Konzept und Implementierung einer Sprache zur Umsetzung von Multi-Client-Datenübertragung

Diplomarbeit

Denis Mederake

Mathematische Fakultät
der
Ruhr-Universität Bochum

Essen im April
Betreuung: Prof. Dr. Eberhard Bertsch

Inhaltsverzeichnis

Einleitung

Ausgangssituation

Das Internet hat sich in den letzten 10 Jahren ausgesprochen rasant entwickelt. Wurde es anfänglich überwiegend für E-Mails und hauptsächlich von einem kleinen Benutzerkreis verwendet, ist es inzwischen beinahe allgegenwärtig. Jeder Berufssparte und auch jedem Privatanwender stehen ein unglaublicher Pool an Informationen zur Verfügung, der fast allumfassend scheint.

Obwohl die grundlegenden Techniken bereits zur Jahrtausendwende zur Verfügung standen, hat das Internet ungefähr 2004 einen so großen Sprung in der Nutzung und der durch das Web zur Verfügung gestellten Möglichkeiten erfahren, dass an vielen Stellen von "Web 2.0" gesprochen wird [Web2].
Meistens sind damit Anwendungen gemeint, die im Internetbrowser laufen und durch entsprechende Programmierung eine Interaktion erlauben, die über das übliche Anfragen von Text hinaus geht. So ist es zum Beispiel möglich, bei Google Docs eine komplette Textverarbeitung im Browser zu benutzen. Dies funktioniert sogar mit mehreren Benutzern, die das gleiche Dokument bearbeiten [GDocs].

Zusätzlich wird das Internet aber auch stark von Computerspielern genutzt, die über das Internet Spielerlebnisse in virtuellen Welten teilen. Dafür wird in der Regel eine "Server-Multiclient"-Beziehung aufgebaut. Das bedeutet, dass mehrere Clients (die Spieler) an einem Server zentral zusammengeführt werden. Normalerweise muss dafür auf dem Server spezielle Software betrieben werden, die den Daten- und Informationsfluss steuert.

Viele Literaturhinweise im vorliegenden Text beziehen sich auf Internetquellen.

In den meisten Fällen handelt es sich dabei um Hinweise auf weiterführende Informationen, die die angesprochenen Themen vertiefen sollen.
Da das Internet ständigen Änderungen unterliegt, kann nicht sichergestellt werden, dass die angegebenen Quellen auch weiterhin existieren. Aus diesem Grund sind auch die Internetquellen mit einer Jahreszahl versehen. Diese Jahreszahl orientiert sich an dem Datum der letzten Änderung der Seite, wenn dies auf der Seite ersichtlich ist. Ansonsten wird die aktuelle Jahreszahl bzw. die Jahreszahl aus der Copyrightangabe übernommen.

Begriffserklärung

Für die am häufigsten verwendete Technik hinter "Web 2.0" hat sich das Schlagwort "AJAX" (Asynchronous JavaScript and XML) etabliert. Dies bezeichnet die Fähigkeit der gängigen Browser, HTTP Anfragen an den Server zu senden (die dann häufig mit XML beantwortet werden), ohne die komplette Seite neu aufbauen zu müssen [Request].

PHP (rekursives Akronym: PHP Hypertext Preprocessor) ist eine im Internet weit verbreitete Skriptsprache, die hauptsächlich benutzt wird, um HTML Dokumente zu erzeugen. Obwohl es zwar möglich ist, mit PHP auch Kommandozeilenprogramme und (mit entsprechender Erweiterung) Desktop-Applikationen zu entwickeln, wird hier darauf nicht näher eingegangen, da hier nur der Einsatz auf Webservern betrachtet wird.

Bei vielen Internet-Hosting-Anbietern gibt es die Möglichkeit, neben statischen HTML-Seiten auch durch PHP dynamisch erzeugte Seiten abzulegen. Diese PHP-Skripte beziehen häufig Daten aus einer MySQL Datenbank. Da der verwendete Web-Server meist Apache ist, wird diese Web-Hosting-Kombination mit AMP (**A**pache **M**ySQL **P**HP) bezeichnet.

Wenn in dieser Arbeit von Echtzeitfähigkeit gesprochen wird, so ist damit kein Determinismus in der Laufzeit gemeint. Aufgrund der Asynchronität des Internets kann keine sichere Aussage über Dauer zwischen Anfrage und Antwort gemacht werden.
Vielmehr ist eine möglichst verzögerungsfreie Datenübertragung gemeint, also "weiche" Echtzeit. Durch Messungen der tatsächlichen Dauer der Datenübertra-

gung, Optimierungen der Skripte und Untersuchung der Laufzeit wird versucht, Daten schnellstmöglich von einem Rechner auf einen anderen zu übertragen. Die zentrale, vermittelnde Instanz ist dabei ein PHP-Webserver. Die Daten sollen dabei möglichst so schnell am jeweils anderen Rechner ankommen, dass dort eine weitgehend reibungslose Darstellung möglich ist. Wie diese Darstellung genau aussieht, hängt dabei von der Art der übertragenen Daten und dem Zweck der Anwendung ab.

Zielsetzung

Mit dieser Arbeit soll getestet werden, in wie weit eine AMP-Konstellation dazu geeignet ist, als Server für eine Multiuser-Umgebung genutzt zu werden.
Dies bedeutet, dass über ein AMP-System Daten übertragen werden sollen, über die mehrere Benutzer gleichzeitig in einer virtuellen Umgebung interagieren können. Aufgrund der hohen Verfügbarkeit und der relativ geringen Kosten von AMP-Umgebungen ist eine Implementierung einer solchen Echtzeit-Multiplayer-Umgebung besonders für nicht-professionelle Benutzer interessant.
Da eine harte Echtzeit im Internet nicht möglich ist und die Gesamtlaufzeit aufgrund der serverseitig verwendeten PHP-Skripte zusätzlich verlängert wird, ist nicht zu erwarten, dass die gleichen Anwendungen abgebildet werden können, die mit einer dedizierten Servertechnik möglich sind. Vielmehr soll es darum gehen, neben der reinen Machbarkeit zu untersuchen, welche Verzögerung zu erwarten ist und Beispiele für Anwendungen zu liefern, die damit umgesetzt werden.
Das Ergebnis soll ein Werkzeug in Form eines Compilers sein, mit dem die serverseitigen Komponenten einfach erzeugt werden können. Durch eine einfache Beschreibungssprache kann bestimmt werden, wie die zu übertragenden Daten aussehen. Der Compiler ermittelt daraus die benötigte Datenbankstruktur und erzeugt die PHP-Dateien, die anschließend benutzt werden können, um den Server aufzubauen.

Versuche mit PHP-Skripten zum Datenabgleich wurden bereits unternommen [BuSu]. Allerdings ging es dabei mehr um die Interaktion über AJAX und somit um Browsertechniken. Es wurde weniger Fokus auf die Auslotung der serverseitigen Möglichkeiten und Beschränkungen gelegt. Außerdem ist der Echtzeitaspekt

hintergründig. Eine Verzögerung der Übertragung hat keine Auswirkung auf die Funktionalität der Seite.
Genauso existieren PHP-basierte Onlinespiele, die generell ein rundenbasiertes Konzept verfolgen, oder zumindest durch einzelne Schritte in Form von separaten Browseranfragen (= Absenden/Anfragen einer Webseite) realisiert sind [QStar]. Viele existierende Implementierungen von Chat-Systemen, die auf PHP basieren, zeigen aber, dass generell weiche Echtzeitanwendungen denkbar sind.

Im Bereich der Browserspiele existieren auch viele Programme, die auf der Flash-Technologie beruhen [Flash]. Auch Multiclient-Anwendungen sind vorhanden, setzen aber meistens, ähnlich wie PC-Multiplayer-Spiele, einen Server mit dedizierter Software voraus, zum Beispiel einen Smart-Fox-Server [SFox]. Dies weicht von der Grundidee dieser Arbeit ab, einen günstigen und einfach verfügbaren Server zu nutzen.

Insbesondere sind die oben erwähnten Beispiele explizit für einen Nutzen erstellte Programme. Die Serverkomponenten wurden speziell für einen Zweck entwickelt.
Durch die Bereitstellung eines Compilers kann das erzeugte Framework für allgemeine Aufgaben verwendet werden.

Aufbau der Arbeit

Die Untersuchung der Echtzeitfähigkeit des PHP-Servers ist in drei Schritte unterteilt:

1. Machbarkeitsanalyse
   Zuerst wird ein Spiel für zwei Spieler mit JavaScript realisiert. Das gibt die Möglichkeit einer ersten einfachen Echtzeitdarstellung. Damit kann die Echtzeitfähigkeit von PHP bestätigt werden. Zusätzlich wird ein Chat implementiert, damit das Spiel auch wirklich einsetzbar ist und die Benutzer Spiele vereinbaren können, ohne sich gegenüber zu sitzen.
   Die Nutzung von JavaScript erzeugt zwar einen gewissen Overhead im Bezug auf die Performance, dafür ist die Entwicklung aber vereinfacht und schneller anpassbar.
   Für diese erste Phase ausreichend erscheint eine Umsetzung des Spiels

Pong [Pong], bei dem jeder Spieler einen Balken steuert, mit dem er verhindern muss, dass ein Ball das Spielfeld verlässt.

2. Compiler zur Erzeugung der Serverdateien
   Ziel ist es, ein PHP-Grundgerüst (Framework) zu erstellen, das die Datenübertragung ermöglicht. Dies wird erreicht, indem ein Übersetzer gebaut wird, der aus einer einfachen Beschreibungssprache die benötigten PHP-Dateien generiert. Das bedeutet, dass in der Beschreibungssprache definiert wird, welche Daten übertragen werden müssen, welche Informationen davon vom Client und welche vom Server generiert werden und im letzten Fall auch, wie der Server die Daten erzeugen kann. Die vom Übersetzer erstellten Dateien können dann auf einen Webserver übertragen werden, um von den Clients für den Datenabgleich angesprochen zu werden. Zusätzlich werden PHP-Dateien erzeugt, die auf dem Server die benötigten Datenbankstrukturen bereitstellen.
   Auf diese Weise wird nur der serverseitige Ablauf definiert. Die Client-Software ist dadurch vollkommen flexibel und kann somit in einer beliebigen Sprache entwickelt werden.

3. Testprogramm
   Für die Überprüfung des PHP-Frameworks wird ein Testprogramm entwickelt.
   Mit Hilfe der im zweiten Teil definierten Grammatik wird eine Datenmenge festgelegt, um dann mit dem erstellten Framework die benötigten PHP-Dateien zu erstellen.
   Außerdem wird ein Java-Client implementiert, um die Funktionalität zu testen.

Kapitel 1

Umgebung

1.1 Entwicklung

Da das Projekt Umsetzungen in PHP, JavaScript und Java erfordert, bietet sich als Entwicklungsumgebung "Eclipse" an. Von Haus aus mit voller Java Unterstützung ausgestattet kann es durch Add-Ons um Syntax-Highlighting und -Vervollständigung für JavaScript und PHP ergänzt werden.
Umgesetzt werden die Komponenten auf einem Desktop PC, wobei die folgende Software eingesetzt wird:

- Apache 2
- MySQL 5
- PHP 5
- Eclipse

Um die Kompatibilität der entwickelten JavaScript Module möglichst browserübergreifend sicherstellen zu können, werden bei der Entwicklung folgende Browser zum Testen eingesetzt:

- Mozilla Firefox 3
- Opera 9
- Microsoft Internet Explorer 6 (in einer VMWare virtuellen Instanz von WindowsXP SP 2)

1.2 Zielumgebung

Da es sich bei dem Projekt um eine Web basierte Entwicklung handelt, sollte das Ergebnis weitgehend systemunabhängig lauffähig sein. Generell ist dabei zwischen den Server- und den Client-Komponenten zu unterscheiden.

Als Webserver muss mindestens ein HTTP-Server mit PHP4 Unterstützung vorliegen. Dabei ist es nicht zwingend notwendig, dass tatsächlich Apache zugrunde liegt. So wäre z.B. auch eine Umsetzung auf einem Microsoft Internet Information Server (IIS) denkbar, wenn dort eine PHP-Engine installiert ist. Aufgrund der dafür anfallenden Lizenzkosten ist aber bei den meisten Hosting-Anbietern eher Apache anzufinden.

Die Tests im Rahmen dieses Projektes finden auf einem Web-Server von Planet Hosting (www.planet-hosting.de) statt, auf dem folgende Software eingesetzt wird:

- Apache 2
- MySQL 4
- PHP 5

Für den abschließenden Test in Kapitel 4 wurde zusätzlich ein Web-Server von all-inkl.com eingesetzt, um einen Vergleich der Verbindungswerte zu erhalten. Die dort eingesetzte Software enspricht der des Angebots von Planet Hosting.

Um möglichst viele der gängigen Web-Server-Angebote zu berücksichtigen, werden lediglich Funktionen aus dem Umfang von PHP4 verwendet. Dabei wird aber auch darauf geachtet, bei den benutzten Aufrufen eine Syntax zu verwenden, die sowohl von PHP4 als auch von PHP5 unterstützt wird.

Die Clientunterstützung gestaltet sich in den zwei Projektphasen unterschiedlich:

In Phase eins wird JavaScript verwendet, um die Interaktion mit dem Server abzubilden. Dies bedeutet, dass das zugrunde liegende System nicht vorgeschrieben werden sollte. JavaScript wird normalerweise von einem Browser ausgeführt und hat aus Sicherheitsgründen in der Regel nur eingeschränkten Zugriff auf Systemressourcen. Darum muss das Programm weniger an spezielle Sytemum-

gebungen als an die Eigenheiten der verschiedenen Browser angepasst werden. Dieser Anforderung wird durch den Einsatz der gängigsten Browser (Firefox, Internet Explorer, Opera) bei den Tests nachgekommen.

Der Client in Phase zwei wird in Java umgesetzt. Dadurch erhält man auf der einen Seite eine höhere Flexibilität, da das Java Framework einen weitaus größeren Umfang hat, bleibt aber auf der anderen Seite unabhängig von der Zielarchitektur, da Java in eine systemunabhängige Zwischensprache übersetzt wird. So wäre zum Beispiel in Java durch Verwendung der Java3D Bibliothek eine Darstellung in 3D möglich, was JavaScript nicht erlaubt. Das erstellte Programm kann dann auf allen System verwendet werden, die eine entsprechende Java Laufzeit Umgebung (Java Runtime Environment RTE) zur Verfügung stellen.

Kapitel 2

Stufe 1: Machbarkeitsanalyse

Zur Überprüfung der Machbarkeit des Projektes wird ein einfaches Programm erstellt, das Daten von einem Client zu einem anderen Client überträgt und als Serversoftware PHP verwendet.

Für diesen Test wird eine Version des Spiels ”Pong” erstellt. Bei diesem einfachen Spiel steuern zwei Spieler jeweils einen Strich (Schläger) mit dem sie einen Ball auf dem Spielfeld halten müssen. Zusätzlich ermöglicht das Programm eine Kommunikation zwischen den Clients über einen einfachen Chat. Dies beinhaltet auch das Anmelden der Benutzer mit einem Benutzernamen und Passwort. Auch wenn dies nur eine Testphase ist, soll die Übertragung des Passwortes nicht im Klartext geschehen. Dies ist möglich, da das Passwort auf dem Server im Klartext gespeichert wird. Dadurch kann man mit folgenden Schritten das Passwort überprüfen, ohne es im Klartext über das Internet übertragen zu

müssen:

1. Benutzer gibt das Kennwort auf dem Client ein
2. Clientprogramm erstellt einen Zeitstempel
3. Das Kennwort wird zusammen mit dem Zeitstempel durch die MD5-Hash-Funktion in eine 32-stellige Zeichenfolge überführt
4. Die Zeichenfolge und der erstellte Zeitstempel werden an den Server übermittelt
5. Da dem Server das Kennwort bekannt ist, kann er darauf und auf den übermittelten Zeitstempel ebenfalls den MD5-Hash anwenden
6. Stimmt die auf dem Server erzeugte Zeichenkette mit der vom Client übermittelten überein, war die Anmeldung erfolgreich

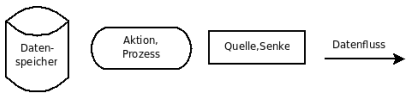
Der Einfachheit halber und um einen kürzeren Entwicklungs- und Debug-Zyklus zu haben, werden nicht nur die Serverkomponenten (PHP) sondern auch das Clientprogramm (JavaScript) in einer Skriptsprache entworfen . Auf dem Client wird also immer ein Browser benötigt, um den Test durchzuführen.

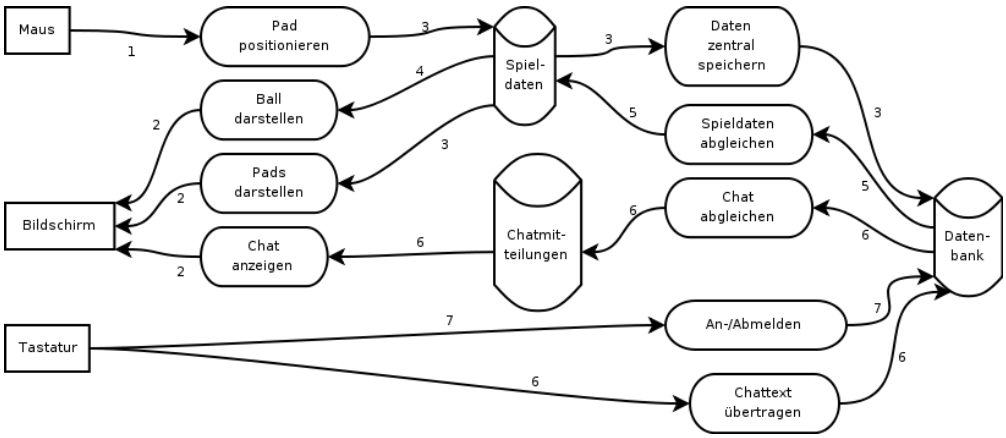
2.1 Datenflussdiagramm

Da aus Performancegründen sowohl auf Client- als auch auf Serverseite auf Objektorientierung verzichtet wird, lässt sich das Konzept am besten mit einem Datenflussdiagramm gemäß der "Structured Analysis" (SA) von Tom deMarco darstellen [Marco78].

Symbole

Für eine übersichtlichere Darstellung wurden die Symbole für Datenspeicher (bei deMarco zwei parallele Linien) von den Quellen und Senken (Rechtecke) abgehoben, indem dafür Zylinder verwendet werden.





Aufgrund der generellen Struktur des Projektes sind außerdem noch folgende Besonderheiten zu berücksichtigen:

- Die Ausgabe findet nicht direkt auf dem Bildschirm statt. Stattdessen geschieht die Darstellung immer über einen Browser.
- Auch die Maus- und Tastatureingabe wird durch den Browser geschleust.
- Das Datenflussdiagramm stellt die Trennung zwischen Client und Server nicht dar. Tatsächlich sind aber die folgenden Funktionen in einen Client- und einen Serverteil aufgeteilt:
  - Daten zum Server übertragen
  - Spieldaten abgleichen
  - Chat abgleichen
  - Chattext zum Server übertragen
  - An-/Abmelden
- Abgesehen von der Datenbank existieren alle Elemente als separate Instanzen pro Benutzer. Die Datenbank ist das zentrale Mittel für die Synchronisierung.

Die Struktur der Datenflüsse wird im Datenlexikon beschrieben:

2.1.1 Datenlexikon

1. Mausposition

2. HTML/CSS Daten
3. Padposition
4. Ballposition
5. Ball-/Padposition
6. Text
7. Logindaten

Mausposition ::==	x-Position
	y-Position
HTML/CSS Daten ::==	Styledaten (Positionierung)
	HTML-Daten (Chattext)
Padposition ::==	y-Position
Ballposition ::==	x-Position
	y-Position
Ball/Padposition ::==	x-Position (Ball)
	y-Position (Ball)
	y-Position (Pad)
Text ::==	Textstring
Logindaten ::==	Loginname
	Passworthash

2.2 Module

Auf dem Client wird JavaScript und auf dem Server PHP eingesetzt. Also werden alle Funktionen zur Darstellung auf dem Benutzerbildschirm und zur Umsetzung der Eingaben über Tastatur und Maus in JavaScript umgesetzt. Außerdem sorgen JavaScript Funktionen für die Übertragung der Daten zum Server und für den Empfang der von Server versendeten Informationen. Auf dem Server werden die Daten von PHP-Dateien verarbeitet, die dann auch die entsprechenden Rückgabewerte zurück zum Client senden. Die PHP Dateien für die serverseitigen asynchronen Abläufe sind mit dem Präfix ”async_” gekennzeichnet.

2.2.1 JavaScript Module (Client)

pong_chat.js

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- Chat anzeigen
- Chat abgleichen
- Chattext übertragen

pong_game.js

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- Pad positionieren
- Ball darstellen
- Pads darstellen
- Daten zentral speichern
- Spieldaten abgleichen

pong_login.js

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- An-/ Abmelden

2.2.2 PHP Module (Server)

asynch_getchat.php

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- Chat abgleichen

asynch_sendchat.php

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- Chattext übertragen

asynch_getgamedata.php

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- Daten zentral speichern
- Spieldaten abgleichen

asynch_logoff.php

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- An- / Abmelden

asynch_logon.php

Bildet diese Funktionen aus dem Datenflussdiagramm ab:

- An- / Abmelden

2.2.3 Zusatzmodule

Teilfunktionen werden aus Gründen der Übersicht und Wiederverwendbarkeit in separate Dateien ausgelagert. Diese Zusatzmodule und die Funktionen, die sie abbilden, werden hier beschrieben.

JavaScript

pong_immediatecalls.js In diesem Modul werden die Aufrufe zusammengefasst, die direkt durchgeführt werden müssen, nachdem das Dokument vom Client geladen wurde. So werden die Anzeigestatus der Schaltflächen

konfiguriert, der Event-Handler für die Mausbewegungen aktiviert und die wiederkehrenden Aufrufe für die Chatanzeige eingerichtet.

ajaxBase.js Hier werden die Funktionen definiert, die für die asynchronen Aufrufe benötigt werden. Dabei werden insbesondere die browserabhängigen Besonderheiten berücksichtigt. Dadurch kann im Hauptteil der clientseitigen Ausführung einfach auf diese Funktionen zurückgegriffen werden, ohne dass der zugrunde liegende Browser bedacht werden muss.

ajaxObjects.js Dieses Modul beinhaltet die Basisdefinitionen für alle asynchronen Aufrufe. Das heißt, dass hier die Objekte definiert werden, die für den Datenverkehr zwischen Client und Server zuständig sind. Hier finden sich also die Komponenten wieder, die die clientseitige Schnittstelle der aufgeteilten Funktionen "Daten zentral speichern", "Spieldaten abgleichen", "Chat abgleichen", "An-/Abmelden" und "Chattext übertragen" bilden. Hervorzuheben ist dabei, dass für die Aufrufe, die erwartungsgemäß häufig auftreten, Objekt-Arrays definiert werden. Dadurch wird sichergestellt, dass auch bei eventuellen Verzögerungen im Datenverkehr keine Überschneidungen geschehen, durch die noch laufende Übertragungen unterbrochen werden. Dies betrifft die Aufrufe zum Senden eines neuen Chattextes und zum Aktualisieren der Spieldaten. Gerade die Aktualisierungen müssen besonders häufig durchgeführt werden, um einen flüssigen Ablauf zu ermöglichen. Durch die Verwendung mehrerer Objekte kann bereits die nächste Aktualisierung gestartet werden, bevor die vorhergehende abgeschlossen ist.

md5.js Um das Kennwort für das Benutzerlogin auch ohne eine SSL Verbindung übertragen zu können, wird nicht das Kennwort im Klartext versendet, sondern ein Hash daraus. Dieser wird aus dem Kennwort und dem aktuellen Timestamp gebildet. Auf diese Weise wird sichergestellt, dass ein abgefangener Anmelde-Datenstrom nicht von Dritten für eine spätere Authentifizierung missbraucht werden kann. Die hierfür verwendete Hash-Funktion ist MD5. Der bekannte Algorithmus wurde nicht selbst umgesetzt. Stattdessen wurde die Implementierung von [JSMD5] verwendet.

pong_debug.js Die Funktionen und Einstellungen, die für die Fehlersuche und die Auswertung verwendet werden, sind in dieser Datei zusammenge-

fasst.

PHP

pong_db.php In dieser Datei sind die Informationen hinterlegt, die benötigt werden, um eine Verbindung mit der Datenbank aufzubauen. Außerdem wird beim Einbinden dieser Datei direkt eine Datenbankverbindung hergestellt.

pong_db_functions.php Diese Datei enthält eine Hilfsfunktion, die das Schreiben in die Datenbank vereinfachen soll.

pong_check_logon.php Wenn PHP Aufrufe nur nach einem erfolgreichen Anmelden durchgeführt werden dürfen, wird diese Datei eingebunden. Hier wird getestet, ob bereits eine Authentifizierung stattgefunden hat.

pong_functions.php Diese Datei ist als Container für allgemeine Hilfsfunktionen gedacht. Tatsächlich beinhaltet sie nur die Funktion "getMicrotime", die verwendet wird, um die aktuelle Systemzeit zu ermitteln. Dabei ist der Rückgabewert die Anzahl der Sekunden seit Beginn der UNIX-Epoche (0:00:00 January 1, 1970 GMT); die Millisekunden werden als vierstelliger Nachkommawert zurückgegeben. Zu beachten ist dabei, das die verwendete PHP-Funktion "microtime" mit der PHP4 konformen Syntax verwendet wird, und nicht der erst mit PHP5 eingeführte Parameter benutzt wird.

pong_controls.php Beinhaltet den HTML-Code zum Darstellen der Kontrollelemente für den Chat.

pong_game.php Beinhaltet den HTML-Code zum Darstellen der eigentlichen Pong Spielfläche.

pong_playerlist.php Beinhaltet den HTML-Code zum Darstellen der Liste der an den Server angemeldeten Spieler.

asynch_getchatterlist.php Diese PHP -Datei gibt eine Liste der an den Server angemeldeten Benutzer zurück.

2.3 Auswertung

Das entwickelte Programm wird während der Implementierungsphase hauptsächlich auf dem lokalen System getestet. Für die maßgeblichen Tests mit einem tatsächlichen Webserver wird ein Standardpaket von planet-hosting.de verwendet. Dieses bietet einen Webserver mit wahlweise PHP4 oder PHP5 und MySQL Datenbank. Dies entspricht also der Minimalvoraussetzung, die bei den meisten Hosting-Anbietern gefunden werden kann.

Für die Auswertung wird bei den einzelnen Aktionen der Programmmodule der jeweilige Zeitstempel im Unix-Format mit übergeben. Die Debug-Komponente gibt die einzelnen Zeitstempel der Serveranfrage als Liste aus. Diese Liste kann dann z.B. in einer Tabellenkalkulation ausgewertet werden.

Die Zeitstempel werden in diesen Momenten festgehalten:

- Beim Senden der Anfrage an den Server (Request Start)
- Beim Start des PHP Skriptes (Request Server)
- Beim Beenden des PHP Skriptes (Answer Start)
- Beim Annehmen der Serverantwort (Answer Client)

Die Zeiten "Request Start" und "Answer Client" werden also clientseitig mit JavaScript festgehalten, während "Request Server" und "Answer Start" vom PHP Skript auf dem Server aufgenommen werden.

2.3.1 lokale Messungen

Bei den lokalen Tests (d.h. Client und Server sind der gleiche Rechner) gibt es keine Diskrepanz bezüglich der Zeitsynchronität der sendenden und der empfangenden Komponente. Darum können die zurückgegeben Zeitwerte einfacher in einen Zusammenhang gebracht werden.

Die Daten werden in folgender Form festgehalten und ausgewertet:

Request Start	Request Server	Answer Start	Answer Client	Zeitdifferenz		
				Client Server	PHP	Server Client
1205092908064	1205092908090	1205092908093	1205092908100	26	3	7
1205092908212	1205092908237	1205092908239	1205092908241	25	2	2
		...				
1205092949019	1205092949052	1205092949053	1205092949064	33	1	11
1205092949167	1205092949199	1205092949200	1205092949203	32	1	3
			Durchschnitt	33,51	1,86	19,66

Der oben angegebene Durchschnitt wurde aus einer Auswertung mit über 250 Messwerten errechnet.

Für die Auswertung herangezogen werden die letzten drei Spalten, die die Differenz zwischen den jeweils aufeinanderfolgenden Zeitstempeln angeben. Dies ist in der Form nur möglich, da die vier Werte auf der selben Uhr basieren. Bei einer echten Client-Server Beziehung muss von einer Diskrepanz in den Systemzeiten ausgegangen werden.

Auffallend ist, dass generell die Zeit zwischen "Request Start" und "Request Server" länger ausfällt als die Zeit zwischen "Answer Start" und "Answer Client"; besonders da es keinen Netzwerk-Unterschied zwischen Server-Client- und Client-Server-Verbindung geben dürfte, solange die Messungen auf dem gleichen System stattfinden. Ursache für diesen Unterschied ist die PHP-Initialisierung. Da PHP als Skriptsprache ausgelegt ist, wird beim Aufrufen des Dokumentes die Datei eingelesen und der PHP Interpreter gestartet. Die Zeitspanne zwischen "Answer Start" und "Answer Client" misst also auch die Analyse des PHP-Codes mit.

Die als "PHP" Zeitdifferenz ausgezeichneten Werte in der vorletzten Spalte geben also nur die tatsächliche Ausführungszeit der Skriptanweisungen an, und nicht die von der PHP Engine durchzuführenden Vorbereitungen. Durch den Vergleich der Anfrage- und der Antwortzeiten erkennt man, dass die Initialisierung der PHP-Skripte im Durchschnitt mit ungefähr 10-20 ms zu Buche schlägt.

Aber auch trotz dieses Overheads befindet sich die Dauer von der Anfrage an den Server bis zum Erhalten der Antwort in der Regel im Bereich von ca. 50 ms. Die folgenden Tests mit einem tatsächlichen Webserver zeigen, inwiefern die

Werte dort abweichen.

2.3.2 Messungen im Internet

Bei den Tests gegen einen tatsächlichen Webserver ist zu beachten, dass nicht davon ausgegangen werden kann, dass die Zeit auf dem Client synchron mit der Zeit auf dem Server läuft. Darum kann bei dieser Auswertung keine Aussage über die genaue Dauer der einzelnen Netzwerkzeiten der Anfrage gemacht werden. Die Werte, die der Server zurückgibt, basieren auf einer unterschiedlichen Zeitbasis. Der Unterschied mag gering sein, da aber für diese Betrachtung Millisekunden relevant sind, ist es sinnlos, die Werte wie bei der lokalen Auswertung zu subtrahieren.

Stattdessen werden lediglich jeweils die Clientdaten und die Serverdaten voneinander abgezogen. Auf diese Weise erhält man Auskunft über die gesamte Laufzeit einer Anfrage und über die Dauer, die das PHP-Skript für die Abfertigung benötigt.

Die Daten werden in einer Tabelle mit der folgenden Form festgehalten:

				Zeitdifferenz		
Request Start	Request Server	Answer Start	Answer Client	Gesamt	PHP	Traffic
1205186313297	1205186306686	1205186306687	1205186313385	88	1	87
1205186313440	1205186306827	1205186306828	1205186313520	80	1	79
		...				
1205186345386	1205186338770	1205186338772	1205186345472	86	2	84
1205186345535	1205186338922	1205186338923	1205186345621	86	1	85
			Durchschnitt	89,18	2,18	87

Auch hier wurden über 250 Messwerte ausgewertet, um den angegebenen Durchschnitt zu erhalten.

Wie erwartet ist die Gesamtlaufzeit angestiegen. Leider kann aus oben genannten Gründen keine Abschätzung über die Dauer für die Initialisierung der PHP-Engine gemacht werden. Geht man von dem aus der lokalen Analyse erhaltenen Wert von 10-20 ms aus, folgt, dass die Datenübertragung über das Internet ungefähr jeweils 30-40 ms in Anspruch nimmt.

2.4 Fazit

Für diesen ersten Test wurden einfache Daten übertragen, nämlich zwei X-Koordinaten für die Pads und die Position des Balls. Da es sich dabei sowohl um von Spielern generierte Daten, als auch auf dem Server berechnete Daten handelt, umfasst dieser kleine Test bereits alle für die zweite Phase geplanten Komponenten. Lediglich der Umfang wird aufgrund der Skalierbarkeit größer sein.

Da die eigentliche Ausführung der PHP Skripte nur einen sehr geringen Anteil der gesamten Ausführungszeit beansprucht, ist zu erwarten, dass auch umfangreichere Prozeduren die Ausführungszeit nicht zu immens erhöhen. Aber auch der eigentliche Netzverkehr wird durch zusätzliche Daten nur geringfügig mehr belastet, da einige zusätzliche Informationen kaum mehr Speicher verbrauchen im Vergleich zum relativ hohen Overhead durch das HTTP Gerüst der Übertragung.

Generell sollte man bei der Betrachtung von etwa 100 ms für eine Transferphase Client-Server-Client ausgehen. Dies würde im Worst-Case eine Dauer von 200 ms für die Übertragung von Informationen von einem Client zum anderen bedeuten. Dieser Maximalwert wird aber in der Praxis kaum erreicht, da es zum einen Überschneidungen zwischen den Anfragen der beiden Clients gibt und zum anderen mehrere Anfragen parallel laufen.

Die zu erwartende Diskrepanz in der Synchronisierung muss in der Client-Software entsprechend berücksichtigt werden. Mögliche Maßnahmen wären z.B. ein Algorithmus, der durch Interpolationsverfahren basierend auf den letzten bekannten Daten den nächsten zu erwartenden Wert errechnet. Welches Mittel letztendlich verwendet wird, hängt stark von der Anwendung und der Art der Daten ab.

Zu erwähnen ist auch noch, dass JavaScript mit der eingesetzten Pong-Implementierung bereits an seine Grenzen zu stoßen scheint. Es gibt Unterschiede in der Performance der JavaScript-Engines der verschiedenen Browser. Dadurch ist das Verhalten der Clients nicht mit jedem Browser identisch. Komplexere Umsetzungen mit mehr dynamischen Elementen und einer größeren Datenmenge, die per AJAX synchronisiert werden muss, können eventuell den Browser insofern

überfordern, dass die Anwendung praktisch nicht mehr nutzbar ist.

Auf jeden Fall ist mit bestimmten Einschränkungen ein Echtzeit-Datenabgleich über PHP mit einer MySQL-Datenbank als zentralen Instanz möglich. Abhängig von der geplanten Anwendung muss aber eine entsprechende Client-Implementierung berücksichtigt werden.

Kapitel 3

Stufe 2: Compiler zur Erzeugung der Serverdateien

3.1 Anforderungen

Nachdem die generelle Machbarkeit mit erwarteten Einschränkungen in der allgemeinen Geschwindigkeit bestätigt wurde, wird nun eine Grammatik definiert, die es ermöglicht, die zu übertragenden Daten festzulegen.
Diese Grammatik wird dann benutzt, um mit dem Compiler-Generator JavaCC [JCC] einen Compiler zu erstellen.

Dieser Compiler soll die PHP-Dateien erzeugen, die benötigt werden, um

1. initial die verwendeten Datenbanktabellen zu erstellen
2. auf Anfragen mit den entsprechenden aktuellen Werten aus der Datenbank zu antworten
3. Datenänderungen entgegen zunehmen und damit die Datenbank zu aktualisieren
4. servergesteuerte Veränderungen der Daten vorzunehmen

Bei der Festlegung der Grammatik müssen also folgende Anforderungen berücksichtigt werden:

Datentypen Für jedes Datum wird ein Datentyp angegeben. Dabei beschränkt

sich die Grammatik auf die elementaren Datentypen Integer, String, Float und Boolean.

Gruppierung Die Daten werden jeweils einer Gruppe zugeordnet. Dadurch sind logische Zusammenschlüsse möglich. Es können also z.B. Daten wie "Name", "X-Position", "Y-Position" einer Gruppe "Spieler" zugeordnet werden, um damit Spielerdaten zu kennzeichnen.

Datenattribute Jedem Datum können Attribute zugeordnet werden, die Ihre Verwendung näher beschreiben:

- **prio** Mit dem "prio"-Attribut versehene Daten sind für eine Übertragung bei Standardanfragen vorgesehen. Das bedeutet, dass diese Daten bei einer Anfrage nicht konkret angefragt werden müssen, sondern immer teil der Antwort sind. Dies gilt auch für den Fall, dass der Client keine eigentliche Anfrage gesendet hat, sondern Daten für eine Aktualisierung an den Server geschickt hat. Auf diese Weise kann ein Client einen gewissen Satz relevanter Daten aktuell halten, indem er selbst nur Aktualisierungen schickt. Er muss also nicht noch zusätzliche Anfragen versenden. Das "prio"-Attribut kann von Randbedingungen abhängig gemacht werden. So ist es z.B. möglich, dass ein Client nur Aktualisierungen von User-Daten erhält, die in einer "Friendlist" gespeichert sind.

- **change** Damit wird festgelegt, ob der Client diesen Wert ändern darf, bzw. unter welchen Umständen das möglich ist.

- **update** Bestimmte Daten werden vom Server geändert. Mit dem "update"-Attribut werden Regeln für diese Datenänderungen festgelegt.

Gruppenattribute Auch den Datengruppen können Eigenschaften zugewiesen werden:

- **static** Eine als "static" markierte Gruppe wird als übergeordneter Datenspeicher verwendet. Die Zugriffe hierauf geschehen immer auf den gleichen Datensatz, unabhängig davon, woher der Zugriff geschieht. Statische Gruppen können also zur Speicherung von allgemeingültigen Informationen wie z.B. Anzahl der Mitspieler verwendet werden.

ref Ein Datensatz kann einen Bezug zu einer anderen Datensammlung haben. Dafür wird mit dem "ref" Attribut eine Datengruppe angegeben, wodurch eine Verknüpfung des aktuellen Datensatzes mit einem Datensatz aus der referenzierten Gruppe hergestellt werden kann. Es muss unterschieden werden, ob der Datensatz dieser Gruppe eindeutig im Bezug auf die Fremdgruppe ist, oder vielleicht mehrere Datensätze der Fremdgruppe zugeordnet werden können.

Es wird zwei Referenzoperatoren geben, um die Beziehung zwischen zwei Datengruppen zu verdeutlichen. Wenn in Gruppe A eine Referenz zu Gruppe B definiert wird, haben die Operatoren jeweils die folgende Bedeutung:

ref In A gibt es maximal einen Datensatz mit einem Verweis auf einen Datensatz aus B, d.h. es besteht eine 1 zu 1 Beziehung.
In der Umsetzung bedeutet dies, dass ein Tabellenfeld für einen Fremdindex vorgesehen wird und bei Aktualisierungen kein neuer Datensatz erzeugt wird, wenn der Fremdindex bereits vorhanden ist. Es wird stattdessen ein UPDATE durchgeführt.

mref In A gibt es mehrere Datensätze, die jeweils dem gleichen Datensatz aus B zugeordnet werden können.
Auch dies wird über einen Fremdindex realisiert. Allerdings werden neue Daten hinzugefügt ohne bestehende Datensätze zu überschreiben.

3.2 Die Grammatik

Anhand eines Beispieles wird verdeutlicht, wie über die Grammatik die Datenhaltung definiert werden kann. Dieses Beispiel wird danach erweitert, um dann die Basis für das in Kapitel 4 entwickelte Testprogramm zu bilden. Die formale Definition erfolgt zuerst über die Extended-Backus-Naur-Form (EBNF), um eine übersichtliche Darstellung der Grammatik zu erhalten.

Damit auf Basis dieser Grammatik mit JavaCC ein Compiler erzeugt werden kann, muss die Grammatik eine LL(k)-Grammatik sein. Dies bedeutet, dass bei der Analyse die Eingabe von links nach rechts gelesen wird und sukzessiv die linken Symbole ersetzt werden. Dabei werden bis zu k Zeichen eingelesen, um

zu bestimmen, welche Ersetzung als nächstes passt.

Um zu untersuchen, ob die Grammatik eine LL(k)-Grammatik ist, werden die first- und follow-Mengen der nicht-terminalen Symbole erstellt. Daraus ist dann ersichtlich, ob die zu verwendende Regel anhand der nächsten k Zeichen bestimmt werden kann.

Dafür wird die definierte Grammatik in der einfachen Backus-Naur-Form (BNF) aufgeschrieben.

3.2.1 Beispiel

Diese einfachste Stufe des Beispiels soll einen Chat abbilden. Die definierte Datenstruktur muss also die Benutzer und mögliche Textnachrichten berücksichtigen.

```
GROUP user {
    STRING username;
    STRING userpwd;
}

GROUP messages {
    STRING message;
    STRING sender;
    STRING receiver;
}
```

Damit ist zwar die benötigte Datenstruktur definiert, es besteht aber noch kein Zusammenhang zwischen den Gruppen. Dafür wird ein Referenzoperator verwendet, der einen Bezeichner und die zu referenzierende Gruppe bestimmt:

```
GROUP user {
    STRING username;
    STRING userpwd;
}

GROUP messages {
    mref sender - user;
```

```
    mref receiver - user;
    STRING message;
}
```

3.2.2 Extended-Backus-Naur-Form

Eine kompakte Definition der Grammatik wird in der Extended-Backus-Naur-Form (EBNF) gegeben. Um nicht alle Buchstaben und Ziffern aufzählen zu müssen, werden drei Punkte benutzt, um die Liste anzudeuten.

Optionen werden durch einen senkrechten Strich (”|”)(ganze Ausdrücke) bzw. eckige Klammern (”[” , ”]”)(Teilausdrücke) und optionale Wiederholungen durch die Verwendung von eckigen Klammern mit einem Sternchen (”[”, ”]*”) dargestellt.

Damit die Übertragung in die Syntax von JavaCC einfacher ist, werden auch für die reservierten Wörter, die Identifier und die Literale nicht-terminale Symbole eingeführt.

Für die eigentliche Definition der Datenmengen und -zuordnungen genügen eigentlich die Buchstaben, Zahlen und einige Sonderzeichen wie Klammern, Punkt und Semikolon. Allerdings müssen in die Beschreibungen auch PHP-Skripte eingebettet werden können, um Datenaktualisierungen und Bedingungen zu beschreiben. Darum sind zumindest in diesen PHP Bereichen sämtliche UTF-8 Zeichen möglich. Um diese Bereiche eindeutig von den restlichen Abschnitten abzugrenzen, werden sie von CDATA-Token (**c**haracter **data**) eingeklammert. Diese Token sind ein geläufiges Mittel, um Parsern mitzuteilen, das bestimmte Texte von der Übersetzung ausgeschlossen werden sollen. Der CDATA-Block beginnt mit den Zeichen
<![CDATA[
und wird mit
]]>
abgeschlossen.
Da die meisten dieser Zeichen als Metazeichen in der BNF-Darstellung verwendet werden, werden die CDATA-Token in der Definition mit CDATAstart und CDATAend angegeben. Auch von einer Definition des Inhalts des CDATA-

Blocks wird abgesehen, da dort jedes beliebige Zeichen (außer der Zeichenfolge "]]>") vorkommen kann. Dieser Teil wird mit CDATAcontent angegeben.

Dadurch besteht die Menge der terminalen Symbole aus allen Buchstaben, den Zahlen und Sonderzeichen wie geschwungenen Klammern oder Gleichheitszeichen. Diese Sonderzeichen, die ja alle innerhalb eines CDATA-Blocks vorkommen können, werden in der Definition nicht explizit aufgezählt. Ebenso werden nicht alle Buchstaben und Ziffern aufgelistet, obwohl diese alle als terminale Symbole verwendet werden. Die Liste wird jeweils mit drei Punkten abgekürzt, statt jedes Zeichen einzeln darzustellen.

Zur Kennzeichnung wird den Symbolen der reservierten Wörtern ein "Res" angehängt. Die reservierten Wörter werden später in JavaCC mit der "TOKEN" Anweisung übernommen. Alle nicht-terminalen Symbole werden mit spitzen Klammern ("<" , ">") gekennzeichnet.

Das Startsymbol ist "<Definition>".

<Definition> ::= <Optionblock> <Groupdefinitions>
<Optionblock> ::= <OptionRes> { <OptionList> }
<OptionList> ::= [<OptionStatement>;]*
<OptionStatement> ::= <Identifier> = "<OptionValue>"
<OptionValue> ::= [<Char> | <Digit> | _]*
<Groupdefinitions> ::= [<GroupBlock>]*
<GroupBlock> ::= <GroupRes> <Identifier>
{ <GroupAttributeList> <GroupDataList> }
<GroupAttributeList> ::= [<StaticRes>;] [<GroupAttribute>]*
<GroupAttribute> ::= <RefRes> <Identifier> - <Identifier> <DataAttributeList>; |
<MrefRes> <Identifier> - <Identifier> <DataAttributeList>; |
<GroupDataList> ::= [<DatatypeRes> <Identifier> <DataAttributeList>;]*
<DataAttributeList> ::= [<DataAttributePrio>]
[<DataAttributeCng>]
[<DataAttributeUpd>]
<DataAttributePrio> ::= <prioRes> { CDATAstart CDATAcontent CDATAend }
<DataAttributeCng> ::= <changeRes> { CDATAstart CDATAcontent CDATAend }
<DataAttributeUpd> ::= <updateRes> { CDATAstart CDATAcontent CDATAend }
<Identifier> ::= <Char> [<IdentifierRest>]*
<IdentifierRest> ::= <Char> | <Digit> | _
<OptionRes> ::= OPTIONS
<GroupRes> ::= GROUP
<StaticRes> ::= static
<RefRes> ::= ref
<MrefRes> ::= mref
<prioRes> ::= prio
<changeRes> ::= change
<updateRes> ::= update
<DatatypeRes> ::= INT
<DatatypeRes> ::= FLOAT
<DatatypeRes> ::= STRING
<DatatypeRes> ::= BOOL
<Char> ::= A | ... | Z | a | ... | z
<Digit> ::= 0 | ... | 9

3.2.3 Backus-Naur-Form

Um die first- und follow-Mengen ermitteln zu können, wird die Grammatik nun in der einfachen Backus-Naur-Form (BNF) dargestellt.

Gegenüber der EBNF haben sich dabei weder die Menge der terminalen Symbole noch das Startsymbol geändert. Lediglich Optionen und Wiederholungen mussten für die BNF-Darstellung angepasst werden, wofür auch das nicht-terminale Symbol <GroupAttributeListRest> ergänzt wurde.

Um auch hier die Darstellung nicht unnötig aufzublähen, wurden Definitionen zum Teil vereinfacht, indem Listen nicht komplett aufgeschrieben werden. Stattdessen werden das erste und das letzte Element und dazwischen drei Punkte ("...") dargestellt.

(1) <Definition> ::= <Optionblock> <Groupdefinitions>

(2) <Optionblock> ::= <OptionRes> { <OptionList> }

(3) <OptionList> ::= <OptionStatement>;<OptionList>

(4) <OptionList> ::= ε

(5) <OptionStatement> ::= <Identifier> = "<OptionValue>"

(6) <OptionValue> ::= <Char><OptionValue>

(7) <OptionValue> ::= <Digit><OptionValue>

(8) <OptionValue> ::= _<OptionValue>

(9) <OptionValue> ::= ε

(10) <Groupdefinitions> ::= <GroupBlock> <Groupdefinitions>

(11) <Groupdefinitions> ::= ε

(12) <GroupBlock> ::= <GroupRes> <Identifier>
{ <GroupAttributeList> <GroupDataList> }

(13) <GroupAttributeList> ::= <StaticRes>;
<GroupAttributeListRest>

(14) <GroupAttributeList> ::= <GroupAttributeListRest>

(15) <GroupAttributeListRest> ::= <GroupAttribute> ;
<GroupAttributeListRest>

(16) <GroupAttributeListRest> ::= ε

(17) <GroupAttribute> ::= <RefRes> <Identifier> - <Identifier>
<DataAttributeList>

(18) <GroupAttribute> ::= <MrefRes> <Identifier> - <Identifier>
<DataAttributeList>

(19) <GroupDataList> ::= <DatatypeRes> <Identifier>
<DataAttributeList>; <GroupDataList>

(20) <GroupDataList> ::= ε

(21) <DataAttributeList> ::= <DataAttributePrio>
<DataAttributeCng> <DataAttributeUpd>

(22) <DataAttributePrio> ::= <prioRes>
{ CDATAstart CDATAcontent CDATAend }

(23) <DataAttributePrio> ::= ε

(24) <DataAttributeCng> ::= <changeRes>
{ CDATAstart CDATAcontent CDATAend }

(25) <DataAttributeCng> ::= ε

(26) <DataAttributeUpd> ::= <updateRes>
{ CDATAstart CDATAcontent CDATAend }
(27) <DataAttributeUpd> ::= ε
(28) <Identifier> ::= <Char><IdentifierRest>
(29) <IdentifierRest> ::= <Char><IdentifierRest>
(30) <IdentifierRest> ::= <Digit><IdentifierRest>
(31) <IdentifierRest> ::= _<IdentifierRest>
(32) <IdentifierRest> ::= ε

(33) <OptionRes> ::= OPTIONS
(34) <GroupRes> ::= GROUP
(35) <StaticRes> ::= static
(36) <RefRes> ::= ref
(37) <MrefRes> ::= mref
(38) <prioRes> ::= prio
(39) <changeRes> ::= change
(40) <updateRes> ::= update
(41) <DatatypeRes> ::= INT
(42) <DatatypeRes> ::= FLOAT
(43) <DatatypeRes> ::= STRING
(44) <DatatypeRes> ::= BOOL
(45) <Char> ::= A
(46) <Char> ::= ...
(47) <Char> ::= Z
(48) <Char> ::= a
(49) <Char> ::= ...
(50) <Char> ::= z
(51) <Digit> ::= 0
(52) <Digit> ::= ...
(53) <Digit> ::= 9

Alle nicht-terminalen Symbole dieser Grammatik lassen sich auf eine terminale Kette abbilden. Auch ist kein überflüssiges nicht-terminales Symbol definiert. Die Symbole der Grammatik sind also terminalisierbar und ableitbar. Außerdem

wurde bei den rekursiven Definitionen zusammen mit der leeren Zeichenfolge ε darauf geachtet, dass keine zirkulären Ableitungen möglich sind. Es liegt also eine zyklenfreie Grammatik vor.

3.2.4 first- und follow-Mengen

JavaCC ist nur in der Lage, eine LL(k)-Grammatik auszuwerten. Um dies sicherzustellen, müssen die Produktionsregeln der nicht-terminalen Symbole betrachtet werden. Diese Regeln müssen so gestaltet sein, dass bei der lexikalischen Analyse nur k Symbole eingelesen werden müssen, um zu bestimmen, welche Produktionsregel angewandt werden muss.

Laut [Aho08] sind die folgenden drei Bedingungen hinreichend, um eine LL(1)-Grammatik zu erhalten:

1. Für ein nicht-terminales Symbol existieren keine zwei Produktionsregeln, die beide mit dem gleichen terminalen Symbol beginnen.
2. Für ein nicht-terminales Symbol existieren keine zwei Produktionsregeln, die beide zum leeren String (ε) abgeleitet werden können.
3. Wenn eine der Produktionsregeln für ein nicht-terminales Symbol zum leeren String abgeleitet werden kann, so existiert keine Produktionsregel für dieses nicht-terminale Symbol, das mit dem gleichen terminalen String beginnt, wie ein nicht-terminales Symbol, das möglicherweise folgen kann.

Durch die dritte Regel soll sichergestellt werden, dass beim Abarbeiten einer Produktionsregel die danach folgende Regel eindeutig bestimmt werden kann.

Diese Bedingungen lassen sich einfach verifizieren, indem man die first- und follow-Mengen der einzelnen Produktionsregeln bestimmt. Die first-Menge gibt jeweils die terminalen Symbole an, mit denen eine Produktion starten kann, während die follow-Menge beschreibt, welche terminalen Symbole auf eine Produktion folgen können.

Grundlage hierfür sind die Notationen der Produktionsregeln in der BNF, denn hier hat man für jedes nicht-terminale Symbol die möglichen Produktionsregeln in einer separaten Zeile, für die dann jeweils die first- und die follow-Menge ermittelt wird.

Die folgende Tabelle zeigt die first- und follow-Mengen der in Abschnitt 3.2.3 angegebenen Produktionsregeln. Dabei wird der leere String mit ε und die Endmarkierung mit $ angegeben.

Produktion	first-Menge	follow-Menge
(1)<Definition>	OPTION	$
(2)<Optionblock>	OPTION	GROUP $
(3)<OptionList>	A...z a...z	}
(4)<OptionList>	ε	}
(5)<OptionStatement>	A...z a...z	;
(6)<OptionValue>	0...9	”
(7)<OptionValue>	A...z a...z	”
(8)<OptionValue>	_	”
(9)<OptionValue>	ε	”
(10)<Groupdefinitions>	GROUP	$
(11)<Groupdefinitions>	ε	$
(12)<GroupBlock>	GROUP	GROUP $
(13)<GroupAttributeList>	static	} INT FLOAT STRING BOOL
(14)<GroupAttributeList>	ref mref ε	} INT FLOAT STRING BOOL
(15)<GroupAttributeListRest>	ref mref	} INT FLOAT STRING BOOL
(16)<GroupAttributeListRest>	ε	} INT FLOAT STRING BOOL
(17)<GroupAttribute>	ref	;
(18)<GroupAttribute>	mref	;
(19)<GroupDataList>	INT FLOAT STRING BOOL	}
(20)<GroupDataList>	ε	}
(21)<DataAttributeList>	prio change update ε	;
(22)<DataAttributePrio>	prio	change update ;
(23)<DataAttributePrio>	ε	change update ;
(24)<DataAttributeCng>	change	update ;

(25)<DataAttributeCng>	ε	update ;
(26)<DataAttributeUpd>	update	;
(27)<DataAttributeUpd>	ε	;
(28)<Identifier>	A...z a...z	= { ; - prio change update
(29)<IdentifierRest>	A...z a...z	= { ; - prio change update
(30)<IdentifierRest>	0...9	= { ; - prio change update
(31)<IdentifierRest>	_	= { ; - prio change update
(32)<IdentifierRest>	ε	= { ; - prio change update
(33)<OptionRes>	OPTION	{
(34)<GroupRes>	GROUP	A...Z a...z
(35)<StaticRes>	static	;
(36)<RefRes>	ref	A...Z a...z
(37)<MrefRes>	mref	A...Z a...z
(38)<prioRes>	prio	{
(39)<changeRes>	change	{
(40)<updateRes>	update	{
(41)<DatatypeRes>	INT	A...Z a...z
(42)<DatatypeRes>	FLOAT	A...Z a...z
(43)<DatatypeRes>	STRING	A...Z a...z
(44)<DatatypeRes>	BOOL	A...Z a...z
(45)<Char>	A	= { ; - 0...9 a...z A...Z
(46)<Char>	..	= { ; - 0...9 a...z A...Z
(47)<Char>	Z	= { ; - 0...9 a...z A...Z
(48)<Char>	a	= { ; - 0...9 a...z A...Z
(49)<Char>	..	= { ; - 0...9 a...z A...Z
(50)<Char>	z	= { ; - 0...9 a...z A...Z
(51)<Digit>	0	= { ; - 0...9 a...z A...Z
(52)<Digit>	...	= { ; - 0...9 a...z A...Z
(53)<Digit>	9	= { ; - 0...9 a...z A...Z

Die oben beschriebenen Regeln aus [Aho08] haben nun bezogen auf die first- und follow-Mengen eines nicht-terminalen Symbols A diese Bedeutung:

1. Die first-Mengen der Produktionsregeln für A sind paarweise disjunkt.

2. Von den first-Mengen einer Produktionsregel für A enthält nur höchstens eine den leeren String (ε).

3. Wenn eine der Produktionsregeln für A zum leeren String abgeleitet werden kann, so sind bei allen anderen Produktionsregeln für A die first- und die follow-Menge disjunkt.

In der Tabelle kann man nun leicht sehen, dass diese drei Bedingungen erfüllt sind. Bei der hier entwickelten Grammatik handelt es sich also um eine LL(1)-Grammatik. Sie kann mit JavaCC umgesetzt werden.

3.3 Lexikalische Analyse und Parser

Der erste Schritt zur Erzeugung des Parsers ist die Übertragung der Grammatikdefinition in die Syntax von JavaCC. Da die Syntax aber bewusst an die EBNF angelehnt ist, können die Produktionsregeln aus dem Abschnitt 3.2.2 fast zeilenweise übernommen werden.

Die Komponente von JavaCC, die für die lexikalische Analyse zuständig ist, wird als "Token-Manager" bezeichnet. Ursprung des Begriffes ist die Tatsache, dass alle terminalen Symbole als sogenannte "Tokens" definiert werden.

Beispiel:
Um die Regeln für die Optionsparameter (Produktionsregeln 2-9) in JavaCC umzusetzen, müssen zuerst die Token für die terminalen Symbole <Char> (Regeln 45-50) und <Digit> (Regeln 51-53) definiert werden:

```
TOKEN :
{
    < DIGIT: ["0" - "9"] >
| < CHAR: ["a" - "z"] | [ "A" - "Z"] >
}
```

Der senkrechte Strich trennt dabei die verschiedenen Regeln. Es ist aber auch genauso möglich, jede Regel in einen eigenen "TOKEN"-Block zu schreiben. Da JavaCC auch reguläre Ausdrücke auswertet, kann die Definition dieser beiden Token vereinfacht werden, indem ganze Zeichensatz-Bereiche angegeben werden. Die eckigen Klammern ("[" und "]") stehen dabei für für Zeichenklassen, die in

diesem Fall als die Ziffern von 0 bis 9 und die Buchstaben von ”a” bis ”z” bzw. von ”A” bis ”Z” angegeben werden. Innerhalb des regulären Ausdrucks steht der senkrechte Strich (”|”) für ein logisches ”oder”, stellt also eine Option dar.

Nachdem die Token für die terminalen Symbole festgelegt wurden, muss dem Parser mitgeteilt werden, wie diese in den nicht-terminalen Symbolen eingesetzt werden. Dies geschieht in einer Notation ähnlich den Prozeduren eines Java-Programmes. Dabei können analog zur EBNF-Notation Optionen und Wiederholungen beschrieben werden. Diese Produktionsregeln werden später erweitert, um die Semantik für die Erstellung der PHP-Dateien zu ergänzen.

Beispiel:
Die Produktionsregeln für das nicht-terminale Symbol <Optionblock> und die darin verwendeten Symbole werden dann so in JavaCC umgesetzt:

```
void Optionblock() : {}
{
    <tOPTIONS> "{" OptionList() "}"
}

void OptionList() : {}
{
    (  OptionStatement() ";" )*
}

void OptionStatement() : {}
{
    Identifier() "="  "\"" OptionValue()  "\""
}

void Identifier() : {}
{
    <CHAR>( <CHAR> | <DIGIT> | "_" )*
}

void OptionValue() : {}
```

```
{
    ( <CHAR> | <DIGIT> | "_"  )*
}
```

In doppelten Anführungszeichen stehen Zeichen, die so in der Grammatik an den jeweiligen Positionen vorkommen müssen. Zwar hätten auch dafür Token definiert werden können, aber der Einfachheit halber wurde von zusätzlichen Token-Definitionen abgesehen. Dadurch soll die Definition im Gesamten möglichst schlank gehalten werden.

Es lassen sich zudem mehrere Zustände definieren, in die der Parser sich beim Durchlaufen der Eingabe begibt. Durch diese Zustände ist es möglich, das gleiche Eingabe-Token unterschiedlich zu interpretieren. Jeder Zustand kann mit den gleichen Token unterschiedlich umgehen. Der Standardzustand heißt "DEFAULT". Wenn nichts anderes angegeben ist, werden die Token-Definitionen für diesen Zustand angenommen. Wenn ein Token für einen anderen Zustand definiert werden soll, so muss dies explizit angegeben werden.
Benutzt wird die Technik für die CDATA Blöcke, in denen beliebiger PHP-Code zugelassen sein soll. Also sind hier theoretisch alle beliebigen Zeichen möglich. Damit dies nicht mit den bereits definierten Token kollidiert, wird der Zustand CDATA_CONTENT_STATE eingeführt. Dieser Zustand startet mit dem CDATA-Starttoken ("<![CDATA[") und endet mit dem CDATA-Endtoken ("]]>"). Solange sich der Parser im Zustand CDATA_CONTENT_STATE befindet, sind alle beliebigen Zeichen erlaubt und werden vom Token-Manager zugelassen. Um Token einem bestimmten Zustand zuzuordnen, wird der Name des Zustandes in spitzen Klammern vor die Token-Definition geschrieben.
Dadurch erlaubt die folgende Definition jedes beliebige Zeichen, solange sich der Parser im Zustand CDATA_CONTENT_STATE befindet:

```
< CDATA_CONTENT_STATE >
TOKEN:
{
    < CDATAcontent : ~[] >
}
```

Ein weiterer Zustand (COMMENT_STATE) wird benutzt, um Kommentare zuzulassen. Die hierfür definierten Token zum Beginnen ("/*") und Beenden ("

*/") eines Kommentares wurden gegenüber der EBNF-Definition aus Abschnitt 3.2.2 hinzugefügt, da sie eigentlich nicht zur Grammatik an sich gehören. Für die Benutzereingabe sind Kommentare aber sinnvoll und praktisch.
Die Kommentarblöcke werden mit diesen drei Anweisungen definiert:

```
SKIP:
{
    "/*"  : COMMENT_STATE
}

< COMMENT_STATE >
SKIP:
{
    < ~[] >
}

< COMMENT_STATE >
SKIP:
{
    "*/" : DEFAULT
}
```

Die komplette Definition der Grammatik in JavaCC-Syntax steht in Anhang A-1.

3.4 Generierung der PHP-Dateien

3.4.1 Dateienübersicht

Die JavaCC Anweisungen müssen nun so erweitert werden, dass Eingaben nicht nur bezüglich der Grammatik bestätigt werden, sondern auch die benötigten PHP-Dateien erzeugt werden. Bei der Bestimmung der Dateien werden die Erfahrungen aus Kapitel 2 genutzt.
Prinzipiell werden Dateien für diese Zwecke benötigt:

- Login

- Logout
- Daten senden
- Daten abfragen

Um eine übersichtlichere Struktur in den erzeugten PHP-Skripten zu haben, werden die individuellen Konfigurationen in separate Dateien geschrieben, die dann in die Hauptskripte eingebettet werden. Außerdem gibt es zwei generelle Skripte: init.php und checklogin.php.
init.php enthält die generellen Informationen, die von den anderen Skripten benötigt werden, wie die Datenbankkonfiguration und Subfunktionen z.B. für die Generierung der JSON-Ausgabe. checklogin.php wird von allen Skripten eingebunden, um zu überprüfen, ob der Benutzer angemeldet ist, wenn die Anwendung dies erfordert.
Um einmalig in der Datenbank die benötigte Struktur anzulegen, wird das Skript initdbtables.php erzeugt. Dies kann über einen Webbrowser auf dem Server aufgerufen werden und schreibt dann die vorgesehenen Tabellen in die Datenbank.

Insgesamt erzeugt der Parser also die folgenden Dateien:

initdbtables.php Generierung der Datenbanktabellen

init.php.inc Hauptkonfiguration

login.php Benutzeranmeldung

logout.php Benutzerabmeldung

getdata.php Abfragen von Daten

setdata.php Ändern von Daten

getprio.php.inc Beziehen der prio-Daten

setprioinfo.php.inc Konfiguration der prio-Daten

update.php.inc Aktualisierung der servergesteuerten Daten

setupdateinfo.php.inc Konfiguration der servergesteuerten Daten

Die Skripte, die von anderen eingebunden werden, sind mit der Endung .inc gekennzeichnet.

3.4.2 Bedeutung der Grammatik

Eingaben gemäß der in Abschnitt 3.2.2 definierten Grammatik werden eingelesen und daraus die benötigten PHP-Skripte erzeugt.

Dabei entsprechen die definierten Gruppen den Datenbanktabellen, in denen die übertragenen Daten und die zusätzlichen Informationen gehalten werden. Die Gruppen ”user” und ”userdata” spielen eine besondere Rolle.
Da die meisten Anwendungen voraussichtlich einen angemeldeten Benutzer erwarten, stellen die beiden Gruppen ”user” und ”userdata” automatisch Mechanismen zur Benutzerverwaltung zur Verfügung. Wenn in der Eingabedatei eine Gruppe ”user” definiert ist erhält diese immer auch die Felder ”username” für einen Benutzernamen, ”userpwd” für ein Passwort, ”online” zur Markierung des Anmeldestatus und ”tstamp” als Timestamp um Aktualisierungen nachzuvollziehen. Die Gruppe ”userdata” wird automatisch mit der Gruppe ”user” verknüpft. Sie ist vorgesehen für alle zusätzlichen den Benutzer betreffenden Informationen; insbesondere solche, die für eine häufige Aktualisierung oder Abfrage vorgesehen sind.

Abgesehen davon erhält jede Gruppe mit einer update-Anweisung das Feld ”updatetstamp”. Hier wird der Zeitpunkt der letzten Aktualisierung durch den Server festgehalten.

Aktualisierungen an den Daten werden prinzipiell in Form von ”INSERT”-SQL-Anweisungen durchgeführt. Wenn allerdings ein Datenfeld in der Definition mit dem ”ref”-Attribut versehen ist, so wird bei Aktualisierungen der ”UPDATE”-Befehl verwendet. In diesem Fall und bei Verwendung des ”mref”-Attributs wird in der Datenbank ein Feld für einen Fremdindex angelegt. Wenn innerhalb einer Gruppe sowohl das ”ref”- als auch das ”mref”-Attribut vergeben wird, ist die anzuwendende Methode nicht eindeutig. Der Parser konfiguriert diese Gruppe dann so, dass Änderungen als ”INSERT”-Anweisung durchgeführt werden.

Die CDATA Bereiche bei den prio-, change- und update-Parametern in der Definitionsdatei haben unterschiedliche Bedeutungen.

Bei prio-Anweisungen enthalten sie eine where-Bedingung für die SQL-Anweisung, mit der die prio-Daten bezogen werden. Damit kann die Anzahl der Datensätze, die für die prio-Antwort aus der Datenbank bezogen und an den Client

übertragen werden, eingeschränkt werden. Die Felder aus der Gruppe ”userdata” und können hier mit vorangestelltem ”$user_” als Platzhalter eingesetzt werden. Auch die Benutzer-ID steht als ”$user_id” zur Verfügung. Die where-Bedingungen aller prio-Felder einer Gruppe werden mit einem logischen ”AND” verknüpft und als Bedingung für die prio-Abfrage benutzt.

Beispiel:
In der Gruppe ”userdata” ist das Feld ”farbe” definiert. Nun wird in einem der prio-Felder die Bedingung
$user_farbe=”rot”
konfguriert.
Bei jeder Serveranfrage werden dann die prio-Daten aller Datensätze der Gruppe übermittelt, bei denen das Feld ”farbe” auf ”rot” gesetzt ist.

Die change- und update-Anweisungen enthalten in den CDATA-Blöcken PHP-Befehle. Diese Skripte werden dabei zuerst für change und dann für update in der Reihenfolge aufgerufen, in der sie in der Definitionsdatei beschrieben werden.

Für update wird damit festgelegt, welche Daten vom Server verändert werden. All Felder der Gruppe werden als PHP-Variablen (also in der Form ”$” + Feldname) zur Verfügung gestellt und können von den prio-Skripten geändert werden. Wie bei prio-Anweisungen können auch hier die Daten aus der Gruppe ”userdata” als PHP-Variablen mit vorangestelltem ”user_” benutzt werden.

In den change-Blöcken wird über die PHP-Befehle die Benutzer-Eingabe validiert. Sobald PHP-Befehle im CDATA-Teil angegeben sind wird ein Änderung des Wertes nur akzeptiert, wenn die PHP-Variable ”$isValid” auf TRUE gesetzt wird. Dafür können die Felder der Gruppe als PHP-Variablen (also in der Form ”$” + Feldname) gelesen und geändert werden. Zusätzlich stehen die bisherigen Werte mit dem Prefix ”old_” zur Verfügung. Wenn in dieser Gruppe ein ”mref”-Attribut vorkommt, dann wird die Änderung nur durchgeführt, wenn alle angegebenen PHP-Skripte die PHP-Variable ”$isValid” auf TRUE gesetzt haben.

Die Optionen werden direkt in die erzeugten PHP-Skripte übernommen und dort ausgewertet. Die folgende Tabelle gibt einen Überblick über die implementierten Einstellungsmöglichkeiten.

Parameter	**Bedeutung**	**Standardwert**
DBSERVER	MySQL-Servername oder IP	localhost
DBUSER	Name des Datenbankbenutzers	root
DBPWD	Datenbankpasswort	root
DBNAME	Datenbankname	playhp
TABLEPREFIX	Prefix für die Tabellennamen	chat_
USELOGIN	0: freier Zugriff 1: Benutzer muss angemeldet sein	1
LOGINMD5	0: Anmeldung über Name und Passwort 1: Verwendung des MD5 Hashes aus Passwort und Timestamp	1
MAXUSERS	gleichzeitig zugelassene Benutzer	2
PRIOWHEREFORUPDATE	0 : nur die Einschränkung aus der update-Definition wird verwendet 1: zusätzlich gelten die prio-Einschränkungen der Gruppe	0
HTTPREQUEST	Methode der HTTP-Datenübertragung G : GET P : POST GP : GET und POST POST-Werte überschreiben ggf. die GET-Werte PG : POST und GET GET-Werte überschreiben ggf. die POST-Werte	GP
UPDATEMILLISEC	minimales Intervall der Datenänderung durch den Server (update) in ms	100
SHOWDEBUGINFO	0: Standardausgabe 1: zusätzlich Ausgabe der PHP-Ausführungszeit	0

In der Datei init.php.inc werden die Optionen als Konstanten definiert und mit dem Präfix ”const” gekennzeichnet.

3.4.3 Umsetzung in JavaCC

In JavaCC können zwischen die Lexer-Anweisungen Java-Kommandos gesetzt werden, durch die die Ausgabe des erzeugten Compilers generiert wird. Der Compiler zur Erzeugung der PHP-Dateien geht dabei folgendermaßen vor:

Beim Durchlaufen der Eingabedatei werden die eingelesenen Informationen (z.B. Bezeichner und Token) in globalen Variablen gespeichert. Dafür werden an Schlüsselstellen in den Compileranweisungen Java-Befehle eingefügt, die aus den bisher bezogenen Daten Ausgabestrings oder temporäre Werte erzeugen. Diese globalen Variablen dienen als Basis für die variablen Daten der zu erzeugenden PHP-Skripte.

Neben Strings und Listen (Klasse Vector in Java) wird auch die Klasse Hashtable zur Datenhaltung verwendet. Eine Hashtable speichert Schlüssel/Wert-Paare und ist deswegen gut für schnelle, gezielte Zugriffe geeignet.

Für die letztendlich zu erstellenden Skripte liegen Vorgabedateien (Templates) vor, in denen Platzhalter die variablen Stellen kennzeichnen. Hier werden Skriptzeilen in Abhängigkeit von der Eingabedatei eingetragen. Nachdem die Eingabe komplett eingelesen wurde, werden die gesammelten Daten benutzt, um die vorbereiteten Templates zu füllen.

Das Ersetzen der Platzhalter und das Schreiben der erzeugten Dateien wurde in der separaten Klasse TemplateEvaluator implementiert. Jede Instanz der Klasse erhält den Namen der Template-Datei, die eingelesen werden soll, den Namen der Zieldatei und die zu dem Template passenden Daten. Daraus erzeugt das Object das Ziel-Skript.

Der komplette JavaCC-Quelltext steht in Anhang A-2, die vom generierten Compiler verwendete Hilfs-Klasse TemplateEvaluator im Anhang A-3.

Kapitel 4

Stufe 3: Testprogramm

4.1 Datenmodel

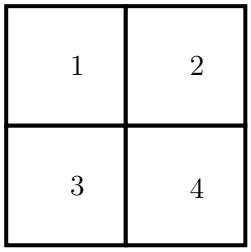
In diesem Kapitel wird anhand eines konkreten Beispiels gezeigt, wie der entwickelte Compiler benutzt werden kann, um die Serverdateien für eine Multi-Client-Datenübertragung zu erstellen. Das in Abschnitt 3.2 eingeführte Beispiel wird dafür aufgegriffen und erweitert.

Der angedachte Chat wird nun um "Avatare" erweitert: Der Benutzer sieht eine Figur, die ihn im Rahmen des Chats repräsentiert. Damit kann er zwischen den Chat-Räumen wechseln. Nachrichten, die er eingibt sind für alle Benutzer sichtbar, die ihren Avatar im gleichen Raum positioniert haben.

```
GROUP userdata {
    INT room;
    INT xpos;
    INT ypos;
}
```

Für den Test wird es vier der Chat-Räume geben. Sie erfüllen keine Aufgabe außer der Verteilung der Benutzer.

Alle Felder der Gruppe userdata müssen mit dem change-Attribut versehen werden, damit der Benutzer die Werte auch ändern kann. Um zu verhindern,



Anordnung der vier Chat-Räume

dass beliebig zwischen den Räumen gesprungen werden kann, wird eine einfache Plausibilitätsprüfung durchgeführt:

```
change {
    <![CDATA[
        $roomDif = abs( $user_room - $room );
        if ( $roomDif == 1 || $roomDif == 2  || $roomDif == 0  ) {
            $isValid = TRUE;
        }
    ]]>
}
```

Diese Prüfung umfasst zwar nicht alle Fälle, reicht aber aus, um das Prinzip der Eingabevalidierung zu demonstrieren. Bei den lediglich vier Räumen könnte man auch alle Fälle separat überprüfen.
Für die userdata-Gruppe wird also eine Datenbanktabelle angelegt, deren Inhalt vom Benutzer geändert werden kann. Die PHP-Skripte verwenden dafür eine UPDATE-SQL-Anweisung, denn die Datensätze aus der userdata-Gruppe sind eindeutig einem Datensatz der user-Gruppe zugeordnet. Die bestehenden Einträge werden also verändert statt neue anzulegen.
Die Position der Avatare in einem Raum sollte häufig an den Client übertragen werden, damit Änderungen schnell dargestellt werden können. Darum werden die Felder xpos und ypos als prio-Felder gekennzeichnet. Da nur die Avatare relevant sind, die im gleichen Raum wie der abrufende Benutzer sind, wird die prio-Abfrage entsprechend eingeschränkt:

```
prio {
        <![CDATA[
                room="$user_room"
```

```
        ]]>
}
```

Da alle prio-Anweisungen einer Gruppe mit AND verknüpft werden, genügt es diese Einschränkung nur entweder bei xpos oder bei ypos einzutragen.

Für die Chat-Mitteilungen wird die Gruppe messages angelegt. Hier ist eine Aktualisierung der Daten nicht sinnvoll. Neue Nachrichten sollen angefügt werden. Dies wird durch die mref-Verknüpfung erreicht.

```
mref sender - user
        change {
                <![CDATA[
                        if ($sender == $user_id)
                                $isValid = TRUE;
                ]]>
        };
```

Zusätzlich werden die Gruppen userlooks und room gebildet, die statische Informationen halten. userlooks speichert die Komponenten für das Aussehen eines Avatars und rooms speichert die Verbindungen zwischen den Räumen.

Mit dieser Konstellation werden fast alle von der Grammatik ermöglichten Aspekte berücksichtigt:

Datenbankaktualisierung mit INSERT und UPDATE, automatische Übertragung von Daten (Position der anderen Avatare), Einschränkung der SQL Anweisung (nur Avatare im gleichen Raum werden übertragen) und Überprüfung der Daten, die vom Benutzer kommen (es kann nur zu angrenzenden Räumen gewechselt werden).

Damit außerdem auch servergenerierte Daten eingesetzt werden, wird ein Geist erzeugt, der durch die Räume schwebt. Die Gruppe ghost speichert dafür ähnlich wie userdata für die Avatare die X- und Y-Position und den aktuellen Raum. Allerdings werden diese Daten nicht vom Client aktualisiert, sondern vom Server.
In der update-Anweisung ist in PHP-Notation der Algorithmus beschrieben, der den Geist durch die Räume bewegt. Dabei bewegt sich der Geist linear in eine

Richtung. Zufällig wird bestimmt, ob er die Richtung wechselt, ohne gegen einen äußeren Rand zu stoßen.

Der gesamte Anweisungsblock für die Definition des Datenmodels ist im Anhang A-4 aufgeführt.

Nachdem diese Datendefinition dem Compiler übergeben wurde, generiert dieser die beschriebenen PHP-Dateien. Diese werden auf den Server übertragen und mit dem initdbtables-Skript die benötigten Datenbanktabellen erzeugt.
Die gewünschten Benutzer müssen per Hand in der Datenbank angelegt werden. Danach steht der Server bereit und kann die Daten für die Clients vorhalten.

4.2 Java Client

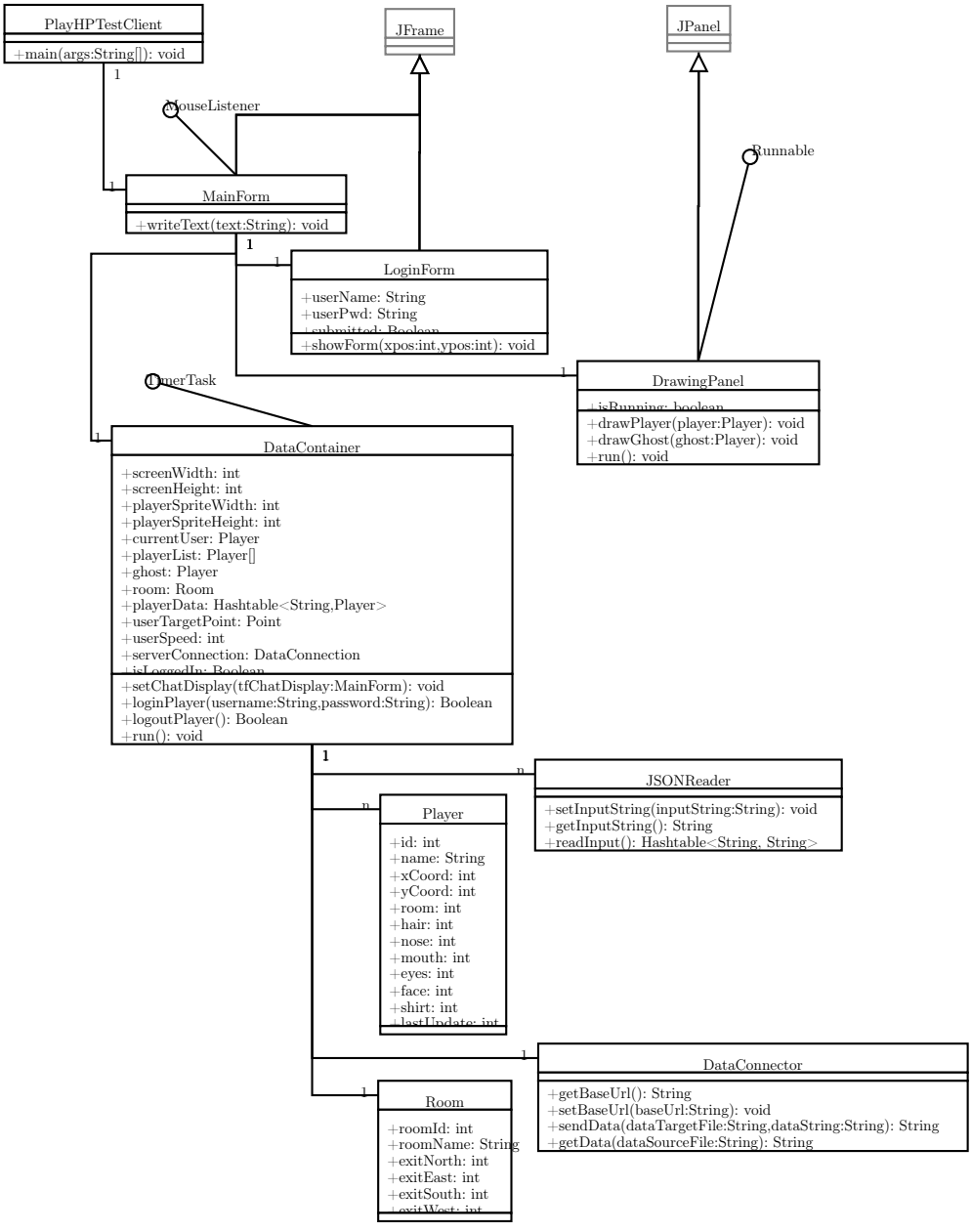
Der Client ist für die Darstellung der erhaltenen Daten zuständig. Wie die Daten letztendlich verarbeitet werden ist unabhängig von der Serverimplementierung. Für das Testprogramm wird Java verwendet. Dies ist zwar durch die Vorkompilierung in eine Zwischensprache nicht so flexibel wie eine reine Skriptsprache, bietet dafür aber einen Geschwindigkeitsvorteil. Außerdem stehen durch die Java-Klassenbibliothek Möglichkeiten zur Verfügung, die innerhalb einer HTML-Seite mit JavaSkript nicht umgesetzt werden können.

Das Java-Programm soll also primär eine Visualisierung der Daten ermöglichen. Die zentrale Klasse dafür ist "MainForm", die das Hauptfenster erzeugt und darauf die GUI-Elemente anordnet. Dies sind ein Eingabefeld für die Eingabe des Chat-Textes, ein Text-Feld für die Darstellung der eingegebenen Texte und and Grafik-Panel zum Zeichnen der Chat-Räume mit den Avataren.
Die Instanz der MainForm verarbeitet auch Maus-Klicks. Daraus werden die Positionsänderungen des eigenen Avatars berechnet.
Das Grafik-Panel wird als Erweiterung von JPanel implementiert. Da es regelmäßig die Position der Avatare aktualisieren muss, implementiert es die Schnittstelle Runnable. Dies ermöglicht das Aufrufen einer Methode als Thread, in dem die Aktualisierung in einem regelmäßigen Intervall durchgefüht wird.
Eine Instanz der Klasse "DataContainer" dient zum Speichern der Liste der sichtbaren Avatare, deren Positionen und der Informationen über die Chat-Räume. Damit diese Daten auch dem Grafik-Panel zur Verfügung stehen, erhält es eine

Referenz auf das verwendete DataContainer-Objekt.
DataContainer enthält auch die Methoden für die Aktualisierung der Daten auf dem Server. Damit dies regelmäßig geschehen kann ist DataContainer eine Erweiterung der Klasse "TimerTask". So kann ein Timer eingerichtet werden, durch den die Server-Routinen von DataContainer in einem wiederkehrenden Intervall aufgerufen werden.
Die Verbindung zum Server wird von einer Instanz der Klasse "DataConnector" hergestellt. Diese Klasse enthält Methoden, um eine HTTP-Verbindung herzustellen und die PHP-Dateien auf dem Server anzusprechen.

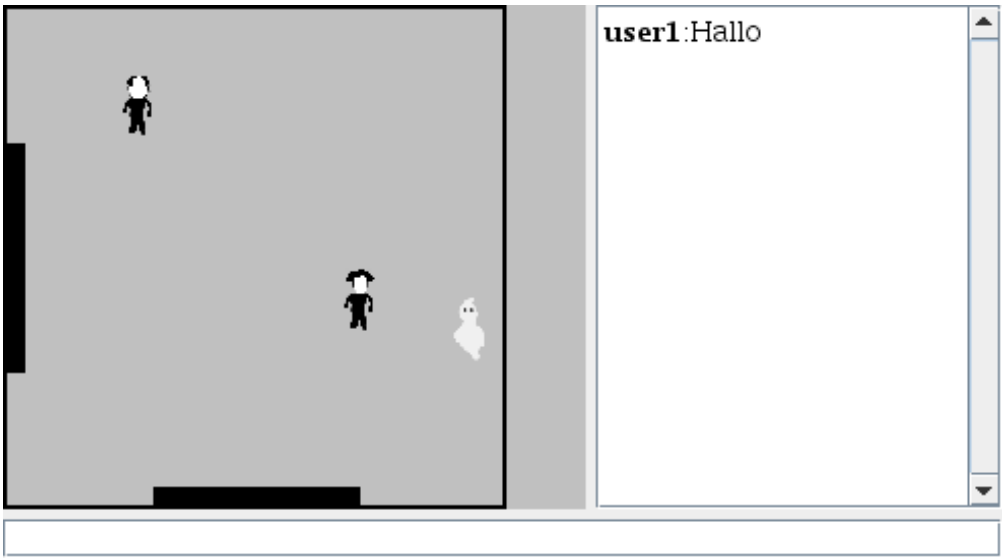
Die weiteren erstellten Klassen sind "LoginForm" zum Abfragen der Benutzerdaten für die Anmeldung, "Player" und "Room" zur Kapselung von Daten und JSONReader zum Auswerten der vom Server übermittelten Daten in JSON-Form.

Das UML-Diagramm stellt alle selbst implementierten Klassen sowie deren öffentliche Methoden und Attribute dar.



UML-Diagramm des Test-Clients

4.3 Auswertung



Die erstellten PHP-Skripte werden einmal auf einem lokalen Desktop-PC eingesetzt und auf den Planet-Hosting-Server kopiert.

Da die PHP-Skripte extra so konzipiert sind, dass sie universell eingesetzt werden können, müssen lediglich die Datenbank-Informationen angepasst werden. Dies kann entweder im Optionen-Block der Eingabedatei geschehen oder nach dem Generieren in der init.php.inc-Datei.

Mit Hilfe der Skripte werden dann die Datenbanktabellen erstellt und anschließend von Hand mit den Benutzerdaten gefüllt.

Auf diese Weise stehen ein lokaler Server und ein Web-Server für ein Java-Chat-System zur Verfügung.

Der Zielserver kann beim Client leicht umkonfiguriert werden, ohne dass eine Neu-Kompilierung notwendig ist. So können Verbindungsdaten für beide Fälle gesammelt werden.
Der Client zeichnet dabei die Zeit auf, die bei einer Serveranfrage vergeht. Dies umfasst das Vorbereiten der HTTP-Daten, den Transfer zum Server und den Empfang der Antwort. Außerdem übermittelt der Server auf Wunsch auch die Ausführungsdauer des PHP-Skriptes. Insgesamt stehen somit drei Werte pro Anfrage zur Verfügung:

netTime Dauer vom Absenden der Anfrage bis zum erhalten der Antwort

serverTime Dauer der Ausführung des PHP-Skriptes auf dem Server

updateTime Gesamtzeit der Anfrage, inklusive Datenverkehr durchs Netzwerk und lokale Auswertung der erhaltenen Daten

Die Messungen mit einem lokalen Webserver, bei dem somit Client und Server auf dem gleichen System liegen, ergaben Werte in dieser Form, wobei die letzte Zeile den Durchschnitt angibt:

netTime	serverTime	updateTime
5	3	5
9	5	9
5	3	5
	...	
6	3	6
5	2	5
6	4	6
6,32	3,15	6,53

Bei der lokalen Auswertung dauern die Anfragen in der Regel unter 10ms. Zwar gibt es immer wieder Ausbrecher nach oben, diese treten aber bei weit unter 10% der Anfragen auf. Sie können als typisch für das zugrunde liegende HTTP-Protokoll betrachtet werden, das keine deterministischen Aussagen über die Laufzeit zulässt.
Die durchschnittliche Antwortzeit beträgt hier ungefähr 7ms. Damit ist auch ohne Optimierung des Clients eine flüssige Bewegung der Avatare gegeben.

Die Werte der Messungen mit einem tatsächlichen Webserver im Internet weichen erheblich von den lokalen Werten ab.

netTime	serverTime	updateTime
128	12	128
119	12	120
125	14	126
	...	
161	12	161
200	19	200
178	16	178
177,14	17,64	177,37

Die Zugriffe auf den Web-Server dauern viel länger. Ein Verbindungszyklus

schlägt hier mit 120ms bis 210ms zu Buche. Auch hier sind Ausbrüche zu beobachten, die vereinzelt bis zu 310ms steigen. Die durchschnittliche Dauer liegt aber bei ungefähr 180ms. Damit sind 5 Aktualisierungen pro Sekunde möglich. Dies äußert sich in entsprechend abgehackten Bewegungen der Figuren. Die generelle Funktionalität ist aber trotzdem gegeben. Man kann Chat-Nachrichten übertragen und sieht anhand der Avatare, wieviele Personen sich im gleichen Raum aufhalten. Eine flüssige Bewegung ist nicht zwingend notwendig, könnte aber durch diverse Optimierungen zumindest annähernd erreicht werden.

Auffallend ist, dass auch die PHP-Ausführungszeit um das Vierfache gestiegen ist. Das deutet darauf hin, dass wahrscheinlich auch die Initialisierung der PHP-Engine relativ viel Zeit in Anspruch nimmt. Dies hat auch entsprechend Auswirkung auf die gemessene Gesamtdauer. Die anscheinend längere Dauer einer einzelnen Datenverbindung setzt sich also aus der Netzwerkzeit und der vermutlich längeren Initialisierungsphase zusammen. Insgesamt liegt die Performance des Webservers offensichtlich unter der des Desktop-Rechners.

Der verwendete Webserver von Planet-Hosting zeigt eine insgesamt geringe Leistung, die unter dem liegt, was aufgrund der Tests in Abschnitt 2.3.2 erwartet wurde. Um einen Vergleich anstellen zu können, wurde wie in Abschnitt 1.2 angekündigt ein zusätzlicher Webserver hinzugezogen. Dieser zeigt bessere Verbindungszeiten:

netTime	serverTime	updateTime
79	6	79
48	6	48
44	6	44
	...	
47	6	47
47	6	47
47	6	47
48,1	6,16	48,83

Auch die tatsächliche Ausführungsdauer des Skriptes ist wieder näher an den auf dem Desktop gemessenen Werten. Die durchschnittliche Gesamtzeit von unter 50ms lässt die Bewegungen beim Client wieder relativ flüssig erscheinen.

Der Vergleich der verschiedenen Hosting-Angebote ist aber nicht das Ziel der Untersuchung. Vielmehr geht es darum, zu zeigen, dass die Antwortzeiten aus-

reichen, um einen Abgleich der Daten zu ermöglichen. Die für den Test gewählte Darstellung der Avatare ist aber in dieser Form mit dem ersten Webserver nicht vollkommen zufriedenstellend, aber dennoch machbar. Durch die Unterschiede in den Webservern sieht man, dass man solche Diskrepanzen bei der Zielanwendung berücksichtigen muss.

Alle Test zeigen, dass die Auswertung auf dem Client nicht nennenswert ins Gewicht fällt. Insofern bietet sich das System auch für komplexere Anwendung an. Dabei ist nur darauf zu achten, dass der Mehraufwand nicht vom Server getragen wird, sondern vom Client übernommen werden kann. So kann zum Beispiel die Darstellung im Client noch verbessert werden (Animationen, bessere Grafiken) ohne die zu übertragende Datenmenge zu verändern. Hier wird auch der Nachteil von JavaScript noch deutlicher. In der Machbarkeitsanalyse hat sich gezeigt, dass komplexere Anwendungen JavaScript überfordern würden, während der Java-Client nun kaum Systemzeit beansprucht.

Kapitel 5

Ergebnis

5.1 Zusammenfassung

Die Übertragung von Daten zwischen mehreren Clients über das Internet ist weit verbreitet. Meist werden dafür aber dedizierte Server und proprietäre Protokolle verwendet, die auf den jeweiligen Zweck zugeschnitten sind. Komplexe Anwendungen sind auch ohne eine Spezialisierung selten umsetzbar.
Für simplere Einsätze ist es aber sinnvoll, auf einfachere Standards zurückzugreifen.

Die hier vorgestellten Techniken setzen auf das Protokoll HTTP auf und stellen eine Verbindung zu einem normalen Webserver her. Eine Dynamik zwischen Client und Server mit diesen Mitteln wird mit AJAX inzwischen auf vielen modernen Internetseiten genutzt. Selten findet man aber dabei Anwendungen für mehrere Clients.

Solche Multi-Client Anwendungen können mit dem hier erstellten Compiler vorbereitet werden. Wenn die geplante Anwendung festgelegt wurde, können über die definierte Sprache die benötigten Datenstrukturen beschrieben werden. Ohne einen festgelegten Client bleibt auch Spielraum für viele verschiedene Einsatzzwecke. Der hier verwendete Chat ist nur das naheliegendste Beispiel. Wichtig ist aber auch, dass viele Bausteine der Client-Implementierung wiederverwendet werden können. Die für den Testclient entwickelten Klassen für die Verbindung zum Server und die Auswertung der Antwort im JSON-Format können z.B. in

anderen Clients zum Einsatz kommen.

Bei der Planung muss die Geschwindigkeit des Servers und der Clientimplementierung berücksichtigt werden. In Abhängigkeit von der erwarteten Dynamik sollte der Server die Antworten entsprechend zügig liefern können. Bei der eingesetzten PHP-Technik ist natürlich wie erwartet nicht mit Antwortzeiten zu rechnen, die mit denen von dedizierten Servertechniken verglichen werden können. Die Anwendungen müssen entsprechend darauf abgestimmt werden und die eingesetzte Client-Technik sollte in der Lage sein, die Unzulänglichkeiten abzufangen. Eine Möglichkeit, um z.B. die Bewegungen zu verbessern, sind mehrere Threads, die parallel Anfragen an den Server senden. Dadurch können in gleicher Zeit mehrere Datenaktualisierungen durchgeführt werden. Ob diese Methode tatsächlich eine Verbesserung bringt, hängt aber auch von der Leistung des Webservers ab. Da die Tests zeigen, dass die reine PHP-Ausführungszeit nach wie vor relativ gering bleibt, wird die Übertragung hauptsächlich vom Netzverkehr ausgebremst.

Mit den erwarteten Einschränkungen ist aber der erstellte Compiler ein brauchbares Werkzeug, das Anwender dabei unterstützt, einfache Multi-Client Programme umzusetzen, wenn man keine Möglichkeit hat, einen speziellen Server einzusetzen.

5.2 Ausblick

Ein besonderes Charakteristikum der entwickelten Multi-Client-Technik ist die Flexibilität. Der Einsatzzweck ist nicht festgelegt und es hängt von der Kreativität des Benutzers ab, welche Anwendungsmöglichkeiten er findet.
Darum ist es möglich, dass in Zukunft PHP-Skripte, die von dem Compiler erzeugt wurden, in vielen verschiedenen Bereichen eingesetzt werden.

Die Sprache zum Festlegen der Datenstrukturen kann noch optimiert werden, um benutzerfreundlicher zu sein. Durch die derzeitige Grammatikdefinition ist z.B. die Reihenfolge einiger Parameter noch vorgegeben. Eine Anpassung der Grammatik würde die Eingabe weniger fehleranfällig machen.
Außerdem könnte es eine Erweiterung geben, durch die man Werte angibt, mit denen die Datenbanktabellen initial gefüllt werden.

Es wurden noch keine Tests mit einer HTTPS-Verbindung durchgeführt. Da die Verschlüsselung aber von der Webserver-Software und vom Client gehandhabt wird, hat die Verwendung von HTTPS keinen Einfluss auf die PHP-Skripte. Es ist also nicht zu erwarten, dass der Compiler für eine Unterstützung der Verschlüsselung modifiziert werden muss.

Anhang A-1: JavaCC Parser-Anweisungen

```
SKIP :
{
     " " | "\r" | "\t" | "\n"
}
TOKEN : /* GENERAL */
{
    < tOPTIONS : "OPTIONS" >
|    < tGROUP : "GROUP" >
|    < tSTATIC : "static" >
|    < tREF : "ref" >
|    < tMREF : "mref" >
|    < tPRIO : "prio" >
|    < tCHANGE : "change" >
|    < tUPDATE : "update" >
}
TOKEN : /* DATATYPES */
{
    < tINT: "INT" >
|   < tFLOAT: "FLOAT" >
|    < tSTRING: "STRING" >
|    < tBOOL: "BOOL" >
}
TOKEN :
{
    < DIGIT: ["0" - "9"] >
|    < CHAR: ["a" - "z"] | [ "A" - "Z"] >
}

/* CDATA */
TOKEN :
{
    < CDATAstart: "<![CDATA[" > : CDATA_CONTENT_STATE
}

< CDATA_CONTENT_STATE >
TOKEN:
{
    < CDATAend: "]]>" > : DEFAULT
}

< CDATA_CONTENT_STATE >
TOKEN:
{
    < CDATAcontent : ~[] >  /* allow any character */
}

/* COMMENTS */
SKIP:
{
     "/*"  : COMMENT_STATE
}

< COMMENT_STATE >
SKIP:
{
    < ~[] >
}

< COMMENT_STATE >
SKIP:
{
    "*/" : DEFAULT
}
```

```
int parseInput() : {}
{
  Definition()  { return 0; }
}

void Definition() : {}
{
  Optionblock() Groupdefinitions()
}

void Optionblock() : {}
{
    <tOPTIONS> "{" OptionList() "}"
}

void OptionList() : {}
{
    (  OptionStatement() ";" )*
}

void OptionStatement() : {}
{
    Identifier() "="  "\"" OptionValue()  "\""
}


void OptionValue() : {}
{
    ( <CHAR> | <DIGIT> | "_"  )*
}


void Groupdefinitions() : {}
{
    ( GroupBlock() )*
}

void GroupBlock() : {}
{
    <tGROUP> Identifier() "{" GroupAttributeList() GroupDataList() "}"
}

void GroupAttributeList() : {}
{
    ( <tSTATIC> ";" )? ( GroupAttribute() )*
}

void GroupAttribute() : {}
{
    <tREF> Identifier() "-" Identifier() DataAttributeList() ";" |
    <tMREF>Identifier() "-" Identifier() DataAttributeList() ";"
}

void GroupDataList() : {}
{
    ( DatatypeRes() Identifier() DataAttributeList() ";" )*
}

void DataAttributeList() : {}
{
    ( DataAttributePrio() )? ( DataAttributeCng() )? ( DataAttributeUpd() )?
}

void DataAttributePrio() : {}
{
```

```
    <tPRIO> "{" <CDATAstart> CDATAcontent() <CDATAend> "}"
}

void DataAttributeCng() : {}
{
    <tCHANGE> "{" <CDATAstart> CDATAcontent() <CDATAend> "}"
}

void DataAttributeUpd() : {}
{
    <tUPDATE> "{" <CDATAstart> CDATAcontent() <CDATAend> "}"
}

void DatatypeRes() : {}
{
        <tINT> | <tFLOAT> |    <tSTRING> | <tBOOL>
}


void CDATAcontent() : {}
{
    ( <CDATAcontent> )*
}

void Identifier() : {}
{
    <CHAR>    (     <CHAR> | <DIGIT> | "_"     )*
}
```

Anhang A-2: JavaCC Quelltext

```
PARSER_BEGIN(phpServParser)
import java.io.*;
import java.util.*;

public class phpServParser {

    // define the path for the templates
    final static String TEMPLATE_PATH = "/templates/";

    // global variable that hold the names of the templates
    private static String[] fileNameList = {
                "checklogin.php.inc",    "getdata.php",    "getprio.php.inc",    "init.php.inc.tmpl",    "initdbtables.php.tmpl",
                "login.php",    "logout.php",    "setchangeinfo.php.inc.tmpl",    "setdata.php",    "setprioinfo.php.inc.tmpl",
                "setupdateinfo.php.inc.tmpl",    "update.php.inc"
            };

    // global varaibles where all the info for the creation of the PHP files are stored
    private static Hashtable<String, String> replacementData;
    private static Hashtable<String, Vector<Hashtable<String, String>> > sectionData;

    // varaibles that help to keep trace of the current parser action
    private static String currentGroup = "";
    private static String currentField = "";
    private static String currentDatatype = "";
    private static String currentTableType = ""; // UPDATE / INSERT

    private static StringBuilder currentFullFieldList;
    private static StringBuilder currentSqlFieldList;
    private static StringBuilder currentUpdateFieldList;
    private static StringBuilder currentPrioFieldList;
    private static StringBuilder currentChangeFieldArray;
```

```
private static StringBuilder currentPrioWhereString;


/**
*    main method that prepares the in- and outputstreams and invokes the actual parsing.
*/
public static void main(String args[]) throws ParseException {

    try {
        //String filePath = System.getProperty("user.dir");
        //System.out.println( filePath);

        String inFile = "default.pin";
        String outDir = "default/";

        // read the parameters
        if (args.length > 0){
            inFile = args[0];
        }

        if (args.length > 1){
            outDir = args[1];

            // remove trailing backslash
            if (outDir.endsWith("\\")){
                outDir = outDir.substring(0, outDir.length() -1);
            }

            // prepend slash if not present
            if (! outDir.endsWith("/")){
                outDir = outDir + "/";
            }
        }

        // check for input file and output dir
        File inFileObj = new File(inFile);
        File outDirObj = new File(outDir);
        if (! inFileObj.exists()){
            throw (new Exception("File '" + inFile + "' not found!"));
        }
        if (! outDirObj.exists()){
            if (! outDirObj.mkdir()){
                throw(new Exception("Output directory '" + outDir + "' could not be created!"));
            }
        }

        // initialize the global data arrays/hashtables
        replacementData = new Hashtable<String, String>();
        sectionData = new Hashtable<String, Vector<Hashtable<String, String>>>();

        // start reading input
        FileInputStream filestream = new FileInputStream( inFile );
        InputStream in = new DataInputStream(filestream);

        //phpParser parser =
        new phpServParser(in);

        System.out.println("Reading from "+ inFile);


        try {
            if (phpServParser.parseInput() == 0) {
                System.out.println("Parser successful");

                // use gathered data to create the PHP files
                for (String fileName : fileNameList){
```

```
                    String inputName = TEMPLATE_PATH + fileName;
                    String outputName = outDir + fileName;

                    // determine output filename
                    if (fileName.substring( fileName.length() -5).equals(".tmpl")    ){
                        outputName = outDir + fileName.substring( 0, fileName.length() -5);
                    }

                    System.out.println("writing " + outputName);

                    // create file from template
                    TemplateEvaluator tempTE = new TemplateEvaluator(inputName, outputName, replacementData, sectionData);
                    try {
                        tempTE.evaluateTemplate();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                }
            }
        } catch (Exception e) {
            System.out.println("Exception:");
            System.out.println(e.getMessage());
        } catch (Error e) {
            System.out.println("Compiler Error:");
            System.out.println(e.getMessage());
        }



    } catch (Exception e) {
        System.out.println("Input Error:");
        System.out.println(e.getMessage());
    }
}

/**
 * this adds a new replacement-block to the "sectiondata" datastructure
 * @param sectionKey    the key that starts this section in the template file
 * @param sectionDataHash    a hashtable that holds the replacements for one section occurance
 */
private static void addSectionDataToHashtable(String sectionKey, Hashtable<String,String> sectionDataHash){
        Vector<Hashtable<String, String>> tempSectionList;

        // check if there already exists data for this section
        if (sectionData.containsKey(sectionKey)){
            // get existing data ( the new data will be appended)
            tempSectionList = sectionData.get(sectionKey);
        }
        else {
            // create a new vector for this section
            tempSectionList = new Vector<Hashtable<String, String>>();
        }

        // put data to sectionData
        tempSectionList.add(sectionDataHash);
        sectionData.put(sectionKey, tempSectionList);
}
}
PARSER_END(phpServParser)

SKIP :
{
     " " | "\r" | "\t" | "\n"
}
TOKEN : /* GENERAL */
```

```
{
    < tOPTIONS : "OPTIONS" >
|    < tGROUP : "GROUP" >
|    < tSTATIC : "static" >
|    < tREF : "ref" >
|    < tMREF : "mref" >
|    < tPRIO : "prio" >
|    < tCHANGE : "change" >
|    < tUPDATE : "update" >
}
TOKEN : /* DATATYPES */
{
    < tINT: "INT" >
|   < tFLOAT: "FLOAT" >
|    < tSTRING: "STRING" >
|    < tBOOL: "BOOL" >
}
TOKEN :
{
    < DIGIT: ["0" - "9"] >
|    < CHAR: ["a" - "z"] | [ "A" - "Z"] >
}

/* CDATA */
TOKEN :
{
    < CDATAstart: "<![CDATA[" > : CDATA_CONTENT_STATE
}

< CDATA_CONTENT_STATE >
TOKEN:
{
    < CDATAend: "]]>" > : DEFAULT
}

< CDATA_CONTENT_STATE >
TOKEN:
{
    < CDATAcontent : ~[] >  /* allow any character */
}

/* COMMENTS */
SKIP:
{
     "/*"  : COMMENT_STATE
}

< COMMENT_STATE >
SKIP:
{
    < ~[] >
}

< COMMENT_STATE >
SKIP:
{
    "*/" : DEFAULT
}


int parseInput() : {}{
  Definition()  { return 0; }
}

void Definition() : {}{
  Optionblock() Groupdefinitions()
}
```

```
void Optionblock() : {}{
    <tOPTIONS> "{" OptionList() "}"
}

void OptionList() : {
    Vector<Hashtable<String,String>> optionListVector = new Vector<Hashtable<String,String>>();
    Hashtable<String, String> singleOptionData;

    Hashtable<String, String> OptionDataList = new Hashtable<String, String> ();

    // setting the default option parameters
    OptionDataList.put("DBSERVER","\"localhost\"");
    OptionDataList.put("DBUSER","\"root\"");
    OptionDataList.put("DBPWD","\"root\"");
    OptionDataList.put("DBNAME","\"playhp\"");
    OptionDataList.put("TABLEPREFIX","\"chat_\"");
    OptionDataList.put("USELOGIN","\"1\"");
    OptionDataList.put("LOGINMD5","\"1\"");
    OptionDataList.put("MAXUSERS","\"2\"");
    OptionDataList.put("PRIOWHEREFORUPDATE","\"0\"");
    OptionDataList.put("HTTPREQUEST","\"GP\"");
    OptionDataList.put("UPDATEMILLISEC","\"100\"");
    OptionDataList.put("SHOWDEBUGINFO","\"0\"");
}{
    ( OptionStatement(OptionDataList) ";" )*
    {
        Enumeration<String> allOptionNames = OptionDataList.keys();

            while( allOptionNames.hasMoreElements() ){
                String optionName = allOptionNames.nextElement();

                singleOptionData = new Hashtable<String, String>();
                singleOptionData.put("###OPTIONNAME###" , optionName );
                singleOptionData.put(" ###OPTIONVALUE###" , OptionDataList.get(optionName));

                optionListVector.add(singleOptionData);
            }

        sectionData.put("###SECTION_OPTIONS_BEGIN###" , optionListVector);
    }
}

void OptionStatement(Hashtable<String, String> OptionDataList) : {
    String optionName ;
    String optionValue;
}{
    {
        optionName = "";
    }
    optionName = Identifier() "="  "\"" optionValue = OptionValue()  "\""
    {
        OptionDataList.put(optionName.toUpperCase() ,  "\"" + optionValue.toString() + "\"");
    }
}


String OptionValue() : {
    Token singleChar;
    StringBuilder optionValue;
}{
    {
        optionValue = new StringBuilder();
    }
    (
        singleChar= <CHAR> {optionValue.append(singleChar.image); } |
        singleChar= <DIGIT> {optionValue.append(singleChar.image) ;} |
```

```
                    "_" {optionValue.append("_") ;}
    )*
    {
        // clear references to let the garbage collector do its work
        singleChar = null;

        return optionValue.toString();
    }
}


void Groupdefinitions() : {}{
    ( GroupBlock() )*
}

void GroupBlock() : {}{
    {
        // reset the variables with infos about the group
        currentGroup = "";
        currentField = "";
        currentDatatype = "";
        currentTableType = ""; // UPDATE / INSERT

        currentFullFieldList = new StringBuilder();
        currentSqlFieldList = new StringBuilder();
        currentUpdateFieldList = new StringBuilder();
        currentPrioFieldList = new StringBuilder();
        currentChangeFieldArray = new StringBuilder();

        currentPrioWhereString = new StringBuilder();
    }
    <tGROUP> currentGroup=Identifier() "{" GroupAttributeList() GroupDataList() "}"
    {
        // store group data in hash datastructure
        //Vector<Hashtable<String,String>> tempDataVector;
        Hashtable<String, String> tempSectionData;

        // prepare Strings
        String FullFieldString = currentFullFieldList.toString();
        if (currentFullFieldList.length() > 0){
            FullFieldString = currentFullFieldList.substring(0,currentFullFieldList.length() -1);
        }
        String groupString = currentGroup.toLowerCase();

        // SQL info
        String SqlFieldListString = currentSqlFieldList.toString();
        if (groupString.equals("user")){
            SqlFieldListString = ",username VARCHAR( 50 ) NOT NULL ,userpwd VARCHAR( 25 ) NOT NULL,"+
             " online TINYINT NOT NULL,  tstamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP" +
             SqlFieldListString;
        }
        else if (groupString.equals("userdata")){
            SqlFieldListString = ",uid BIGINT UNSIGNED NOT NULL" + SqlFieldListString;
            currentTableType = "UPDATE";

            // create a list of global PHP-Variables
            replacementData.put("###GLOBAL_VARIABLES###" , "$user_id,$user_" + FullFieldString.replace(",", ",$user_") );
        }

        // groupfield info
        if (!groupString.equals("user")){

            String groupTableString = "constTABLEPREFIX.'" + groupString + "'";
            if (currentGroup.toLowerCase().equals("userdata")){
                groupTableString = "constTABLEPREFIX.'user LEFT JOIN "+
                 "'.constTABLEPREFIX.'userdata ON "+
                 "'.constTABLEPREFIX.'user.id='.constTABLEPREFIX.'userdata.uid'";
```

```
        FullFieldString = "username, " + FullFieldString;
    }

    tempSectionData = new Hashtable<String, String>();
    tempSectionData.put("###GROUPNAME###" , groupString);
    tempSectionData.put("###FIELDLIST###" , FullFieldString );
    tempSectionData.put("###SQLTABLESTRING###" , groupTableString);

    addSectionDataToHashtable("###SECTION_GROUPFIELDS_DEFINITION_BEGIN###" , tempSectionData);
}

// setarray info
if (currentChangeFieldArray.toString().trim().length()>0){

    String tempWhereString = "";
    if (groupString.equals("userdata")){
        tempWhereString = "uid='$user_id'";
    }
    else if (groupString.equals("user")){
        tempWhereString = "id='$user_id'";
        currentTableType = "UPDATE";
    }
    if (currentTableType.length()==0){
        currentTableType = "INSERT";
    }
    tempSectionData = new Hashtable<String, String>();
    tempSectionData.put("###GROUPNAME###" , groupString);
    tempSectionData.put("###GROUPTYPE###" , currentTableType );
    tempSectionData.put("###FIELDARRAY###" , currentChangeFieldArray.toString());
    tempSectionData.put("###GROUPWHERE###" , tempWhereString);

    addSectionDataToHashtable("###SECTION_SETARRAY_DEFINITION_BEGIN###" , tempSectionData);
}

// prioarray info
if (currentPrioFieldList.length() > 0){
    String tempPrioFieldList = "";
    if (groupString.equals("userdata")){
        tempPrioFieldList = currentPrioFieldList.append("uid").toString();
    }
    else {
        // remove trailing ","
        tempPrioFieldList =  currentPrioFieldList.substring(0,currentPrioFieldList.length()-1);
    }
    // remove trailing "AND "
    String tempPrioWhereString= currentPrioWhereString.substring(0,currentPrioWhereString.toString().length()-4).trim();

    tempSectionData = new Hashtable<String, String>();
    tempSectionData.put("###GROUPNAME###" , groupString);
    tempSectionData.put("###FIELDLIST###" , tempPrioFieldList );
    tempSectionData.put("###PRIOSQLWHERE###" , tempPrioWhereString);

    addSectionDataToHashtable("###SECTION_PRIOARRAY_DEFINITION_BEGIN###" , tempSectionData);
}

// updatearray info
if (currentUpdateFieldList.length() >0){
    // remove trailing ","
    String tempUpdateFieldList = currentFullFieldList.toString() + "updatetstamp";

    tempSectionData = new Hashtable<String, String>();
    tempSectionData.put("###GROUPNAME###" , groupString);
    tempSectionData.put("###FIELDLIST###" , tempUpdateFieldList );

    addSectionDataToHashtable("###SECTION_UPDATEARRAY_DEFINITION_BEGIN###" , tempSectionData);

    // all tables that may be updated by the server get an update field
```

```
            SqlFieldListString = SqlFieldListString + ",updatetstamp BIGINT UNSIGNED NOT NULL" ;
        }

        tempSectionData = new Hashtable<String, String>();
        tempSectionData.put("###GROUPNAME###" , groupString);
        tempSectionData.put("###SQLFIELDDEFINITION###" , SqlFieldListString);

        addSectionDataToHashtable("###SECTION_SQLCOMMANDS_DEFINITION_BEGIN###" , tempSectionData);
    }
}

void GroupAttributeList() : {}{
    ( <tSTATIC> ";" {currentTableType = "UPDATE" ;} )? ( GroupAttribute() )*
}

void GroupAttribute() : {}{
    <tREF>{
            // if any field has attribute MREF the tableType has already been set to "INSERT"
            if (currentTableType.equals("")){
                currentTableType = "UPDATE" ;
            }
            currentDatatype = "INT";
    }
    currentField = Identifier() "-" Identifier() DataAttributeList() ";"
|
    <tMREF>{
            currentTableType = "INSERT" ;
            currentDatatype = "INT";
    }
    currentField = Identifier() "-" Identifier() DataAttributeList() ";"
}

void GroupDataList() : {}{
    ( currentDatatype=DatatypeRes() currentField = Identifier() DataAttributeList() ";" )*
}

void DataAttributeList() : {} {
    {
        currentFullFieldList.append(currentField + ",");

        // create the corresponding SQL Field according to the datataype
        if (currentDatatype.equals("INT")){
            currentSqlFieldList.append( "," + currentField + " BIGINT NOT NULL");
        }
        if (currentDatatype.equals("FLOAT")){
            currentSqlFieldList.append( "," + currentField + " FLOAT(53) NOT NULL");
        }
        if (currentDatatype.equals("STRING")){
            currentSqlFieldList.append( "," + currentField + " TEXT NOT NULL");
        }
        if (currentDatatype.equals("BOOL")){
            currentSqlFieldList.append( "," + currentField + " TINYINT NOT NULL, ");
        }
    }
    ( DataAttributePrio() )? ( DataAttributeCng() )? ( DataAttributeUpd() )?
}

void DataAttributePrio() : {
    String cdataString;
}{
    <tPRIO> "{" <CDATAstart> cdataString = CDATAcontent() <CDATAend> "}"
    {
        currentPrioFieldList.append(currentField + ",");
        if (cdataString.trim().length() > 0){
            currentPrioWhereString.append(cdataString.replace("\"" , "\\\"") + " AND ");
        }
    }
```

```
}

void DataAttributeCng() : {
String cdataString;
}{
    <tCHANGE> "{" <CDATAstart> cdataString = CDATAcontent() <CDATAend> "}"
    {
        currentChangeFieldArray.append( "'" + currentField + "' => '',\r\n");

        //only if there is a restriction for the change, a check function will be created
        if (!cdataString.trim().equals("")){

            Hashtable<String, String> tempSectionData = new Hashtable<String, String>();

            tempSectionData.put("###GROUPNAME###" , currentGroup);
            tempSectionData.put("###FIELDNAME###" , currentField);
            tempSectionData.put("###USER_INPUT_CHANGE###" , cdataString);

            addSectionDataToHashtable("###SECTION_FUNCTION_TESTSET_BEGIN###",tempSectionData );
        }
    }
}

void DataAttributeUpd() : {
    String cdataString;
}{
    <tUPDATE> "{" <CDATAstart> cdataString = CDATAcontent() <CDATAend> "}"
    {
        currentUpdateFieldList.append(currentField + ",");
        //only if there is an instruction for the update, an update function will be created
        if (!cdataString.trim().equals("")){
            Hashtable<String, String> tempSectionData = new Hashtable<String, String>();

            tempSectionData.put("###GROUPNAME###" , currentGroup);
            tempSectionData.put("###FIELDNAME###" , currentField);
            tempSectionData.put("###USER_INPUT_UPDATE###" , cdataString);

            addSectionDataToHashtable("###SECTION_FUNCTION_UPDATE_BEGIN###",tempSectionData );
        }
    }
}

String DatatypeRes() : {}
{
        <tINT> { return "INT"; }
    |    <tFLOAT> { return "FLOAT"; }
    |    <tSTRING> { return "STRING"; }
    |    <tBOOL> { return "BOOL"; }
}


String CDATAcontent() : {
    Token dataContentToken;
    StringBuilder dataContentString;
}{
    {
            dataContentString = new StringBuilder();
    }
    (
        dataContentToken = <CDATAcontent>
        {
            dataContentString.append( dataContentToken.image );
        }
    )*

    {
        // clear references to let the garbage collector do its work
```

```
        dataContentToken = null;

        return dataContentString.toString() ;
    }
}

String Identifier() : {
    Token chidToken;
    StringBuilder id;
}{
    {
        id = new StringBuilder();
    }
    chidToken = <CHAR> {id.append( chidToken.image) ;}
    (
        chidToken =  <CHAR> {id.append( chidToken.image);} |
        chidToken =  <DIGIT> {id.append( chidToken.image);} |
        "_" {id.append( "_");}
    )*

    {
        // clear references to let the garbage collector do its work
        chidToken = null;

        return id.toString();
    }
}
```

Anhang A-3: Hilfsklasse TemplateEvaluator

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

public class TemplateEvaluator {

    public String inputFileName = "";
    public String outputFileName = "";
    public Hashtable<String, Vector<Hashtable<String,String>> > sectionData = null;
    public Hashtable<String, String> replacementData = null;

    /*
     * constructors
     */

    /**
     *
     */
    public TemplateEvaluator(){

    }

    /**
     *
     * @param inputFileName    file to read the template from
```

```
 */
public TemplateEvaluator(String inputFileName){
    this.inputFileName = inputFileName;
}

/**
 *
 * @param inputFileName    file to read the template from
 * @param outputFileName    file to write the output to
 */
public TemplateEvaluator(String inputFileName, String outputFileName){
    this.inputFileName = inputFileName;
    this.outputFileName = outputFileName;
}

/**
 *
 * @param inputFileName    file to read the template from
 * @param outputFileName    file to write the output to
 * @param replacementData    hashtable with replacements for general placeholder markers
 */
public TemplateEvaluator(String inputFileName, String outputFileName, Hashtable<String, String> replacementData){
    this.inputFileName = inputFileName;
    this.outputFileName = outputFileName;
    this.replacementData = replacementData;
}

/**
 *
 * @param inputFileName    file to read the template from
 * @param outputFileName    file to write the output to
 * @param replacementData    hashtable with replacements for general placeholder markers
 * @param sectionData    a list of replacements for multiple sections
 */
public TemplateEvaluator(String inputFileName,
 String outputFileName,
 Hashtable<String, String> replacementData,
 Hashtable<String, Vector<Hashtable<String, String>> > sectionData){
    this.inputFileName = inputFileName;
    this.outputFileName = outputFileName;
    this.replacementData = replacementData;
    this.sectionData = sectionData;
}

/**
 * read the template, replace all markers and write the result
 * @throws IOException
 */
public void evaluateTemplate() throws IOException {

    StringBuilder outputData = new StringBuilder();

    InputStream templateInput = this.getClass().getResourceAsStream(inputFileName);

    if (templateInput == null){
        throw new IOException("File '" + inputFileName + "' not found");
    }

    BufferedReader templateReader = new BufferedReader(new InputStreamReader(templateInput));

    String inputLine=null;
    while ( (inputLine = templateReader.readLine()) != null ){

        if ( sectionData.containsKey( inputLine.trim() ) ){
            // this line starts a defined section , so the complete block will be read

            String sectionStartTag = inputLine.trim();
```

```
            String sectionEndTag = inputLine.trim().substring( 0, inputLine.trim().length() - 9) + "_END###";

            String tempLine = null;
            StringBuilder sectionBlock = new StringBuilder();

            do {
                if ( (tempLine = templateReader.readLine()) != null ){
                    if (tempLine.trim().equals(sectionEndTag)){
                        // the end of this section was reached
                        tempLine = null;
                    }
                    else {
                        sectionBlock.append(substituteReplacementData(tempLine) + "\r\n");
                    }
                }

            } while (tempLine != null);

            outputData.append( substituteSectionData(sectionBlock, sectionData.get(sectionStartTag)) );
        }
        else {
            //in each line the placeholders are replaced, then the line is added
            inputLine = substituteReplacementData(inputLine) + "\r\n";
            outputData.append(inputLine);

        }
    }

    if (outputFileName.length() == 0){
        System.out.println("Error: no output filename given for " + inputFileName);
        return;
    }
    FileOutputStream tempStream = new FileOutputStream( outputFileName );
    OutputStream outStream = new DataOutputStream(tempStream);

    writeStringToStream(outStream, outputData.toString());
}

/**
* Takes the given input string and sends each byte to the OutputStream.
*
* @param    out    OutputStream    The stream where the string data is sent to
* @param    text    String    The string that will be sent to the OutputsStream
*
*/
private static void writeStringToStream(OutputStream out, String text){
    try {
        for (int i=0; i< text.length() ; i++){
            out.write((byte)text.charAt(i));
        }
    } catch (Exception e) {
        System.out.println ("Output Error:");
        System.out.println(e.getMessage());
    }
}

/*
 * Subfunctions for string replacement
 */

/**
 * replaces all playeholders according to "replacementData"
 * @param input    the string where the placeholders should be replaced
 * @return    the string with the replaced placeholders
 */
private String substituteReplacementData(String input){
    Enumeration<String> replaceKeys = replacementData.keys();
```

```
    while (replaceKeys.hasMoreElements()){
        String hashKey = replaceKeys.nextElement();

        input = input.replace(hashKey,  replacementData.get(hashKey) );
    }
    return input;
}

/**
 * replaces the placeholders in a whole section. As the input is a vector of hashtables, the replacement takes place multiple times.
 * This results in multiple section with different replacements each.
 * @param input A StringBuilder that holds the section
 * @param currentSectionData    a vector of hashtables, each hashtable stands for the replacements in one occurance of the section
 * @return    A StringBuilder that holds all the sections that where built according to the input data
 */
private StringBuilder substituteSectionData(StringBuilder input, Vector<Hashtable<String, String>> currentSectionData){

    StringBuilder output = new StringBuilder();

    // run through the section data
    for (Hashtable<String, String> singleSectionData : currentSectionData){

        Enumeration<String> replaceKeys = singleSectionData.keys();
        String tempInput = input.toString();

        // do all replacements for this section
        while (replaceKeys.hasMoreElements()){
            String hashKey = replaceKeys.nextElement();

            tempInput = tempInput.replace(hashKey,  singleSectionData.get(hashKey) );
        }
        output.append(tempInput);
    }
    return output;
}

/*
 * get/set methods
 */

public void setSectionData(
        Hashtable<String, Vector<Hashtable<String, String > > > sectionData) {
    this.sectionData = sectionData;
}

public void setReplacementData(Hashtable<String, String> replacementData) {
    this.replacementData = replacementData;
}

public void setOutputFileName(String outputFileName) {
    this.outputFileName = outputFileName;
}

public void setInputFileName(String inputFileName) {
    this.inputFileName = inputFileName;
}

public Hashtable<String, Vector<Hashtable<String, String>> > getSectionData() {
    return sectionData;
}

public Hashtable<String, String> getReplacementData() {
    return replacementData;
}

public String getOutputFileName() {
```

```
        return outputFileName;
    }

    public String getInputFileName() {
        return inputFileName;
    }
}
```

Anhang A-4: Datenmodel für das Testprogramm

```
OPTIONS {
        DBSERVER = "localhost" ;
        DBNAME = "playhp" ;
        DBUSER = "root" ;
        DBPWD = "root";
        TABLEPREFIX = "chat1_" ;
        MAXUSERS = "5" ;
        USELOGIN = "1" ;
        LOGINMD5 = "1" ;
        HTTPREQUEST = "GP" ;
        UPDATEMILLISEC = "100";
        PRIOWHEREFORUPDATE =  "0";
}

GROUP user {
        /* Group 'user' automatically creates fields 'username' , 'userpwd', 'online' */
}

/* Group 'userdata' provides data, that all PHP-Scripts and SQL queries may access ( plus "$user_id")
        a reference to 'user' will be created, too */
GROUP userdata {
        INT room
                change {
                        <![CDATA[
                                $roomDif = abs( $user_room - $room );
                                if ( $roomDif == 1 || $roomDif == 2  || $roomDif == 0  ) {
                                        $isValid = TRUE;
                                }
                        ]]>
                };

        INT xpos
                prio {
                        <![CDATA[
                                room="$user_room"
                        ]]>
                }
                change {
                        <![CDATA[
                        ]]>
                };

        INT ypos
                prio {
                        <![CDATA[

                        ]]>
                }
                change {
                        <![CDATA[
                        ]]>
                };
}
```

```
GROUP userlooks {
        ref person - user;

        INT hair;
        INT eyes;
        INT nose;
        INT mouth;
        INT face;
        INT shirt;
}

GROUP messages {
        mref sender - user
                change {
                        <![CDATA[
                        if ($sender == $user_id)
                                $isValid = TRUE;
                        ]]>
                };
        mref position - room
                change {
                        <![CDATA[
                        ]]>
                };
        STRING message
                change {
                        <![CDATA[
                        if ($sender != '')
                                $isValid = TRUE;
                        ]]>
                };
        INT tstamp
                change {
                        <![CDATA[
                                $tstamp = getMillisecs();
                                $isValid = TRUE;
                        ]]>
                };
}

GROUP room {
        mref exitnorth - room;
        mref exiteast - room;
        mref exitsouth - room;
        mref exitwest - room;

        STRING roomname;
}

/*      For testing the update PHP-Script a group "ghost" is created.
        This group creates data for a ghost that roams in the chatrooms
        (it is up to the client implementation what to do with the data) */
GROUP ghost {
        INT room
                prio {
                        <![CDATA[
                        room = '$user_room'
                        ]]>
                };
        INT xpos
                prio {
                        <![CDATA[

                        ]]>
                }
                update {
```

```
                <![CDATA[
                srand();
                $xpos +=  $xdir * (getMillisecs() - $updatetstamp)/ 5 ;

                if ($xpos < 1){

                        if ($room == 2 || $room == 4){
                                $xpos = 800-50;
                                $room = $room-1;
                        }
                        else {
                                $xpos = 1;
                                $xdir = 1;
                        }
                }

                if ($xpos > 800-50){

                        if ($room == 1 || $room == 3){
                                $xpos = 1;
                                $room = $room +1;
                        }
                        else {
                                $xpos = 800-50;
                                $xdir = -1;
                        }
                }

                $changeDirProb = round( rand( 0 , 100 ) );
                if ($changeDirProb > 95) $xdir = -1;
                else if ($changeDirProb > 90) $xdir = 1;
                else if ($changeDirProb > 85) $xdir = 0;
                ]]>
        };

INT ypos
        prio {
                <![CDATA[

                ]]>
        }
        update {
                <![CDATA[
                srand();
                $ypos += $ydir *  (getMillisecs() - $updatetstamp)/ 5 ;
                if ($ypos < 1){

                        if ($room == 3 || $room == 4){
                                $ypos = 800-100;
                                $room = $room-1;
                        }
                        else {
                                $ypos = 1;
                                $ydir = 1;
                        }
                }

                if ($ypos > 800-100){

                        if ($room == 1 || $room == 2){
                                $ypos = 1;
                                $room = $room +1;
                        }
                        else {
                                $ypos = 800-100;
                                $ydir = -1;
                        }
```

```
                    }

                    $changeDirProb = round( rand( 0 , 100 ) );
                    if ($changeDirProb > 95) $ydir = -1;
                    else if ($changeDirProb > 90) $ydir = 1;
                    else if ($changeDirProb > 85) $ydir = 0;
                    ]]>
            };
        INT xdir;
        INT ydir;
}
```

Literaturverzeichnis

Bücher

[Aho08] Alfred V. Aho, Jeffrey Ullmann, Ravi Sethi, Monica S. Lam
Compiler: Prinzipien, Techniken und Werkzeuge
2008
ISBN 3827370973, 9783827370976

[Marco78] Tom DeMarco
Structured Analysis and System Specification
1978
ISBN 0917072073, 9780917072079

Internet Links

[BuSu] Bunte Suppe
Das Internet Mosaik
2009
http://www.buntesuppe.de

[Flash] Adobe Flash
Entwicklung Multimedialer Inhalte
2009
http://www.adobe.com/de/products/flash

[GDocs] Google Docs
Google Text & Tabellen
2009
http://docs.google.com

[JCC] Java Parser/Scanner Generator für Java
JavaCC

2007
https://javacc.dev.java.net/

[JSMD5] Homepage Paul Andrew Johnston
JavaScript Implementierung von MD5
2008
http://pajhome.org.uk/crypt/md5/

[PHP] PHP.net
PHP Dokumentation
2009
http://www.php.net/docs.php

[PHP5] PHP.net
PHP5 Backward Incompatible Changes
2009
http://www.php.net/manual/en/migration5.incompatible.php

[Pong] Wikipedia
Wikipedia Artikel: Pong
2009
http://de.wikipedia.org/wiki/Pong_(Computerspiel)

[QStar] Quantum Star
PHP Online Multiplayer Framework
2006
http://www.quantum-star.com/news.php

[Request] Wikipedia
Wikipedia Artikel: HTTP Request
2009
http://de.wikipedia.org/wiki/XMLHttpRequest

[selfHTML] Self HTML
HTML-Dateien selbst erstellen
2007
http://de.selfhtml.org

[SFox] Smart Fox Server
Server Software für Flash basierte Anwendungen
2008
http://www.smartfoxserver.com/

[Web2] Wikipedia
Wikipedia Artikel: Web2.0
2009
http://de.wikipedia.org/wiki/Web_2.0

Hiermit versichere ich, die Arbeit selbständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt zu haben.

Denis Mederake Essen, 27.04.2009