

Peer-to-Peer Netzwerke

Vorstellung von Peer-to-Peer-Modellen und Entwicklung eines P2P-Marktes

Maturaarbeit Severin Hacker, Kantonsschule Zug,
mit Rolf Peterhans (betreuende Lehrperson)
2002/2003

Inhaltsverzeichnis

1. Einleitung	3
2. Einführung	4
2.1 Was heisst Peer-to-Peer?.....	4
2.2 Was macht Peer-to-Peer interessant?.....	4
2.3 Verschiedene Netzwerkmodelle	5
Client / Server.....	5
Napster	5
Gnutella	6
Andere	7
3. Analyse und Entwicklung	7
3.1 Idee eines Peer-to-Peer-Marktes.....	7
3.2 Netzwerkmodell	8
3.3 Implementation	9
Packages.....	9
Der Ladevorgang.....	9
Anmeldung beim Hauptserver.....	11
Abruf der Datenserverliste	12
Objektupload auf den Datenserver	13
Objekterstellung	14
Suche	16
Up-/Download.....	17
Beenden des Programms	21
Schlusswort	22
Quellenverzeichnis	23
Lehrbücher für Java	23
Texte aus dem Internet	23
Glossar	24
Anhang	25
UML-Diagramme.....	25
Quellcode	25
ch.calderon.objects Package	26
ch.calderon.server Package.....	28
ch.calderon.thegathering Package.....	30
ch.calderon.visualization Package.....	76

1. Einleitung

Das Ziel dieser Maturaarbeit ist es, einen Peer-to-Peer-Markt zu entwerfen und ihn zu implementieren. Ich werde erklären, was ich unter einem Peer-to-Peer-Markt verstehe, wie ich auf diese Idee kam und wie ein solcher implementiert werden kann

Um einen solchen Markt zu implementieren, brauchte ich zunächst vertieftes technisches Wissen. Nach einer Analyse der Bedürfnisse an die Programmiersprache (vollständig objektorientiert, starke Netzwerk-API's, einfache Kommunikation mit Datenbanksystemen, „gleichzeitiges“ Ausführen von Prozessen) entschied ich mich für Java. In der ersten Phase eignete ich mir Wissen über Netzwerkprogrammierung, Serialisierung, Datenbankanbindungen mit JDBC, Java-Threads zum „gleichzeitigen“ Ausführen von Prozessen, GUI's mit Swing, I/O-Schnittstellen, sowie einige fortgeschrittene objektorientierte Techniken an (innere Klassen, Design-Patterns). Des Weiteren hätte man auch noch weitere, abstraktere Techniken wie RMI beziehungsweise J2EE lernen können, dieser Aufwand stünde dann aber in einem schlechten Verhältnis zum Ertrag. Ausserdem kann der ganze Peer-to-Peer-Part des Projekts nicht über RMI abgewickelt werden, weil dazu jeder Peer einen RMI-Server starten müsste.

Während den Sommerferien 2002 schrieb ich die erste Beta-Version des Programms. Diese Version hatte bereits alle grundlegenden Funktionen eines Peer-to-Peer-Marktes, allerdings hatte sie noch relativ viele Fehler. Diese begann ich ab den Sommerferien konsequent auszumerzen. Ausserdem kommentierte ich alle Klassen und führte kleinere Design-Änderungen durch.

Das Projekt wurde nicht mit dem Ziel implementiert sicher zu sein. Auch wenn ich gewisse Sicherheitsfunktionen eingebaut habe (z. B. keine direkte Kommunikation mit Users-Tabelle auf Hauptserver), wäre es wohl ohne allzu grossen Aufwand möglich das System zu knacken und Änderungen an heiklen Daten (z. B. Passwörtern) vorzunehmen.

Der erste Teil meiner Arbeit befasst sich mit einer Vorstellung der bisherigen Peer-to-Peer-Modelle, der zweite Teil mit meiner Eigenentwicklung. Dies entspricht auch der Reihenfolge meines Vorgehens. Zuerst musste ich schauen, wie andere Peer-to-Peer-Netzwerke (insbesondere Napster) funktionieren um dann die Stärken und Schwächen zu erkennen. Meine Eigenentwicklung versucht eine der Schwächen zu beheben, nämlich die mangelnde Bereitschaft der Benutzer eigene Dateien mit anderen Benutzern zu teilen.

Ab Kapitel 3.3 verwende ich Problem-Lösungen-Kästen, in denen die Wirkung des Codes kurz und einfach verständlich gezeigt werden soll.

Da das Projekt recht umfangreich ist (mehr als 60 A4-Seiten Code), kann ich nicht auf alle Details eingehen und beschränke mich, wo möglich, auf die netzwerkrelevanten Themen. Auf Seite 23 findet sich ein Glossar mit den wichtigsten Fachwörtern sowie im Anhang der gesamte Source-Code des Projektes.

2. Einführung

2.1 Was heisst Peer-to-Peer?

Peer-to-Peer (kurz: P2P) bedeutet frei übersetzt: Ebenbürtig zu Ebenbürtig. Peer, als Adjektiv bedeutet gleichrangig.

Definition: Als Peer-to-Peer bezeichne ich ein Kommunikationsmodell, bei welchem verschiedene, in einem Netzwerk verbundene Rechner gleichwertig und dezentral Daten austauschen. Die einzelnen Rechner in einem Peer-to-Peer-Netzwerk werden Peers, Nodes oder Servents genannt, da sie immer sowohl Client als auch Server sind.

Einen anderen Weg Peer-to-Peer zu beschreiben schlägt Clay Shirik vor:

„1) Does it treat variable connectivity and temporary network addresses as the norm, and 2) does it give the nodes at the edges of the network significant autonomy? If the answer to both of those questions is yes, the application is P2P. If the answer to either question is no, it's not P2P.“ (Clay Shirik: What Is P2P - And What Isn't)

Das Internet, oder das was die meisten darunter verstehen, nämlich das Abrufen von Informationen über einen Browser benutzt das Client-Server-Modell. Die verschiedenen Modelle werden im Kapitel 2.3 genauer betrachtet. Ausserdem werde ich im weiteren Verlauf dieser Arbeit in erster Linie von Filesharing-Netzwerken sprechen. Dabei gibt es eine Reihe weitere Anwendungsmöglichkeiten; Instant-Messenger wie ICQ, AIM oder auch Systeme zur Umgehung von Internet-Zensur wie Peek-A-Booty.

2.2 Was macht Peer-to-Peer interessant?

Die Eigenschaft, dass die Daten nicht auf zentralen Servern liegen, sondern direkt auf den einzelnen verbundenen Rechnern ist **der** entscheidende Vorteil gegenüber dem Client/Server-Modell. Es ermöglicht nämlich den Aufbau von sogenannten Filesharing („Dateitausch“-)Netzwerken. Über diese Netzwerke können dann alle Arten von Daten getauscht werden, in erster Linie aber Musikdateien. Die Musik liegt dann meist in Form einer Mpeg-Layer-3 (MP3)-Datei auf den Festplatten der einzelnen teilhabenden Rechnern. Durch eine Suchmaschine findet man die anderen Peers und kann von diesen **direkt** (d. h. ohne die Verwendung eines zentralen Servers) die gewünschten Musikdateien herunterladen.

Ein weiterer Vorteil ist, dass der Aufwand, um Daten ins Netz zu stellen (Upload-Vorgang), für den Benutzer ausserordentlich gering ist. In der Regel braucht man nur ein Programm herunterzuladen und kann mit diesem dann die eigenen Dateien tauschen. Bei einem normalen Client/Server-Netzwerk bräuchte man neben einem eigenen Server, eine Standleitung und eine Domain und recht viel technisches Wissen. Die Domain hat allerdings einen gewichtigen Nachteil; bei den Domainregistrationen (in der Schweiz: switch.ch), werden Informationen wie zum Beispiel echter Name und Adresse verlangt. Dies verunmöglicht vollständige **Anonymität**.

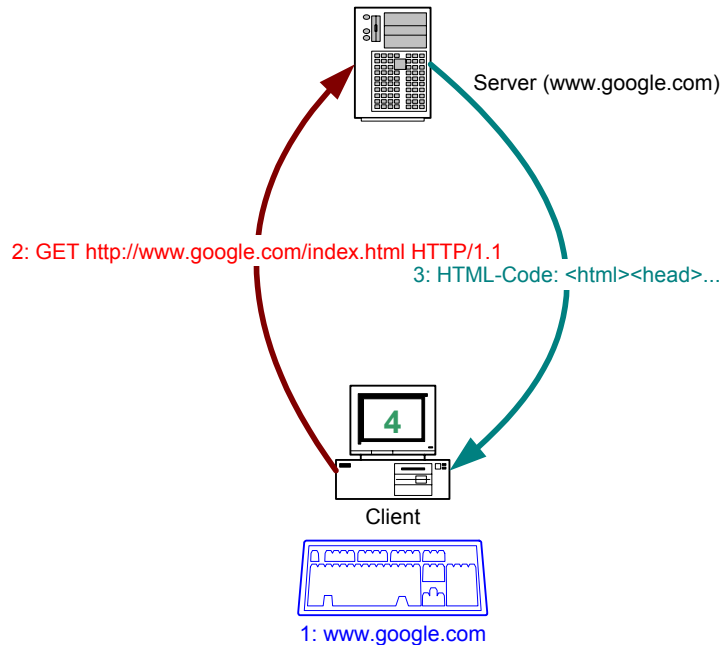
In den Peer-to-Peer-Netzen dagegen herrscht weitgehend die Anonymität, welche das Internet ursprünglich auch mal hatte. Diese Anonymität gewährt erst die für die Benutzer von Filesharing-Netzwerken notwendige Sicherheit, um auch **illegale Daten**, wiederum in erster Linie Musik, zu tauschen. Unter illegalen Daten verstehe ich Daten, welche unrechtmässig kopiert und vervielfältigt werden.

Während das normale Client/Server-Modell viel Leistung und Speicher auf dem Server benötigt, lagert das Peer-to-Peer-Konzept diesen Ressourcen-Hunger auf die immer schneller werdenden Heimcomputer aus. Kazaa zum Beispiel hat an einem Stichtag (21.11.02 17:05) 3.2 Millionen Benutzer weltweit, welche 4.6 Petabytes Daten (entspricht rund 7 Millionen CD's) zur Verfügung stellen.

2.3 Verschiedene Netzwerkmodelle

Client / Server

Client-Server ist das Standard-Netzwerkmodell des Internets. Es ist das Gegenteil des P2P-Prinzips. Es wird zum Beispiel beim Benutzen eines Internet-Browsers verwendet.



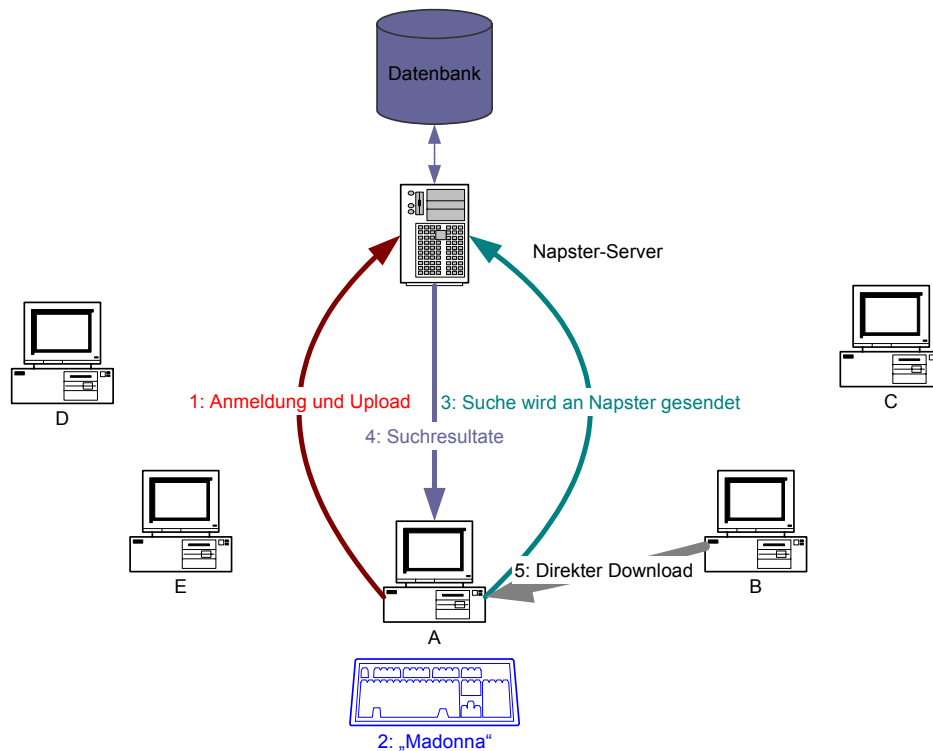
Übliches Modell beim Benutzen eines Internet-Browsers (vereinfacht)

1. Der Benutzer tippt eine URL ein (z. B. www.google.ch)
2. Der Client sendet dem Server seinen Request (GET <http://www.google.com/index.html> HTTP/1.1)
3. Der Server sendet die angeforderte Seite zurück (HTML-Code)
4. Der Browser liest die HTML-Daten und stellt sie auf dem Bildschirm dar

Napster

Die Grundlage des folgenden Abschnitts ist eine Packetanalyse des Stanford-Studenten David Weekly.

Die Vorteile der P2P-Technologie wurden zunächst von Shawn Fanning, dem Gründer von Napster entdeckt. Napster verwendet ein recht einfaches P2P-Netzwerk, sowohl die Verbindung als auch die Suche läuft zentral über einen Server von Napster. Nur der eigentliche Tausch der Dateien läuft peer-to-peer.



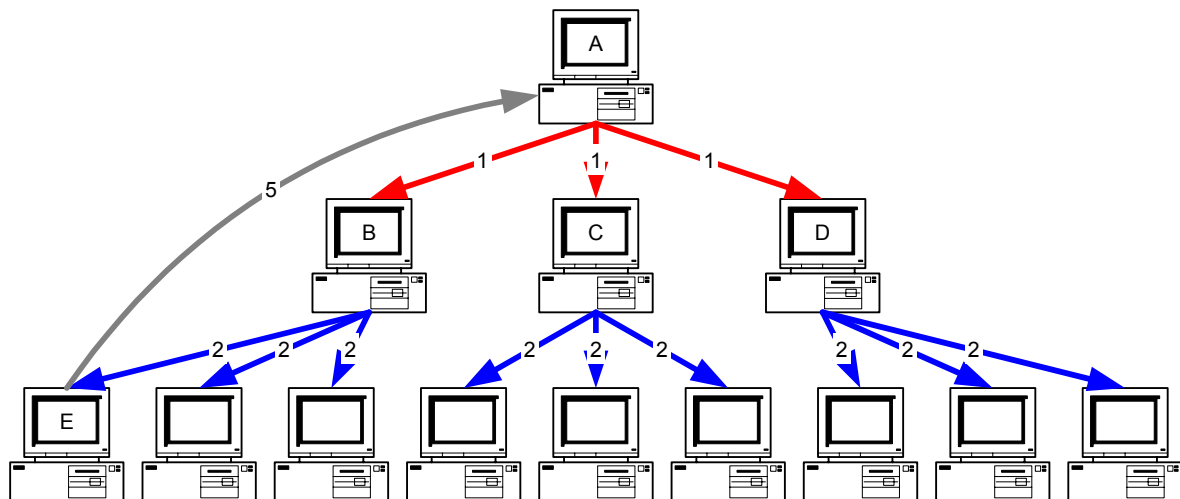
Ein übliches Anwendungsmuster:

1. Benutzer A meldet sich beim Napster Server an, Upload der Beschreibungen der eigenen Dateien (z.B. „Enigma – Push the Limits“, etc.):
2. Benutzer A tippt ein Suchwort ein (z. B. „Madonna“)
3. Suche wird an Napster-Datenbank geschickt
4. Napster-Datenbank sendet die Suchresultate an A, darin ist ein Eintrag von B.
5. Benutzer wählt den Eintrag von B aus und lädt die Datei von B direkt herunter.

Das Napster-Prinzip erfreute sich lange grosser Beliebtheit, da das Programm einfach zu bedienen und ressourcenschonend war. Napster war während Monaten die meistheruntergeladene Anwendung der Welt.

Gnutella

Napster wurde im Jahre 2000 von verschiedenen Musikvertriebsfirmen in den USA verklagt. Der Gerichtsentscheid führte schliesslich zur Abschaltung des Napster-Servers und damit des ersten P2P-Filesharing-Netzwerkes. Die Schwäche, dass es einen zentralen Server gab, und es ein Leichtes war diesen ausser Betrieb zu nehmen, führte zum ersten völlig dezentralen P2P-Netzwerk, dem Gnutella-Netzwerk. Entwickelt wurde es ursprünglich von Nullsoft, einer späteren Tochter von AOL. Später wurde die Entwicklung bei AOL eingestellt, doch der Quellcode hatte sich bereits verbreitet. Aufgrund dieser Basis entwickelte dann Gene Kan die Idee weiter zum heutigen Gnutella-Netzwerk. Das Gnutella-Netzwerk ist im Vergleich zu Napster relativ komplex. Die Grundlage folgender Ausführungen sind die „Gnutella Protocol Specifications 0.4“. Das Protokoll behandelt nicht die Suche nach dem ersten Peer im Netzwerk, dies hängt von den verschiedenen Implementationen des Protokolls ab.



Der Suchprozess:

1. A sendet eine Suchanfrage an seine bekannte Peers B, C, D (1. Schicht)
2. B, C, D schicken die Suchanfrage an ihre bekannten Peers weiter (2. Schicht)
3. die Peers schicken die Suchanfrage jeweils an ihre bekannten Peers weiter, solange das TTL (Time To Life) ungleich 0 ist (n-te Schicht)
4. ein Peer E der x-ten hat die entsprechende Datei und sendet die Antwort über den gleichen Weg zurück wie er die Anfrage erhalten hat
5. A baut eine direkte Verbindung zu E auf, und lädt die Datei herunter.

Die beliebtesten Gnutella-Programme sind LimeWire und BearShare. Aufgrund der dezentralen Netzwerkstruktur werden dem Gnutella-Netzwerk die besten Erfolgsaussichten – gerade bezüglich Gerichtsverfahren – gemacht. Es ist beinahe unmöglich das Netzwerk mit heutigen technischen Mitteln abzuschalten.

Andere

Daneben gab es immer auch andere Entwicklungen. Zunächst gab es die FastTrack-Technologie des gleichnamigen niederländischen Unternehmens. Kazaa, das heute mit Abstand beliebteste Filesharing-Programm basiert ebenfalls auf der FastTrack-Technologie. Allerdings sehe ich die Zukunft dieses Netzwerks nicht so rosig, da es zu wenig dezentralisiert ist. Die Beliebtheit ist ein weiterer Nachteil, da die Musikindustrie zuerst gegen die beliebtesten Programme vorgehen wird.

Eine relativ neue Entwicklung ist eDonkey des Amerikaners Ted McCaleb. Das eDonkey-Netzwerk ist ähnlich wie Napster aufgebaut, allerdings dezentraler (es werden Listen mit Datenservern verwendet). Neu bei eDonkey ist, dass die einzelnen Dateien in kleinere Teile aufgesplittet werden. Diese werden dann heruntergeladen und sobald die ersten Daten ankommen, wieder freigegeben, was zu einer rasanten Verbreitung von stark nachgefragten Dateien führt. Der Nutzen ist besonders bei grossen Dateien gross, insbesondere bei Filmen.

3. Analyse und Entwicklung

3.1 Idee eines Peer-to-Peer-Marktes

Alle damaligen Peer-to-Peer-Modelle hatten einen entscheidenden Nachteil. Es war möglich die eigenen Dateien nicht mit den anderen zu tauschen, sondern nur deren Dateien herunterzuladen. In der Regel genügt es eine Einstellung in den Optionen zu ändern, um von diesem Vorteil zu profitieren. Denn jeder Upload schmälert die Bandbreite für das eigene Surfen. Liess man den Peer-to-Peer-Programmen freie Hand, war es schliesslich nicht mehr möglich parallel das Internet zu nutzen, vor allem bei schwachen Internet-Verbindungen über Modem. Und jeder, der dies wusste, änderte die

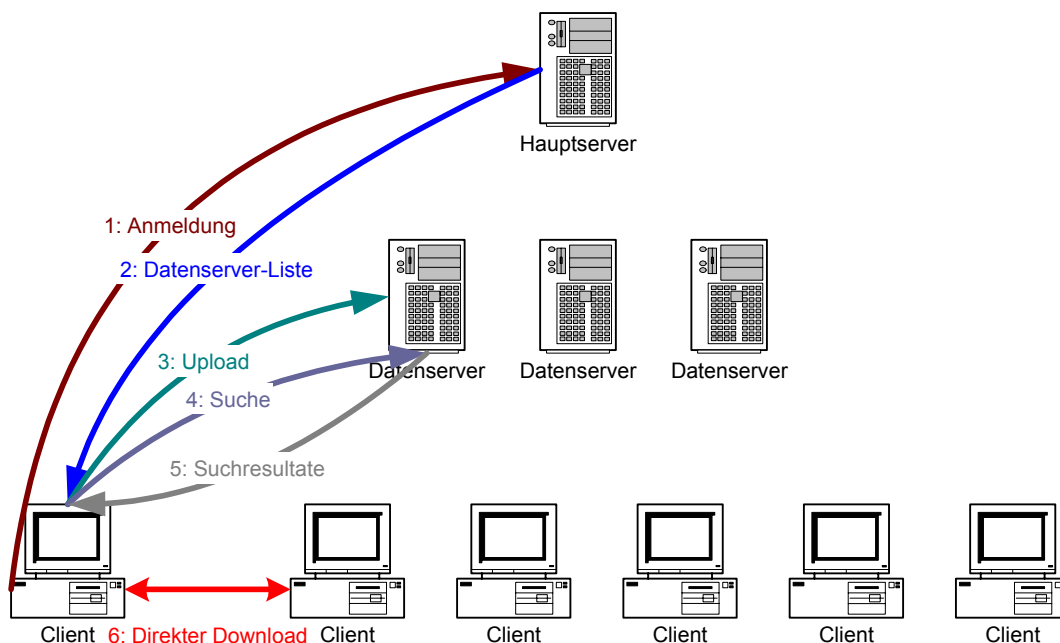
Einstellung in den Optionen. Das heisst, es konnte eigentlich nicht mehr von einem Tauschnetzwerk gesprochen werden.

Dazu ein Vergleich mit einem üblichen Sammlermarkt. Am Anfang, solange der Markt klein ist, können die Sammelobjekte normal getauscht werden, Sammelobjekt gegen Sammelobjekt. Es fällt sofort auf, wenn ein Sammler nur nimmt und selten oder gar nie gibt. Wächst dieser Markt aber stetig, so dass er schliesslich von einer anonymen Masse übernommen wird, deren Anhänger sich untereinander nicht kennen, wächst die Versuchung nur zu nehmen und nicht zu geben. Schliesslich werden nur noch die neuen Sammler, die nicht wissen, dass sie Ihre Objekte auch verbergen können, Objekte freigeben. Man darf den andern Sammlern aber keinen Vorwurf machen; sie nützen die Schwäche des Systems einfach aus. Was aus dem Dilemma führt, ist die Einführung eines Punktesystems. Wer viel gibt soll auch viel nehmen können. Jetzt besteht noch das Problem, dass die Sammelobjekte oft nicht den gleichen Wert haben. Deshalb muss ein Transaktionsmittel, das als Recheneinheit und Wertmassstab gelten kann, geschaffen werden: Geld.

Was bisher allen Peer-to-Peer-Netzwerken fehlt ist ein Bonussystem. Es gibt zwar Ansätze ein solches einzuführen (z. B. Up-, Downloadratio 1:4 bei eDonkey), keiner geht aber genügend weit. Für eine maximale Tauschrate ist es nötig Geld und eine Währung einzuführen. Das ist die Idee eines **Peer-to-Peer-Marktes**.

3.2 Netzwerkmodell

Mein Projekt verwendet ein abgeändertes Napster-Netzwerkmodell.

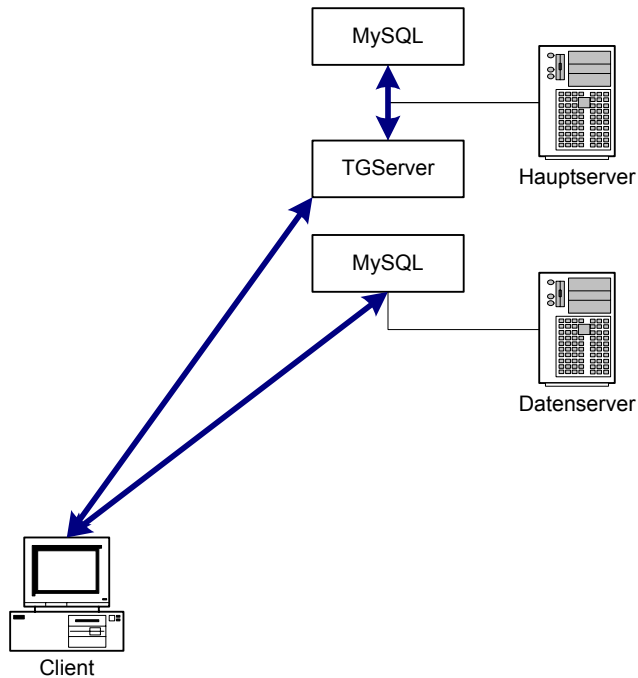


Übliches Anwendungsmuster:

1. Anmeldung beim Hauptserver (mou.ksz.ch)
2. Hauptserver sendet Liste mit aktuellen Datenserver zurück
3. Upload der eigenen Objektbeschreibungen auf einen beliebigen, auswählbaren Datenserver
4. Suche auf einem beliebigen, auswählbaren Datenserver
5. Datenserver sendet Suchergebnisse
6. Direkter Download, Peer-to-Peer

Auf dem Hauptserver (mou.ksz.ch) läuft im Hintergrund eine MySQL-Datenbank. Diese darf aber aus Sicherheitsgründen nicht direkt über das Internet angefragt werden. Deshalb gibt es eine zweite Schicht, den TGServer. Dies ist ein einfaches Java-Programm, welches die Anfragen der Clients an die Datenbank weiterleitet, und die Antworten den Clients zurückschickt. Beim Datenserver spielt

Sicherheit allerdings eine kleinere Rolle, weshalb man direkt auf die Datenbank über den JDBC-Treiber für MySQL zugreift.



3.3 Implementation

Packages

Das Projekt umfasst insgesamt 4 Packages:

- ch.calderon.objects.*** : Die Handelsobjekte (Klassen EconomicGood und Erben) und die Handelspartner (Klasse Peer)
- ch.calderon.thegathering.*** : Das Hauptpackage mit der ganzen GUI- und Netzwerkfunktionalität und der Main-Klasse TheGathering
- ch.calderon.server.*** : Beinhaltet nur die Klasse TGServer, die Schnittstelle zwischen Client und Datenbank auf dem Hauptserver.
- ch.calderon.visualization.*** : Beinhaltet nur die Klasse ObjectVisualizer, zum dynamischen Darstellen und Ändern eines Objektes.

Der Implementationsteil wird in verschiedene Teilprobleme unterteilt werden. Jeweils am Anfang des Teilproblems steht die Problemstellung und eine kurze Zusammenfassung der Lösung, die dann detaillierter mit Code erläutert wird.

Der Ladevorgang

Problem:	Das Programm muss gestartet werden, dann soll es dem Benutzer eine ansprechende graphische Benutzerschnittstelle zur Verfügung stellen. Die Wartezeit soll über einen Ladebalken dargestellt werden. Die Handelsobjekte der letzten Session sollen wieder geladen werden.
Lösung:	Verwendung von Java WebStart zum Starten, Benutzung der Swing-API für die GUI, Darstellung eines SplashScreen mit einer JProgressBar, Deserialisierung der Handelsobjekte mit ObjectInputStream aus einer Datei.

Das Programm wird üblicherweise über einen Java WebStart-Link gestartet. Für das Ausführen ist die Java Runtime Environment (JRE) notwendig. Als erstes wird die Main-Methode der `ch.calderon.thegathering.TheGathering` ausgeführt. Diese macht zunächst eine neue SplashScreen-Instanz `sp` (privates Feld) durch den Aufruf des Konstruktors von SplashScreen. Diese Klasse wurde

ursprünglich von Guido Krüger in dem Buch „Handbuch der Java-Programmierung“ verwendet. Ich habe sie leicht abgeändert und dem Projekt beigelegt.

(in TheGathering.java)

```
sp =new SplashScreen("icons/logo.gif", "Loading...");
```

Danach wird in main() die erste und einzige Instanz von TheGathering erzeugt:

```
new TheGathering();
```

Dies ruft den Konstruktor TheGathering() auf. Dort wird als erstes die Methode deserialize() aufgerufen. Sie liest aus der economicGoods.dat-Datei mittels der Java-Objektserialisierung die eigenen Handelsobjekte aus, die beim letzten Starten oder vorher erstellt wurden und speichert sie in den öffentlichen Feldern economicGoods und sellGoods.

(in TheGathering.java)

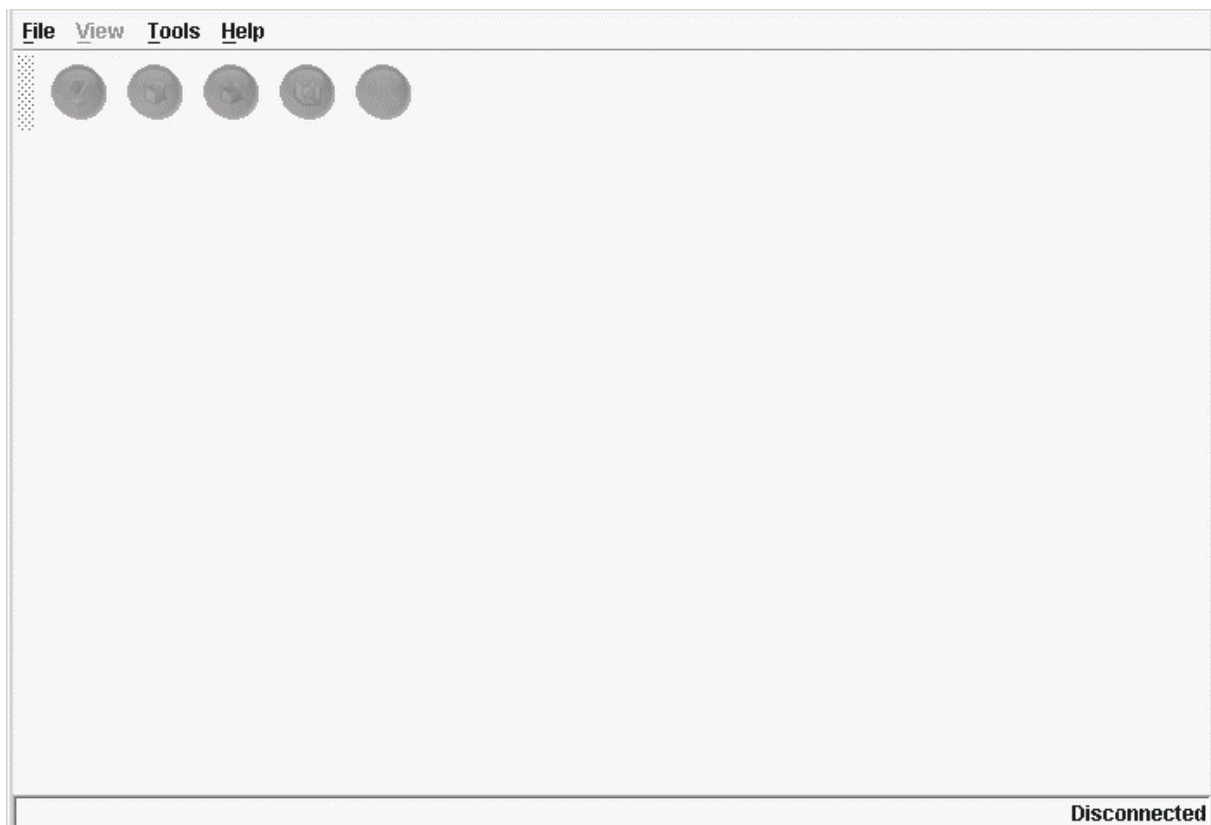
```
FileInputStream fis = new FileInputStream("economicGoods.dat");  
ObjectInputStream ois = new ObjectInputStream(fis);  
economicGoods = (ArrayList) ois.readObject();  
sellGoods = (ArrayList) ois.readObject();
```

Es folgen die privaten Methoden zur Erstellung der GUI:

(in TheGathering.java)

```
createMenus();  
createToolbar();  
createPanels();  
createStatusbar();
```

Nach dem Konstruktor sieht die Oberfläche so aus:



Die Users-Tabelle auf dem Hauptserver sieht, zum Beispiel, so aus:

	NAME	FNAME	ADDRESS	ZIP	CITY	NATION	IP	USERNAME	PASSWORD	MONEY
1	Hacker	Severin	Ringstrasse 6	6300	Zug	CH	offline	seve	passwort	1446.00

Anmeldung beim Hauptserver

Problem: Auf dem Hauptserver muss die eigene IP-Adresse eingetragen werden. Mit der IP-Adresse können dann spätere Käufer von unseren Objekten eine direkte Verbindung aufnehmen. Bei der Anmeldung muss der Benutzername und das Passwort überprüft werden. Der eigene Kontostand soll gezeigt werden.

Lösung: Die IP-Adresse wird mit Hilfe der Klasse UsersTable, welche indirekt über die Klasse Server und TGServer auf die Users-Tabelle auf dem Hauptserver zugreift eingetragen, das heisst der Wert in der Tabelle ändert von ‚offline‘ auf z.B. 212.4.74.64. Mit der Klasse UsersTable wird ausserdem das eigene Peer-Objekt heruntergeladen dann wird die Eingabe des Benutzernamens und Passwortes mit dessen Daten verglichen. Der Kontostand wird aus dem eigenen Peer-Objekt herausgeholt und unten links angezeigt.

Der Benutzer wählt dazu aus dem File-Menü die Option „Connect“. Dies führt zur Ausführung des bei der Erstellung des Connect-Buttons mitgegebenen anonymen ActionListeners:

(in TheGathering.java)

```
connectM.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        LoginDialog ls = new LoginDialog(tg);
        ls.setLocation(200, 200);
    }
});
```

Der ActionListener instanziiert den LoginDialog. Der LoginDialog enthält ein Username- und ein Password-Eingabefeld. Nach dem Eingeben eines Usernames und Passwortes (z.B. seve/passwort) und Drücken auf Ok im Dialog wird dann die Methode login() der Klasse LoginDialog aufgerufen.

(in LoginDialog.java)

```
private void login() throws LoginException {
    ...
    pe = UsersTable.getUsersTable().getPeer(usernameT.getText());
    ...
}
```

Dieser Aufruf verbindet mit der Users-Tabelle auf dem Hauptserver, aber wie bereits erwähnt nicht direkt.

(in UsersTable.java)

```
public Peer getPeer(String username) throws UserNotFoundException {
    Peer pe = new Peer();

    String sql = "SELECT * FROM USERS WHERE USERNAME='" + username + "'";
    ArrayList al = mou.sendQuery(sql);
    ...
    Object[] oa = (Object[])al.get(0);
    pe.name = (String)oa[0];
    ...
}
```

Diese Methode gibt zu dem vorhandenen Username das Peer-Objekt aus der Users-Tabelle zurück, wo alle Informationen gespeichert sind (wie Adresse, Ort, Passwort, etc.). Sie verwendet das Objekt „mou“ des Typs ch.calderon.thegathering.Server (nicht zu verwechseln mit ch.calderon.server.TGServer!). Dieses Objekt verbindet mit dem TGServer auf dem Hauptserver über den Port 14999.

(in Server.java)

```
public ArrayList sendQuery(String sql) {
```

```

ArrayList al = null;
try {
    Socket s = new Socket("mou.ksz.ch", 14999);
    ObjectOutputStream oos =
        new ObjectOutputStream(s.getOutputStream());
    oos.writeObject(sql);

    ObjectInputStream ios = new
    ObjectInputStream(s.getInputStream());
    al = (ArrayList) ios.readObject();
    ...
}

```

Das sendet den SQL-String wieder über Java-Objektserialisierung an den TGServer. Der führt die Anfrage an die MySQL-Datenbank weiter und schickt die Resultate als `java.util.ArrayList` zurück. Dann geht's zurück nach `getPeer()` und zurück nach `login()`. Dort wird überprüft, ob das aus der Datenbank stammende Passwort mit der Eingabe im Eingabefeld übereinstimmt. Falls dies so ist, wird in der Users-Tabelle auf dem Server der Eintrag IP von „offline“ auf die eigene IP-Adresse geändert:

(in `LoginDialog.java`)

```

pe.ip = Options.getOptions().publicIP;
UsersTable.getUsersTable().updateIP(pe);

```

Dieser letzte Befehl führt wieder zur gleichen Kette wie oben, über Server, TGServer, MySQL-Datenbank, TGServer, Server und UsersTable.

Danach sieht die Users-Tabelle auf dem Hauptserver so aus:

	NAME	FNAME	ADDRESS	ZIP	CITY	NATION	IP	USERNAME	PASSWORD	MONEY
1	Hacker	Severin	Ringstrasse 6	6300	Zug	CH	212.4.81.80	seve	passwort	1446.00

Abruf der Datenserverliste

Problem: Wir müssen ab jetzt auf einkommende Nachrichten hören, also auf Downloadanforderungen von anderen Benutzern. Um überhaupt die eigenen Objekte für andere Benutzer sichtbar zu machen, müssen sie auf einen Datenserver geladen werden. Zuerst müssen wir also eine Liste aller Datenserver haben. Die Liste soll in einer Tabelle graphisch dargestellt werden.

Lösung: Der `MessageReceiver` wird gestartet. Die Datenserverliste wird über `DataServerTable` als `ResultSet` abgerufen, in ein `TableModel` konvertiert und angezeigt.

Nach der `login()`-Methode des `LoginDialog` folgt folgender Aufruf:

(in `LoginDialog.java`)

```
tg.loggedIn();
```

Diese Methode der Klasse `TheGathering` macht 3 Dinge. Als erstes wird ein `MessageReceiver` gestartet, der ab sofort auf dem Port 60784 auf einkommende Nachrichten wartet. Er läuft als eigenständiger Thread im Hintergrund:

(in `TheGathering.java`)

```
mr = new MessageReceiver(tg);
```

Als nächstes füllt er die Start-Tabelle auf dem Bildschirm mit den vorhandenen Datenservern, über den Aufruf:

(in `TheGathering.java`)

```
stp.fillPanel();
```

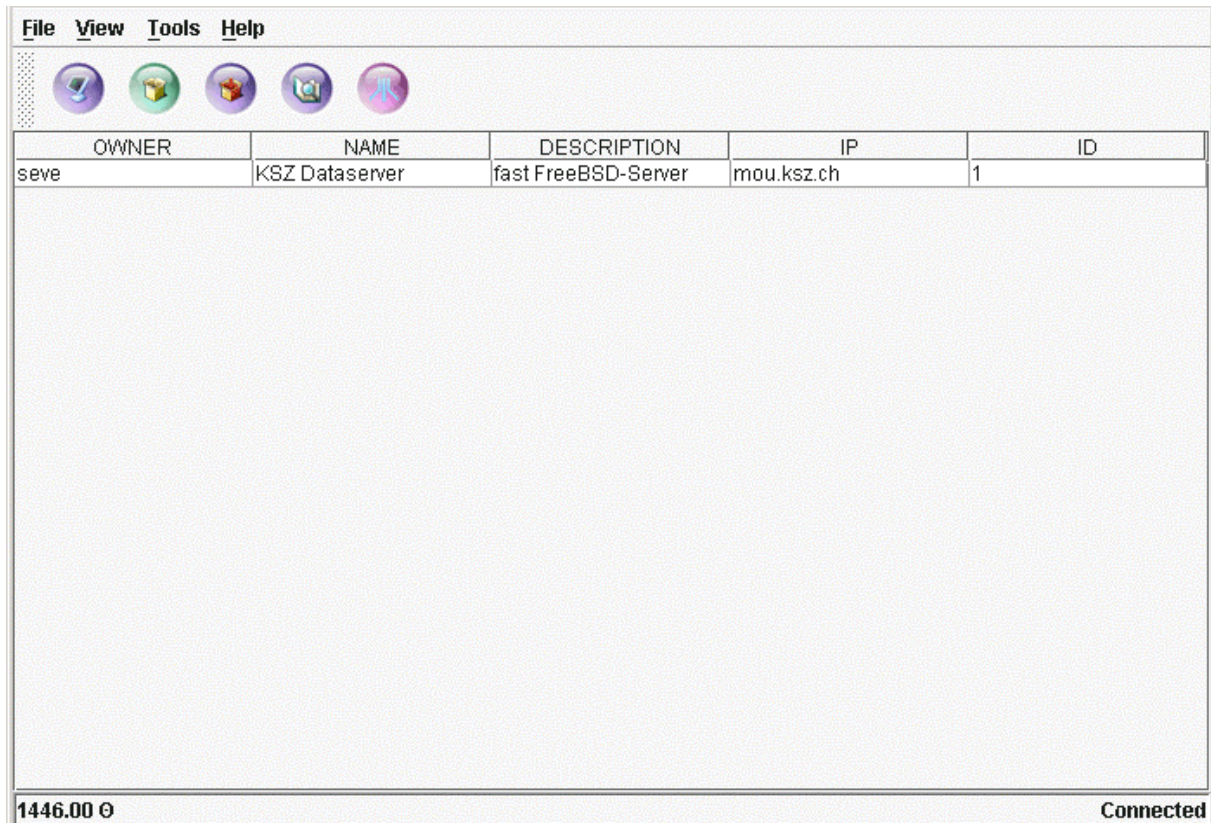
Die Methode `fillPanel()` der Klasse `StartPanel` baut dann über die Klasse `DataServerTable` eine eigenständige direkte JDBC-Verbindung mit dem Hauptserver auf, fragt die Datenserver-Tabelle ab

und schreibt durch die Konvertierung des entstehenden ResultSet mit der Hilfsklasse ResultSetTableModel die Ergebnisse in die Tabelle:

(in StartPanel.java)

```
ResultSet rs = new DataServerTable().getDataServersRS();  
table.setModel(new ResultSetTableModel(rs));
```

Die Oberfläche sieht schliesslich so aus:



The screenshot shows a Java Swing window with a menu bar (File, View, Tools, Help) and a toolbar with five icons. The main area contains a table with the following data:

OWNER	NAME	DESCRIPTION	IP	ID
seve	KSZ Dataserver	fast FreeBSD-Server	mou.ksz.ch	1

At the bottom left of the window, the text "1446.00 Θ" is displayed, and at the bottom right, the text "Connected" is displayed.

Die Tabelle zeigt den wegen des Entwicklungsstadiums des Projektes, einzigen Datenserver mou.ksz.ch. Das ist gleichzeitig auch der Hauptserver. Die Zahl unten links gibt den Kontostand an. Das Zeichen dahinter ist ein griechisches Theta, es dient als Währungszeichen. Ausserdem wurde die Toolbar und das Menü freigeschaltet.

Objektupload auf den Datenserver

Problem: Der Benutzer soll seinen Datenserver auswählen und die Objekte, die zum Verkauf stehen sollen dort eingeschrieben werden.
Lösung: Der Klasse SearchAndUpload wird der ausgewählte Datenserver übergeben, danach werden die Objekte mit dieser Klasse hinaufgeladen.

Als drittes lädt die Methode loggedIn() nun alle Objekte, die zum Verkauf stehen, auf den ausgewählten Datenserver.

(in TheGathering.java)

```
sau = new SearchAndUpload(tg, (String) stp.table.getValueAt(0, 3));  
sau.upload(sellGoods);
```

Für die gesamte Kommunikation Peer-Datenserver ist die Klasse SearchAndUpload verantwortlich. Deren Methode upload löscht zuerst alle eigenen Objekte, damit es nicht zu Überschneidungen

kommen kann. Dann holt sie aus jedem Objekt den Typ (MP3File oder OggVorbisFile). Für jeden Typ liegt auf jedem Datenserver eine Tabelle.

(in SearchAndUpload.java)

```
public void upload(ArrayList al) {
    //eine andere Methode von SearchAndUpload, um alle eigenen //Objekte
    //zunächst zu löschen
    clear();
    ...
    // jedes Objekt der Liste wird ausgewählt
    Iterator it = al.iterator();
    while (it.hasNext()) {
        Object o = it.next();
        //der Klassenname = Tabellenname geholt
        String className =
            o.getClass().getName().replaceAll(
                o.getClass().getPackage().getName() + ".",
                "");

        //alle öffentlichen Felder (wie artist, songTitle,etc.) //geholt
        Field[] fa = o.getClass().getFields();
        try {
            ResultSet rs = st.executeQuery("SELECT * FROM " + className);
            rs.moveToInsertRow();
            //und jedes Feld in der Tabelle mit dem Wert
            //gesetzt mittels Reflection
            for (int i = 0; i < fa.length; i++) {
                rs.updateObject(fa[i].getName(), fa[i].get(o));
            }
            rs.insertRow();
            ...
        }
    }
}
```

Objekterstellung

Problem:	Die Handelsobjekte sollen graphisch erstellt, editiert und verkauft werden können. Jedes Objekt soll eine eindeutige ID haben. Bei Musikdateien sollen Künstlername, Titel etc. eingegeben werden können. Der Preis wird festgelegt.
Lösung:	Die graphische Darstellung übernimmt die Klasse ObjectVisualizer. Die Erstellung läuft über die Klasse ObjectsPanel. Die ID ist gleich der Anzahl Millisekunden seit 1970. Die Musikdateien werden durch die Klasse MusicFile repräsentiert. Der Preis steht bereits in der Mutterklasse EconomicGood. Jetzt wird ganz einfach z.B. ein MP3File-Objekt erstellt, (welches auch ein EconomicGood-Objekt ist) und dem ObjectVisualizer übergeben. ObjectsPanel speichert die Daten ab und fügt das Objekt der Objektliste hinzu. Falls es verkauft wird, geht es in eine spezielle Verkaufsliste „sellGoods“.

Mein Projekt erlaubt momentan nur den Handel mit den Musikformaten MPEG-Layer 3 und Ogg Vorbis, dem Open-Source-Complement. Es ist aber theoretisch nicht an Dateien gebunden. Genauso könnten, und das ohne allzu grossen Aufwand, reale Güter wie Bücher oder Bildschirme gehandelt werden. Als nächstes wollen wir aber eine normale MP3-Datei zum Verkauf anbieten und danach selber von uns abkaufen. Dazu drücken wir auf den grünen Knopf mit der Kiste und wählen New Object. Das führt den Listener des Buttons des ObjectPanels aus:

(in ObjectsPanel.java)

```
newObjectB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        cod = new ChooseObjectDialog(new OPListener());
    }
});
```

Es folgt ein kleiner ChooseObjectDialog zum Auswählen des Objekttyps (MP3 oder Ogg Vorbis). Der Dialog führt über einen Callback den OPListener in ChooseObjectDialog aus.

(in ObjectsPanel.java)

```
class OPListener implements ActionListener {
    public void actionPerformed(ActionEvent ae) {
        try {
            Class c =
                Class.forName(
                    "ch.calderon.objects."
                    + (String)
cod.chooserC.getSelectedItem());
            Object o = c.newInstance();
            new ChangeObjectDialog(tg,o);
            cod.hide();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Der OPListener macht zunächst eine neue leere Instanz des gewählten Objekttyps und gibt diese dann dem sichtbaren ChangeObjectDialog weiter. Dieser Dialog kann zu einem völlig beliebigen Objekttyp eine Eingabeschablone machen. Das heisst, die Eingabeschablone wird jedes Mal neu dynamisch erstellt. Die Eingabeschablone hat eine einfache Form: Links die öffentlichen Feldnamen des Objekts, rechts die Eingabefelder. Da dahinter recht viel Technik steckt und um grösstmögliche Wiederverwendbarkeit zu erhalten, wurde dies in der Klasse ObjectVisualizer im Package ch.calderon.visualization.* gebündelt.

(in ChangeObjectDialog.java)

```
cv = new ObjectVisualizer(o);
fieldP = cv.visualize();
```

Der ObjectVisualizer gibt also ein fertiges JPanel mit der Eingabeschablone für das neue Objekt o zurück. Wie dies im Detail funktioniert, möchte ich nicht erklären, da es wenig mit der Peer-to-Peer-Entwicklung zu tun hat.

Der Benutzer gibt dann die Informationen, bestimmt den Verkaufspreis (hier 50.0) und den Pfad der Datei (location).

New Object																							
Type	Owner	Name	Price																				
<div style="border: 1px solid black; padding: 5px;"> <p>Changing ch.calderon.objects.MP3File@4a75bb object X</p> <p>class ch.calderon.objects.MP3File</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 30%;">artist</td> <td><input type="text" value="Ryuichi Sakamoto"/></td> </tr> <tr> <td>songTitle</td> <td><input type="text" value="Merry Christmas Mr. Lawrence"/></td> </tr> <tr> <td>album</td> <td><input type="text"/></td> </tr> <tr> <td>location</td> <td><input type="text" value="c:\Ryuichi Sakamoto - Merry Christmas Mr. Law"/></td> </tr> <tr> <td>size</td> <td><input type="text" value="0"/></td> </tr> <tr> <td>extension</td> <td><input type="text" value="mp3"/></td> </tr> <tr> <td>price</td> <td><input type="text" value="50.0"/></td> </tr> <tr> <td>owner</td> <td><input type="text"/></td> </tr> <tr> <td>name</td> <td><input type="text"/></td> </tr> <tr> <td>id</td> <td><input type="text" value="1036857351133"/></td> </tr> </table> <p style="text-align: center;">Save</p> </div>				artist	<input type="text" value="Ryuichi Sakamoto"/>	songTitle	<input type="text" value="Merry Christmas Mr. Lawrence"/>	album	<input type="text"/>	location	<input type="text" value="c:\Ryuichi Sakamoto - Merry Christmas Mr. Law"/>	size	<input type="text" value="0"/>	extension	<input type="text" value="mp3"/>	price	<input type="text" value="50.0"/>	owner	<input type="text"/>	name	<input type="text"/>	id	<input type="text" value="1036857351133"/>
artist	<input type="text" value="Ryuichi Sakamoto"/>																						
songTitle	<input type="text" value="Merry Christmas Mr. Lawrence"/>																						
album	<input type="text"/>																						
location	<input type="text" value="c:\Ryuichi Sakamoto - Merry Christmas Mr. Law"/>																						
size	<input type="text" value="0"/>																						
extension	<input type="text" value="mp3"/>																						
price	<input type="text" value="50.0"/>																						
owner	<input type="text"/>																						
name	<input type="text"/>																						
id	<input type="text" value="1036857351133"/>																						

Die ID des Objekts ist einfachheitshalber gleich der Anzahl Millisekunden seit dem 1. Januar 1970. Die Wahrscheinlichkeit, dass 2 Objekte in der gleichen Millisekunde erstellt werden ist äusserst gering im Entwicklungsstatus. (Richtigerweise müsste dies aber ein berechneter Hash-Wert der Datei sein.) Nach dem Drücken auf Save werden die Informationen in dem Objekt gespeichert. Zuerst wird überprüft, ob die Datei überhaupt existiert, danach wird die Grösse der Datei bestimmt, der Besitzer gesetzt und das Feld ‚name‘ mit dem Dateinamen (ohne Pfad) besetzt. Das Objekt wird dann in die Liste der eigenen Objekte ‚economicGoods‘ aufgenommen:

(in ChangeObjectDialog, Methode changeObject)

```
tg.economicGoods.add(cv.getObject());
```

Bis jetzt gehört das Objekt nur zur Liste der eigenen Objekte, es wird aber noch nicht verkauft. Dazu muss man in der Tabelle rechtsklicken und auf ‚Sell‘ drücken. Nun wird es in der Liste economicGoods gelöscht und in die Liste sellGoods aufgenommen. Alle Objekte der Liste sellGoods werden nach dem Anmelden, wie bereits gesehen, auf den ausgewählten Datenserver geladen. Man kann dies aber auch manuell nachträglich ausführen. Im Menü wählt man dafür unter Tools ‚Upload‘.

Dann sieht die MP3File-Tabelle auf dem Datenserver so aus:

id	name	owner	price	extension	location	size	album	artist	songTitle
1036857351133	Ryuichi Sakamoto -	seve	50.00	mp3	c:\Ryuichi Sakamot	4593664		Ryuichi Sakamoto	Merry Christmas Mr.

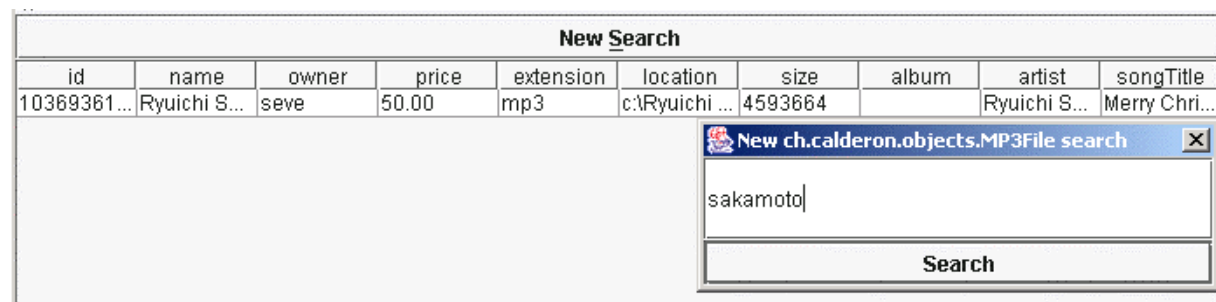
Unsere MP3-Datei kann nun von jedem Nutzer von TheGathering gekauft werden, sofern dieser das nötige Geld hat und auf dem gleichen Datenserver sucht.

Suche

Problem: Der Benutzer soll nach Begriffen wie „madonna“ oder „music“ auf dem ausgewählten Datenserver suchen können. Die Resultate sollen in einer Tabelle dargestellt werden. Mit einem Rechtsklick sollen sie heruntergeladen werden können.

Lösung: Die Suche läuft über die Klasse SearchAndUpload, die gibt ein ResultSet zurück. Dieses wird wieder über ResultSetTableModel in ein TableModel konvertiert und dargestellt. Der Rechtsklick sendet eine Download-Anforderung an den Verkäufer.

Dazu drückt man auf die Lupe in der Toolbar, auf ‚New Search‘ wählt den Dateityp aus und gibt den Suchtext ein. Zum Beispiel ‚lawrence‘, oder ‚sakamoto‘:



Für die Suche sind die Klassen SearchPanel, NewSearchDialog und SearchAndUpload verantwortlich. Ausserdem kann die Klasse ChooseObjectDialog zum Wählen des Dateityps wiederverwendet werden. Nach dem Drücken auf ‚Search‘ wird folgende Routine abgearbeitet:

(in NewSearchDialog.java)

```
searchB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        String name = s.replaceAll("ch.calderon.objects.",
        "");
        ResultSet rs =
            tg.sau.search(
                "SELECT * FROM "
                + name
```

```

        + " WHERE name like '%"
        + sqlT.getText()
        + "%' ";
        ...
*      tg.scp.table.setModel(new ResultSetTableModel(rs));
        ...
    });
}

```

tg.sau ist dabei ein Objekt der Klasse SearchAndUpload. Es wird also in unserem Falle der SQL-Befehl `SELECT * FROM MP3File WHERE name like 'sakamoto'` gesendet. Das heisst, momentan kann nur das Feld ‚name‘ der Tabellen zur Suche verwendet werden. Alle anderen werden ignoriert. Im ‚name‘-Feld befindet sich bei den Dateien aber deren Dateiname, bei den MP3-Dateien also sowohl Künstlername und Titel des Stücks.

Die Methode search von SearchAndUpload ist recht einfach:

(in SearchAndUpload.java)

```

public ResultSet search(String sqlQuery) {
    Statement st = null;
    ...
    st = conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
    ...
    ResultSet rs = null;
    ...
    rs = st.executeQuery(sqlQuery);
    ...
    return rs;
}

```

Der Suchbefehl wird an das JDBC-Statement gedockt und gesendet. Das Resultat im JDBC-ResultSet gespeichert und an den Methodenaufrufer zurückgegeben.

Die Zeile beim * dagegen wandelt das ResultSet wieder mit Hilfe der Klasse ResultSetTableModel in ein Modell für die Tabelle um. Im Tabellenmodell befinden sich dann die Daten, so dass es schliesslich dargestellt werden kann.

Up-/Download

Problem:	Die Datei und das Beschreibungsobjekt sollen heruntergeladen werden können. Eventuell muss das eigene Programm jemand anderem eine von ihm gewünschte Datei verkaufen. Die Kontostände müssen aktualisiert werden. Die Up-/Downloads sollen graphisch in einer Tabelle dargestellt werden.
Lösung:	Zuerst muss eine Download-Anforderung mit einer Nachricht beantwortet werden. In dieser Nachricht steht der Port für den Download drin. Dies übernimmt die Klasse MessageReceiver. Den eigentlichen Up- bzw. Download übernehmen die Klassen UploadThread und DownloadThread. Die Transaktion wird über die Klasse UsersTable und Transaction abgefertigt. Die Darstellung der Transfers übernimmt die Klasse TransferPanel.

In der Tabelle kann nun der Benutzer auf den gewünschten Eintrag rechtsklicken und die Datei bezahlen und herunterladen. Das führt den folgenden Button-Listener aus:

```

downlM.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        ...
        //welche Zeile wurde selektiert?
        int row = table.getSelectedRow();
        //wem gehört das Objekt
        String username = (String) table.getModel().getValueAt(row, 2);
        ...
        //welche ID hat das Objekt
    }
}

```

```

id=Long.parseLong((String)table.getModel().getValueAt(row,0));
//wie gross ist die Datei? (wenn es eine Datei ist)
size =
Integer.parseInt((String)table.getModel().getValueAt(row, 6));

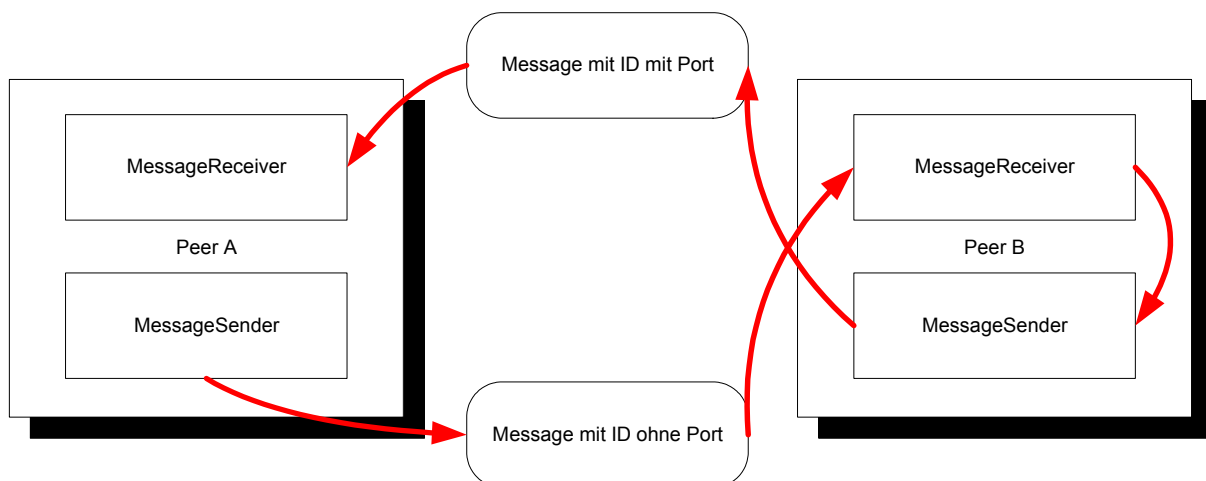
//wie heisst die Datei?
name = (String)table.getModel().getValueAt(row,1);

//und wie teuer ist sie?
price =
Float.parseFloat((String)table.getModel().getValueAt(row,3));
...
Peer receiver = null;
...
//jetzt holen wir die ganze Info über den Verkäufer (vor allem
//brauchen wir seine IP-Adresse) aus der Users-Tabelle auf dem
//Hauptserver
receiver = UsersTable.getUsersTable().getPeer(username);
...
//jetzt starten wir die Geldtransaktion
Transaction t = new Transaction();
t.sender = tg.me;
t.receiver = receiver;
t.amount = price;
...
//dies nimmt dem Käufer das Geld vom Konto, gibt es aber noch
//nicht dem Empfänger. Das heisst, wenn etwas schief geht,
//fehlt das Geld im Markt.
UsersTable.getUsersTable().startTransaction(t);
...
//falls der Käufer zuwenig Geld hat
JOptionPane.showMessageDialog(null, "You don't have enough
money", "Illegal Transaction",JOptionPane.ERROR_MESSAGE);
return;
}
//Es wird ein neues Message-Objekt erstellt mit der Info aus
//der Tabelle
Message m = new Message(id, t.sender,receiver,size,name,price);
//und schliesslich gesendet
new MessageSender().send(tg,m);
...

```

Die Geldtransaktion wollen wir nicht genauer besprechen, dafür die Nachrichtenkommunikation welche nun peer-to-peer abläuft. Das Problem ist, dass wir dem Verkäufer zunächst sagen müssen, welche Datei wir von ihm wünschen, dazu benötigen wir die ID der Datei. Der Verkäufer dagegen muss uns sagen, auf welchem Port er seinen Upload anbieten wird. Schliesslich soll es möglich sein, dass mehrere Up-/Downloads gleichzeitig funktionieren. Jeder Up-/Download wird deswegen über einen eigenen Port abgefertigt.

Das führt zu folgendem Diagramm:



Peer A ist der Käufer, Peer B der Verkäufer.

Der MessageReceiver von B hört auf Port 60784 und bekommt schliesslich die Message von A. Er schaut, ob bereits ein Port drin steht. Falls keiner drin steht, weiss er, dass er der Verkäufer ist. Er öffnet dann einen neuen UploadThread auf dem Port X. Der UploadThread stellt die Datei mit der richtigen ID zur Verfügung. Den Port schreibt er in die Message rein und sendet sie über seinen eigenen MessageSender an Peer A. Jetzt weiss A, dass er auf dem Port X einen DownloadThread starten und somit seine gesuchte Datei herunterladen kann. Die Portnummer X wird bei jeder neuen Instanzierung eines UploadThreads um eins grösser und beginnt bei 15000.

Im Code sieht dies dann so aus:

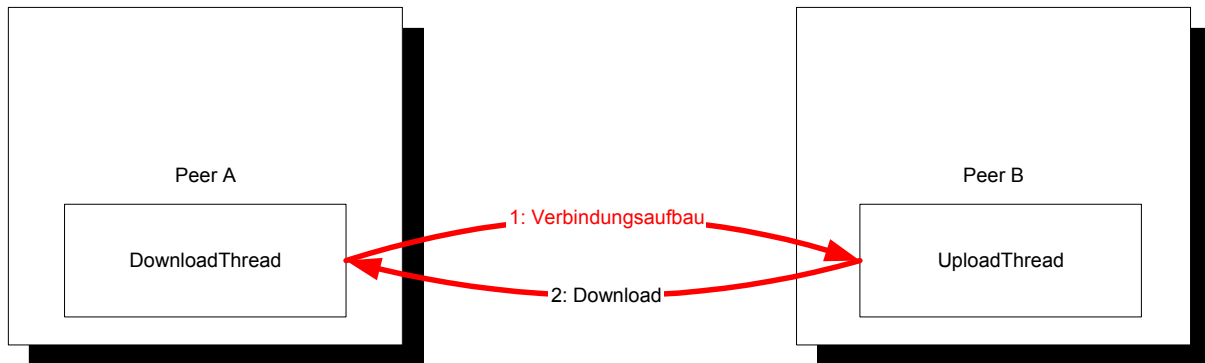
(in MessageSender.java)

```
public void send(TheGathering tg, Message m) {
    tg.log.info("Sending message to :" + m.getReceiver().ip);
    //hole IP des Empfängers
    String ip = m.getReceiver().ip;
    ...
    Socket s = new Socket(ip, PORT);
    oos = new ObjectOutputStream(s.getOutputStream());
    //serialisiere und sende
    oos.writeObject(m);
    oos.flush();
    oos.close();
    s.close();
    ...
}
```

Der MessageReceiver ist ein Thread, der beim Starten des Programms mitgestartet wird. Er läuft also stets parallel zu den anderen Prozessen. Beim Start wird die Methode run() ausgeführt.

```
public void run() {
    while (true) {
        //diese Methode blockt, bis sie einen Anrufer erkennt.
        Socket s = ss.accept();
        //kapsle das ganze in einen OIS
        ois = new ObjectInputStream(s.getInputStream());
        ...
        //und lese das Objekt
        Message m = (Message) ois.readObject();
        ...
        //falls m.getPort() = 0 ist, sind wir Verkäufer, sonst
        //Käufer
        if (m.getPort() == 0) {
            //öffne einen UploadThread
            UploadThread ut = new UploadThread(tg,m);
            ...
            //und in die Liste der UploadThreads aufnehmen
            tg.trp.uAL.add(ut);
            //hier wird der Port gesetzt
            m.setPort(ut.getPort());
            //und "Return to sender!"
            ch.calderon.objects.Peer sender = m.getSender();
            m.setSender(m.getReceiver());
            m.setReceiver(sender);
            new MessageSender().send(tg,m);
        } else {
            //wir sind Käufer, also DownloadThread öffnen
            ...
            DownloadThread dt = new DownloadThread(tg,m);
            //und der Liste hinzufügen
            tg.trp.dAL.add(dt);
        }
    }
    ...
}
```

Der DownloadThread ist, wie der Name sagt, ebenfalls ein Thread, der parallel zu den anderen Prozessen ablaufen muss. Er verbindet mit dem UploadThread des Verkäufers.



Im Code:

(in UploadThread.java)

```

//Konstruktor
public UploadThread(TheGathering tg, Message m) {
    this.tg = tg;
    port++;
    this.m = m;
    ...
    ss = new ServerSocket(port);
    ...
    this.start();
}

//run()
public void run() {
    //blockt bis DownloadThread verbindet
    Socket s = ss.accept();
    ...
    //finde die Datei in der Liste (getObject() ist eine Methode von
    //UploadThread)
    EconomicGood eg = getObject();
    //und sende die Datei
    oos.writeObject(eg);
    ...
}
  
```

(in DownloadThread.java)

```

public void run() {
    ...
    //eine kurze Zeit warten, um zu versichern, dass Sender bereits Port
    //offen hat
    sleep(TheGathering.options.downloadDelayTime);
    //baut die Verbindung zum ServerSocket des Upload-Threads auf
    Socket s = new Socket(m.getSender().ip, m.getPort());
    //kapsle hier meinen CountInputStream, um die Bytes zählen zu können
    cis = new CountInputStream(s.getInputStream());
    //kapsle das ganze in einen Serialisierungsstrom
    ois = new ObjectOutputStream(cis);
    //und lese schliesslich ein.
    EconomicGood eg = (EconomicGood) ois.readObject();
    ...
    //sobald alles da ist, wird die Rechnung bezahlt, d.h. der Verkäufer
    //bekommt das Geld
    Transaction t = new Transaction();
    t.sender = tg.me;
    t.receiver = m.getReceiver();
    t.amount = m.getPrice();
}
  
```

```

UsersTable.getUsersTable().finishTransaction(t);
...
//füge es den eigenen Objekten hinzu
tg.economicGoods.add(eg);
...

```

Sowohl Upload- als auch DownloadThread implementieren das Interface Progress. Das heisst, sie müssen bestimmte Methoden implementieren. Diese Methoden werden vom TransferPanel verwendet, um sie dann grafisch als Tabelle darzustellen. Oberhalb des Balkens sind die DownloadThreads, darunter die UploadThreads. Das sieht dann so aus:

Name	Peer	Progress	Size
Ryuichi Sakamoto - Merry Ch...	seve	4594246	4593664
Name	Peer	Progress	Size
Ryuichi Sakamoto - Merry Ch...	seve	4594297	4593664

Der Unterschied zwischen der Zahl bei Progress und Size entspricht dabei der Grösse des Mantel-Objekts, das die Daten der Datei enthält.

Beenden des Programms

Problem:	Die IP-Adresse auf dem Hauptserver muss auf ‚offline‘ geschaltet werden. Beim Beenden müssen die eigenen Objekte gesichert werden, so dass sie beim nächsten Start wieder erreichbar sind. Die graphische Oberfläche soll verschwinden.
Lösung:	Die IP-Adresse wird mit UsersTable in TheGathering auf offline geschaltet. Die eigenen Objekte werden mit Serialisierung in einer Datei gespeichert. Die Oberfläche wird in TheGathering abgebaut.

Beim Beenden des Programms müssen drei Sachen erledigt werden. Als erstes werden alle eigenen Objekte auf dem Datenserver gelöscht werden. Dann wird die eigene IP in der Users-Tabelle auf ‚offline‘ gesetzt. Beides geschieht in der Methode logout() der Klasse TheGathering.

(in TheGathering.java)

```

private void logout() {
    ...
    //zuerst alle Objekte auf dem Datenserver löschen
    sau.clear();
    ...
    //ersetze die IP durch "offline"
    p.ip = "offline";
    ut.updateIP(p);
}

```

Dann werden sowohl die eigenen Objekte, die nicht verkauft werden, als auch die Verkaufsobjekte in der Datei economicGoods.dat gespeichert, so dass sie beim nächsten Start wieder vorhanden sind. In der Methode serialize().

(in TheGathering.java)

```

private void serialize() {
    ...
    FileOutputStream fos = new FileOutputStream("economicGoods.dat");
}

```

```
OutputStream oos = new OutputStream(fos);  
oos.writeObject(economicGoods);  
oos.writeObject(sellGoods);  
...
```

Über `System.exit(0)`; in `TheGathering.java`, wird das Programm schliesslich beendet.

Schlusswort

Das Ziel meiner Maturaarbeit wurde erreicht. Der Peer-to-Peer-Markt wurde beschrieben, entwickelt und er funktioniert. Ich habe damit den Beweis erbracht, dass es technisch möglich ist, eine recht komplexe Netzwerkanwendung innerhalb kurzer Zeit eigenhändig zu entwickeln, ohne dass das Wissen über die technischen Hilfsmittel vollständig vorhanden sein muss.

Man muss aber anfügen, dass eine solche Applikation nicht veröffentlicht werden könnte. Die wichtigen Punkte Sicherheit und Benutzerfreundlichkeit wurden zu Gunsten der Einfachheit vollständig vernachlässigt. Die Anwendung behält daher ihren Prototyp-Charakter und kann nicht als Ersatz für bestehende Anwendungen betrachtet werden.

Trotzdem glaube ich, mit diesem Projekt gezeigt zu haben, in welche Richtung kommende Peer-to-Peer-Anwendungen gehen könnten.

Quellenverzeichnis

Lehrbücher für Java

- Eckel, Bruce: Thinking in Java, Second Edition. New Jersey 07458: Prentice Hall PTR, 2000.
- Roberts, Simon und Heller, Philip und Ernest, Michael: Complete Java 2 Certification, Study Guide. Alameda CA 94501: SYBEX Inc., 2000
- White, Seth; Fisher, Maydene; Cattell, Rick; Hamilton, Graham; Hapner, Mark: JDBC API Tutroial and Reference, Second Edition.
- Krüger, Guido: Handbuch der Java-Programmierung 3. Auflage (HTML-Ausgabe). München : Addison Wesley, 2002.

Texte aus dem Internet

- David Weekly: The Napster Protocol. <http://david.weekly.org/code/napster.php3> (10.11.02)
- Danielle Roy, IDG News Service, Boston Bureau: Napster Timeline. <http://www.idg.net/idgns/2001/04/02/NapsterTimeline.shtml> (10.11.02)
- Graeme Wearden: Napster's legacy: P2P poised to rule corporate nets. <http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2803865,00.html> (10.11.02)
- LimeWire : Modern Peer-to-Peer File-Sharing over the Internet <http://www.limewire.com/index.jsp/p2p> (10.11.02)
- OpenP2P, Clay Shirky: What Is P2P - And What Isn't <http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html?page=2> (21.11.02)

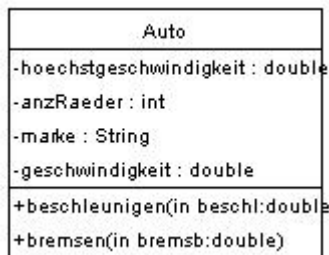
Glossar

Quellen:	Computerlexikon: http://www.computerlexikon.com 21.11.02 Commando GmbH: http://www.commando.de/glossar 21.11.02 Foldoc, Free Online Dictionary of Computing http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=rmi 21.11.02
API:	Application Program Interface. Eine Schnittstelle mit deren Hilfe, Applikationen auf Dienste des Betriebssystemes oder andere Dienste zurückgreifen können.
Client:	Ein Computer oder ein Prozess der Dienste eines anderen Computers (Server) über ein Protokoll in Anspruch nimmt.
Domain:	ein eindeutiger Name, der einer IP-Adresse zugeordnet wird z.b. google.ch
GUI:	GUI ist die Abkürzung für "graphical user interface", die grafische Benutzeroberfläche.
J2EE:	Java 2 Platform, Enterprise Edition. Java-Standard für die Programmierung von Geschäftsapplikationen. „The J2EE platform manages the infrastructure and supports the Web services to enable development of secure, robust and interoperable business applications.“ (Sun Microsystems: Java 2 Enterprise Edition http://java.sun.com/j2ee 21.11.02)
Messenger:	Instant Messenger, Programme die es dem Benutzer erlauben mit seinen Freunden zu kommunizieren solange beide online sind. Die Nachrichten werden dabei in beinahe Echtzeit (< 1 s Übermittlungszeit) übermittelt. Die beliebtesten Programme sind ICQ („I seek you“), AIM (AOL Instant Messenger) und MSN (Microsoft Network) Messenger.
IP-Adresse:	32-Bit-Adresse eines Computers in einem Netzwerk gemäss dem Internet-Protocol z.B. 212.4.74.64
Java:	Eine einfache (im Vergleich zu C++), objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, multithread-fähige, dynamische, Programmiersprache von Sun Microsystems 1995 entwickelt.
JDBC:	Java Database Connectivity. Java-API für den universalen Zugriff auf relationale Datenbankmanagementsysteme wie Oracle, ODBC oder MySQL. Benötigt für jedes RDBMS einen JDBC-Treiber.
MySQL:	Open-Source Datenbank (siehe http://www.mysql.org)
RMI:	Remote Method Invocation. Teil der Programmiersprache Java, der es einem Java-Programm erlaubt Objekte eines anderen Java-Programms auf einem anderen Computer zu benutzen.
Server:	Ein Computer oder ein Programm, das den Clients Dienste über ein Protokoll zur Verfügung stellt.
Swing:	Java-API zur Erstellung von GUI's
Thread:	Eine Java-Klasse, mit deren Hilfe man mehrere Prozesse gleichzeitig auf einer oder mehreren CPU's ausführen kann. (sogenanntes Multithreading)

Anhang

UML-Diagramme

Um einen Überblick über die Klassenstruktur des ganzen Projekts zu erlangen, erstellte ich mit dem UML-Tool „Poseidon UML“ von Gentleware (<http://www.gentleware.de> 21.11.02) für jedes Package ein Diagramm. Hier ein kleiner Überblick für eine Beispielklasse Auto:



Im ersten Block steht der Klassenname (hier: Auto), im zweiten Block sind die Felder (Attribute) aufgeführt, im dritten die Methoden (Funktionen). Ein kleines Minuszeichen vor dem Bezeichner (Identifizier) zeigt, dass das Element private ist, es ist also nur innerhalb der eigenen Klasse sichtbar. Ein kleines Plus zeigt, dass das Element public ist, es ist also für alle Objekte sichtbar und benutzbar. Hinter dem Doppelpunkt steht jeweils der Typ des Elements. Kleingeschriebene Typen sind primitive Datentypen, grossgeschriebene sind Klassen.

Quellcode

Der gesamte Quellcode des Projektes ist auch in digitaler Form (auf CD oder E-Mail) erhältlich. Der Code wurde (bis auf die Klasse SplashScreen.java) vollständig von mir geschrieben. Für die Klasse SplashScreen habe ich die ausdrückliche Genehmigung von dem Autor Guido Krüger.

To: Severin Hacker <homehacker@datazug.ch>
Subject: Re: Java-Buch, Maturaarbeit, Klasse SplashScreen
References: <000801c291a5\$23a03b10\$0206a8c0@SEVERIN>
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 8bit
X-RCPT-TO: <homehacker@datazug.ch>
Status: U
X-UIDL: 335015710

Severin Hacker wrote:

> Für meine Maturaarbeit (entspricht deutschem Abitur) hab ich Ihre
> Klasse SplashScreen
> aus dem Buch Java-Handbuch verwendet. Dabei hab ich sie ein ganz
> wenig verändert
> (Ladebalken am unteren Rand). Ich habe dies in meiner Maturaarbeit
> erwähnt, und möchte
> jetzt aber trotzdem noch ausdrücklich nach Ihrer Erlaubnis fragen.
> Anbei liegt eine Kopie
> der abgeänderten Klasse.
Bin damit einverstanden.

Guido

PGP Fingerprint: 4D99 9BE7 0A3E 1770 A6F2 C649 1F16 0CB0 7012 AF26