Informatik mit Java

Eine Einführung mit **BlueJ** und der Bibliothek **Stifte und Mäuse**

Band 3

Bernard Schriek

Informatik mit Java

Eine Einführung mit

BlueJ

und der Bibliothek

Stifte und Mäuse

Band III

Nili-Verlag, Werl

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet unter <http://dnb.ddb.de> abrufbar.

bernard.schriek@t-online.de
www.nili-verlag.de/sum

ISBN 978-3-00-022995-4

Bernard Schriek
Ostlandstr. 52
59457 Werl

Geschrieben mit dem Programm Ragtime 6 der Ragtime GmbH, Hilden
<http://www.ragtime.de>
Druck und Verarbeitung: SDV - Die Medien AG, Dresden
<http://www.sdv.de>
Printed in Germany

Inhalt

Danksagung

Das Paket *Stifte und Mäuse (SuM)* wurde in den 90er Jahren im Rahmen der Lehrerfortbildung von Ulrich Borghoff, Dr. Jürgen Czischke, Dr. Georg Dick, Horst Hildebrecht, Dr. Ludger Humbert und Werner Ueding entwickelt. Die Bibliothek wurde zuerst in Object Pascal und Oberon geschrieben und dann auf andere objektorientierte Sprachen portiert. Die Urfassung des SuM-Kern-Pakets in Java wurde mir von Dr. Georg Dick zur Verfügung gestellt. Im Rahmen der Planung einer weiteren Fortbildungsreihe, an der ich auch beteiligt war, wurde die Bibliothek SuM-Kern um weitere Pakete erweitert. Fokke Eschen schrieb die erste Version des Programmgenerators, die dann im Rahmen einer besonderen Lernleistung von Alexander Bissaliev (Abi 2005) neu geschrieben wurde. Die Automatensimulatoren SuM-Transduktor und SuM-Akzeptor wurden von Dr. Michael Grinder an der Montana State University in Bozeman, MT, USA entwickelt und von mir überarbeitet. Der Transduktor wird zur Zeit noch verbessert.

Alle in diesem Buch behandelten Projekte wurden von den oben genannten Kollegen bei mehreren Fortbildungen zur objektorientierten Programmierung entwickelt und auch im Unterricht erprobt. Umfangreiche methodisch-didaktische Materialien wurden auf dem Bildungsserver Learnline bereitgestellt <http://www.learnline.de/angebote/oop/>. Dort gibt es auch ein Forum zum Gedankenaustausch. Der OOP-Bereich auf Learnline wird von Horst Hildebrecht verwaltet. Dieses Buch soll ein weiterer Baustein zum SuM-Paket sein. Ich bedanke mich bei allen Entwicklern dieses Pakets sowie den Kollegen in den verschiedenen Fortbildungsgruppen, die durch anregende Diskussionen die Weiterentwicklung unterstützt haben.

Die freie Java-Entwicklungsumgebung *BlueJ* wurde Ende der 90er Jahre an der Monash-Universität in Australien speziell für den Unterricht entwickelt und auch ständig weiter entwickelt. Besonderer Dank geht an Michael Kölling von der University of Kent, der die BlueJ-Webseiten <http://www.bluej.org> pflegt und mit David J. Barnes ein hervorragendes Lehrbuch zur objektorientierten Programmierung mit BlueJ geschrieben hat.

Die Programmiersprache *Java* wurde von Sun Microsystems, Santa Clara, Kalifornien entwickelt. Neben der Objektorientierung und Plattformunabhängigkeit ist die einfache, klare und konsistente Syntax und das konsequente Sprachkonzept ein besonderer Vorzug dieser Programmiersprache, die inzwischen an fast allen Hochschulen und Fachhochschulen eingesetzt wird. Mein Dank gilt allen Javaentwicklern sowie der Firma Sun, die das Java-Paket kostenlos zur Verfügung stellt.

Besonderer Dank geht an Torsten Schultz, der das Manuskript kritisch durchsah und mir viele Anregungen für Verbesserungen gab.

Nicht zuletzt gilt mein Dank meiner Frau, die meine stundenlangen Sitzungen am Computer mit viel Geduld ertragen hat und mich immer wieder zur Weiterarbeit ermutigte.

Werl, im Oktober 2007

Bernard Schriek

Vorwort an Schüler

Die Gefahr, dass der Computer so wird wie der Mensch, ist nicht so groß wie die Gefahr, dass der Mensch so wird wie der Computer.

Konrad Zuse

Nachdem Sie in Band I die Grundlagen der Objektorientierten Programmierung und insbesondere die Ereignisorientierte Programmierung kennen gelernt haben und in Band II die wichtigsten Datenstrukturen und die zugehörigen Algorithmen erarbeitet haben, sollen Sie in diesem Band III einige Spezialbereiche der Informatik kennen lernen.

Dabei handelt es sich um vier unabhängige Themengebiete der Informatik:

- Netzwerkprogrammierung
- Relationale Datenbanksysteme
- Kryptographie
- Automatentheorie und Compilerbau

Aus Zeit- und Platzgründen war es leider nicht möglich, Kapitel über Graphentheorie und maschinennahe Programmierung in dieses Buch zu integrieren.

Die Kapitel über Netzwerkprogrammierung informieren Sie zuerst über die theoretischen Grundlagen der Datenübertragung zwischen zwei und mehreren Computern. Dabei ist es egal, ob diese Computer im gleichen Raum stehen oder über das Internet verbunden sind. Sie werden anschließend eigene Programme entwickeln, mit denen Sie u.a. Email abrufen und versenden können. Sie werden ein Chatsystem und mehrere Netzwerkspiele entwickeln.

In den Kapiteln über Datenbanken lernen Sie zuerst, wie man eines der bekanntesten frei verfügbaren Datenbanksysteme 'MySQL' auf Ihrem Computer installiert. Danach lernen Sie die wichtigsten SQL-Befehle zur Arbeit mit Datenbanken kennen. Zum Schluss sollen Sie lernen, wie man eigene Datenbanken so entwirft, dass möglichst keine Fehler und Unstimmigkeiten entstehen können.

Das Kapitel über Kryptographie zeigt Ihnen, wie schon die alten Römer Nachrichten verschlüsselt haben. Aber auch moderne Verschlüsselungssysteme, wie sie heute von Banken und Internetfirmen benutzt werden, werden behandelt. Dabei kommen Sie allerdings nicht um ein bisschen Mathematik herum.

Zum Schluss sollen Ihnen die Kapitel über Automatentheorie einen Einstieg in die Theoretische Informatik ermöglichen. Dieses Teilgebiet der Informatik beschäftigt sich u.a. mit formalen, also künstlichen Sprachen. Sie werden lernen, wie sich Grammatiken zu formalen Sprachen mit Automaten darstellen lassen. Dazu stehen Ihnen zwei leistungsfähige Automatensimulatoren als Werkzeuge zur Verfügung. Anschließend werden die gerade gewonnen theoretischen Kenntnisse umgesetzt in die Scanner- und Parserentwicklung, einen wichtigen Teil des Compilerbaus. Als Letztes werden Sie ein mächtiges Werkzeug zur Entwicklung von Programmiersprachen, das Programm ANTLR kennen und benutzen lernen.

Da das letzte Schuljahr in der Gymnasialen Oberstufe wegen der Abiturprüfung verkürzt ist, wird es Ihnen wahrscheinlich nicht möglich sein, alle Kapitel ausführlich durchzuarbeiten. Trotzdem sollten Sie dies versuchen, wenn Sie sich später einmal ein Informatikstudium anstreben. An der Hochschule werden diese Teilgebiete der Informatik meist wesentlich theoretischer und abstrakter behandelt.

Die Beispiele und Aufgaben wurden in mehreren Jahren vom Autor und mehreren anderen Lehrern im Unterricht erprobt und von mehreren Schülern vor der Drucklegung getestet. Wenn Sie aber Verbesserungsvorschläge zu machen haben oder Fehler finden, schreiben Sie eine Email an

bernard.schriek@t-online.de

Viel Erfolg bei Ihrer Informatikausbildung

Bernard Schriek

Werl, im Oktober 2007

Vorwort an Lehrer

Dieses Buch versucht Lehrerinnen und Lehrer im Informatikunterricht zu unterstützen. Viele Lehrkräfte haben ihre erste Unterrichtserfahrung mit Pascal bzw. vor langer Zeit mit Basic gesammelt. In den 80er Jahren wurde mit großem Aufwand die strukturierte Programmierung mit Pascal unter der Informatiklehrerschaft eingeführt. Das Leitmotiv war damals: Lösen von Problemen durch Zerlegung in Teilprobleme. In den 90er Jahren gelang der objektorientierten Programmierung, die mit der Programmiersprache Smalltalk ein Nischendasein geführt hatte, mit der Verbreitung von Borland Delphi und Sun Java der Durchbruch. Insbesondere *Java* wurde zur Standardprogrammiersprache an den Universitäten und Fachhochschulen. Jetzt ist das neue Leitmotiv: Objekte schicken sich Botschaften und reagieren darauf. Die Konsequenz ist eine neue Form des Softwareentwurfs. Dazu stellt die Unified Modeling Language (UML) die passenden Diagrammformen zur Verfügung. Zur Dokumentation liefert die Java-Entwicklungsumgebung das passende Werkzeug gleich mit: JavaDoc. Die bei vielen Entwicklern unbeliebte Testphase wird durch spezielle Hilfsmittel zum automatischen Testen unterstützt. In diesem Buch wird großer Wert auf Entwurfstechniken, Dokumentation und Tests gelegt.

Lange Zeit musste der Informatikunterricht mit Werkzeugen arbeiten, die zur professionellen Softwareentwicklung gedacht waren. Inzwischen wurde mit *BlueJ* eine Java-Entwicklungsumgebung speziell für den Unterricht geschaffen. Neben einem leistungsstarken Debugger gibt es jetzt die Möglichkeit interaktiv Objekte zu erzeugen, diese zu inspizieren und ihnen Nachrichten zu schicken. BlueJ wird inzwischen an vielen Schulen und Hochschulen für die Ausbildung der Informatikstudenten genutzt.

Speziell für den Schulunterricht wurde die Bibliothek *Stifte und Mäuse* entwickelt. In ihr ist eine Turtlegrafik integriert. Die in Java komplizierte Ereignisverwaltung wurde in fertige Klassen integriert. Zur Bibliothek entstanden im Rahmen von Lehrerfortbildungen zahlreiche vielfach erprobte Unterrichtsprojekte, die in diesem Buch behandelt werden.

Der vorliegende Band III behandelt die bei den Schülerinnen und Schülern sehr beliebte Netzwerkprogrammierung, relationalen Datenbanksysteme (SQL), Kryptographie (RSA) sowie die Grundlagen der Automatentheorie. Für die Simulation von endlichen Automaten wurden zwei BlueJ-Erweiterungen entwickelt.

An mehreren Stellen im Buch gibt es Hinweise auf Referatsthemen. Schüler können sich dazu die passenden Informationen im Internet suchen.

Zur Unterstützung der Lehrkräfte gibt es eine CD-ROM mit Musterlösungen, die gegen Nachweis der Unterrichtstätigkeit angefordert werden kann.

Die aktuellen SuM-Bibliotheken können aus dem Internet geladen werden.
`<http://www.mg-werl.de/sum/>` bzw. `<http://www.nili-verlag.de/sum/>`

Methodisch didaktische Erläuterungen zum Konzept von "Stiften und Mäusen" und vielen behandelten Projekten finden Sie auf dem Bildungsserver *Learnline*. Dort gibt es auch ein Forum zum Gedankenaustausch.
`<http://www.learnline.de/angebote/oop/>`

Kapitel 1 Installation der Bibliotheken

BlueJ ist eine Entwicklungsumgebung für Java, die an der Monash University in Australien mit dem Ziel entwickelt wurde, Schülern und Studenten den Einstieg in die objektorientierte Programmierung (OOP) zu erleichtern. BlueJ ist also eine Lernumgebung und nicht dazu gedacht, professionelle Javaprogramme zu erstellen. Dazu gibt es geeignetere Werkzeuge wie zum Beispiel Eclipse <http://www.eclipse.org>. In den folgenden Kapiteln werden Sie die besonderen Möglichkeiten von BlueJ kennen und schätzen lernen.

Auch die SuM-Bibliothek (SuM = Stifte und Mäuse) wurde konzipiert um das Erlernen der OOP zu vereinfachen. Das SuM-Konzept wurde in den 90er Jahren von einer Gruppe von Lehrern aus Nordrhein-Westfalen entwickelt, um den Wechsel von der bis dahin üblichen imperativen Programmierung zur objektorientierten Programmierung zu unterstützen. Für verschiedene Programmiersprachen (Object-Pascal, Delphi, Java und Python) wurden die entsprechenden Bibliotheken erstellt. Im Rahmen mehrerer Lehrerfortbildungen wurden eine Reihe von Projekten erarbeitet, um die Schüler in die verschiedenen OOP-Konzepte einzuführen. Weitere ausführliche Informationen zu SuM finden Sie unter <http://www.learnline.de/angebote/oop>.

Um mit SuM unter BlueJ Javaprogramme entwickeln zu können, müssen zuerst drei Pakete auf ihrem Computer installiert sein:
- BlueJ, die Entwicklungsumgebung
- JDK 1.5.0 oder höher, das Java Development Kit und die API-Dokumentation
- die SuM-Bibliotheken, Werkzeuge und Hilfetexte.

1.1 Installation von BlueJ

Die aktuelle Version von BlueJ können Sie sich von der BlueJ Website <http://www.bluej.org/download/download.html> herunterladen.

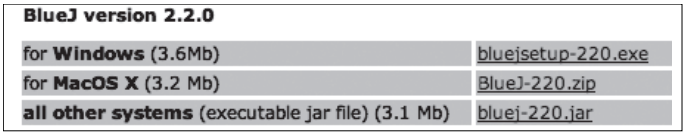

Abbildung 1.1: Download von BlueJ

Für Windows erhalten Sie einen Installer, mit dem Sie BlueJ an einen Ort ihrer Wahl installieren können. Damit allerdings später bestimmte Hilfefunktionen problemlos funktionieren, sollte BlueJ **auf der Festplatte C: im Ordner Programme** installiert werden. Benutzer mit Apple Macintosh und OSX sollten den BlueJ-Ordner **in den Ordner Programme** legen. Starten Sie BlueJ jetzt noch nicht.

1.2 Installation des JDK und der API-Dokumentation

Als Nächstes muss das aktuelle Java Development Kit (JDK) auf dem Computer installiert werden. Sie sollten aber vorher überprüfen, ob sich dieses Paket schon auf Ihrem Computer befindet. Dazu öffnen Sie bei Windows Computern die Eingabeaufforderung (das DOS-Fenster). Beim Macintosh öffnen Sie das Terminal. Geben Sie jetzt den Befehl `java -version` ein und betätigen Sie die Returntaste. Falls das JDK installiert ist, erhalten Sie eine Meldung über die installierte Javaversion. Sie sollte mindestens die Versionsnummer 1.5.0 besitzen. Falls Sie allerdings eine Fehlermeldung erhalten, müssen Sie das JDK erst noch auf ihrem Rechner installieren. Windowsbenutzer können sich das JDK von `<http://java.sun.com/javase/downloads/index.jsp>` herunterladen (J2SE = Java 2 Standard Edition) und installieren. Beachten Sie, dass Sie `JSE 1.5.0` oder höher auswählen (und nicht J2EE = Java 2 Enterprise Edition). Macintoshbenutzer können sich das aktuelle JDK von `<http://www.apple.com/java/>` herunterladen. Normalerweise ist aber das JDK schon unter OSX vorinstalliert.

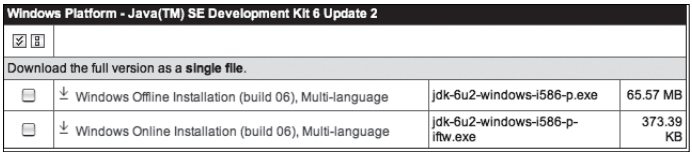

Abbildung 1.2: Download von Java (J2SE SDK)

Für beide Plattformen (Mac und Windows) benötigen Sie jetzt noch die Dokumentation der Javastandardbibliotheken, also die API-Dokumentation (API = Application Programming Interface). Diese finden Sie im gleichen Fenster, von dem aus Sie das JDK geladen haben, etwas weiter unten
`<http://java.sun.com/javase/downloads/index.jsp>`.

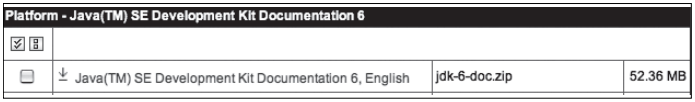

Abbildung 1.3: Download der Dokumentation zu Java

Nach dem Download erhalten Sie einen Ordner, dem Sie den Namen `docs` geben und den Sie in den BlueJ-Ordner legen.

1.3 Installation der SuM-Bibliotheken

Die aktuellen SuM-Bibliotheken finden Sie unter `<http://www.mg-werl.de/sum/>`. Wenn Sie die zip-Datei runter geladen und entpackt haben, erhalten Sie einen Ordner `SuMWin 6.x` bzw. `SuMMac 6.x`. Dessen Inhalt muss an bestimmte Stellen kopiert werden:

- Den Ordner `doc` legen Sie in den BlueJ-Ordner.
- Den Ordner `docs` legen Sie in den BlueJ-Ordner.

Windows-Benutzer öffnen jetzt den Ordner `lib` im BlueJ-Ordner.

Mac-Benutzer machen einen Rechtsklick bzw. ctrl-Klick auf das Programm BlueJ im BlueJ-Ordner und wählen im Kontextmenü `Paketinhalt zeigen`. Doppelklicken Sie `Contents - Resources - Java`.

- Ersetzen Sie den Ordner `german` durch den Ordner mit dem gleichen Namen.
- Ersetzen die Datei `bluej.defs` durch die Datei mit dem gleichen Namen.
- Legen Sie die folgenden drei jar-Dateien in den Ordner `extensions`: `SuMGenerator.jar, SuMAkzeptor.jar, SuMTransduktor.jar`.
- Legen Sie die folgenden acht jar-Dateien in den Ordner `userlib`: `SuMKern.jar`, `SuMEreignis.jar`, `SuMWerkzeuge.jar`, `SuMKomponenten.jar`, `SuMNetz.jar`, `SuMSql.jar`, `SuMStrukturen.jar`, `SuMMultimedia.jar`.

Schließen Sie die Ordner und starten Sie BlueJ. Falls Sie mehrere JDKs auf Ihrem Rechner installiert haben, werden Sie gefragt, mit welchem JDK Sie arbeiten wollen.

1.4 Test der Installation

Als Nächstes sollen Sie kontrollieren, ob alle Komponenten korrekt installiert wurden.

Öffnen Sie die BlueJ-Einstellungen und wählen Sie `Bibliotheken`. Im unteren Teil des Fensters sollte dann stehen, dass die acht SuM-Bibliotheken geladen wurden. Schließen Sie die Einstellungen und wählen Sie im Hilfemenü `Dokumentation sum.kern`. Jetzt sollte sich der Internet-Browser öffnen und die Dokumentation zum Paket `sum.kern` anzeigen. Schließen Sie das Browserfenster. Falls die Dokumentationen nicht angezeigt werden, haben Sie den BlueJ-Ordner nicht an die oben angegebenen Stelle gelegt oder der Ordner `doc` ist nicht im BlueJ-Ordner.

Wählen Sie im Hilfemenü `Java Klassenbibliotheken`. Im Internetbrowser sollte jetzt die Javadokumentation aus dem Ordner `docs/api/index.html`, der im BlueJ-Ordner liegt, angezeigt werden. Falls dies nicht der Fall ist, können Sie sich im Einstellungsfenster von BlueJ unter `Diverses` bei der Eigabe `URL der Javadokumentation` zur oben stehenden Seite durchklicken. Schließen Sie das Browserfenster.

Wählen Sie im Menü `Werkzeuge` den `SuM-Programmgenerator`. Jetzt sollte sich ein Fenster öffnen, in dem Sie verschiedene Komponenten anlegen und das zugehörige SuM-Programm erzeugen können.

Stellen Sie unter `Einstellungen - Editor - Zeilennummern anzeigen` an.

Damit ist die Installation beendet.

Kapitel 2 Netzwerkprogrammierung I

In diesem Kapitel sollen Sie lernen:

- was man unter dem OSI-Schichtenmodell für Netzwerke versteht
- welche Schichten das OSI-Schichtenmodell enthält
- welche Aufgaben die Schichten haben
- wozu IP-Nummern dienen
- welche Aufgaben ein Router im Internet erledigt
- welche Aufgaben ein Nameserver übernimmt
- was Ports sind und wozu sie dienen
- was man unter einem privaten Netz versteht
- wozu Teilnetzmasken dienen
- wozu man dynamische Adressierung benutzt

Dieses Kapitel erläutert den Aufbau und die Wirkungsweise des Internets. Dazu wird zuerst das OSI-Schichtenmodell für Netzwerke erklärt. Die Wirkungsweise des IP-Protokolls und des TCP-Protokolls werden ausführlich behandelt. Die Bedeutung des DNS-Dienstes wird erläutert.

2.1 Das OSI-Schichtenmodell

Das Internet besteht aus zur Zeit (2007) aus ca. 500 Millionen Computern weltweit, die vernetzt sind. Die Computer in Ihrem Informatikraum sind vernetzt und mit einem Router verbunden, der über die Telefonleitung unter Nutzung von DSL (Digital Subscriber Line = Digitale Teilnehmeranschlussleitung) mit einem Computer der Telekom verbunden ist. Dieser Telekom-Computer ist wiederum mit einem Computer der DE-CIX (Deutscher Comercial Internet Exchange) in Frankfurt/M vernetzt, mit dem andere ISPs (Internet Service Provider) in Deutschland, aber auch die zentralen Vermittlungscomputer in anderen Ländern und Kontinenten verbunden sind. Die Vernetzung erfolgt lokal über Ethernetkabel, auf weiteren Strecken über Lichtwellenleiter oder Richtfunkstrecken und sogar über Satellitenverbindungen.

Die Vernetzung von so vielen unterschiedlich schnellen Computern weltweit ist nur möglich, indem klar definierte Regeln eingehalten werden, die die relativ problemlose Integration von neuen Technologien (z.B. DSL statt analoger bzw. ISDN-Übertragung) ermöglichen. Diese Regeln werden im OSI-Schichtenmodell festgelegt (OSI = Open System Interconnection). Das Schichtenmodell enthält 7 Schichten.

Um das Schichtenmodell besser zu verstehen, soll zuerst ein anderes Kommunikationsmodell, das die Kommunikation zwischen dem Lehrer und einem Schüler beschreibt, betrachtet werden. Der Lehrer fragt den Schüler nach dem Jahr der Entstehung des Internets und der Schüler antwortet 1969. Die Frage kann der Lehrer akustisch (mit seiner

Stimme) stellen, wenn der Schüler in der Nähe ist, er kann die Frage aber auch auf ein Blatt Papier schreiben, das er dem Schüler gibt. Die Frage könnte auch in Morsezeichen umgewandelt und mit Lichtsignalen übertragen werden. Man spricht hier von der physikalischen Schicht und es muss sicher gestellt sein, dass der Sender (Lehrer) und der Empfänger (Schüler) diese Nachricht empfangen kann. Bei einer akustischen Nachricht muss der Sender sprechen und der Empfänger hören können, bei der geschriebenen und der mit Blinkzeichen übermittelten Nachricht muss der Empfänger sehen können.

Falls der Schüler Englisch spricht, kann die Frage auch in Englisch statt Deutsch gestellt werden. Das gilt natürlich für beliebig viele Sprachen. Hier kann man von der Codierungsschicht sprechen.

Wenn die Nachricht zu einer entfernten Person übermittelt werden soll, so muss beim Telefonieren die Rufnummer bekannt sein und die Verbindung hergestellt werden. Falls ein Brief geschickt wird, muss die Adresse mit Postleitzahl bekannt sein. Dies ist die Transportschicht.

Wesentlich bei diesem Beispiel ist es, dass in jeder Schicht ein anderes Verfahren (in der Informatik spricht man von einem Protokoll) benutzt werden kann, **ohne dass die anderen Schichten verändert werden müssen**. Dies gilt auch für das OSI-Schichtenmodell.

2.2 Die Bitübertragungsschicht

Die erste Schicht im OSI-Schichtenmodell heißt *Bitübertragungsschicht* (engl. physical layer). Alle Nachrichten werden in Form von Bits also den Zeichen 0 und 1 übertragen. Bei einer Drahtverbindung werden diese Zeichen durch verschiedene Spannungen realisiert, in einem Lichtwellenleiter durch Lichtimpulse. Eine häufig benutzte Übertragungstechnologie ist das kabelgebundene *Ethernet*, mit dem man in den zur Zeit häufigsten Fällen 100 MBit/s übertragen kann. Es sind allerdings auch schon Ethernetverbindungen mit 10 GBit/s in Entwicklung. Die maximale Leitungslänge beträgt beim Ethernet 100 m, für längere Leitungen werden Lichtwellenleiter aus Glasfasern eingesetzt.

Weit verbreitet sind auch sogenannte WLAN-Verbindungen (WLAN = Wireless Local Area Net), wobei die Daten über Funk übertragen werden. Allerdings ist die Übertragungskapazität im WLAN auf 50 MBit/s beschränkt, soll in nächster Zeit auf 300 MBit/s ausgeweitet werden. Ein großes Problem bei der WLAN-Nutzung ist die Abhörsicherheit. WLAN-Verbindungen sollten deshalb möglichst verschlüsselt betrieben werden.

Der Datenverkehr zwischen Computernetzen und auf größere Entfernungen verläuft über sogenannte Backbones, breitbandige Kabelverbindungen mit 10 GBit/s. Unterwasserkabel verbinden die verschiedenen Kontinente mit Backbones. Alternativ werden die Signale über Satelliten übertragen.

Ein Problem bei der Übertragung von Daten in dieser Schicht besteht darin, dass Kollisionen erkannt werden müssen, wenn an beiden Leitungsenden gleichzeitig gesendet wird. Normalerweise sendet der Computer an einem Leitungsende nur dann Daten, wenn vom anderen Leitungsende gerade nicht gesendet wird. Falls aber von beiden Seiten aus gleichzeitig eine Sendung begonnen wird, brechen beide Sender ab und warten eine zufällige Zeit, um die Sendung dann erneut zu beginnen.

2.3 Die Sicherungsschicht

Die zweite Schicht heisst *Sicherungsschicht* (eng. data link layer). Sie sorgt dafür dass bei der empfangenen Information überprüft wird, ob sie korrekt empfangen wurde. Dazu wird die Nachricht mit einer Prüfsumme versehen, die beim Empfänger verglichen wird. Bei Unstimmigkeiten wird die Nachricht noch mal verschickt. Ein übliches Verfahren zur Bildung einer Prüfsumme funktioniert folgendermaßen:

Die Nachricht `01001100011110000111100000111110` (32 Bit) soll verschickt werden. Dazu wird sie in 4 Päckchen von je 8 Bit zerlegt: `01001100 01111000 01111000 00111110`. Man schreibt die ersten beiden Byte (= 8 Bit) untereinander und macht eine sogenannte XOR-Verknüpfung der übereinander stehenden Bits. Falls die Bits gleich sind (beide 0 oder beide 1), so ist das Ergebnis 0, falls die Bits verschieden sind, ist das Ergebnis 1.

`01001100` Byte 1
`01111000` Byte 2
`00110100` Byte 1 XOR Byte 2

Anschließend wird das Ergebnis mit Byte 3 "gexort". Danach wird das Gleiche mit Byte 4 und eventuellen anderen Bytes gemacht.

`00110100` Byte 1 XOR Byte 2
`01111000` Byte 3
`01001100` voriges Ergebnis XOR Byte 3.

`00110100` voriges Ergebnis
`00111110` Byte 4
`00001010` voriges Ergebnis XOR Byte 4.

Jetzt hat man das Prüfbyte erhalten. Dieses Prüfbyte wird vor der eigentlichen Nachricht übermittelt und zum Schluss mit dem Prüfbyte beim Empfänger verglichen. Die Prüfsumme kann natürlich länger als ein Byte sein. Mit der Länge der Prüfsumme wächst die Wahrscheinlichkeit, eventuelle Fehler bei der Datenübertragung zu erkennen. Man sieht sofort, dass die Sicherungsschicht von der Art der physikalischen Bitübermittlung unabhängig ist.

Übung 2.1 Bestimmen Sie das Prüfbyte zu folgender Nachricht: 10110111 01011000 10101100 01011100 10111101 11010111 00100010 11010011.

2.4 Die Vermittlungsschicht

Die dritte Schicht heißt *Vermittlungsschicht* (engl. network layer). Diese Schicht sorgt dafür, dass eine Nachricht vom Empfänger das Ziel erreicht. Jede Nachricht enthält, genau wie ein Brief, Angaben über den Absender und den Empfänger. Das bekannteste Protokoll in dieser Schicht ist das IP-Protokoll (IP = Internet Protocol). Ein weiteres Protokoll ist das ICMP-Protokoll (ICMP = Internet Control Message Protocol), mit dem man andere Computer "anpingen" kann und so feststellt, ob eine Verbindung zwischen diesen Computern hergestellt werden kann. Computer werden im Internet über ihre IP-

Nummer identifiziert. Eine IP-Nummer ist also die Adresse eines Computers. IP-Nummern bestehen aus vier Zahlen zwischen 0 und 255, die durch Punkte getrennt sind, z.B. 80.146.99.217. Sie entsprechen genau 4 Byte.

Übung 2.2 Wieviele unterschiedliche IP-Nummern sind theoretisch möglich?

Wenn jetzt vom Computer 80.146.99.217 eine Nachricht zum Computer 237.123.34.9 geschickt werden soll, muss der Weg des Datenpakets bestimmt werden. Im Internet wird zwischen zwei Computern keine feste Verbindung geschaltet, wie Sie das vom Telefonieren kennen, sondern es werden Datenpakete auf die Reise geschickt wie Pakete bei der Post. Internettelefonie funktioniert nur deshalb so gut, weil die entsprechenden Datenpakete bevorzugt behandelt und schneller weitergeleitet werden. Es handelt sich dabei also um eine Art von Expresspaketen. Wenn Sie auf der Insel Amrum bei der Post ein Paket nach Amöneburg in Hessen verschicken, so weiß der Mann am Schalter, der das Paket in Empfang nimmt, natürlich nicht, welchen genauen Weg das Paket nimmt. Anhand der Postleitzahl erkennt er allerdings, dass das Paket die Insel verlassen muss.

Eine ähnliche Aufgabe übernimmt der Switch, mit dem alle Computer im Computerraum der Schule verbunden sind. Anhand der IP-Nummer erkennt er, dass er das Datenpaket an den Router, der mit der Telefonleitung verbunden ist weiterleiten muss. IP-Nummern in einem lokalen Netz (LAN = local area net) unterscheiden sich nur in der letzten Zahl, die ersten 3 Zahlen sind identisch. Ein Switch speichert eine Tabelle die lokalen Adressen und weiß so, über welche Leitung er lokale Pakete weiterleiten muss. Dies unterscheidet einen Switch von einem Hub, der genau wie ein Switch aussieht. Ein Hub kennt keine Adressen und sendet deshalb eine empfangene Nachricht an alle angeschlossenen Computer weiter. Dadurch entsteht ein unnötig hoher Netzverkehr, so dass Hubs inzwischen kaum noch verwendet werden.

Das am Anfang dieses Abschnitts erwähnte deutsche Vermittlungszentrum DE-CIX in Frankfurt/M besteht also im Prinzip aus einer Reihe von sehr leistungsfähigen Switches, die Nachrichten von einem Netz z.B. dem Telekom-Netz zu einem anderen Netz z.B. Freenet weiterleiten. Dazu muss der Switch natürlich in einer Adresstabelle speichern, welche Anfangs-IP-Nummern zu den verschiedenen Netzen gehören. Einen Switch, der mehrere Rechnernetze koppelt, bezeichnet man als Router. Der Router an ihrer Schule oder bei Ihnen zu Hause, koppelt das lokale Netz (LAN) mit dem Telekomnetz bzw. dem Netz Ihres ISPs. (ISP = Internet Service Provider, eine Firma, die Endanwendern den Internetanschluss zur Verfügung stellt.)

Viele ISPs haben weniger IP-Nummern reserviert, als Computer bei ihren Kunden vorhanden sind, da sie davon ausgehen, dass viele Kunden nur kurzfristig online sind. Aus diesem Grund erhalten Computer, die sich im Netz anmelden, jedesmal eine neue IP-Nummer, die gerade nicht benutzt wird. Dies erledigt das DHCP (DHCP = Dynamic Host Configuration Protocol), ein Protokoll, mit dem sich Computer im Internet anmelden können. Hier bietet sich eine **Referat** zum DynDNS-Dienst an, der die Nachteile einer dynamischen (regelmäßig wechselnden) IP-Nummer für einen Computer zu beheben versucht.

2.5 Die Transportschicht

Die vierte Schicht heißt *Transportschicht* (engl. transport layer). Diese Schicht kümmert

sich um die Zerlegung einer Nachricht in kleinere Segmente von ca. 1500 Bytes. Die maximale Größe von 1500 Bytes wird vom Ethernet-Protokoll (Schichten 1 und 2) gefordert. Die Datenpakete werden dann einzeln auf die Reise zum Empfänger geschickt. Da das IP-Protokoll aus der dritten Schicht (Vermittlungsschicht) sich um den Reiseweg der Pakete kümmert und dabei versucht Netzlasten auszugleichen, kann es passieren, dass die Segmente einer Nachricht unterschiedliche Wege im weltweiten Netz nehmen und auch nicht in der korrekten Reihenfolge beim Empfänger ankommen. Die Transportschicht kümmert sich um die Nummerierung der einzelnen Segmente und die richtige Zusammensetzung an der Empfangsstelle. Wenn Sie mit ihrem Webbrowser ein größeres Bild empfangen, bemerken Sie manchmal Pausen beim Bildaufbau, nach denen das Bild dann wieder schneller aufgebaut wird. Dies wird durch verspätet ankommende Segmente beim Empfänger bewirkt.

Übung 2.3 Ein Foto aus einer Digitalkamera besteht aus 3072 x 2304 Pixeln mit 24 Bit Farbtiefe. Berechnen Sie, in wieviele Segmente das Bild bei einer unkomprimierten Übertragung im Internet aufgespalten wird.

In der Transportschicht gibt es zwei wichtige Protokolle TCP und UDP. Das *TCP-Protokoll* (TCP = Transmission Control Protocol) kümmert sich neben der Segmentierung um die Überprüfung, ob ein Segment richtig angekommen ist. Dazu erhält jedes Segment eine Kennung und der Empfänger bestätigt dem Absender den Empfang eines Segments. Falls der Absender nicht innerhalb einer bestimmten Zeit die Empfangsbestätigung erhält, wird das entsprechende Segment noch mal verschickt. Bevor die Segmente verschickt werden, kümmert sich TCP zuerst darum, ob eine Verbindung zwischen Sender und Empfänger überhaupt hergestellt werden kann. Da der Datenverkehr in beide Richtungen läuft, spricht man hier von einer bidirektionalen Verbindung.

Wie können Pakete verloren gehen? Eine Möglichkeit ist es, dass der Weg des Pakets von den Routern aus der Vermittlungsschicht zu weit gewählt wurde, so dass die Lebenszeit (TTL = time to live) überschritten wurde und das Paket gelöscht und nicht mehr weitergeleitet wird. In der Vermittlungsschicht erhält jedes Paket (= Segment) neben Absender- und Empfänger-IP-Nummer auch eine TTL-Nummer. Diese Nummer wird von jedem Router bei der Weiterleitung um 1 erniedrigt. Falls die 0 erreicht wird, wird das Paket nicht mehr weitergeleitet und somit gelöscht. So wird verhindert, dass "herrenlose Pakete" die Datenautobahn verstopfen.

Das *UDP-Protokoll* (UDP = User Datagram Protocol) ist im Gegensatz zum TCP-Protokoll nicht verbindungsorientiert. Die Daten werden segmentiert und vom Absender zum Empfänger geschickt. Dabei wird nicht geprüft, ob der Empfänger überhaupt existiert und ob die Segmente korrekt angekommen sind. Dieses Protokoll ist natürlich bedeutend schneller und wird bei der Übertragung von Ton- und Filmsequenzen benutzt, wo sich das Fehlen eines Segments kaum bemerkbar macht.

2.6 Die Sitzungsschicht

Die fünfte Schicht heißt *Sitzungsschicht* (engl. session layer). Wenn bei einer Verbindung mit einem anderen Computer mehrere Nachrichten ausgetauscht werden, spricht man von einer Sitzung. Die Nachrichten einer solchen Sitzung werden mit einer gemeinsamen Kennung versehen. Eine solche Sitzung kann zum Beispiel der Besuch einer Website sein, bei der verschiedene Webseiten angeschaut werden. Auch der Kontakt mit einem

Email-Server ist eine Sitzung. Es wird nachgefragt, ob neue Emails angekommen sind und falls ja, werden diese Emails runtergeladen. Während einer Sitzung merken sich die beteiligten Computer die zugehörigen IP-Nummern. Diese Schicht wird üblicherweise vom Anwendungsprogrammierer implementiert, da ihre Aufgaben eng mit der Anwendung zusammenhängen.

2.7 Die Darstellungsschicht

Die sechste Schicht heißt *Darstellungsschicht* (engl. presentation layer). Diese Schicht ist verantwortlich dafür, dass sich Computer verschiedener Systeme verstehen können. Dazu werden die Informationen in ein einheitliches Format umgewandelt. Ein einfaches Beispiel ist die Codierung eines Zeilenumbruchs in einem Text. Unter Windows und dem schon betagten Betriebssystem DOS wird ein Zeilenumbruch durch die beiden Zeichen <CR> <LF> gekennzeichnet. <CR> steht für *Carriage Return*, auf deutsch Wagenrücklauf und <LF> für *Linefeed* auf deutsch Zeilenvorschub. Diese Begriffe stammen noch aus der Zeit der Fernschreiber aus den 50er Jahren des vorigen Jahrhunderts. Unter Unix bzw. Linux wird für den Zeilenumbruch nur das Zeichen <LF> benutzt. Auf Apple-Rechnern wurde bis vor kurzem <CR> benutzt, inzwischen wird <LF> eingesetzt, da die aktuellen Betriebssysteme von Apple-Computern von Unix abstammen. Bei Übertragungen im Internet wird ein Zeilenvorschub standardmäßig als <CR><LF> übertragen. Der Anwendungsprogrammierer muss dies dann berücksichtigen. Auch Sonderzeichen wie die deutschen Umlaute werden in verschiedenen Betriebssystemen unterschiedlich codiert und sollten in der Darstellungsschicht vereinheitlicht werden. Dass dies nicht immer perfekt funktioniert, sehen Sie, wenn auf Webseiten Umlaute durch seltsame Symbole dargestellt werden.

Eine weitere Aufgabe der Darstellungsschicht ist die Verschlüsselung. Dabei müssen sich Sender und Empfänger über das Verschlüsselungsverfahren einigen. Banktransaktionen laufen üblicherweise nur verschlüsselt ab. Kapitel 7 dieses Buchs beschäftigt sich mit den entsprechenden Verschlüsselungsverfahren.

2.8 Die Anwendungsschicht

Die siebte und letzte Schicht heißt *Anwendungsschicht* (eng. application layer). Die Anwendungsschicht besteht aus den verschiedenen Internetprogrammen wie z.B. Firefox als Browser und Eudora als Emailprogramm. Für viele Internetdienste gibt es unzählige Programme, die die Möglichkeiten der Dienste ausnutzen. Dazu muss die Funktionalität und das Protokoll eines solchen Dienstes genau beschrieben sein. Im nächsten Kapitel werden Sie sich hauptsächlich mit dieser Schicht beschäftigen und zuerst bekannte Dienste wie Email analysieren. Anschließend werden Sie für bestimmte Probleme eigene Internetdienste mit Hilfe von Protokollen definieren und die zugehörigen Programme implementieren.

Auf den meisten Computern laufen gleichzeitig mehrere Internetprogramme. Man kann seine Email abrufen, während gleichzeitig im Browser eine Webseite geladen wird. Damit sich die entsprechenden Datenübertragungen nicht in die Quere kommen, werden zusätzlich zu den IP-Nummern, die einen Computer identifizieren, noch Portnummern benutzt, die einen speziellen Internetdienst auf dem Computer kennzeichnen. Portnummern (kurz *Ports*) sind ganze Zahlen zwischen 0 und 65535. Die Ports zwischen 0 und 1023

werden als *well known ports* bezeichnet und festen Diensten zugeordnet. Zum Beispiel benutzen Browser zur Betrachtung von Webseiten den Port 80. Email wird über Port 25 verschickt und Port 110 empfangen.

Es besteht die Möglichkeit bestimmte Ports zu sperren und somit gewisse Internetdienste zu verbieten. Eine *Firewall* dient zur Sperrung bestimmter Ports, oder genauer: Alle Ports werden gesperrt und nur bestimmte, wenige Ports werden geöffnet. So soll verhindert werden, dass unerwünschte Programme auf dem Computer heimlich Daten ausspionieren und im Internet mit anderen Computern austauschen. Leider benutzen viele Privatanwender die Firewall nicht, so dass kriminelle Täter auf diesen Rechnern heimlich Trojaner installieren können. Das sind Programme, die von einem anderen Computer ferngesteuert werden können. Es gibt inzwischen einen richtigen kriminellen Markt, bei dem es Anbieter ermöglichen, gleichzeitig auf Tausenden von Computern Aktionen auszulösen, z.B. Spammail (unerwünschte Werbung) zu versenden oder Millionen von Anfragen an ein Firmennetz zu senden, so dass dessen Netzverbindungen zusammenbrechen (Denial of service attack).

2.9 Das Domain Name System

Jeder Computer, der mit dem Internet verbunden ist, muss eine eindeutige Adresse haben, die IP-Nummer. Da Menschen sich solche IP-Nummern nur schwer merken können, besteht die Möglichkeit, den IP-Nummern verständliche Namen zuzuordnen. Zum Beispiel ist die IP-Nummer 217.79.215.140 dem Bundestag (www.bundestag.de) zugeordnet. So wie eine IP-Nummer in 4 Zahlen aufgeteilt ist, ist auch der IP-Name durch Punkte aufgeteilt. Der letzte Namensbestandteil de steht für Deutschland. Er wird als *Top-Level-Domain* (TLD) bezeichnet. Andere TLDs sind us (für USA), com (commercial), org (organisation), fr (france), jp (japan), edu (education), to (Tonga), vu (Vanuatu, ein Inselstaat im Südpazifik) und so fort. Der zweite Bestandteil des IP-Namens also das Wort bundestag, wird als *Domain* bezeichnet.

Wenn man eine eigene Internetpräsenz aufbauen will, muss man eine solche Domain beantragen. Dies kann man nicht selbst, sondern man muss genau wie an der Börse einen Makler, Provider genannt, beauftragen. Es gibt eine zentrale Instanz, die den deutschen Bereich, also alle Domains, die mit .de enden, verwaltet, das ist die DENIC (Deutsches Network Information Center). Die Webseite `<http://www.denic.de>` liefert Informationen, welche Namen schon vergeben sind. Wenn man sehen will, wer Namen aus anderen Top-Level-Domains, z.B. com oder org, für sich registriert hat, leistet `<http://www.onlinewhois.info/>` gute Dienste.

Beim DENIC waren 2006 schon mehr als 10 Millionen Domain-Namen registriert. Da Router, die Nachrichten im Internet weiterleiten, nur mit IP-Nummern arbeiten, muss bei einer Anfrage mit IP-Namen möglichst schnell in einer Datenbank, die man als *DNS* (*Domain Name System*) bezeichnet, nachgeschaut werden, welche IP-Nummer zu diesem IP-Namen gehört. Das DNS ist hierarchisch aufgebaut. Zuerst wird beim DNS-Server, der bei den Netzwerkeinstellungen des Computers eingetragen ist, nachgeschaut, ob der IP-Name dort bekannt ist. Ist dies nicht der Fall, leitet dieser DNS-Server die Anfragen an den nächsten übergeordneten DNS-Server weiter und so fort, bis der Eintrag gefunden oder eine Fehlermeldung geliefert wird. So wird vermieden, dass die Datenbanken der DNS-Server zu groß werden.

Sie können übrigens erkennen, dass der DNS-Dienst nur zusätzlich vorhanden ist, indem Sie im Browser statt des IP-Namens die IP-Nummer direkt eingeben. Wenn Sie also <http://217.79.215.140> eingeben, erhalten Sie die gleiche Webseite, wie wenn Sie <http://www.bundestag.de> eingeben. Das Internet funktioniert ganz ohne den DNS-Dienst, er wurde nur geschaffen, damit wir uns Internetadressen besser merken können.

2.10 Lokale Netze (Intranet)

Umgekehrt besteht die Möglichkeit, zu einer IP-Nummer den zugehörigen IP-Namen zu ermitteln, auch dies erledigt der DNS-Server. Bei <http://www.dnswatch.info/de> können Sie zu IP-Namen die IP-Nummern und umgekehrt ermitteln. Dabei wird auch angezeigt, bei welchen DNS-Servern nachgefragt wurde.

Viele Firmen und auch Schulen benutzen ein *lokales Netzwerk* (*LAN*), in dem mehrere Computer über Switche vernetzt sind. Ein Router bildet dann die Schnittstelle nach draußen zum *WAN* (*Wide Area Network*), also dem restlichen Internet. Das lokale Netz wird oft als *Intranet* bezeichnet. Dieses lokale Netz ist üblicherweise über eine einzige IP-Nummer von außen zu erreichen. Innerhalb des Intranets werden sogenannte private IP-Nummern benutzt, normalerweise aus dem Adressbereich 10.x.x.x oder 192.168.x.x. Diese Adressen sind für lokale Netze reserviert, können deshalb in vielen Intranets gleichzeitig benutzt werden, sie sind nämlich nicht nach außen sichtbar. Wenn ein Computer aus dem Intranet eine Nachricht an einen Computer außerhalb des Intranets sendet, so wandelt der Router die lokale IP-Nummer in der Absenderangabe des Datenpakets in die globale IP-Nummer des Intranets um. Zusätzlich wird die Portnummer des Absenders verändert, so dass der Router eine Antwort an den passenden Computer zurücksenden kann. Dies bezeichnet man als *IP-Masquerading*.

Die IP-Nummer eines Computers besteht aus dem Netzwerkteil und dem Geräteteil. Nur Computer mit gleichem Netzwerkteil können im Intranet Daten austauschen. Die Teilnetzmaske legt fest, welche Anteile der IP-Nummer zum Netzwerkteil und welche Anteile zum Geräteteil gehören. Üblicherweise wird die Teilnetzmaske 255.255.255.0 verwendet. Dies bedeutet, dass die ersten drei Bytes der IP-Nummer zum Netzwerkteil gehören. In einem solchen lokalen Netz müssen also die ersten 3 Bytes der IP-Nummer für alle Computer übereinstimmen. Für das letzte Byte gibt es 256 Möglichkeiten, also kann da lokale Netz nur 256 Computer enthalten. In einem Netz mit der Maske 255.255.0.0 müssen nur die ersten 2 Bytes der IP-Nummer übereinstimmen. Es kann also 256*256 = 65536 Computer im lokalen Netz geben.

IP-Nummer	192.168.38.123
Teilnetzmaske	255.255.255.0
Netzwerkteil	192.168.38
Geräteteil	123

Abbildung 2.1: Aufgabe der Teilnetzmaske

Die IP-Nummer 127.0.0.1 ist auch reserviert und bezeichnet immer den lokalen Computer. Der zugehörige IP-Name ist *localhost*. So kann eine Computer eine Nachricht an sich selbst verschicken.

Wenn ein Computer in ein lokales Netz eingebunden werden soll, werden 4 IP-Nummern

benötigt:
• die IP-Nummer des Computer selbst (oft Host-IP genannt)
• die Teilnetzmaske (meistens 255.255.255.0)
• die IP-Nummer der Routers, der die Verbindung mit dem restlichen Internet herstellt. Hier findet man manchmal satt Router die Bezeichnung Gateway.
• die IP-Nummer des Nameservers (DNS)

Die IP-Nummer des Routers und des DNS-Servers müssen den gleichen Netzwerkteil haben, wie die IP-Nummer des neu angeschlossenen Computers. Normalerweise sind diese IP-Nummern gleich, da ein Router meistens auch als Nameserver dient.

2.11 Dynamische Adressierung

Standardmäßig werden Router und neue Computer mit einer Einstellung ausgeliefert, die die Einstellung dieser IP-Nummern überflüssig macht. Der Router weist angeschlossenen neuen Computern jedesmal, wenn sie eingeschaltet werden, eine neue gültige IP-Nummer mit passender Teilnetzmaske zu und informiert diese Computer über die IP-Nummern des Routers und Nameservers. Diesen Dienst bezeichnet man als *DHCP* (Dynamic Host Control Protocol). Bei den meisten Routern kann man den passenden Adressraum einstellen. Bei den Netzwerkeinstellungen eines Computers kann man DHCP (dynamische Adressierung) einstellen.

Ein Router hat normalerweise 2 IP-Nummern. Die eine ist die IP-Nummer, unter der das lokale Netz von außen angesprochen wird, die andere ist die IP-Nummer, die zu dem lokalen Adressraum des Intranets gehört. Die IP-Nummer, unter der das Netz von außen angesprochen wird, ist üblicherweise eine dynamische Adresse, die meistens alle 24 Stunden (von T-online) gewechselt wird. Dadurch ist es nicht möglich, einem Nameserver den IP-Namen des Schulnetzes mitzuteilen. Da die IP-Nummer täglich gewechselt wird, muss die Zuordnung IP-Nummer - IP-Name täglich geändert werden. Also ist das Schulnetz und damit der Schulserver, nicht von außen erreichbar (außer man kennt die aktuelle IP-Nummer). Hier ist eine neuer Dienst, *DynDNS,* entstanden, der dieses Problem behebt. Unter `<http:///www.dyndns.org>` kann man das lokale Netz kostenlos anmelden und einen IP-Namen für das Netz eintragen, z.B. bspgymnasium.dyndns.org. Im Router muss man einstellen, dass er sich regelmäßig bei `dyndns.org` meldet und die aktuelle IP-Nummer mitteilt. Außerdem muss man im Router mit NAT (Network Address Translation) einstellen, dass eine Nachricht von außen an den lokalen Server (des Schulnetzes) weitergeleitet wird. Wenn man jetzt auf dem lokalen Server des Schulnetzes Internetdienste anbietet, können diese von außen in Anspruch genommen werden. Dies werden Sie im nächsten Kapitel noch genauer kennen lernen.

2.12 Zusammenfassung

In diesem Kapitel haben Sie gelernt, dass die Datenübertragung im Internet nach dem OSI-Schichtenmodell abläuft. Oft werden Schicht 1-3 zur sogenannten physikalischen Schicht und die Schichten 6-8 zur Anwendungsschicht zusammengefasst. Dies nennt man das vereinfachte Schichtenmodell, auch TCP-Schichtenmodell.

Der große Vorteil dieses Modells besteht in der Unabhängigkeit der Schichten. So konnte das Internet ohne große Probleme ausgebaut und die Datenübertragung beschleunigt

werden.

OSI-Schichtenmodell	vereinfachtes Schichtenmodell
7. Anwendungsschicht	4. Anwendungsschicht
6. Darstellungsschicht	
5. Sitzungsschicht	
4. Transportschicht	3. Transportschicht
3. Vermittlungsschicht	2. Internetschicht
2. Sicherungsschicht	1. Netzzugangsschicht
1. Bitübertragungsschicht	

Abbildung 2.2: Das Schichtenmodell

Im letzten Teil dieses Kapitels haben Sie den DNS-Dienst und weitere Hilfsdienste kennen gelernt, die den Umgang mit dem Internet erleichtern. Sie sollten jetzt die wesentlichen Konfigurationseinstellungen bei der Einbindung eines neuen Computers ins Internet/Intranet erklären können.

Neue Begriffe in diesem Kapitel

• **Schichtenmodell** Die Aufgaben der Datenübertragung werden im OSI-Schichtenmodell auf 7 Schichten aufgeteilt, die voneinander unabhängig sind. So kann in jeder Schicht das Übertragungsverfahren verbessert werden, ohne dass die anderen Schichten geändert werden müssen.

• **IP-Nummer** Jeder Computer im Internet besitzt eine eindeutige IP-Nummer, über die er angesprochen werden kann. Man kann die IP-Nummer mit der Adresse einer Person oder Firma vergleichen, die benötigt wird, um ein Paket zuzustellen.

• **DNS** Das Domain Name System dient dazu, den IP-Nummern leichter merkbare IP-namen zuzuordnen.

• **TCP** Das TCP-Protokoll dient dazu, Daten im Internet sicher zu übertragen, indem der Empfänger Meldungen über den Empfang einzelner Teilpakete an den Absender zurücksendet. Falls ein Paket nicht ankommt, wird es dann nochmal versendet.

• **UDP** Das UDP-Protokoll wird benutzt, um Daten wie Musiksendungen oder Filme zu übertragen, bei denen auch kleinere Fehler auftreten können, ohne dass die Qualität stark beeinflusst wird. UDP ist bedeutend schneller als TCP, da auf Empfangsquittungen verzichtet wird.

• **Private Netze** Bestimmte IP-Nummern sind für die Nutzung in privaten Netzen reserviert. Nach aussen erscheint dieses LAN wie ein einzelner Computer mit nur einer IP-Nummer.

• **Port** zusätzlich zur IP-Nummer muss bei einer Datenübertragung noch der Port angegeben werden, der den zugehörigen Dienst kennzeichnet. So können auf einem Computer gleichzeitig mehrere Datenübertragungen statt finden.

Kapitel 3 Netzwerkprogrammierung II

In diesem Kapitel sollen Sie lernen:

- was man unter dem Server-Client-Konzept versteht
- wie das Daytime-, QOTD- und Echo-Protokoll funktioniert
- wie man Daytime-, QOTD- und Echo-Clients schreibt
- wie das POP3-Protokoll funktioniert
- wie das SMTP-Protokoll funktioniert
- welche Schwächen die beiden Mail-Protokolle besitzen
- wie POP3-vor-SMTP-Authentifizierung funktioniert

Nachdem Sie im letzten Kapitel die theoretischen Grundlagen zur Datenübertragung im Internet kennen gelernt haben, sollen Sie in diesem Kapitel eigene Netzwerkprogramme schreiben. Dabei werden Sie immer in der Anwendungsschicht arbeiten. Die Bibliothek sum.netz der "Stifte und Mäuse"-Umgebung wird die Arbeit mit der Netzwerkschnittstelle von Java stark vereinfachen.

3.1 Ein Daytime-Client

Es gibt im Internet zwei Möglichkeiten, das aktuelle Datum und die aktuelle Uhrzeit zu erhalten. Eine Möglichkeit ist die Nutzung des *NTP*-Dienstes (NTP = *Network Time Protocol*), der von den meisten Betriebssystemen benutzt wird, um die interne Uhr des Computers zu synchronisieren. Das Protokoll für diesen Dienst ist ziemlich komplex, da auch Zeitverschiebungen durch die Signallaufzeiten im Internet berücksichtigt werden. Der zweite Dienst ist der *Daytime*-Dienst, der nur einen String mit Angaben über das Datum und die Uhrzeit liefert. Den Computer, der diesen String liefert, bezeichnet man als einen *Server*. Dieser Name wird aber auch für das Programm zur Bereitstellung der Uhrzeit benutzt. Die Bezeichnung Server gilt also für einen Computer aber auch für ein Programm auf diesem Computer, das im Internet Dienste zur Verfügung stellt. Der Dienst läuft immer auf einem bestimmten Port. Der NTP-Server läuft auf Port 123, der Daytime-Server läuft auf Port 13. Eine Liste der (well known) Ports mit den zugehörigen Diensten finden Sie bei `<http://de.wikipedia.org/wiki/Port_%28Protokoll%29>`.

Ein Computer, aber auch ein Programm, das einen Dienst nutzt, der von einem Server angeboten wird, bezeichnet man als *Client* (dt. Kunde). Allgemein spricht man von einem Client-Server-System. Im Gegensatz dazu gibt es auch Verbindungen zwischen zwei "gleichwertigen" Computern. Dies bezeichnet man als Peer-to-Peer-System (kurz P2P, peer = Gleichgestellter). Solche Systeme werden meistens von Tauschbörsen benutzt. In diesem Buch werden nur Client-Server-Systeme betrachtet.

Ein Protokoll beschreibt die Regeln, nach denen Clients und Server Daten austauschen. Das Daytime-Protokoll ist extrem einfach. Der Server wartet auf neue Verbindungen. Wenn sich ein Client beim Server anmeldet, schickt der Server einen String mit den Da-

tumsinformationen und trennt anschließend die Verbindung.

Starten Sie in BlueJ den Programmgenerator und erzeugen Sie die folgende Programmoberfläche. Sie besteht aus 3 Etiketten, 2 Knöpfen und einem Textfeld zur Eingabe der Serveradresse. Das Textfeld soll `hatTextfeldServer` heißen. Speichern Sie das Projekt als `DaytimeClient`.

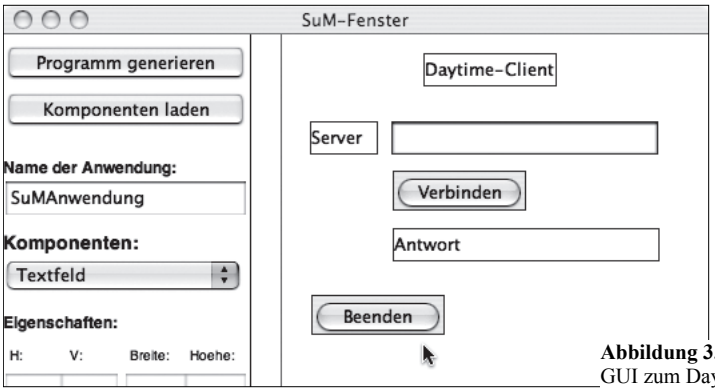

Abbildung 3.1: GUI zum Daytime-Client

Im Quelltext der Klasse `SuMAnwendung` müssen Sie oben die Zeile

```
import sum.netz.*;
```

ergänzen. Der Dienst `hatKnopfVerbindenGeklickt` lautet folgendermaßen:

```
public void hatKnopfVerbindenGeklickt()
{
  Verbindung lVerbindung;
  String lServeradresse;
  String lAntwort;

  lServeradresse = hatTextfeldServer.inhaltAlsText();
  lVerbindung = new Verbindung(lServeradresse, 13, true);
  if (lVerbindung != null)
  {
    lAntwort = lVerbindung.empfangeneNachricht();
    hatEtikettAntwort.setzeInhalt(lAntwort);
    lVerbindung.gibFrei();
  }
}
```

Die Klasse `Verbindung` ist im SuM-Netz-Paket enthalten. Sie ermöglicht es, sich als Client mit einem Server zu verbinden, diesem Nachrichten zu senden, Nachrichten vom Server zu empfangen und die Verbindung zu trennen. Genauere Informationen finden Sie im BlueJ-Hilfe-Menü in der Dokumentation des SuM-Netz-Pakets.

Übung 3.1 Informieren Sie sich in der Dokumentation über die Klasse `Verbindung` und implementieren Sie den DaytimeClient. Testen Sie mit den Servern `time.fu-berlin.de` und `mgwerl.dyndns.org`.

Der Konstruktor der Klasse `Verbindung` erwartet 3 Parameter: Der erste ist die Adresse des Daytime-Servers als IP-Name oder als IP-Nummer. Leider gibt es inzwischen nur noch wenige Daytime-Server und es kann nicht garantiert werden, ob die in Übung 3.1 angegebenen Server noch funktionieren. Der zweite Parameter ist der Daytime-Port 13, der letzte Parameter gibt an, ob die Kommunikation mit dem Server zu Testzwecken mitprotokolliert werden soll. Ein solches Protokoll sieht dann so aus:

```
Öffne Verbindung: - Eigene IP <10.0.0.3> - Partner-IP <
130.133.1.10> - Eigener Port: 50059 Partner-Port: 13
Verbindung liest: Wed Jun 27 18:55:25 2007
Schließe Verbindung mit Socket: Socket[addr=time.fu-berlin.de/
130.133.1.10,port=13,localport=50059]

Öffne Verbindung: - Eigene IP <10.0.0.3> - Partner-IP <91.1.203.3>
- Eigener Port: 50060 Partner-Port: 13
Verbindung liest: 27.6.2007 18:55:46
Schließe Verbindung mit Socket: Socket[addr=mgwerl.dyndns.org/
91.1.203.3,port=13,localport=50060]
```

Das Protokoll wird in einem zusätzlichen Fenster, dem Konsolenfenster, angezeigt, das bei Ihnen eventuell von dem SuM-Fenster verdeckt wird. Wie Sie am Protokoll sehen können, hat nur der Server die Portnummer 13. Clients erhalten eine zufällige relativ hohe Portnummer. Zu den IP-Namen des Servers werden auch die entsprechenden IP-Nummern angegeben. Sie können in das Textfeld für die Serveradresse auch die IP-Nummern statt der IP-Namen angeben. `mgwerl.dyndns.org` ist die dynamische Adresse (siehe Abschnitt 2.11) des Daytime-Servers, der auf dem Server des Marien-Gymnasiums in Werl läuft.

In den USA bietet das NIST (National Institute of Standards and Technology) eine ganze Reihe von Time-Servern an, die auch das Daytime-Protokoll beherrschen. Sie finden die Adressen unter <http://tf.nist.gov/service/time-servers.html>. Beachten Sie allerdings zwei wichtige Dinge:
(1) Der Server antwortet mit zwei Nachrichten. Zuerst wird eine leerer String zurückgeschickt, anschließend erst ein String mit der Uhrzeit. Sie müssen also Ihr Programm etwas abändern.
(2) Falls ein Computer **mehr als 4-mal pro Sekunde** nach der Zeit fragt, wird dies als ein Netzwerkangriff gewertet und die zugehörige IP-Nummer des Absenders dauerhaft gesperrt.

Ein zum Daytime-Dienst sehr ähnlicher Dienst ist der *QOTD-Dienst*. Wenn sich ein Client beim QOTD-Server anmeldet, erhält er einen zufälligen Spruch (Quote of the day = Spruch des Tages) zurück. Anschließend trennt der Server die Verbindung.

Übung 3.2 Schreiben Sie einen QOTD-Client. Der QOTD-Port hat die Nummer 17. Als mögliche Serveradressen können Sie `quotes4all.net`, `ota.iambic.com` und `mgwerl.dyndns.org` nehmen. Die Antwort soll dabei in einem Zeichenbereich (statt Etikett) angezeigt werden.

3.2 Ein Echo-Client

Ein weiterer einfacher Internetdienst ist der Echodienst. Dabei wird zuerst die Verbindung mit einem Echoserver hergestellt. Anschließend kann man dem Echoserver einen

Text senden, den dieser dann (wie ein Echo) zurücksendet. Zum Schluss trennt der Client die Verbindung. Dieser Dienst läuft auf Port 7.

Zwei Dinge sind hier neu:
• Zum ersten Mal versendet der Client einen String, dazu besitzt die Klasse `Verbindung` den Dienst `sende`. Um den zu sendenden String einzugeben, wird ein zusätzliches Textfeld und ein weiterer Knopf mit der Aufschrift *Senden* benötigt.
• Die Verbindung zum Server wird gehalten und erst durch den Client (`gibFrei`) getrennt. So kann man sich mehrere verschiedene Strings als Echo zurückgeben lassen. Da beide KnopfGeklickt-Dienste die Verbindung benötigen, muss sie als Bezugsobjekt global deklariert werden: `Verbindung hatVerbindung`. Beim Daytime-Server und QOTD-Server war die Verbindung ein lokales Objekt im Dienst `hatKnopfVerbinden-Geklickt`.

In Abbildung 3.2 ist eine Programmoberfläche mit dem Programmgenerator erzeugt worden.

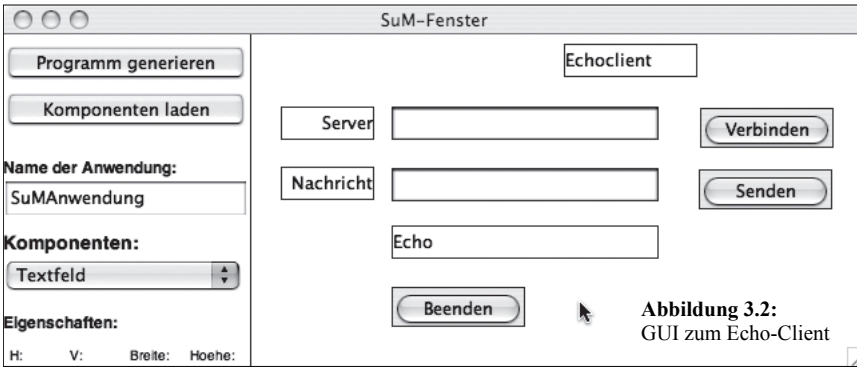

Abbildung 3.2: GUI zum Echo-Client

Übung 3.3 Schreiben Sie einen Echo-Client. Nennen Sie das Projekt `EchoClient1`. Der Echo-Port hat die Nummer 7. Als mögliche Serveradressen `mgwerl.dyndns.org` nehmen. Die Antwort soll dabei in einem Etikett angezeigt werden. Wenn mit einem anderen Echoserver verbunden werden soll, muss erst die alte Verbindung freigegeben werden.
Zusatz: Der Sendedienst soll nur ausgeführt werden, wenn wirklich eine Verbindung mit einem Echoserver hergestellt wurde. Benutzen Sie dazu ein boolsches Attribut `zVerbunden`.

Echoserver können leicht missbraucht werden. Durch ständige Anfragen mit langen Texten kann man so einen Computer lahm legen. Aus diesem Grund gibt es an Universitäten keine Echoserver mehr. Es kann auch nicht garantiert werden, dass der Echoserver auf `mgwerl.dyndns.org` noch läuft, wenn Sie mit diesem Buch arbeiten. Sie sollten dann den Netzadministrator an Ihrer Schule bitten, einen lokalen Echoserver im Intranet anzulegen.

Der hier programmierte Echoclient hat eine große Schwäche: Wenn der Auftrag

`hatEtikettEcho.setzeInhalt(hatVerbindung.empfangeneNachricht());` ausgeführt wird, ohne dass der Server eine Antwort (Echo) zurücksendet, bleibt das Programm bei `hatVerbindung.empfangeneNachricht()` hängen und lässt sich nur noch über den Rechtsklick im Barberpole des Projektfensters abbrechen. Das können Sie einfach ausprobieren, indem Sie im Echoclient die entsprechende Zeile duplizieren, so dass zweimal eine Nachricht empfangen werden soll. Andererseits gibt es aber Server, die zwei Strings zurücksenden, wie die NIST-Timeserver aus dem vorigen Abschnitt.

Deshalb soll die Klasse `Verbindung` in Zukunft **nicht** mehr benutzt werden. Die SuMNetz-Bibliothek enthält stattdessen die Klasse `Clientverbindung`. Diese Klasse ist **abstrakt**, denn Sie besitzt den abstrakten Dienst `bearbeiteNachricht()`, der in einer selbst erstellten Unterklasse implementiert werden muss. Das Wörtchen *bearbeite* zeigt dabei schon an, dass es sich hier um die Reaktion auf ein **Ereignis** handelt. Das Ereignis ist der Empfang einer Nachricht vom Server.

Das Beziehungsdiagramm für den Echoclient sieht dann folgendermaßen aus:

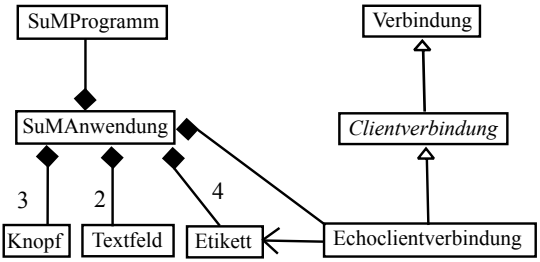

Abbildung 3.3: Beziehungsdiagramm für den Echoclient

Die `Echoclientverbindung` ist eine Unterklasse der Klasse `Clientverbindung` aus dem Paket `sum.netz`. Um die empfangene Nachricht ausgeben zu können, muss sie das Echoetikett aus der `SuMAnwendung` kennen. Der Quelltext der Klasse `Echoclientverbindung` ist nur kurz:

```
import sum.netz.*;
import sum.komponenten.*;
/**
 * @author Bernard Schriek
 * @version 28.06.2007
 */
public class Echoclientverbindung extends Clientverbindung
{
  // Bezugsobjekte
  Etikett kenntEtikettEcho;

  // Attribute

  // Konstruktor
  public Echoclientverbindung(String pServer, int pPort,
                    boolean pMitProtokoll, Etikett pEcho)
  {
    super(pServer, pPort, pMitProtokoll);
    kenntEtikettEcho = pEcho;
  }

  // Dienste
```

```
  public void bearbeiteNachricht(String pNachricht)
  {
        kenntEtikettEcho.setzeInhalt(pNachricht);
  }
}
```

Es wird nur der abstrakte Dienst `bearbeiteNachricht` aus der abstrakten Klasse `Clientverbindung` implementiert. Da die Klasse `Clientverbindung` eine Unterklasse der Klasse `Verbindung` ist, erbt sie u.a. die Dienste `sende` und `gibFrei`.

In der SuMAnwendung wird jetzt statt einer Verbindung eine Echoclientverbindung erzeugt, der das Echoetikett als vierter Parameter übergeben wird. Die Echoclientverbindung muss das Echoetikett kennen, da sie die empfangene Nachricht anzeigen muss.

Übung 3.4 Speichern Sie das EchoClient-Projekt als `EchoClient2` und implementieren Sie die ereignisorientierte Lösung mit der Klasse `Echoclientverbindung`.

3.3 Ein allgemeiner Client

Als Nächstes soll eine etwas allgemeinerer Client (`TCPClient`) entwickelt werden. Bei diesem Client soll die Portnummer des Servers in einem Textfeld eingegeben werden. Zu sendende Strings werden in eine Textfeld eingegeben und mit dem Knopf Senden verschickt. Empfangene Nachrichten sollen an ein Zeichenfeld angehängt werden. Ein Statusfeld soll anzeigen, ob gerade eine Verbindung besteht. Die Knöpfe `Verbinden` und `Trennen` sollen eine Verbindung aufbauen und trennen. Die Programmoberfläche soll folgendermaßen aussehen:

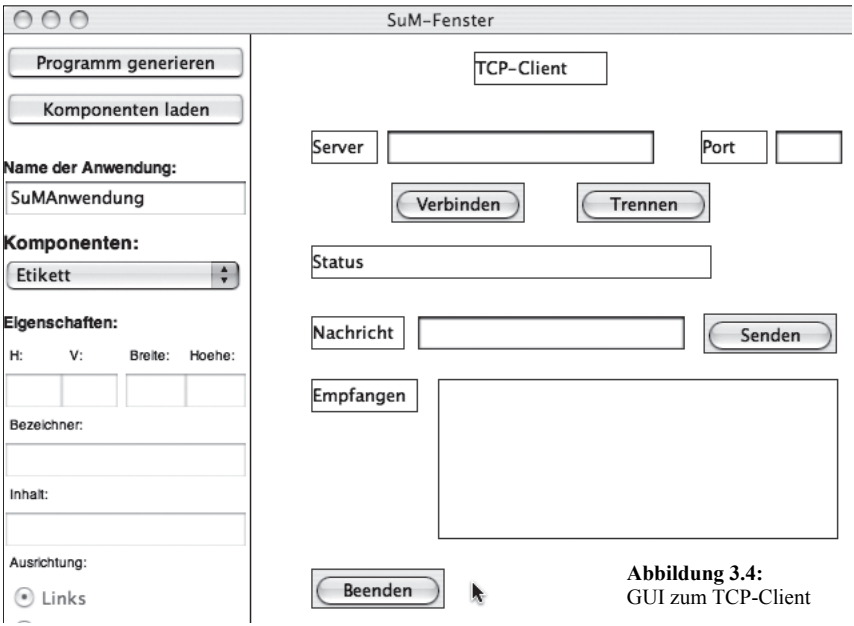

Abbildung 3.4: GUI zum TCP-Client

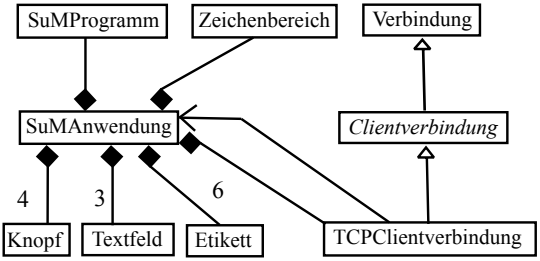

Abbildung 3.5: Beziehungsdiagramm für den TCPClient

Übung 3.5 Erzeugen Sie mit dem Prorammgenerator die Oberfläche aus Abbildung 3.4 und speichern Sie das Projekt als TCPClient1. Ergänzen Sie eine Klasse TCPClientverbindung und implementieren Sie die Dienste der SuMAnwendung. Testen Sie das Programm mit den vorher behandelten Diensten `Daytime`, `QOTD` und `Echo`. Wenn eine Verbindung hergestellt ist, soll der Knopf *Verbinden* unsichtbar werden, besteht keine Verbindung, sollen die Knöpfe *Trennen* und *Senden* unsichtbar sein.

Der Trennen-Dienst soll hier angegeben werden:

```
public void hatKnopfTrennenGeklickt()
{
  hatVerbindung.gibFrei();
  hatKnopfTrennen.verstecke();
  hatKnopfVerbinden.zeige();
  hatKnopfSenden.verstecke();
  hatEtikettStatus.setzeInhalt("nicht verbunden");
  hatVerbindung = null;
}
```

Mit Hilfe dieses Programms können jetzt beliebige Dienste und ihre Protokolle getestet werden. Dies soll am POP3-Protokoll (POP3 = Post Office Protokoll 3) gezeigt werden. Das POP3-Protokoll dient dazu Email vom Mailserver zu empfangen. Für das Senden von Email gibt es das SMTP-Protokoll, das später behandelt wird.

Um das POP3-Protokoll zu testen benötigen Sie eine gültiges Email-Konto. Falls Sie keines besitzen, können Sie bei `web.de` oder `gmx.de` ein kostenloses Email-Konto anlegen. Sie benötigen drei Informationen über ihr Email-Konto:

(1) den Benutzernamen (= Emailadresse, z.B. willi.wusel@t-online.de)

(2) das Passwort (das übrigens im Klartext übertragen wird, also sehr unsicher ist)

(3) die Adresse des POP3-Servers (z.B. pop3.t-online.de bzw. pop3.web.de usw.)

Das Pop3-Protokoll funktioniert folgendermaßen:

```
Der Client <C> meldet sich beim Server an.
Der Server <S> antwortet mit +ok und einem Zusatztext zur Informa-
tion.
<C> USER willi.wusel@web.de
<S> +ok Gib Passwort an
<C> PASS meinGeheimwort
```

```
<S> +ok Passwort wurde akzeptiert
<C> STAT
<S> +ok 5 46313  ( 5 Nachrichten mit zusammen 46313 Bytes)
<C> LIST
<S> +ok Listing folgt  (Nr. der Email und Größe in Bytes)
            1 5397
            2 2000
            3 6609
            4 7324
            5 24983
<C> LIST 2
<S> +ok 2 2000 (2. Email mit 2000 Byte)
<C> RETR 1
<S> +ok Text folgt (Mehrere Zeilen, die letzte Zeile enthält nur
einen Punkt)
<C> QUIT
<S> +ok (Server trennt danach die Verbindung)
```

Beachten Sie, dass die Emails auf dem Mailserver **nicht** gelöscht werden. Will man z.B. die zweite Email löschen, muss man den Befehl `DELE 2` senden. Sobald die Verbindung mit `QUIT` getrennt wird, löscht der Mailserver dann die zweite Nachricht. Weitergehende Informationen zum POP3-Protokoll mit zusätzlichen Befehlen finden Sie im Internet bei Wikipedia `<http://de.wikipedia.org/wiki/POP3>`. Falls Sie Informationen über die Servernamen und Anmeldeverfahren verschiedener Emaildienste benötigen, finden Sie diese bei `<http://www.patshaping.de/hilfen_ta/pop3_smtp.htm>`.

Übung 3.6 Benutzen Sie den TCPClient, um auf einem POP3-Server ihre Email abzurufen. Versuchen Sie auch eine Email zu löschen. Der POP3-Port ist 110.

Jetzt sollen Sie mit dem TCPClient eine Mail versenden. Dazu wird das *SMTP*-Protokoll (SMTP = Simple Mail Transport Protocol) benutzt. Beim Versand von Mail wird kein Passwort benötigt, allerdings haben die meisten Mailserver einen Schutzmechanismus eingebaut, so dass nur autorisierte Personen eine Mail verschicken können. So versucht man, der SPAM-Flut (SPAM = überflüssige Mail, unerwünschte Werbungsmail) Herr zu werden. Web.de benutzt die POP3-vor-SMTP-Authentifizierung. Man muss, bevor man Mail versendet, erst einmal Mail über POP3 abrufen. Bei POP3 muss man ein Passwort übermitteln. Der Mailserver merkt sich jetzt die IP-Nummer dieser POP3-Clients und für eine gewisse Zeit danach kann der gleiche Client Mail mit SMTP versenden. Es gibt auch andere Verfahren, bei denen die Passwörter allerdings erst vorher verschlüsselt werden müssen. Verschlüsselungen werden in Kapitel 7 behandelt.

Das SMTP-Protokoll funktioniert folgendermaßen:

```
Der Client <C> meldet sich beim Server smtp.web.de an.
Der Server <S> antwortet mit 220 smtp.web.de und zusätzlichen In-
formationen
<C> HELO smtp.web.de
<S> 250 smtp.web.de Hello und Angaben über die Verbindungsdaten des
Clients
<C> MAIL FROM:<willi.wusel@web.de>
<S> 250 <willi.wusel@web.de> ist syntaktisch korrekt
<C> RCPT TO:<willi.wusel@t-online.de>
<S> 250 <willi.wusel@t-online.de> Empfänger verifiziert
<C> DATA
<S> 354 Gib Nachricht ein, ende mit einer Zeile mit einem Punkt "."
<C> From: <willi.wusel@web.de>
```

```
<C> To: <willi.wusel@t-online.de>
<C> Subject: Testmail
<C> Das ist eine Testmail
<C> Das ist die 2. Zeile
<C> .
<S> 250 OK id=1I4boj-0001ya-00
<C> QUIT
<S> 221 smtp.web.de schließt die Verbindung (Server trennt danach
die Verbindung)
```

Die oben angegebene Beispielsitzung funktioniert allerdings nur, wenn Sie kurz vorher den Server mit POP3 angesprochen haben. Die Antworten des Servers beginnen immer mit einer 3-stelligen Zahl xyz. Emailprogramme werten nur diese Zahl aus. Der Zusatztext dient nur zur Hilfe, wenn man den Server anders z.B. mit dem TCP-Client anspricht.

Diese Zahlen bedeuten:

1yz Der Server hat die Nachricht akzeptiert, erwartet aber noch eine Bestätigung.
2yz Die Nachricht wurde erfolgreich bearbeitet.
3yz Die Nachricht wurde verstanden, es werden zur Verarbeitung aber noch weitere Informationen benötigt.
4yz Ein Fehler wurde festgestellt. Die Nachricht muss wiederholt werden.
5yz Schwerer Fehler. Die Nachricht kann nicht verarbeitet werden.

Genauere Informationen über die Bedeutung dieser Codes findet man in der offiziellen Beschreibung des SMTP-Protokolls. Offiziell Internetprotokolle werden immer in sogenannten *RFC-Dokumenten* beschrieben (RFC = Request For Comment). Diese Bezeichnung zeigt schon, dass solche Protokollvereinbarungen vor ihrer Verabschiedung intensiv diskutiert werden. Dies passiert bei der IETF (= Internet Engineering Task Force), deren Aufgaben auch in einem RFC-Dokument beschrieben werden. Das SMTP-Protokoll finden Sie als RFC2841 `<http://tools.ietf.org/html/rfc2821>`. Das POP3-Protokoll finden Sie als RFC1081 `<http://tools.ietf.org/html/rfc1081>`.

Übung 3.7 Benutzen Sie den TCPClient, um auf einem SMTP-Server eine Email zu senden. Beachten Sie, dass viele SMTP-Server die Verbindung nach relativ kurzer Inaktivität unterbrechen. Der SMTP-Port ist 25.

Übung 3.8 Warum wird zum Schluss einer Email immer eine Zeile mit einem einzigen Punkt übertragen? Wie kann dann man das Problem lösen, wenn man eine Zeile mit einem Punkt mitten in einer Nachricht übermitteln will. Im Internet finden Sie dazu Informationen.

Übung 3.9 Wenn der Server die Verbindung trennt, empfängt der Client statt eines Strings das Symbol `null`. Dann wird der Dienst `bearbeiteVerbindungsverlust` der Klasse `Clientverbindung` aufgerufen. Speichern Sie das Projekt als `TCPClient2` und überschreiben Sie den Dienst `bearbeiteVerbindungsverlust` in der Klasse `TCPClientverbindung`. Ein Verbindungsverlust soll wie ein Klick auf den Trennenknopf behandelt wird. Dazu muss die Klasse `TCPClientverbindung` die Klasse `SuMAnweisung` kennen.

3.4 Ein Email-Client

Ein Email-Programm (Outlook oder Eudora) funktioniert im Prinzip genau so, allerdings wird das Protokoll vor dem Benutzer versteckt. Die Anmeldung erfolgt mit Daten, die sich das Programm aus einer Einstellungsdatei holt. Sie sollen jetzt zwei Programme erstellen, einen POP3-Client und einen SMTP-Client, um komfortabel Email abzuholen und zu verschicken. Dazu sind aber ein paar Vorüberlegungen notwendig.

Wie Sie bei der Untersuchung des POP3- und des SMTP-Protokolls gesehen haben, besteht die Nachrichten von einem Server immer aus einem oder mehreren Strings. Die einzelnen Informationsteile (Worte) in den Nachrichten sind durch Leerzeichen getrennt. Beim POP3-Protokoll antwortet der Server auf die STAT-Anfrage des Clients mit einem String, der aus 3 Worten besteht: der Ok-Bestätigung, dann die Zahl der wartenden Mails und zum Schluss die Gesamtlänge in Bytes. Wenn man jetzt herausfinden will, wieviele Mails warten, muss man den 2. Teil der Nachricht herausfiltern und in eine ganze Zahl (int) umwandeln. Dazu leistet die Klasse `Textwerkzeug` aus dem Paket `sum.werkzeuge` gute Dienste. Dies soll in einem kleinen Programmausschnitt gezeigt werden:

```
Textwerkzeug hatTW;
String lNachricht;   // z.B. +ok 17 35476
int lAnzahlMails;

if (hatTW.wortanzahl(lNachricht) == 3
          && hatTW.istGleich(hatTW.wortAn(lnachricht, 1), "+ok"))
  lAnzahlMails = hatTW.alsGanzeZahl(hatTW.wortAn(lNachricht, 2));
else
  lAnzahlMails = -1;
```

Umgangssprachlich ausgedrückt:

Wenn `lNachricht` aus 3 Worten besteht und das erste Wort +ok ist,
  dann erhält `lAnzahlMails` das in eine ganze Zahl umgewandelte, zweite Wort von `lNachricht`,

ansonsten
  erhält `lAnzahlMails` den Wert -1.

Das Beziehungsdiagramm für den POP3Client könnte so aussehen:

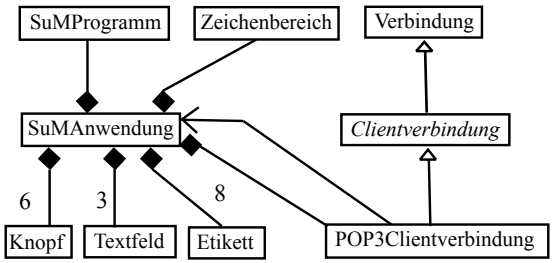

Abbildung 3.6: Beziehungsdiagramm für den POP3Client

Eine mögliche Oberfläche für den POP3Client könnte so aussehen:

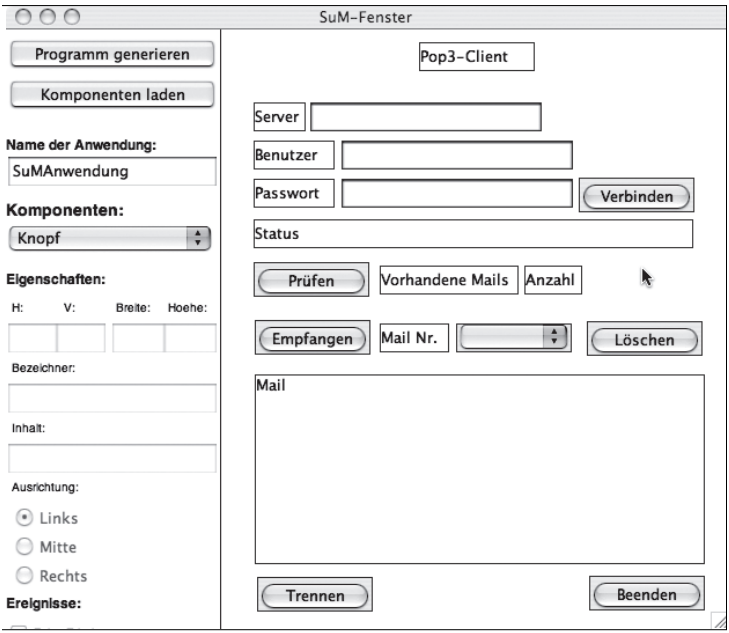

Abbildung 3.7:
GUI zum POP3-Client

Hinter dem Etikett `Mail Nr.` befindet sich eine `Auswahl`, die die Zahlen 1 bis Anzahl der Mails zur Auswahl zur Verfügung stellt. Das Textfeld für das Passwort ist ein `Kennwortfeld` (siehe `sum.komponenten`).

Ein kleines Problem tritt bei der Implementierung der Klasse `POP3Clientverbindung` auf. Wenn der Mailserver eine Nachricht schickt, wird der Dienst `verarbeiteNachricht` aufgerufen. Dabei müssen aber unterschiedliche Aufgaben erledigt werden. Wenn der Prüfenknopf geklickt wurde, sendet der Client des STAT-Befehl an den Server, die Antwort muss dann ausgewertet werden und die Auswahlkomponente muss aktualisiert werden. Wenn der Empfangenknopf geklickt wird, sendet der Client den RETR-Befehl an den Server. Dieser antwortet mit einer Reihe von Nachrichten. Für jede Zeile der Mail wird eine eigene Nachricht geschickt. Diese Zeilen müssen dann im Zeichenbereich angezeigt werden.

Für diese unterschiedlichen Reaktionen auf Nachrichten benutzt man am besten ein ganzzahliges Attribut `zStatus` in der `POP3Clientverbindung`. Der Wert 0 bedeutet dann, dass die empfangene Nachricht ignoriert werden kann. Der Wert 1 bedeutet, die Antwort auf den STAT-Befehl muss ausgewertet werden. Der Wert 2 bedeutet, dass die Nachrichten auf einen RETR-Befehl ausgewertet werden müssen. Zur Hilfe soll hier eine mögliche Implementierung des Auftrags `bearbeiteNachricht` in der `POP3Clientverbindung` angegeben werden:

```
public void bearbeiteNachricht(String pNachricht)
{
  switch (zStatus)
  {
    // Reaktion auf den STAT-Befehl
    case 1: kenntAnwendung.setzeAuswahlMax(hatTW.alsGanzeZahl
                                    (hatTW.wortAn(pNachricht, 2)));
            zStatus = 0;
            break;
    // Reaktion auf den RETR-Befehl
    case 2: if (hatTW.istGleich(pNachricht, "."))
              zStatus = 0;
            else
              kenntAnwendung.zeigeMail(pNachricht);
            break;
  }
}
```

Der Dienst `hatKnopfEmpfangenGeklickt` der Klasse `SumAnwendung` lautet dann so:

```
public void hatKnopfEmpfangenGeklickt()
{
  int lMailNr;

  if (hatAuswahl.zeilenAnzahl() > 0)
  {
    hatZeichenbereichMail.loescheAlles();
    lMailNr = hatAuswahl.index();
    hatVerbindung.setzeStatus(2);
    hatVerbindung.sende("RETR " + lMailNr);
  }
}
```

Übung 3.10 Implementieren Sie den POP3Client.

Jetzt soll ein SMTP-Client implementiert werden. Sein Beziehungsdiagramm ist dem POP3Client sehr ähnlich. Eigentlich ändert sich nur der Name der Unterklasse der Clientverbindung.

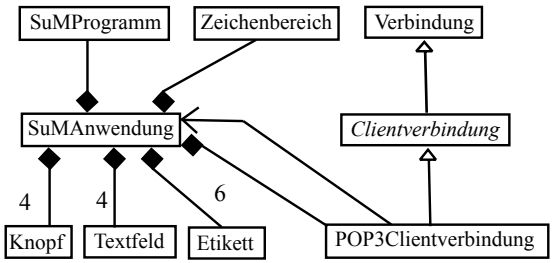

Abbildung 3.8: Beziehungsdiagramm für den SMTPClient

Es ist nicht notwendig, sich beim SMTP-Server anzumelden, deshalb entfallen die Textfelder für den Benutzer und das Passwort. Allerdings funktioniert dieser Client nur, wenn der Server die POP3-vor-SMTP-Authentifizierung zulässt (z.B. web.de), und kurz vorher die Anmeldung beim entsprechenden POP3-Server stattgefunden hat.

Die Klasse `SMTPClientverbindung` empfängt Nachrichten vom Server und schickt diese an die SuMAnwendung weiter. Die SuMAnwendung zeigt diese Nachrichten dann im Statusetikett an.

Die Oberfläche des SMTPClients soll folgendermaßen aussehen:

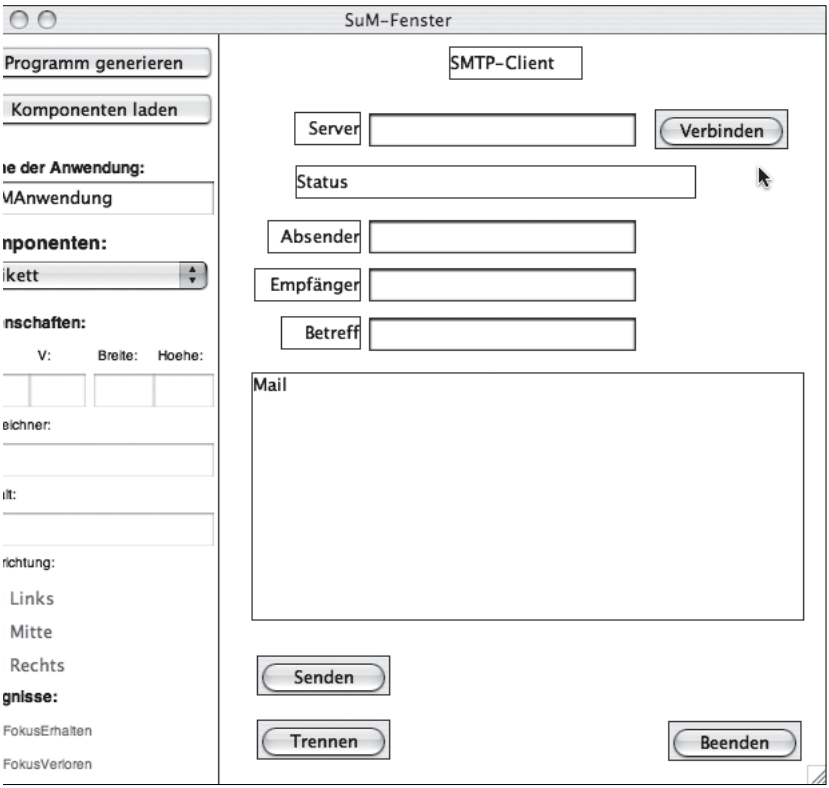

Abbildung 3.9:
GUI zum SMTP-Client

Übung 3.11 Implementieren Sie den SMTPClient.

3.5 Zusammenfassung

In diesem Kapitel haben Sie das Prinzip der Server-Client-Kommunikation kennen gelernt. Ein Server wartet, bis sich ein Client bei ihm anmeldet. Anschließend findet eine Kommunikation statt, die nach festen Regeln verläuft. Diese Regeln bezeichnet man als ein Protokoll. Die Protokolle für die Standardinternetdienste kann man in sogenannten RFCs nachlesen. Sie müssen vollständig und eindeutig sein.

Sie haben die einfachen Protokolle für die Daytime-, QOTD- und Echo-Dienste sowie die komplexen Protokolle für die POP3- und SMTP-Dienste kennen gelernt und angewandt.

Um zu verhindern, dass Programme beim Warten auf eine Nachricht vom Server hängen bleiben, wurde die abstrakte Klasse `Clientverbindung` benutzt, in deren Unterklasse der ereignisorientierte Dienst `bearbeiteNachricht` implementiert wurde. Um die Kommunikation zwischen Client und Server besser beobachten zu können, besteht die Möglichkeit, im Konstruktor der Verbindung einen Schalter zu setzen, der dafür sorgt, dass die Nachrichten im Konsolenfenster angezeigt werden. So kann man im Nachhinein eine Sitzung analysieren.

Das Projekt `TCPClient` diente dazu, einen allgemeinen Protokolltester zu erstellen. Mit diesem Programm kann interaktiv die Kommunikation mit Servern getestet werden.

Neue Begriffe in diesem Kapitel

• **Client-Server-System** Ein Server (Anbieter) bietet im Netz Dienste an, die vom Client (Kunden) in Anspruch genommen werden. Dies ist der Standard bei der Internetkommunikation. Es gibt allerdings auch Verbindungen, bei denen die beteiligten Computer gleichwertig sind (Peer-to-Peer-Verbindungen). Welchen Dienst des Servers der Client nutzen möchte, teilt er mit der IP-Nummer mit.

• **Daytime** Der Daytime-Dienst ist ein Dienst im Internet, bei dem das Datum und die Uhrzeit als String an den Client geliefert werden.

• **QOTD** Der Quote-of-the-Day-Dienst liefert Sprüche (Zitate) an den Client. Nach jeder Anmeldung eines Clients wird diesem ein zufälliger Spruch geschickt, anschließend trennt der Server die Verbindung.

• **Echo** Nach der Anmeldung eines Clients, kann dieser dem Server Nachrichten schicken, die wie eine Echo zurückgeschickt werden. Die Verbindung wird gehalten, bis sie der Client beendet,

• **POP3** Das Post Office Protokoll in der Version 3 dient dazu, Email von einem Mailserver abzurufen. Das Passwort wird unverschlüsselt übertragen.

• **SMTP** Das Simple Mail Transfer Protokoll dient dazu, Emails an einen Mailserver zu versenden. Da keine Benutzerauthentifizierung vorgesehen ist, wurden verschiedene Methoden entwickelt, um diese Schwäche zu verhindern.

• **POP3-vor-SMTP** Verfahren zu Benutzerauthentifizierung beim Emailversand. Bevor man Emails versenden kann, muss man sich vorher mit POP3 anmelden.

Kapitel 4 Netzwerkprogrammierung III

In diesem Kapitel sollen Sie lernen:

- wie man einen Netzwerkserver implementiert
- wie man Protokolle entwickelt und darstellt
- wie man mit dem Server Daten von einem Client an alle anderen Clients weiterleitet
- wie Threads funktionieren
- wie man dialogorientierte Serverprotokolle implementiert

Nachdem Sie im letzten Kapitel mehrere Netzwerkclients programmiert haben, sollen Sie in diesem Kapitel lernen, wie man Server programmiert. Zu Beginn wird ein Daytime- und ein QOTD-Server entwickelt, danach ein Echoserver, der sich sehr einfach zu einem Chatserver erweitern lässt, zu dem dann der passende Client entwickelt wird. Anschließend werden Client und Server zu zwei Netzwerkspielen entworfen. Dabei liegt der Schwerpunkt auf der Entwicklung und Dokumentation eines passenden Protokolls.

4.1 Daytime- und QOTD-Server

Ein Daytime- und ein QOTD-Server reagieren fast gleich: Sobald sich ein Client anmeldet, wird ihm eine Nachricht geschickt, anschließend trennt der Server die Verbindung. Welche Dienste sollte ein Server zur Verfügung stellen? Im Kapitel 3 haben Sie schon gesehen, dass die ereignisorientierte Sichtweise Vorteile bringt. Deshalb kann man die Frage umformulieren: Auf welche Ereignisse sollte der Server reagieren?

Ereignis	Serverdienstbezeichnung
Ein Client hat sich angemeldet	bearbeiteVerbindungsaufbau
Ein Client hat eine Nachricht geschickt	bearbeiteNachricht
Der Server hat die Verbindung beendet	bearbeiteVerbindungsende
Der Server hat die Verbindung verloren	bearbeiteVerbindungsverlust

Abbildung 4.1
Ereignisse für den Server

Zusätzlich wird natürlich auch noch der Sende-Dienst benötigt. Allerdings können bei einem Server **gleichzeitig mehrere Clients** angemeldet sein. Deshalb gibt es zwei sende-Dienste: `sendeAnEinen(String pClientIP, String pNachricht)`, bei dem die Nachricht an ein bestimmten, durch die IP-Nummer gekennzeichneten Client geschickt wird und `sendeAnAlle(String pNachricht)`, bei dem die Nachricht an alle zur Zeit angemeldeten Clients geschickt wird.

Die SuM-Netz-Bibliothek stellt eine Klasse `Server` zur Verfügung bei der die bearbeite-Dienste allerdings als leere Dienste und nicht als abstrakte Dienste implementiert sind. So ist es nicht notwendig, in einer konkreten Unterklasse der Klasse `Server` alle bear-

beite-Dienste zu implementieren.

Eigentlich benötigt ein Server keine eigene Oberfläche, da Benutzereingaben normalerweise nicht vorgesehen sind. Bei den hier entwickelten Servern soll aber zumindest ein Etikett anzeigen, um was für einen Server es sich handelt, außerdem sollen zwei Knöpfe zum Starten des Servers und zum Beenden des Programms vorhanden sein.

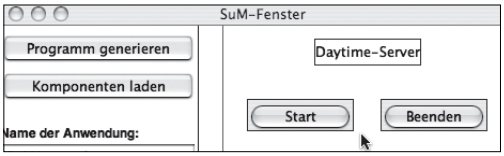

Abbildung 4.2:
GUI zum Daytime-Server

Die KnopfGeklickt-Dienste der beiden Knöpfe lautet folgendermaßen:

```
public void hatKnopfStartGeklickt()
{
  hatServer = new Daytimeserver();
}

public void hatKnopfBeendenGeklickt()
{
  if (hatServer != null)
      hatServer.gibFrei();
  this.beenden();
}
```

Das Beziehungsdiagramm zum DaytimeServer-Projekt sieht so aus:

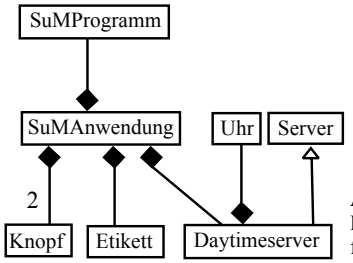

Abbildung 4.3:
Beziehungsdiagramm
für den Daytime-Server

Die meisten Betriebssysteme verhindern den Start von Servern im Portbereich 0-1024. Solche Dienste darf nur der Administrator (root) starten. Deshalb wird für den Daytime-Dienst statt Port 13 der Port 20013 und für den QOTD-Dienst Port 20017 statt 17 gewählt. Natürlich müssen dann auch die Ports der zugehörigen Clients angepasst werden.

Der Quelltext der Klasse `Daytimeserver` lautet folgendermaßen:

```
import sum.netz.*;
import sum.werkzeuge.*;
/**
 * @author Bernard Schriek
 * @version 03.07.2007
 */
```

```
public class Daytimeserver extends Server
{
  // Bezugsobjekte
  Uhr hatUhr;

  // Attribute

  // Konstruktor
  public Daytimeserver()
  {
    super(20013, true); // Port 20013 mit Protokoll
    hatUhr = new Uhr();
  }

  // Dienste
  public void bearbeiteVerbindungsaufbau(String pIPNr)
  {
    this.sendeAnEinen(pIPNr, hatUhr.datum() + " " + hatUhr.zeit());
    this.beendeVerbindung(pIPNr);
  }
}
```

Die Klasse `Uhr` wird benötigt, um Datum und Uhrzeit zu ermitteln. Im Konstruktor wird mit `super()` der Konstruktor der Oberklasse `Server` aufgerufen. Dabei werden als Parameter die Portnummer, auf der der Server laufen soll, und `true` für die Protokollierung des Netzwerkverkehrs im Konsolenfenster übergeben.

Es ist möglich auf einem Computer gleichzeitig Server und Client laufen zu lassen. Dazu müssen Sie beim Client als Serveradresse `localhost` oder `127.0.0.1` angeben. So können Sie den Server auch auf Ihrem Computer zu Hause testen. Im Computerraum der Schule kann der Server auf einem Computer und die Clients auf anderen Computern laufen. Es muss allerdings die Adresse des Serverrechners bekannt sein. Falls die Dienste auf dem Schulcomputersystemen nicht funktionieren, ist wahrscheinlich eine Firewall daran Schuld. Sie sollten dann im Intranet diese Ports freischalten lassen.

Übung 4.1 Implementieren Sie den Daytime-Server und testen Sie mit einem Daytime-Client, bei dem der Port angepasst wurde.

Ein QOTD-Server funktioniert gleichermaßen. Statt der Uhr wird allerdings ein Feld oder eine lineare Liste mit Strings (Sprüchen) benötigt, aus denen zufällig ein Spruch ausgewählt wird. Sie benötigen also auch einen Rechner für die Erzeugung von Zufallszahlen, um auszuwählen, welcher Spruch verschickt wird.

Übung 4.2 Implementieren Sie einen QOTD-Server und testen Sie mit einem QOTD-Client, bei dem der Port angepasst wurde.

4.2 Echoserver und Chatserver

Bei einem Echoserver bleibt die Verbindung bestehen, bis der Client sich abmeldet. Dann tritt für den Server das Ereignis *Verbindungsverlust* ein, auf das er aber nicht reagieren muss, da die verlorene Verbindung in der Klasse `Server` automatisch aus der Liste der offenen Verbindungen entfernt wird.

Beim Empfang einer Nachricht muss die Nachricht an den Absender zurück geschickt werden.

```
public void bearbeiteNachricht(String pIPNr, String pNachricht)
{
    this.sendeAnEinen(pIPNr, pNachricht);
}
```

Der Server ist also sehr einfach zu implementieren.

Übung 4.3 Implementieren Sie einen Echo-Server und testen Sie mit einem Echo-Client, bei dem der Port angepasst wurde (20007). Testen Sie auch, dass gleichzeitig mehrere Clients mit dem Server verbunden sein können.

Bei einem Chatsystem sind gleichzeitig mehrere Clients mit dem Chatserver verbunden. Wie beim Echoclient schickt eine Teilnehmer eine Nachricht an den Server. Der Server schickt diese Nachricht allerdings an alle angeschlossenen Clients. Damit die Clients sehen können, wer diese Nachricht schickt, wird an den Anfang der Nachricht die IP-Nummer des ursprünglichen Absenders angehängt.

```
public void bearbeiteNachricht(String pIPNr, String pNachricht)
{
    this.sendeAnAlle(pIPNr + "schreibt: " + pNachricht);
}
```

Um den Chat besser verfolgen zu können, sollten die empfangenen Nachrichten beim Client in einem Zeilenbereich dargestellt werden. Das haben Sie schon beim TCPClient so gemacht. Der Chatserver soll auf Port 20001 laufen.

Wenn sich ein neuer Client beim Chat-Server anmeldet, erhält er eine Begrüßungsnachricht. Alle angemeldeten Clients werden darüber informiert, dass sich dieser Client (IP-Nr) beim Chat angemeldet hat.

Übung 4.4 Implementieren Sie einen Chat-Server und einen Chat-Client und testen Sie das Chat-System. Nennen sie das Server-Projekt `ChatServer1`.

Es gibt jetzt viele Möglichkeiten, das Chat-System zu verbessern. Es können Benutzernamen (Nicknames) vergeben werden, Nachrichten können je nach Absender in einer anderen Farbe dargestellt werden, es können während des Chats private Nachrichten zwischen Benutzern ausgetauscht werden (whisper). Hier soll nur die Möglichkeit der Verwendung von Nicknames beschrieben werden.

Wenn ein Client sich einen Nickname geben will, z.B. 127.0.0.1 soll Superchatter heißen, muss er dies dem Client in Form einer Nachricht mitteilen. Damit der Server bei der Nachricht erkennt, dass es sich um eine Nicknameinformation handelt, muss eine Protokollvereinbarung getroffen werden: Wenn die Nachricht mit `@nick` beginnt, soll das zweite Wort der Nachricht der Nickname sein. Der Client muss also die Nachricht

```
@nick Superchatter
```

an den Server senden. Im Dienst `bearbeiteNachricht` der Klasse `Chatserver` muss bei jeder eintreffenden Nachricht überprüft werden, ob es sich um eine `@nick`-Anwei-

sung oder eine normale Nachricht handelt.

Der Server muss sich jetzt die Zuordnung 127.0.0.1 und Superchatter merken und diese den angeschlossenen Clients mitteilen. Wenn Superchatter später eine Nachricht an den Server schickt, weiß dieser, von welcher IP-Nummer die Nachricht kommt. Jetzt muss der Chatserver den zu dieser IP-Nummer zugehörigen Nickname suchen und die Nachricht mit der Nicknamekennung an alle angeschlossenen Clients weiterleiten. Welche Datenstruktur eignet sich zum Verwahren dieser Zuordnungen? Da es sich um ein Suchproblem handelt, bietet sich ein Suchbaum und eine Hashtabelle an. Da es nicht auf die Reihenfolge der IP-Nummern ankommt, ist die Hashtabelle die geeignete Struktur. Der Schlüssel ist dann die IP-Nummer. Die Zuordnung soll in Objekten einer Klasse `Nick` verwahrt werden, diese muss den Dienst `schluessel` des Interface `Schluesselobjekt` implementieren. Dann können Objekte der Klasse `Nick` in einer Hashtabelle verwahrt werden.

```
import sum.strukturen.*;
/**
 * @author Bernard Schriek
 * @version 04.07.2007
 */
public class Nick implements Schluesselobjekt
{
  // Bezugsobjekte

  // Attribute
  String zIPNr;
  String zNickname;

  // Konstruktor
  public Nick(String pIPNr, String pNickname)
  {
    zIPNr = pIPNr;
    zNickname = pNickname;
  }

  // Dienste
  public Object schluessel()
  {
    return zIPNr;
  }

  public String nickname()
  {
    return zNickname;
  }
}
```

Auch die Klasse `Chatserver` soll hier angegeben werden, um als Anregung für weitere Erweiterungen zu dienen:

```
import sum.netz.*;
import sum.strukturen.*;
import sum.werkzeuge.*;
/**
 * @author Bernard Schriek
 * @version 03.07.2007
 */
public class Chatserver extends Server
{
```

```
  // Bezugsobjekte
  Hashtabelle hatHash;
  Textwerkzeug hatTW;

  // Attribute

  // Konstruktor
  public Chatserver()
  {
    super(20001, true);
    hatHash = new Hashtabelle();
    hatTW = new Textwerkzeug();
  }

  // Dienste
  public void bearbeiteVerbindungsaufbau(String pIPNr)
  {
    this.sendeAnEinen(pIPNr, "Willkommen beim SuM-Chat.");
    this.sendeAnAlle(pIPNr + " hat sich beim Chat angemeldet.");
  }

  public void bearbeiteVerbindungsverlust(String pIPNr)
  {
    Nick lNick;

    lNick = (Nick) hatHash.suche(pIPNr);
    if (lNick != null)
    {
      this.sendeAnAlle(lNick.nickname() + " hat sich abgemeldet.");
      hatHash.loesche(pIPNr);
    }
    else
      this.sendeAnAlle(pIPNr + " hat sich beim Chat abgemeldet.");
  }

  public void bearbeiteNachricht(String pIPNr, String pNachricht)
  {
    Nick lNickAlt, lNickNeu;

    if (hatTW.wortanzahl(pNachricht) == 2 &&
        hatTW.istGleich(hatTW.wortAn(pNachricht, 1), "@nick"))
    {
      lNickNeu = new Nick(pIPNr, hatTW.wortAn(pNachricht, 2));
      lNickAlt = (Nick) hatHash.suche(pIPNr);
      if (lNickAlt != null)
        hatHash.loesche(pIPNr);
      hatHash.fuegeEin(lNickNeu);
      this.sendeAnAlle(pIPNr + " heisst jetzt: " +
                       lNickNeu.nickname());
    }
    else
    {
      lNickAlt = (Nick) hatHash.suche(pIPNr);
      if (lNickAlt != null)
        this.sendeAnAlle(lNickAlt.nickname() + " schreibt: " +
                                               pNachricht);
      else
        this.sendeAnAlle(pIPNr + " schreibt: " + pNachricht);
    }
  }
}
```

Übung 4.5 Speichern Sie das Projekt `ChatServer1` als `Chatserver2` und implementieren Sie die Nickname-Verwaltung mit Hashtabellen. Der Client bleibt unverändert.

Übung 4.6 Speichern Sie das Projekt `ChatServer2` als `Chatserver3` und implementieren Sie die whisper-Funktion, bei der nur ein augewählter Client die Nachricht empfängt. Die entsprechenden Anweisungen heißen `@whisper Superchatter` bzw. `@unwhisper Superchatter`.

Es ist auch relativ einfach einen Kick-Befehl zu implementieren, mit dem ein Client aus dem Chat herausgeworfen wird.

4.3 Malen im Netz

Als nächstes Projekt soll ein Client-Server-System entwickelt werden, bei dem mehrere Clients gleichzeitig an einem Bild malen. Das Malen soll so funktionieren, wie das Freihandzeichnen aus Band I. Allerdings soll das Bild gleichzeitig auf den Bildschirmen aller angemeldeten Clients entstehen.

Bei der Entwicklung des Protokolls ist zu berücksichtigen, dass der Server möglichst wenig eigene Arbeit leisten soll. Wenn möglich, sollen die Zeichenanweisungen an alle angeschlossenen Clients weiter gereicht werden. Dadurch verkraftet der Server mehr Clients.

Das Protokoll soll an einem kleinen Beispiel erläutert werden:

```
Client1 <C1> mit IP 10.0.0.51 meldet sich beim Server <S> an.   (1)
<S> an alle: 10.0.0.51 @neu

Client2 <C2> mit IP 10.0.0.52 meldet an.                        (2)
<S> an alle: 10.0.0.52 @neu
<S> an <C2>: 10.0.0.51 @neu

Client3 <C3> mit IP 10.0.0.53 meldet sich beim Server <S> an.   (3)
<S> an alle: 10.0.0.53 @neu
<S> an <C3>: 10.0.0.51 @neu
<S> an <C3>: 10.0.0.52 @neu

<C1>: @pos 100 200                                              (4)
<S> an alle: 10.0.0.51 @pos 100 200

<C1>: @runter                                                   (5)
<S> an alle: 10.0.0.51 @runter

<C3>: @pos 123 234                                              (6)
<S> an alle: 10.0.0.53 @pos 123 234

<C3>: @runter                                                   (7)
<S> an alle: 10.0.0.53 @runter

<C2>: @farbe 7                                                  (8)
<S> an alle: 10.0.0.52 @farbe 7

<C1>: @breite 4                                                 (9)
<S> an alle: 10.0.0.51 @breite 4
<C2> beendet das Programm und trennt so die Verbindung         (10)
<S> an alle: 10.0.0.52 @getrennt
```

```
<C3> @pos 70 30                                                    (11)
<S> an alle: 10.0.0.53 @pos 70 30

Client4 <C4> mit IP 10.0.0.54 meldet sich an.                      (12)
<S> an alle: 10.0.0.54 @neu
<S> an <C4>: 10.0.0.51 @neu
<S> an <C4>: 10.0.0.53 @neu
```

Bei den Protokollteilen (1), (2), (3) und (12) meldet sich ein neuer Client an. Der Server informiert im Dienst `bearbeiteVerbindungsaufbau` alle Clients (`sendeAnAlle`) über diesen neuen Client, also auch den neuen Client. Bei (2), (3) und (12) sind noch andere Clients angemeldet, also wird der neue Client über diese Clients informiert. Dies ist eine Aufgabe des Servers, der die angeschlossenen Clients in einer Datenstruktur verwalten muss, die dynamisch ist, also wachsen und schrumpfen kann, und die einfach durchlaufen werden kann. Hier bietet sich die Datenstruktur `Liste` an. Die Liste enthält dann Strings mit den IP-Nummern der angemeldeten Clients.

Wenn sich ein Client abmeldet (10), werden alle Clients informiert und die IP-Nr des abgemeldeten Clients wird aus der Liste entfernt. Dies wird im Dienst `bearbeiteVerbindungsverlust` des Servers erledigt.

Die anderen Protokollteile dienen der Übermittlung der Zeichenbefehle bzw. Anweisungen an den Stift. Der Server versieht die Anweisungen mit der IP-Nummer des Absenders und sendet sie an alle Clients weiter. Dies erledigt der Server im Dienst `bearbeiteNachricht`.

Das Beziehungsdiagramm für den Server sieht dann folgendermaßen aus:

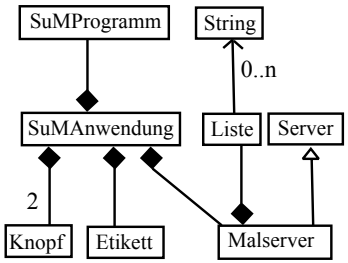

Abbildung 4.4: Beziehungsdiagramm für den Malserver

Der Client hat zwei Aufgabenbereiche:

1. Die Maus wird ständig überprüft und die entsprechenden Zeichenbefehle werden an den Server geschickt. Es wird nicht direkt gezeichnet.

2. Die vom Server ankommenden Zeichenbefehle werden an den/die Stifte weitergeleitet. Dabei ist es einfacher, für jeden Client einen eigenen Stift zu benutzen, da die Positions-, Farb- und Linienbreitenverwaltung stark vereinfacht wird. Zur IP-Nr, die vom Server geliefert wird, muss dann allerdings der passende Stift ermittelt werden. Dies geschieht in einer Hashtabelle.

Da die SuMAnwendung eine Ereignisanwendung ist, erbt sie die Ereignisbearbeiterdienste `bearbeiteMausDruck`, `bearbeiteMausLos`, `bearbeiteMausBewegt` und

`bearbeiteTaste`. Damit können dann die Zeichenanweisungen generiert werden. Dies soll an drei kleinen Programmausschnitten (der SuMAnwendung des Clients) gezeigt werden:

```
public void bearbeiteMausDruck(int pH, int pV)
{
  if (zVerbunden)
  {
    hatVerbindung.sende("@pos " + pH + " " + pV);
    hatVerbindung.sende("@runter");
    zUnten = true;
  }
}

public void bearbeiteMausBewegt(int pH, int pV)
{
  if (zVerbunden && zUnten)
    hatVerbindung.sende("@pos " + pH + " " + pV);
}

public void bearbeiteTaste(char pTaste)
{
  if (zVerbunden)
    switch (pTaste)
    {
      case 'r': hatVerbindung.sende("@farbe " + Farbe.ROT); break;
      case ...
      ...
    }
}
```

Da MausBewegt-Ereignisse nur übermittelt werden müssen, wenn die Maus unten ist, also gezeichnet wird, wird ein boolsches Attribut `zUnten` benötigt. Wenn die Maus gedrückt wird, muss der Stift deshalb erst an die entsprechende Position bewegt werden. Deshalb werden zwei Nachrichten (`@pos` und `@runter`) generiert.

Für jeden am Server angemeldeten Client wird beim Client ein eigener Stift verwaltet. Die Zuordnung IP-Nummer - Stift erledigt eine Hashtabelle. Dies wird wie bei der Nicknameverwaltung im Chatserver (siehe Seite 44 und 45) erledigt. Sie benötigen also eine Klasse `Zeichner`, die diese Zuordnung verwaltet und das Interface `Schluesselobjekt` implementiert ähnlich der Klasse `Nick` im Chatserver.

Der Dienst `bearbeiteNachricht` des Malclients empfängt die Nachrichten, spaltet sie auf in IP-Nr, Anweisung und Parameter. Dann wird in den passenden Hilfsdienst verzweigt. Dabei leistet ein Textwerkzeug gute Dienste.

```
public void bearbeiteNachricht(String pNachricht)
{
  String lAnweisung, lIPNr;
  int lParam1, lParam2;

  lParam1 = 0;
  lParam2 = 0;
  lIPNr = hatTW.wortAn(pNachricht, 1);
  lAnweisung = hatTW.wortAn(pNachricht, 2);
  if (hatTW.wortanzahl(pNachricht) >= 3)
    lParam1 = hatTW.alsGanzeZahl(hatTW.wortAn(pNachricht, 3));
  if (hatTW.wortanzahl(pNachricht) == 4)
```

```
    lParam2 = hatTW.alsGanzeZahl(hatTW.wortAn(pNachricht, 4));
  if (hatTW.istGleich(lAnweisung, "@neu"))
    this.verarbeiteNeu(lIPNr);
  else if (hatTW.istGleich(lAnweisung, "@getrennt"))
    this.verarbeiteGetrennt(lIPNr);
  else if (hatTW.istGleich(lAnweisung, "@pos"))
    this.verarbeitePos(lIPNr, lParam1, lParam2);
  else if (hatTW.istGleich(lAnweisung, "@hoch"))
    this.verarbeiteHoch(lIPNr);
  else if (hatTW.istGleich(lAnweisung, "@runter"))
    this.verarbeiteRunter(lIPNr);
  else if (hatTW.istGleich(lAnweisung, "@farbe"))
    this.verarbeiteFarbe(lIPNr, lParam1);
  else if (hatTW.istGleich(lAnweisung, "@breite"))
    this.verarbeiteBreite(lIPNr, lParam1);
}
```

Ein Hilfsdienst soll beispielhaft angegeben werden. Zuerst wird aus der Hashtabelle das zur IP-Nummer passende Zeichner-Objekt geholt. Dieses liefert dann den zugehörigen Stift, der daraufhin bewegt wird.

```
public void verarbeitePos(String pIPNr, int pH, int pV)
{
  Zeichner lZeichner;
  Buntstift lStift;

  lZeichner = (Zeichner) hatHash.suche(pIPNr);
  lStift = lZeichner.stift();
  lStift.bewegeBis(pH, pV);
}
```

Zum Schluss soll noch das Beziehungsdiagramm zum Malclient angegeben werden:

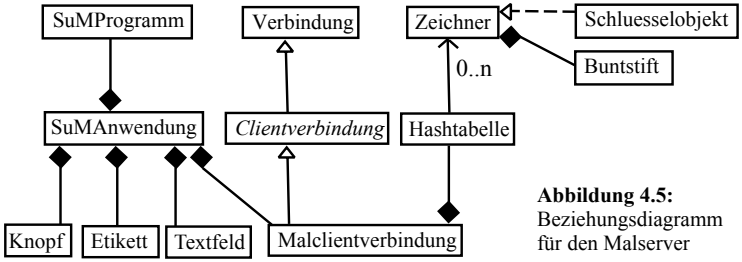

Abbildung 4.5: Beziehungsdiagramm für den Malserver

Das Etikett, das Textfeld und der Knopf dienen zur Anmeldung am Malserver, sie werden nach erfolgreicher Anmeldung unsichtbar gemacht. Das Programm wird durch Schließen des SuMFensters beendet.

Der Dienst `verarbeiteNeu` erzeugt ein neues Zeichnerobjekt, das dann in der Hashtabelle verwahrt wird.

```
public void verarbeiteNeu(String pIPNr)
{
  Zeichner lZeichner;

  lZeichner = new Zeichner(pIPNr);
```

```
  hatHash.fuegeEin(lZeichner);
}
```

Das Klassendiagramm der Klasse `Zeichner` ist einfach:

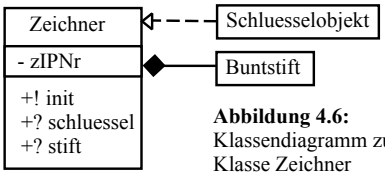

Abbildung 4.6:
Klassendiagramm zur Klasse Zeichner

Übung 4.7 Implementieren Sie arbeitsteilig den Malserver und den Malclient.

Übung 4.8 Erweitern Sie den Malclient so, dass mit dem Druck auf die Leertaste das bis dahin gezeichnete Bild gelöscht wird. Warum muss am Malserver nichts geändert werden?

4.4 Das Spiel Fuchsjagd

Zum Schluss sollen Sie noch ein Netzwerkspiel mit dem Namen *Fuchsjagd* implementieren. Bei diesem Spiel müssen sich fünf Clients beim Server anmelden. Sobald der fünfte Client sich angemeldet hat, erscheinen (auf allen Clientbildschirmen gleich) fünf Punkte an zufälliger Position. Vier Punkte sind schwarz, das sind die Jäger, ein Punkt ist rot, das ist der Fuchs. Jeder dieser Punkte gehört zu einem Client, allerdings weiß kein Client am Anfang, welcher Punkt zu ihm gehört. Dies muss man erst herausfinden, indem man seinen Punkt bewegt. Dazu werden die vier Pfeiltasten benutzt.

Die vier Jäger müssen nun versuchen den Fuchs zu fangen. Keiner der Punkte darf den Bildschirmrand überschreiten, entsprechende Tastendrücke sollen ignoriert werden. Sobald sich ein Jäger dem Fuchs nah genug genähert hat, also wenn die Punkte sich überschneiden, hat dieser Jäger gewonnen und das Spiel beginnt von neuem. Man kann auch eine Zeitbeschränkung einbauen, z.B. wenn der Fuchs nach zwei Minuten noch nicht gefangen wurde, hat dieser gewonnen. Besonders gute Schülergruppen können noch eine Punkteverwaltung einbauen.

Dieses Spiel sollen Sie weitgehend allein entwickeln. Dabei sollen Sie zuerst gemeinsam das Protokoll planen, dann überlegen, wie Sie die Arbeit auf die einzelnen Gruppenmitglieder verteilen und erst zum Schluss implementieren. Es hat sich bewährt, einen Gruppenleiter zu wählen, der nicht selbst implementiert sondern die Arbeit der anderen Gruppenmitglieder überwacht und koordiniert. Die folgenden Fragen sollten Sie in der Planungsphase beantworten:

- Welche Aufgaben sollte der Server, welche der Client übernehmen.
- Wie soll das Protokoll aussehen? Ein Beispiel aufschreiben!
- Wie wird verhindert, dass sich ein sechster Client anmeldet?
- Wie wird erreicht, dass das Spiel erst nach der 5. Anmeldung startet?
- Wo gibt es Ähnlichkeiten mit dem Malserver - Malclient?

Übung 4.9 Implementieren Sie arbeitsteilig den Fuchsjagdserver und -client.

Übung 4.10 Ein einfaches Netzwerkspiel ist Zahlenraten. Der Server denkt sich eine Zahl und die Clients raten. Der Server antwortet mit *zu klein*, *zu groß* und *richtig*.

Übung 4.11 Auch das Spiel Hangman kann man als Netzwerkspiel implementieren.

4.5 Nebenläufigkeiten

Bei den bis jetzt behandelten Beispielen wurde ein Problem außer Acht gelassen: Was passiert, wenn ein Server von mehreren Clients Anfragen nach größeren Datenmengen erhält. Zum Beispiel wollen Sie mehrere Clients von einem FTP-Server größere Dateien runterladen, oder mehrere Benutzer fordern von einem Webserver komplexe Webseiten an, der Übertragung länger dauert.

Es ist wünschenswert, dass ein Server gleichzeitig die Anforderungen mehrerer Clients bearbeiten kann. Es soll also ein kleiner Teil der Daten an Client 1, dann ein kleiner Teil der Daten an Client 2, dann wieder Daten an Client 1 usw. übertragen werden.

Ein ähnliches Problem haben Betriebssysteme, wenn auf einem Computer gleichzeitig mehrere Programme laufen. Sie laden im Hintergrund eine Datei runter und arbeiten gleichzeitig mit einer Textverarbeitung. Dies ist ein relativ altes Problem bei Betriebssystemen und wird mit sogenannten Threads gelöst. Ein *Thread* (dt. Faden) ist ein Teil eines Programms, der unabhängig von anderen Programmteilen läuft. Die Arbeit des Prozessors wird in kleine Zeitabschnitte unterteilt, in denen jeweils ein Teil eines Threads bearbeitet wird. Man spricht hier von einem *Zeitscheibenverfahren* (engl. Timeslice). Dadurch dass die Zeitabschnitte, in denen ein Thread bearbeitet wird relativ klein sind, hat der Benutzer den Eindruck, dass die Threads parallel (nebenläufig) bearbeitet werden.

Der Begriff *Thread* wird auch in Diskussionsforen benutzt und bezeichnet damit die Folge von Beiträgen zu einem bestimmten Thema.

In Java lassen sich Threads relativ einfach programmieren. Entweder benutzt man die Klasse `Thread` direkt oder man bildet Unterklassen von ihr. Es gibt auch die Möglichkeit, das `Interface Runnable` zu implementieren (Mehrfachvererbung). Die Wirkungsweise von Threads soll an einem einfachen Beispiel verdeutlicht werden.

Im Projekt `ThreadDemo1` sollen drei Threads erzeugt werden. Diese Threads sind fast identisch, sie schreiben jeweils eine Zahl in das Konsolenfenster. Allerdings schreibt der erste Thread EINS, der zweite Thread ZWEI und der dritte Thread das Wort DREI. Die einzelnen Threads sollen über Knöpfe gestartet und angehalten werden. In Etiketten soll angezeigt werden, ob ein Thread gerade läuft. Damit der Server diese Zahlen nicht zu schnell sendet, soll vor jeder Sendung eine kurze Zeit gewartet werden. Dies kann man mit dem `sleep`-Dienst der Klasse `Thread` ereichen.

```
try
{
  this.sleep(500);
} catch (InterruptedException e) {}
```

Die Programmoberfläche soll folgendermaßen aussehen:

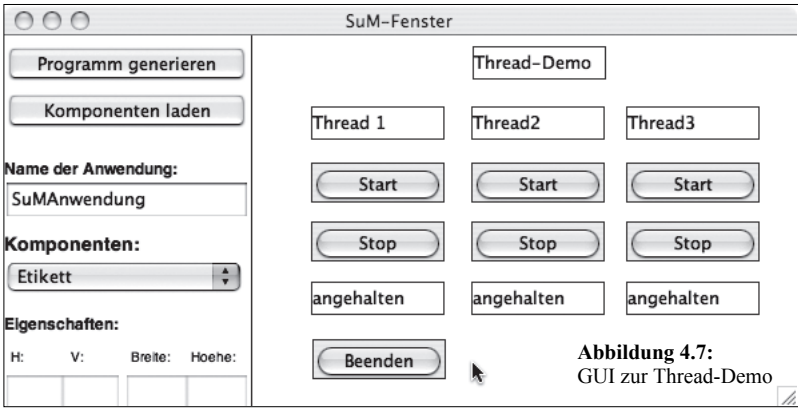

Abbildung 4.7:
GUI zur Thread-Demo

Die Klasse `Thread1` wird folgendermaßen implementiert.

```
/**
 * @author Bernard Schriek
 * @version 05.07.2007
 */
public class Thread1 extends Thread
{
  // Bezugsobjekte

  // Attribute
  boolean zLaeuft;

  // Konstruktor
  public Thread1()
  {
    super();
    zLaeuft = false;
  }

  // Dienste
  public void run()
  {
    zLaeuft = true;
    while (zLaeuft)
    {
      try
      {
        this.sleep(500);
      } catch (InterruptedException e) {}
      System.out.println("EINS"); // ins Konsolenfenster
    }
  }

  public void halteAn()
  {
    zLaeuft = false;
  }
}
```

Der zentrale Dienst eines Threads heißt `run`. Er wird von außen (der SuMAnwendung) mit der Anweisung `start` gestartet. In diesem Dienst `run` befindet sich eine Schleife, die solange durchlaufen wird, bis das boolsche Attribut `zLaeuft` auf `false` gesetzt wird.

In der SuMAnwendungen sehen die Dienste der Start- und Stop-Knöpfe folgendermaßen aus (hier für Thread1)

```
public void hatKnopfStart1Geklickt()
{
  hatThread1.start();
  hatEtikettInfo1.setzeInhalt("laeuft");
}

public void hatKnopfStop1Geklickt()
{
  hatThread1.halteAn();
  hatEtikettInfo1.setzeInhalt("angehalten");
}
```

Übung 4.12 Erzeugen Sie mit dem Programmgenerator die Oberfläche aus Abbildung 4.7 und speichern Sie das Projekt als `ThreadDemo1`. Implementieren Sie die Klassen `Thread1`, `Thread2` und `Thread3` und die `SuMAnwendung`.

Sie werden feststellen, dass man einen angehaltenen (beendeten) Thread nicht wieder neu starten kann, eine Ausweg besteht darin, dass die Threads nicht im Konstruktor der SuMAnwendung sondern im Geklickt-Dienst des Start-Knopfs neu erzeugt werden. So wird jeweils ein neuer Thread erzeugt, der dann mit `start` gestartet werden kann.

```
public void hatKnopfStart1Geklickt()
{
    hatThread1 = new Thread1();
    hatThread1.start();
    hatEtikettInfo1.setzeInhalt("laeuft");
}
```

Übung 4.13 Speichern Sie das Projekt als `ThreadDemo2` und ändern Sie die `hatKnopfStartGeklickt`-Dienste, so dass jeweils neue Threads erzeugt werden.

Wenn Sie das Programm beenden, ohne die Threads angehalten zu haben, sehen Sie, dass die Threads weiterlaufen, obwohl das dazu gehörende Programm beendet ist. Das kann man schnell testen, indem man die Konsole zwischendurch mal löscht. Um diesen Effekt zu verhindern, sollten Sie im Dienst des Beenden-Knopfs die Dienste der Stop-Knöpfe aufrufen, so dass die Threads bei Programmende auch beendet werden.

```
public void hatKnopfBeendenGeklickt()
{
  this.hatKnopfStop1Geklickt();
  this.hatKnopfStop2Geklickt();
  this.hatKnopfStop3Geklickt();
  this.beenden();
}
```

Wie können jetzt Threads dem Server bei der Kommunikation mit Clients helfen? Dazu soll ein Projekt entwickelt werden, bei dem der Server einem angemeldeten Client 100-

mal seine IP-Nr und die zugehörige Portnummer schickt. So wird eine länger dauernde Datenübertragung zum Client erreicht. Wie immer wird eine Unterklasse der Klasse `Server` gebildet und im Dienst `bearbeiteVerbindungsaufbau` dieser Unterklasse wird 1000-mal die IP-Nummer an den Client geschickt.

Übung 4.14 Erzeugen Sie ein Projekt `IPServer`, mit einem Server, der jedem Client bei Anmeldung 100-mal dessen IP-Nummer sendet. Zwischendurch soll der Server mit `sleep` warten. Testen Sie den Server mit mehreren parallelen Clients, indem Sie den TCP-Client aus Abschnitt 3.3 benutzen. Der Server soll auf Port 20003 laufen.

Sie können erkennen, dass der Server mit Threads arbeitet, denn die Daten werden quasi parallel vom Server an die verschiedenen Clients übertragen. Die Klasse `Server` benutzt nämlich die Klasse `Serververbindung`, die eine Unterklasse der Klasse `Verbindung` ist. Die Klasse `Verbindung` ist wieder eine Unterklasse der Klasse `Thread`. Die Serververbindung informiert den Server und damit seine Unterklasse (z.B. Malserver), wenn eine Nachricht eingegangen ist. Jedesmal, wenn ein Client sich anmeldet, wird eine neue Serververbindung erstellt. Der Server verwaltet die Serververbindungen in einer Liste, die abgearbeitet wird, wenn eine Nachricht an alle Clients geschickt werden soll.

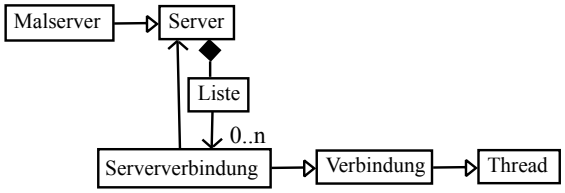

Abbildung 4.8: Beziehungsdiagramm Server - Thread

4.6 Dialogorientierte Protokolle

Bei der bis jetzt benutzten Art der Programmierung ist es allerdings nicht möglich, Server zu programmieren, die stark dialogorientiert sind. Dies ist zum Beispiel der Fall beim POP3-Server. Zuerst muss sich der Client mit der USER-Anweisung identifizieren, danach wird mit der PASS-Anweisung das Passwort übermittelt und erst dann kann der Client die RETR-Anweisung zum Abholen der Emails verwenden. Da ein POP3-Server zu aufwändig ist, soll hier ein einfacheres dialogorientiertes System entwickelt werden.

Es soll ein Matheserver entwickelt werden, der den Clients verschiedene zufällige Additionsaufgabe stellt, z.B. 17 + 38 =. Der Client muss die Lösung angeben und erhält dann eine Rückmeldung (richtig oder falsch). Für richtig gelöste Aufgaben erhält er einen Punkt. Dann wird der Client gefragt, ob er eine neue Aufgabe rechnen will. Falls er mit Nein antwortet, teilt ihm der Server mit, wieviel Prozent der Aufgaben richtig gelöst wurden, und trennt danach die Verbindung.

Übung 4.15 Überlegen Sie, warum lässt sich der Matheserver nicht mit der bisher üblichen ereignisorientierten Form der Serverprogrammierung implementieren. Was würde schief gehen?

Der Server muss sich für jeden angemeldeten Client merken, wieviel Punkte er hat, und in welcher Phase des Protokolls er gerade ist. Wenn eine Lösungszahl empfangen wird, muss geprüft werden, welche Aufgabe der Client vorher erhalten hat. Diese Daten muss man für jeden Client in einer eigenen Datenstruktur verwalten.

Der Algorithmus wäre viel einfacher, wenn der Server sich nur mit einem einzigen Client unterhalten müsste. Und genau das kann man durch den Einsatz von Threads erreichen. Dazu wird eine Unterklasse `Matheserververbindung` der Klasse `Serververbindung` und damit der Klasse `Thread` benötigt. Der entscheidende Dienst in dieser Klasse heißt `bearbeiteProtokoll`. Er ist eine Dienst der Klasse `Seververbindung`, der in der Klasse `Matheserververbindung` überschrieben wird.

```
import sum.netz.*;
import sum.werkzeuge.*;
/**
 * @author Bernard Schriek
 * @version 08.07.2007
 */
public class Matheserververbindung extends Serververbindung
{
  // Bezugsobjekte
  private Rechner hatRechner;
  private Textwerkzeug hatTW;

  // Attribute

  // Konstruktor
  public Matheserververbindung()
  {
    super();
    hatRechner = new Rechner();
    hatTW = new Textwerkzeug();
  }

  // Dienste
  public void bearbeiteProtokoll()
  {
    int lPunkte, lAufgabenzahl, lZahl1,lZahl2, lSumme;
    String lAntwort;
    boolean lNochmal;

    lPunkte = 0;
    lAufgabenzahl = 0;
    this.sende("Willkommen beim Matheserver!");
    do
    {
      lAufgabenzahl++;
      lZahl1 = hatRechner.ganzeZufallszahl(11, 49);
      lZahl2 = hatRechner.ganzeZufallszahl(11, 49);
      lSumme = lZahl1 + lZahl2;
      this.sende("Berechne " + lZahl1 + " + " + lZahl2);
      lAntwort = this.empfangeneNachricht();
      if (hatTW.istGanzeZahl(lAntwort) &&
          hatTW.alsGanzeZahl(lAntwort) == lSumme)
      {
        lPunkte++;
        this.sende("Richtig");
      }
      else
        this.sende("Falsch");
```

```
      this.sende("Neue Aufgabe? (j/n)");
      lAntwort = this.empfangeneNachricht();
      lNochmal = hatTW.istGleich(lAntwort, "j");
    } while (lNochmal);
    this.sende("Ergebnis: " + lPunkte * 100 / lAufgabenzahl
               + "% waren richtig.");
    this.sende("Auf Wiedersehen");
    this.beendeVerbindung();
  }
}
```

Es bleibt noch eine kleines Problem: Wenn sich ein neuer Client beim Matheserver anmeldet, erzeugt dieser eine neue `Serververbindung`. Das erledigt die Oberklasse `Server` automatisch. Es soll aber eine neue `Matheserververbindung` erzeugt werden, damit der dort implementierte Dienst `bearbeiteProtokoll` ausgeführt wird. Dies wird erreicht, indem man den Dienst `neueSerververbindung` der Klasse `Server` in der Unterklasse `MatheServer` überschreibt.

```
public Serververbindung neueSerververbindung()
{
    return new Matheserververbindung();
}
```

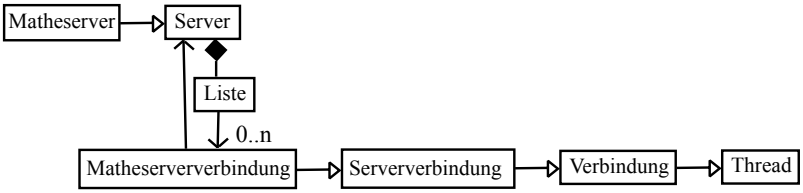

Abbildung 4.9:
Beziehungsdiagramm
Matheserver - Matheserververbindung

Übung 4.16 Erzeugen Sie ein neues Projekt `MatheServer1` und implementiern Sie die Klassen `Matheserver` und `Matheserververbindung`. Der Server soll auf Port 20004 laufen. Testen Sie den Server mit mehreren TCPClients.

Übung 4.17 Speichern Sie das Projekt `MatheServer1` als `MatheServer2`. Erweitern Sie das Protokoll, so dass der Client zu Beginn gefragt wird, ob er Additions- oder Multiplikationsaufgaben lösen will.

4.7 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie man Netzwerkserver implementiert. Zuerst wurden einfache Server implementiert, zum Schluss komplexe Server und Clients. Dabei ist es wichtig, sich Gedanken über die Aufgabenverteilung zu machen. Der Server sollte möglichst entlastet werden, wenn er gleichzeitig mit vielen Clients kommuniziert.

Eine wichtige Aufgabe übernimmt das Protokoll. Zu den Standarddiensten (POP3, SMTP usw.) kann man die Protokolle in sogenannten RFCs nachlesen. Wenn man eine

eigenes Protokoll entwickelt, ist es wichtig, dass das Protokoll vollständig und eindeutig ist. Oft werden Sonderfälle vergessen.

Wenn der Client Nachrichten von einem Client an andere Clients weiterreicht (z.B. bei einem Chatserver), ist der Dienst `sendeAnAlle` der Klasse `Server` sehr hilfreich. Falls aber ein stark dialogorientiertes Protokoll (.z.B. bei einem POP3-Server) benutzt wird, muss man eine Unterklasse der Klasse `Serververbindung` benutzen, in der im Dienst `bearbeiteProtokoll` dieses Protokoll implementiert wird.

Damit Serververbindungen parallel arbeiten können, werden Threads benutzt, deren grundlegende Eigenschaften Sie ebenfalls kennen gelernt haben.

Alle in diesem Kapitel behandelten Beispiele benutzen als Transportschicht das TCP-Protokoll.

Neue Begriffe in diesem Kapitel

• **Nebenläufigkeit** Mehrere Teile eines Programms arbeiten quasi parallel, indem der Prozessor jedem Programmteil für eine kleine Zeitspanne zur Verfügung steht.

• **Thread** (dt. Faden) Java-Klasse zur Implementierung von Nebenläufigkeiten. Dabei benutzen Threads den Dienst `run`, in dem steht, was der Thread leisten soll (normalerweise eine Schleife). Dieser Dienst wird mit dem Aufruf `start` gestartet. Mit Hilfe von boolschen Variablen kann man die Schleife im `run`-Dienst des Thread beenden und damit den Thread beenden. Es ist nicht möglich, den `run`-Dienst zweimal zu starten. Stattdessen muss ein neuer Thread erzeugt werden.

• **Dialogorientiertes Protokoll** Bei dieser Form eines Protokolls befindet sich die Kommunikation mit einem Client jeweils in einem bestimmten Zustand, der bestimmte Protokoll-Anweisungen erfordert. Mit Hilfe einer Unterklasse der Klasse `Serververbindung` lassen sich im Dienst `bearbeiteProtokoll` diese Zustände so implementieren, als ob nur die Verbindung zu einem Client bestünde. Mit Hilfe von lokalen Variablen kann man Eigenschaften des entsprechenden Clients verwalten.

• **RFC** Abkürzung für *Request For Comments* (dt. Forderung nach Kommentaren). Dokument zur Stadardisierung von Protokollen im Internet. RFC 2821 beschreibt z.B. das SMTP-Protokoll. Ein nicht ganz ernst gemeintes RFC hat die Nummer 2549 und beschreibt die drahtlose Übertragung von Informationen mit Hilfe von Brieftauben. Besonders problematisch hat sich dabei allerdings das Antwortverhalten erwiesen. Außerdem hat man inzwischen festgestellt, dass auf diese Art auch leicht Viren übertragen werden können.