Buzzword-Bingo

7. April 2011

Team

Name	MatrikelNr	Studiengang
Lena Pohlmann	0923791	4IB
Stefan Düsing	0912644	5IB
Sebastian Scheuermann	1015014	5IB
Kai Hillenbrand	0922730	4IB
David Stammer	0930140	4IB
Martina Kraus	0926146	4IB

Inhaltsverzeichnis

Kapitel 1

Die Aufgabe

Die Aufgabe wurde im Rahmen des Softwareentwicklungsprojekts von 6 Studenten der Hochschule Mannheim für angewandte Wissenschaften erarbeitet.

1.1 *Die Aufgabenstellung*

Entwicklung einer grafischen Anwendung, mit der die Benutzer gegeneinander Buzzword-Bingo spielen können. Buzzword-Bingo ist ein Klassiker der IT-Geschichte und hilft dabei, die übermäßige Verwendung inhaltsleerer Schlagwörter (häufige Anglizismen[1]) in Vorträgen bzw. Präsentationen sichtbar machen.

1.2 Die Gruppenaufgabe

Die Aufgabe jeder Gruppe war es, ein Buzzword-Bingo als verteilte grafische Anwendung zu entwickeln

Folgende Plattformen durften wir verwenden:

- Mobilanwendung unter Android oder iPhone/iPod Touch
- Webanwendungen in einer Cloud-Plattfom (PaaS[2]) wie Google App Engine oder AWS Elasitc Beanstalk
- Java - Anwendung mit AWT oder Swing
- Kommandozeilenanwendung mit grafischer Bibliothek ncurses
- Anwendung mit GTK+ oder QU

[1] Anglizismen: Einflüsse der Englischen Sprache auf andere Sprachen
[2] Platform as a service

1.2.1 Die Anforderungen

- Über den Client der Anwendung sollen Spieler ein neues Spiel eröffnen, oder einem bereits bestehenden Spiel beitreten.
- Im Client wird festgelegt, wie groß das Spielfeld ist und welche Buzzwords auf dem Spielfeld zufällig verteilt werden sollen.
- Der Client, der ein neues Spiel angelegt hat, steuert den Server.
- Der Spieler soll über ein Netzwerk miteinander spielen können
- Im Client wird die Mitspielerliste angezeigt
- Hat ein Spieler gewonnen, werden die Mitspieler benachrichtigt

Kapitel 2

Analyse

2.1 Plattformwahl

2.1.1 Java - Anwendung mit Swing und AWT

Wir haben uns für eine Java - Anwendung als Plattform entschieden. Für die Oberfläche sollten uns die Bibliotheken javax.swing und java.awt zur Verfügung stehen.

Die Vorteile die dafür sprachen:

- jeder hatte solide Grundkenntnisse in der Programmiersprache Java mit der Entwicklungsumgebung Eclipse
- jeder hatte Erfahrung bereits in Swing und AWT gemacht
- Swing und AWT bieten ein breites Spektrum an Oberflächedesign
- Die Technologie bietet Programmieren den vollen Funktionsumfang aus der J2SE-API und eignet sich gut für Anwendungen in Unternehmensnetzen bei denen alle Anwender die gleiche Java - Version haben

Fakten die dagegen sprachen:

- Java ist eine sehr unperfomante Entwicklungssprache
- Die Vermischung von Swing (leichtgewichtige Komponenten) und AWT (schergewichtige Komponenten) sollte man eigentlich vermeiden, für die Aufgabelösung benötigten wir dennoch beide
- Die Layouts und Fenster selbst durch Code zu erzeugen ist mit Swing und AWT meist sehr umständlich.
- Die Größe der JRE mit 12,5 MB[1] ist der Grund für langsame Netzwerkanschlüsse

[1] Stand JRE 6.0

- Dauer der Initialisierungszeit für die Java Virtual Machine

- Inhalte von Java Applets können nicht von Suchmaschinen erfasst werden

Da es nur eine sehr kleine Anwendung war, mitder wir weder ins Internet noch ein riesiges Netzwerk aufbauen wollten, fielen diese Nachteile nicht sehr ins Gewicht.

Ebenso war es uns unwichtig, dass Inhalte unserer Anwendung von Suchmaschinen erfasst wurden. Die Vorteile, dass jeder mit Swing und AWT und der Programmiersprache Java an sich schon Erfahrungen hatte, überragte eindeutig den Nachteilen.

Dewesgen kamen wir zu dem Entschluss, dass eine Java-Anwendung mit Swing und AWT die beste Lösung für die Aufgabenstellung in unserem Team sei.

2.2 Netzwerktopologie

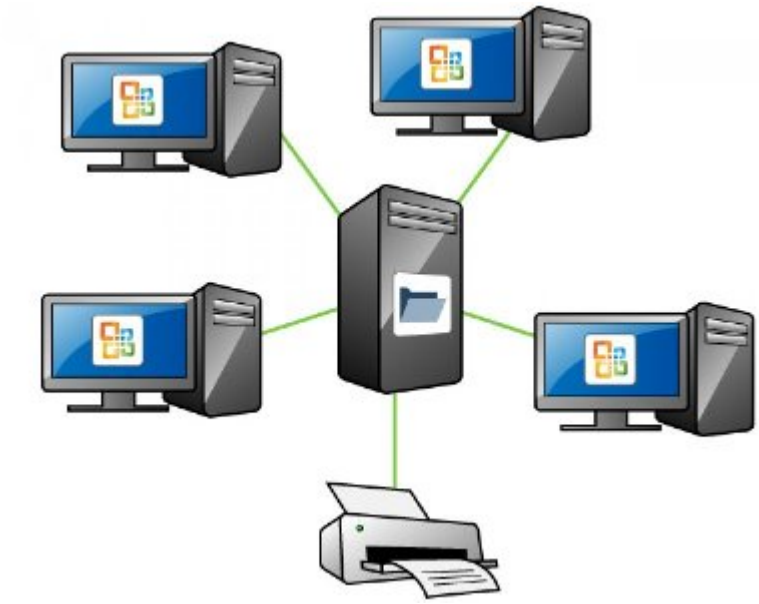
2.2.1 Zentraler Server und thin-Client

2.2.1.1 Client-Server-Modell

Ein Modell, das ermöglicht, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Diese Aufgaben werden von Programmen erledit, die in Clients[2] und Server[3] unterteilt werden. Der Client kann auf Wunsch Aufgaben vom Server anfordern. Der Server der sich auf einem gleichen oder externen Rechner im Netzwerk befindet, stellt diese Aufgabe bereit.

Der Vorteil besteht darin, dass die Programmlogik nicht mehr auf den einzelnen Rechnern liegt, sondern diese sehr simple bzw. „dumm“ gehalten werden können. Sämtliche Logik liegt also zentral auf einem Server, der sich darum kümmert, dass jede Anfrage von einem Client bearbeitet wird.

[2]Kunde, Dienstnutzer
[3]Bediener, Anbieter



2.2.1.2 Vor - bzw. Nachteile

Wir haben uns für eine Client- Server - Architektur entschieden, da

- Grundkenntnisse darüber bereits vorhanden waren
- zentralisierte Ressourcen, da der Server im Zentrum des Netzwerks steht und er so Redundanz und Inkonstistenz vermeidet
- Sicherheit: die Anzahl an Eingangspunkten, die Zugang zu Daten hat, geringer ist
- Die Architektur ermöglicht es, löschen oder hinzufügen von Clients ohne den Betrieb des Netzwerks zu beeinträchtigen.

Gegenüber standen dem folgende Nachteile:

- Kosten durch die technische Komplexität des Servers
- Fällt der Server aus, fällt das gesamte Netzwerk aus.

Letztere Auflistung der Nachteile haben wir entgegengewirkt, dass wir auf jedem Client auch eine Art „Back-Up-Server“ aufgesetzt haben, der schnell hochgefahren werden kann, falls schlimmeres passiert.

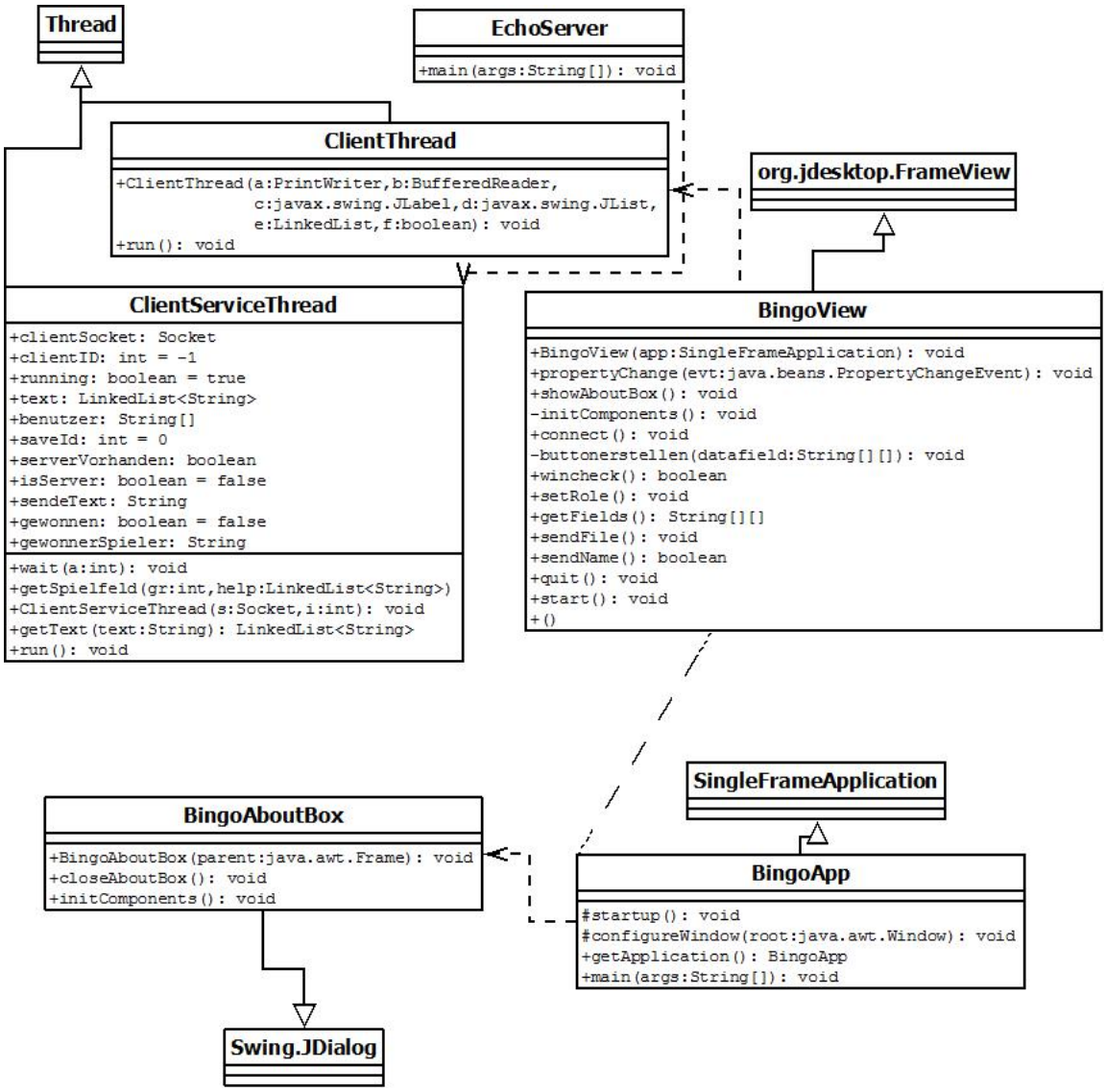
Kapitel 3

Design

3.1 Designentscheidungen

Zu Beginn hatten wir zuviele Klassen dadurch wurde die Implementierung sehr schwierig, deswegen mussten wir das komplette Design noch einmal refactorn und haben uns auf eine Klasse BingoView beschränkt, die so ziemlich die ganze Spiellogik beinhaltet und natürlich zwei Thread Klassen um sich um die Parallelität der Anfragen und Antworten des Clients und Servers zu kümmern.

3.2 Klassendiagramm



Kapitel 4

Implementierung

4.1 Zielanforderungen

Unsere Anforderungen an das System waren unter anderem:

- mehrere Spieler können gleichzeitig an Spielsessions teilnehmen.
- einzelne Spielsessions sind beitretbar
- jeder Spieler hat eine individuelle Textdatei
- Das Spiel BuzzwordBingo funktioniert nach den bekannten Spielregeln
- Thread-Sicherheit (Anfragen und Antworten quasi parallel und in richtiger Reihenfolge Ausführbar)

4.2 Toolumgebung

4.2.1 Entwicklungsumgebung

Eclipse Helios IDE for Java Developers: Version 3.6.2 / NetBeans IDE Version 9.6.1

4.2.2 Hilftools

SVN bei Google- Projekt, Dropbox zum Austausch mancher Dokumente

4.3 Funktionsweise

Das Prinzip ist recht simpel: Der Client schickt eine Anfrage an den Server mit einer Textdatei. Ist der Name bereits vorhanden, wird die Anfrage verweigert. Mit zwei verschiedenen Threads (Anfrage und Antwort) wird sichergestellt, dass die Synchronisation von Client und Server einwandfrei funktioniert.

4.4 Probleme

Folgende Probleme sind uns während der Implementierung aufgekommen:

1. Zuerst war es schwer Möglich das Design umzusetzen, da wir zuerst viel zu viele Klassen eingesetzt haben.
2. Die GUI selbst zu coden wurden immer unübersichlicher und umständlicher.
3. Der Teilcode der in Eclipse für die GUI erstellt wurde, bewirkte bei der Übertragung zu NetBeans, dass die Darstellung nach dem kompilieren eine völlig andere war.
4. Synchronisation von Client und Server

4.5 Lösungen

1. Refactoring des Designs: Weniger Klassen
2. Während wir anfangs noch die GUI mit Swing und AWT selbst gecodet haben, wurden wir uns schnell einig, dass wir Matisse[1] verwenden.
3. GUI in NetBeans angepasst, einzelne codesnippets erneuern
4. Befehle wurde mit dem Server abgestimmt

[1] Oberflächendesignkomponente von NetBeans

Kapitel 5

Quellenverzeichnis

- http://de.wikipedia.org/wiki/Java-Applet
- http://de.wikipedia.org/wiki/Client-Server-Modell
- http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Client-Server-Architektur