzliche Aufgaben bekommen soll.

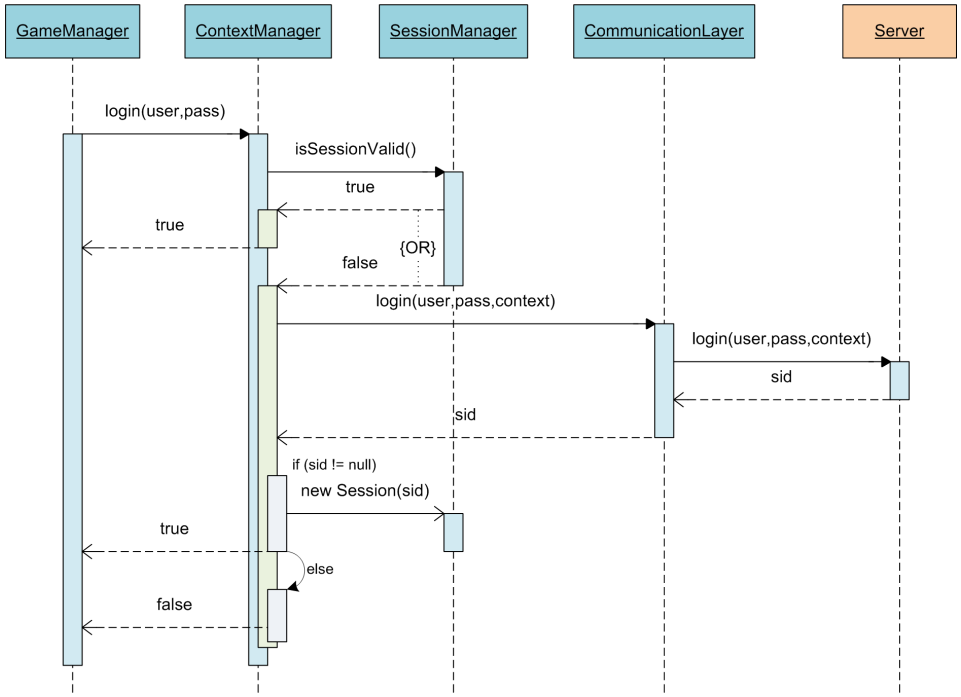
```
byte[] GetData(IContext context, string gsid)
```

Spiel abbrechen: Client teilt einen Spielabbruch an den Server mit. Bei Erfolg wird true zurückgeben, sonst false.

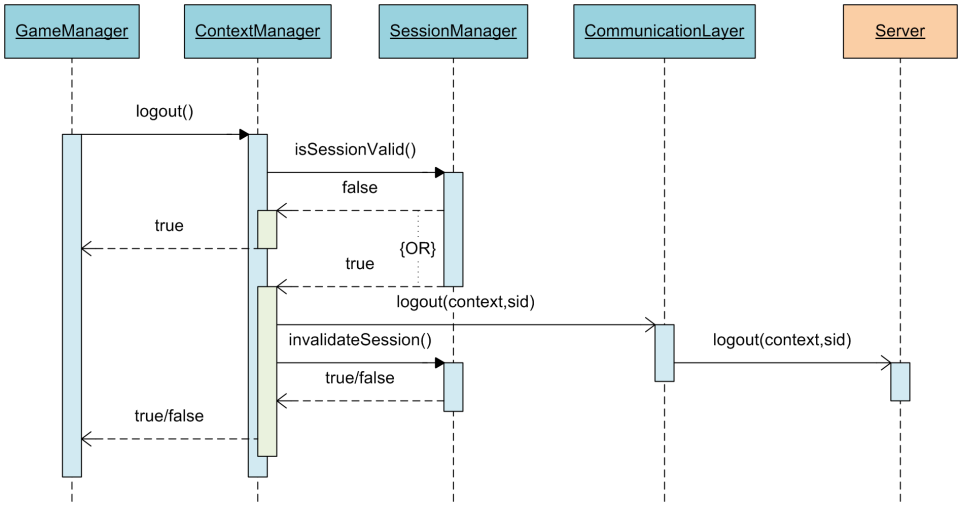
```
bool AbortGame(IContext context, string gsid)
```

A.2 Ablaufdiagramme

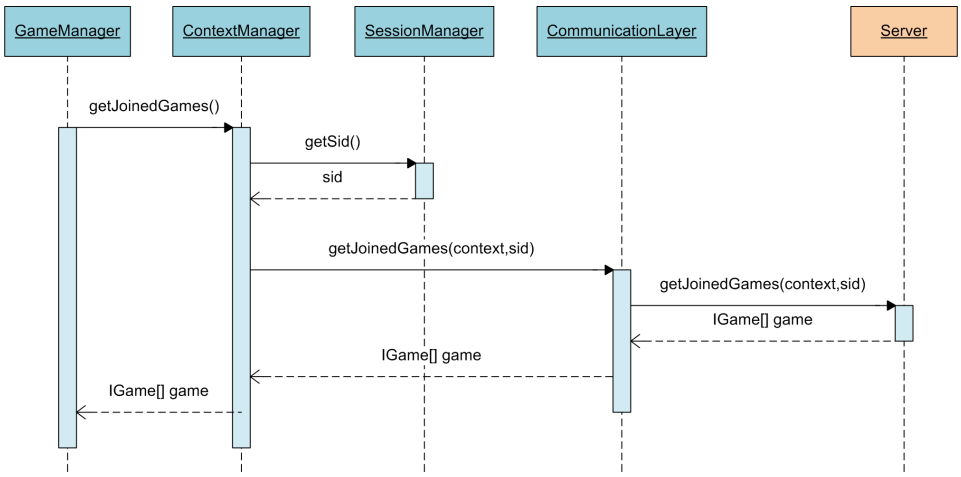
In diesem Kapitel werden die Ablaufdiagramme für die Kommunikation des Clients mit dem Server gezeigt. Dabei wird nur auf die Client-internen Details eingegangen. Die Server-internen Abläufe können in [4] und [10] nachgelesen werden.



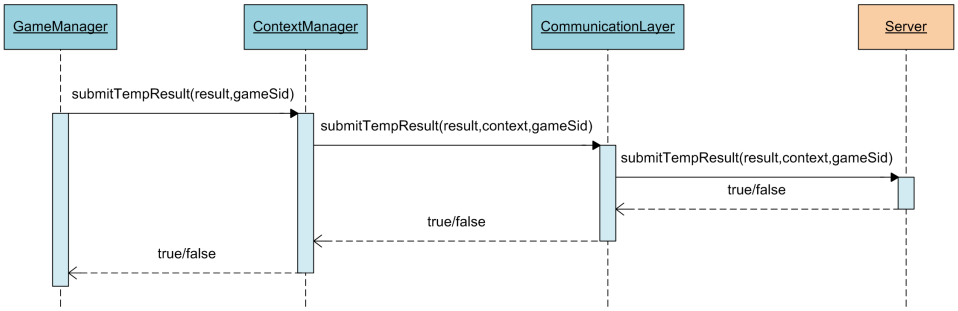
Ablaufdiagramm: Einloggen



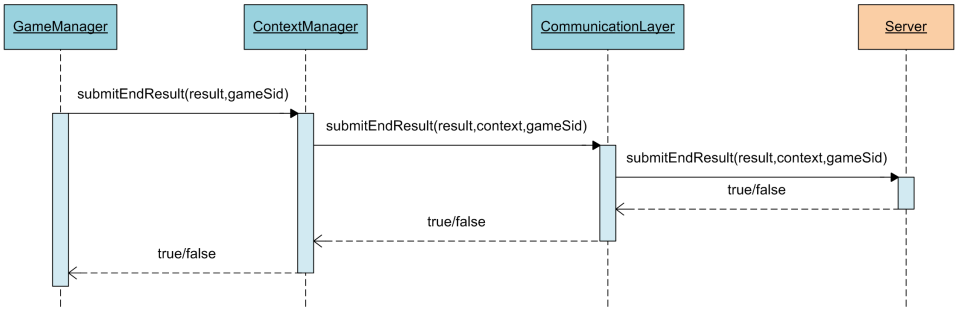
Ablaufdiagramm: Ausloggen



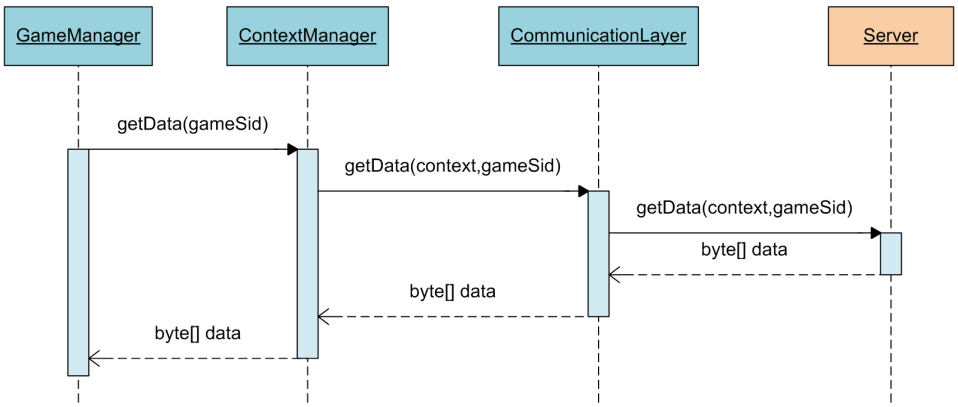
Ablaufdiagramm: Spiele anzeigen



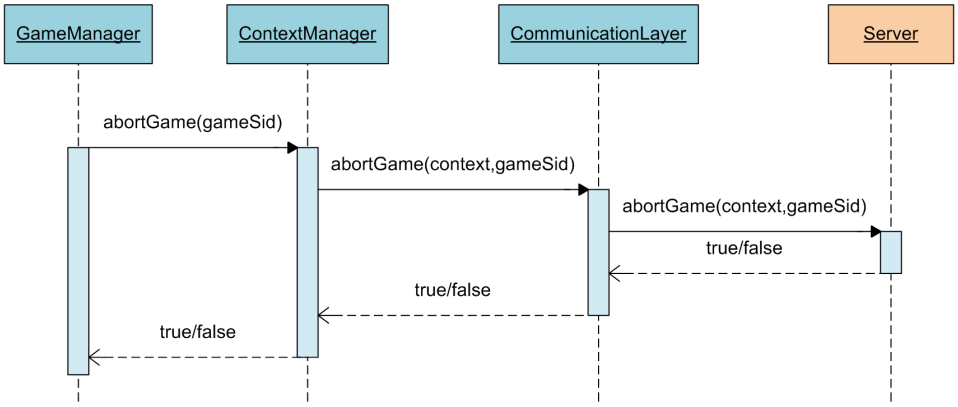
Ablaufdiagramm: Zwischenergebnis hochladen



Ablaufdiagramm: Endergebnis hochladen



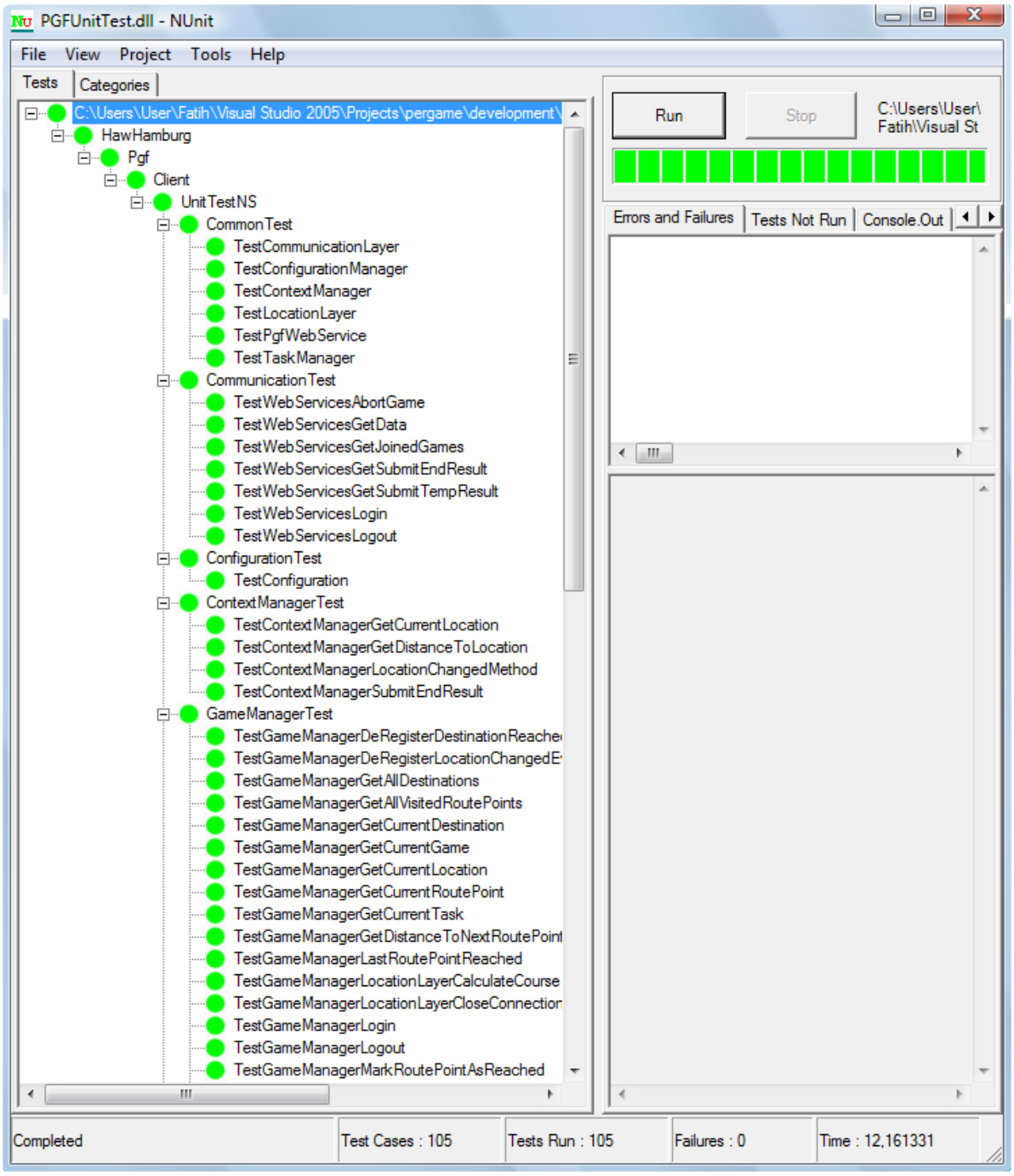
Ablaufdiagramm: Daten vom Server holen



Ablaufdiagramm: Spiel abbrechen

A.3 Tests

In diesem Kapitel ist ein Screenshot der durchgeführten Tests und ein Screenshot der Testabdeckung zu sehen.



Unit-Tests

NCoverExplorer Coverage Report - Pervasive Gaming Framework

Report generated on: Wed 28-Feb-2007 at 18:07:36
NCoverExplorer version: 1.3.6.2004
Filtering / Sorting: None / Name

Project Statistics: **Files:** 18 **NCLOC:** 992 **Classes:** 19 **Functions:** 256 **Unvisited:** 33 **Seq Pts:** 993 **Unvisited:** 312

Project	Acceptable	Unvisited Functions	Function Coverage
Pervasive Gaming Framework	80.0 %	33	**87.1 %**

Modules	Acceptable	Unvisited Functions	Function Coverage
PGFClient.dll	80.0 %	33	**87.1 %**

Module	Acceptable	Unvisited Functions	Function Coverage
PGFClient.dll	80.0 %	33	**87.1 %**
Namespace / Classes			
HawHamburg.Pgf.Client.CommonNS		0	**100.0 %**
ComponentFactory		0	**100.0 %**
HawHamburg.Pgf.Client.CommunicationLayerNS		1	**91.7 %**
AbstractNetworkLayer		0	**100.0 %**
CommunicationLayer		1	**90.9 %**
HawHamburg.Pgf.Client.CommunicationLayerNS.WifiNS		5	**83.9 %**
WifiException		0	**100.0 %**
WifiLayer		5	**83.3 %**
HawHamburg.Pgf.Client.ConfigurationManagerNS		0	**100.0 %**
ConfigurationManager		0	**100.0 %**
HawHamburg.Pgf.Client.ContextManagerNS		1	**94.7 %**
AbstractContextManager		1	**94.4 %**
ContextManager		0	**100.0 %**
HawHamburg.Pgf.Client.GameManagerNS		3	**91.4 %**
AbstractGameManager		3	**91.4 %**
HawHamburg.Pgf.Client.LocationLayerNS.GpsNS		2	**96.7 %**
DegreesMinutesSeconds		0	**100.0 %**
GpsException		0	**100.0 %**
GpsLayer		2	**95.0 %**
GpsLocation		0	**100.0 %**
HawHamburg.Pgf.Client.PgfWebService		17	**55.3 %**
Game		0	**100.0 %**
PGFService		17	**22.7 %**
HawHamburg.Pgf.Client.SessionManagerNS		0	**100.0 %**
Session		0	**100.0 %**
SessionManager		0	**100.0 %**
HawHamburg.Pgf.Client.TaskManagerNS		4	**87.5 %**
AbstractTaskManager		3	**88.5 %**
TaskManager		1	**83.3 %**

Coverage Report

A.4 Event-Mechanismus

Dieses Kapitel zeigt die Grundbestandteile des Event-Mechanismus. Das Beispiel wurde der Klasse „AbstractGameManager“ entnommen.

Initialisierung
Ein ContextManager-Objekt wird initialisiert.

```
private AbstractContextManager contextManager;

public AbstractGameManager()
    {
        contextManager = ComponentFactory.GetContextManager();
    }
```

Events registrieren
Registrierung an den Events des ContextManagers. Wenn dort das LocationChanged-Event gefeuert wird, wird die Methode LocationChanged(ILocation) aufgerufen.

```
public void RegisterLocationChangedEvent()
    {
        contextManager.LocationChanged += LocationChanged;
    }

private void LocationChanged(ILocation loc)
    {
    ...
    }
```

Events deregistrieren
Deregistrierung an den Events des ContextManagers. Wenn dort das LocationChanged-Event gefeuert wird, wird die Methode LocationChanged(ILocation) nicht mehr aufgerufen.

```
public void RegisterLocationChangedEvent()
    {
        contextManager.LocationChanged -= LocationChanged;
    }
```

A.5 XML-Route

Beispiel einer Route in XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<route xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <route_identifier>HAW Quest Route</route_identifier>
  <description>Dies ist die HAW Quest Route. Viel Spass beim spielen.</description>
  <total_time>3</total_time>

  <route_pointsType>
    <location_identifier>Ziel 1</location_identifier>
    <description>Begib Dich zu der Baustelle neben dem HAW Parkplatz.</description>
    <sequence>0</sequence>
    <position>
      <radius>20</radius>
      <x_position>53.55754006</x_position>
      <y_position>10.02443911</y_position>
      <z_position>0</z_position>
    </position>
    <time>1</time>
    <visible_at>
      <position>
        <radius>20</radius>
        <x_position>53.55754006</x_position>
        <y_position>10.02443911</y_position>
        <z_position>0</z_position>
      </position>
      <time>1</time>
      <sequence_number>0</sequence_number>
    </visible_at>
    <tasksType>
      <task_identifier>Frage 1</task_identifier>
      <description>Frage 1</description>
      <question_task>
        <question>
          Wer ist fuer den Bau des Daches des Neubaus (Hydraulikpresse) zustaendig?
        </question>
        <choicesType>
          <answer>Bramfelder Bedachungs GmbH</answer>
          <is_right>0</is_right>
        </choicesType>
        <choicesType>
          <answer>Schroeder Dach- und Haustechnik</answer>
          <is_right>0</is_right>
        </choicesType>
        <choicesType>
          <answer>GfB Pinneberg GmbH</answer>
          <is_right>1</is_right>
        </choicesType>
```

```
        <choicesType>
          <answer>ATH GbR</answer>
          <is_right>0</is_right>
        </choicesType>
      </question_task>
      <time>0.2</time>
      <points>10</points>
    </tasksType>
  </route_pointsType>

  <route_pointsType>
    ...
  </route_pointsType>

  ...

</route>
```

A.6 config.xml

Beispiel einer Konfigurationsdatei für das Framework.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <server-address>
    141.22.11.190
  </server-address>
  <webservice-path>
    /pgfws/pgfservice.asmx
  </webservice-path>
  <location-layer-class-name>
    HawHamburg.Pgf.Client.LocationLayerNS.GpsNS.GpsLayer
  </location-layer-class-name>
  <network-layer-class-name>
    HawHamburg.Pgf.Client.CommunicationLayerNS.WifiNS.WifiLayer
  </network-layer-class-name>
  <task-manager-class-name>
    HawHamburg.Pgf.Client.TaskManagerNS.TaskManager
  </task-manager-class-name>
</configuration>
```

Literatur

[1] ALONSO, Gustavo ; CASATI, Fabio ; KUNO, Harumi ; MACHIRAJU, Vijay: *Web Services - Concepts, Architektures and Applications*. Springer Verlag, 2004

[2] BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Pattern-orientierte Softwarearchitektur - Ein Pattern-System*. Addison-Wesley, 1998

[3] DEY, Anind K. ; ABOWD, Gregory D.: Towards a Better Understanding of Context and Context-Awareness. In: *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, April 2000

[4] GAMM, Stephanie: Ausarbeitung Projekt HAW Hamburg (Veranst.), 2007

[5] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster - Elemente wiederverwertbarer objektorientierter Software*. Addison-Wesley, 1996

[6] HAUSER, Tobias ; LÖWER, Ulrich M.: *Web Services - Die Standards*. Galileo Press, 2004

[7] MAGERKURTH, Carsten ; CHEOK, Adrian D. ; MANDRYK, Regan L. ; NILSEN, Trond: Pervasive games: bringing computer entertainment back to the real world. In: *Computers in Entertainment* 3 (2005), Nr. 3, S. 4

[8] MANSFELD, Werner: *Satellitenortung und Navigation - Grundlagen und Anwendung globaler Satellitennavigationssysteme*. Friedr. Vieweg & Sohn Verlag, 1998

[9] PREE, W.: Meta Patterns - A Means for Capturing the Essentials of Reusable Object-Oriented Design. In: *Proceedings of ECOOP'94*, 1994, S. 150–162

[10] REVOUT, Alexandra: Ausarbeitung Projekt HAW Hamburg (Veranst.), 2007

[11] ROTH, Jörg: *Mobile Computing - Grundlagen, Technik, Konzepte*. dpunkt.verlag, 2005

[12] SATYANARAYANAN, M.: *Pervasive Computing: Vision and Challenges*. Mai 29 2001

[13] SCHMIDT, Albrecht ; BEIGL, Michael ; GELLERSEN, Hans-Werner: There is more to context than location, 1999, S. 893–901

[14] WEINDORF, Maik: Ausarbeitung Projekt. (2007)