TECHNISCHE HOCHSCHULE MITTELHESSEN

Fachbereich
Mathematik, Naturwissenschaften und Informatik

Projektdokumentation Instant-Messaging-Dienst

Entwicklung sicherer, hardwarenaher Anwendungen

Wintersemester 15/16

Stand: 10.05.2016

Projektleiter:
Leonard Schmischke

Kursleitung:
Florian von Zabiensky

Projektteam:
Daniel Kreck
Ewald Bayer
Sebastian Westbrock
Thomas Iffland

Inhaltsverzeichnis

1 Projektidee

Das Ziel ist es einen Instant-Messaging-Dienst zu entwickeln, der es erlaubt, mit anderen Nutzern einzeln oder in Gruppen in Echtzeit zu kommunizieren. Hierbei handelt es sich um ein Gruppenprojekt, dass im Rahmen des Kurses *Entwicklung sicherer, hardwarenaher Anwendungen* in der Programmiersprache *Ada* entwickelt wird.

Der Dienst ist in Client und Server unterteilt. Beide Komponenten können über grafische Benutzerschnittstellen bedient werden. Die des Servers informiert über alle Ereignisse, die auf dem Server eintreten und erlaubt eine rudimentäre Verwaltung von angemeldeten Nutzern. Die Benutzerschnittstelle des Clients gestattet das Registrieren und Einloggen am Server. Im Anschluss bekommt der Nutzer seine Kontaktliste angezeigt, in der er unter anderem andere Nutzer als Freunde hinzufügen oder löschen kann. Bei Doppelklick auf einen befreundeten Nutzer öffnet sich ein seprarates Chatfenster zur direkten Kommunikation. Diese kann zur Gruppenkommunikation erweitert werden, indem weitere Freunde zum Chat eingeladen werden.

2 Vorgehen

Nachdem die Projektidee gefunden war wurde das Projekt analysiert und systematisch eingeteilt. Hierzu wurde sich im Team Gedanken gemacht, was der Instant-Messaging-Dienst im Einzelnen an Funktionaltität bereitstellen soll. Die Gruppe definierte dazu genaue Pflichten und skizzierte anhand dieser die Benutzeroberflächen.

Die daraus resultierenden Aufgabenbereiche wurde im Anschluss auf die Gruppenmitglieder verteilt:

Geschäftslogik Server: Herr Kreck und Herr Schmischke

Benutzeroberfläche Server: Herr Iffland

Geschäftslogik Client: Herr Westbrock

Benutzeroberfläche Client: Herr Bayer

Organisatorisch beschloss das Projektteam wöchentliche Treffen um die bisher entwickelte Funktionalität zusammenzuführen und zu testen. Des Weiteren wurden die Aufgaben für die nächste Woche besprochen und festgesetzt.

3 Pflichtenheft

Folgende Funktionalität wurden bzgl. des Clients festgelegt. Einem Nutzer soll es möglich sein:

- sich durch eine Registrierung ein Konto zu erstellen. Hierfür muss ein freiwählbarer aber eindeutiger Benutzername ausgewählt und ein Passwort zum späteren Login festgelegt werden.
- sich über einen Loginscreen mit seinem Benutzernamen und Passwort in den Chat einwählen zu können.
- nach vorangegangenem Login seine Kontaktliste einsehen zu können, Kontakte daraus zu entfernen oder neue über einen Kontaktanfrage hinzuzufügen zu können.
- über Kontaktanfragen selbstständig entscheiden zu dürfen, d.h. diese entweder annehmen oder ablehnen zu können.
- durch Interaktion mit der Kontaktliste mit seinen Kontakten kommunizieren zu können. Dieses soll er sowohl mit einem einzelnen Kommunikationspartner als auch mit mehreren gleichzeitig vollziehen können.

Folgende Funktionalität wurden bzgl. des Servers festgelegt. Einem Administrator soll es möglich sein:

- den Server von der GUI aus zu starten und zu stoppen.
- auszuwählen auf welchem Port der Server kommunizieren soll.
- die momentan verbundenen Benutzer mit zusätzlichen Informationen einsehen zu können (Client-IP, Kontakte, offene Chaträume).
- einen zuvor selektierten Benutzer zu kicken.
- die Anzahl der momentan verbundenen Benutzer einsehen zu können.
- die momentan zur Kommunikation genutzten Chaträume einsehen zu können.
- einen Überblick über die Aktivitäten des Servers zu erlangen, indem dieser diese zur Kenntnissnahme wichtige Aktivitäten in Informations- und Fehlerfeldern ausgibt.
- einen Mitschnitt über die momentan aublaufende Nutzerkommunikation angezeigt zu bekommen.

Aus diesen für den Nutzer und Administrator zur Verfügung gestellten Funktionen resultiert unter anderem, dass die Geschäftslogik des Servers Nutzer, Chaträume, Kontaktan-

fragen usw. verwalten können muss und dass diese Daten beim Starten und Beenden der Serveranwendung persistent gespeichert bzw. aus einer Datebank geladen werden.

4 Architektur

Der Instant-Messaging-Dienst unterliegt dem Client-Server-Paradigma. Zur Kommunikation zwischen den Clients und dem Server ist ein Protokoll erforderlich, welches den Kommunikationsablauf und entsprechend das Verhalten der Kommunikationsteilnehmer regelt.

4.1 Kommunikationsprotokoll

Server und Client kommunizieren über Nachrichten. Eine Nachricht besteht aus vier Komponenten:

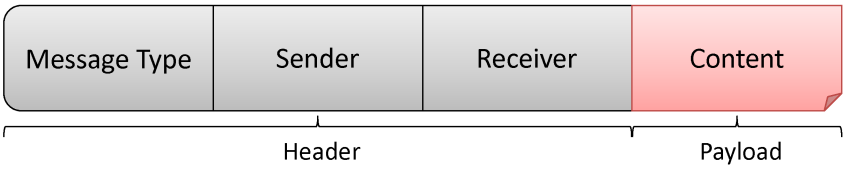


Abbildung 1: Struktur einer Nachricht

Die einzelnen Teile einer Nachricht werden durch ein definiertes Trennzeichen separiert. Hierbei handelt es sich um ein nicht druckbares Zeichen, um den Zeichenraum der Nutzer beim Chatten nicht unnötig einzuschränken. Ein anderes ebenso nicht druckbares Zeichen zeigt das Ende dieser Nachricht an.

Während Bestandteile des Headers nicht mehr weiter unterteilbar sind, wird der Inhalt je nach Nachrichtentyp noch feiner strukturiert, indem in ihm das gleiche Trennzeichen wiederholt zur Anwendung kommt.

Der Nachrichtentyp wird durch die Definition eines Aufzählungstypen mit 13 zu unterscheidenden Werten realisiert. Das Absender-Feld hält den Namen des Benutzers als Zeichenkette fest, wohingegen das Empfänger-Feld einen Chatraum als positive Ganzzahl kodiert. Dies hat den Grund, dass zur Kommunikation immer ein Chatraum erforderlich ist. Alle Nachrichten werden zunächst als Broadcast gehandhabt, allerdings kann durch einen Chatraum von zwei Nutzern Unicast-Kommunikation verwirklicht werden.

Bei diesem Vorgehen nimmt der Standard-Chatraum mit der Nummer *0* eine besondere Rolle ein. Jeder nicht verbundene Client befindet sich in einem individuellen Standard-Chatraum mit dem Server. Er ist die Anlaufstelle für eingehende Verbindungsanfragen, da erst im folgenden Schritt dem anfragenden Client dynamisch ein einzelner Chatraum zur Kommunikation mit dem Server zugewiesen werden kann. Nach dem Verbindungsaufbau verfügt demnach jeder Client über einen eigenen Server-Chatraum und ggf. wenn

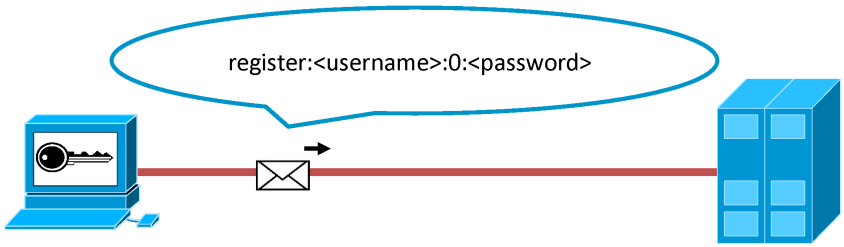
er die Kommunikation mit anderen Clients sucht, über jeweils einen Chatraum für jeden Kommunikationspartner. Diese werden zunächst als Einzelchats zwischen zwei Nutzern erzeugt, können aber durch Benutzerinteraktion um beliebig viele Teilnehmer erweitert werden, wodurch Gruppenkommunikation möglich ist.

Das Protokoll lässt sich um beliebig viele weitere Funktionen erweitern. Hierzu muss nur ein neuer Nachrichtentyp eingeführt und die Kontrollstrukturen des Clients und Servers angepasst werden.

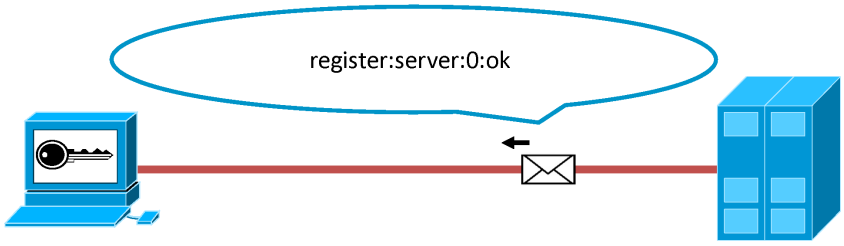
4.1.1 Registrierung

Ein Client meldet sich als ein Benutzer beim Server an. Dieser muss zuvor über eine Registrierung mit frei wählbarem Benutzernamen und Passwort angelegt.

Hierzu schreibt das Protokoll folgendes Verhalten vor: Der Client sendet eine Registrierungsanfrage an den Server. Diese ist vom Nachrichtentyp *register* und trägt im Absender-Feld den gewünschten Benutzernamen. Als Receiver wird der Standard-Chatraum des Servers angegeben. Der Inhalt der Nachricht ist das verschlüsselt zu übertragende Passwort des Benutzers.

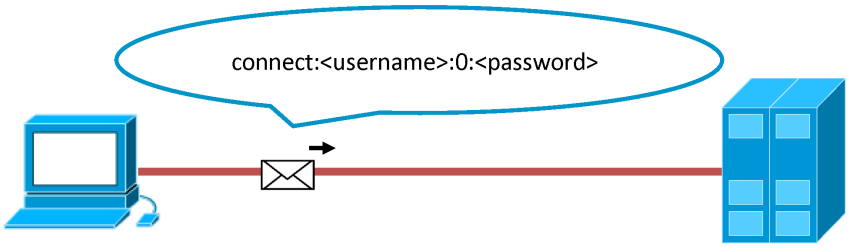


Wenn der Server die Nachricht empfängt, quittiert er den Erfolgsfall, indem ebenfalls eine *register*-Nachricht, mit dem Inhalt „ok“ zurückgeschickt wird. Sollte der Benutzername bereits verwendet werden, wird eine *refused*-Nachricht mit Inhalt „registration failed, name in use“ versendet.



4.1.2 Einloggen

Sobald ein Benutzer registriert wurde, kann sich der Client nun unter Angabe des Benutzernamens und Passworts beim Server per *connect*-Nachricht einloggen. Da noch kein Chat zu diesem existiert, wird der Standard-Chatraum adressiert.



Der Verbindungsversuch hat zwei mögliche Resultate:

- **Einloggen erfolgreich**

  In diesem Fall antwortet der Server ebenfalls mit einer *connect*-Nachricht. Die Empfänger-Nummer stellt die ID des Chatraums dar, indem dieser Client zukünftig mit dem Server kommuniziert, künftig Server-Chatraum genannt. Das erfolgreiche Verbinden wird zusätzlich durch ein „ok“ im Inhalt der Nachricht ausgedrückt.

- **Einloggen fehlgeschlagen**

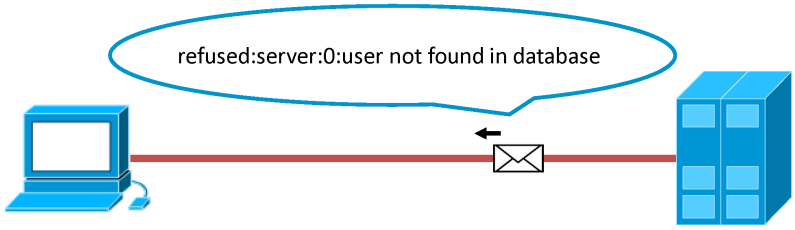
  Sollte das Verbinden nicht gelingen, wird vom Server eine an den Standard-Chatraum adressierte *refused*-Nachricht versendet, deren Inhalt Aufschluss über die Hintergründe des Misserfolgs liefert. Hierfür existieren vier verschiedene Szenarien.

  1. **der Benutzer ist unbekannt**
     Wird ein Benutzername übergeben, welcher dem Server nicht bekannt ist, schlägt das Einloggen fehl. Der Inhalt der Nachricht lautet dann „user not found in database“.

  2. **das Passwort ist nicht korrekt**
     Sollte das übergebene Passwort nicht mit dem in der Datenbank des Servers hinterlegtem Passwort des Benutzers übereinstimmen, kann der Client nicht eingeloggt werden. Der Inhalt der Nachricht lautet dann „invalid password“.

  3. **der angegebene Benutzer ist schon eingeloggt**
     Hier ist bereits ein Benutzer mit dem angegebenen Benutzernamen mit dem Server eingeloggt. Der Inhalt der Nachricht lautet dann „user already logged in“.

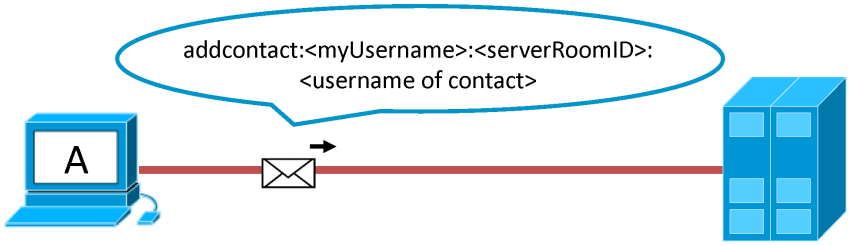
4. **der Client ist bereits eingeloggt**
   Wenn der Client sich bereits als ein Benutzer zum Server verbunden hat, lehnt der Server ein erneutes Einloggen ab. Der Inhalt der Nachricht lautet dann „you are already logged in to an account“.



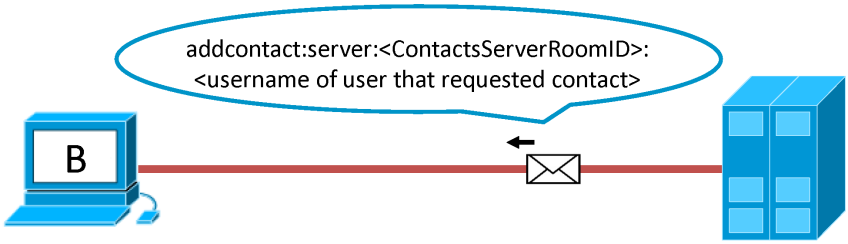
4.1.3 Kontaktanfrage

Sobald der Client eingeloggt ist, kann er nun mit Kontakten chatten. Kontakte sind beidseitig, das heißt, man kann nicht mit einem Benutzer befreundet sein, ohne gleichzeitig auch Kontakt des Benutzers zu sein.

Diese Kontakte müssen vor dem Chatten angelegt werden. Eine *addcontact*-Nachricht, adressiert an den Server, teilt diesem mit, dass dem im Inhalt der Nachricht genannten Benutzer eine Kontaktanfrage gestellt werden soll.



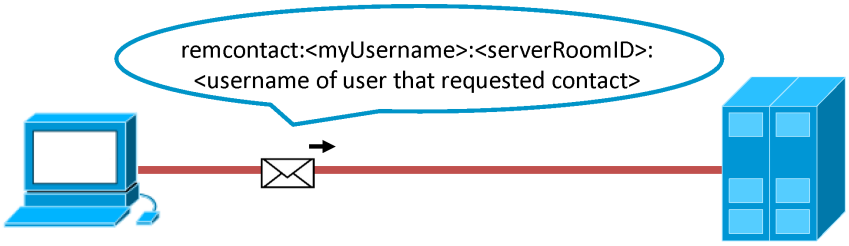
Der Server gibt diese Intention dem requestierten Benutzer weiter, sollte dieser eingeloggt sein.



Eine Kontaktanfrage kann angenommen werden, indem man dem anfragenden User eine

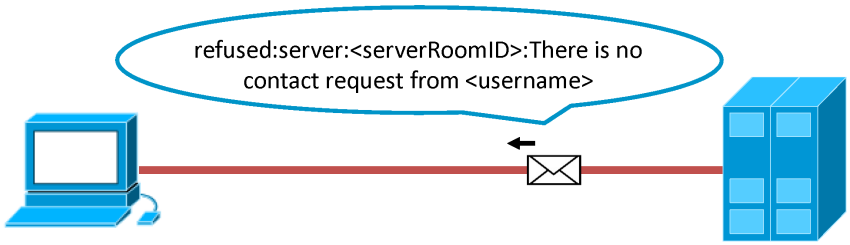
Kontaktanfrage zurück stellt.

Um eine Kontaktanfrage abzulehnen, kann eine *remcontact*-Nachricht an den Server gesendet werden. Als Inhalt ist der Benutzernamen des anfragenden Benutzers zu nennen.

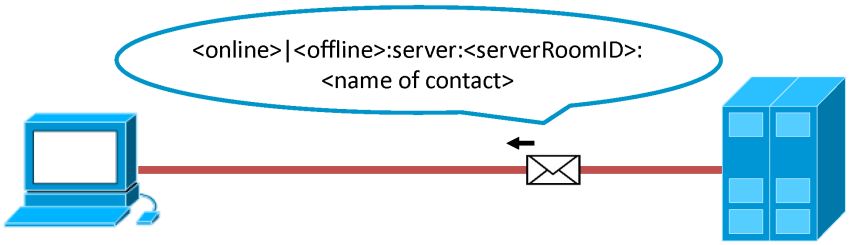


Über eine Nachricht in dieser Form kann auch ein Kontakt entfernt werden. Dies quittiert der Server mit einer *remcontact*-Nachricht, welche den Benutzernamen des entfernten Kontakts als Inhalt trägt. Eine analoge Nachricht wird auch den an entfernten Kontakt gesendet.

Sollte der Client versuchen, einen Benutzer von seiner Kontaktliste zu entfernen, welcher sich nicht auf dieser befindet, oder versuchen, eine Kontaktanfrage von einem Benutzer abzulehnen, welcher keine Anfrage gestellt hat, wird die *remcontact*-Nachricht vom Server mit einer *refused*-Nachricht beantwortet und keine weiteren Aktionen ausgelöst. Der Inhalt der Nachricht lautet dann dementsprechend der Situation „there is no contact request from ‘<username>‘“ beziehungsweise „there is no contact with name ‘<username>‘“.

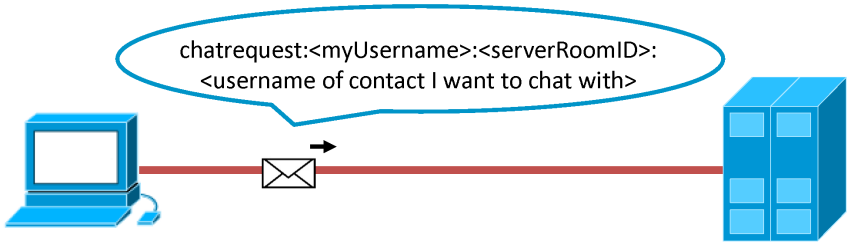


Sollte sich der Online-Status eines Kontaktes ändern, durch Ein-/Ausloggen oder nach dem er als neuer Kontakt hinzugefügt wurde, wird der neue Status dem Benutzer durch eine *offline*- oder *online*-Nachricht mitgeteilt.

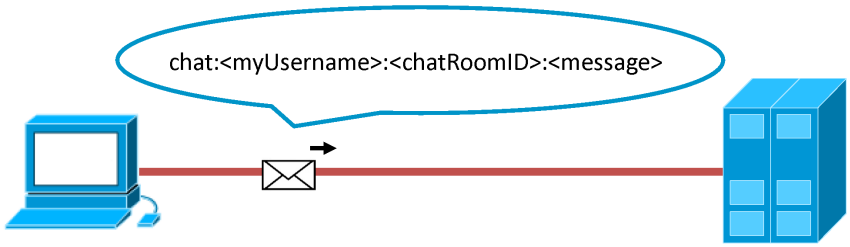


4.1.4 Chatten

Jeder Form des Chattens findet in einem Chatraum statt. Dieser muss zuerst vom Server erstellt werden. Hierzu muss der Client eine *chatrequest*-Nachricht an den Server senden. Adressiert wird diese an den Serverchatraum, der Inhalt gibt an, mit welchem eingeloggten Kontakt man chatten möchte. Mit Benutzern, die nicht Kontakt des Clients sind, kann nicht gechattet werden.



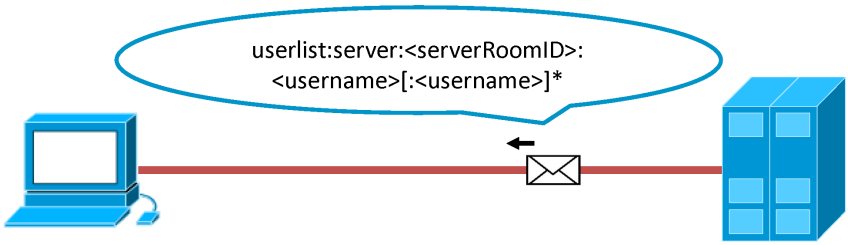
Der Server verarbeitet die Anfrage, in dem er einen Chatraum mit einer einzigartigen ID erstellt und den anfragenden sowie den angefragten Benutzer zu dem Chatraum hinzufügt. Dem anfragenden Benutzer wird anschließend mit einer *chatrequest*-Nachricht geantwortet, die als Empfänger-ID die Nummer des Chatraums trägt. Diese Nummer dient nun als Adresse aller *chat*-Nachrichten, die in diesem Raum ausgetauscht werden.



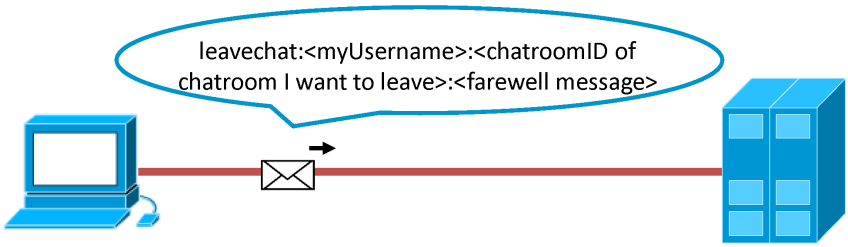
Möchte der Benutzer einen weiteren Kontakt zu diesem Chat hinzufügen, sendet er eine analoge *chatrequest*-Nachricht mit der Chatraum-ID als Empfänger-ID.

Ändert sich die Teilnehmerliste eines Chats, indem ein neuer Benutzer hinzugefügt wurde oder ein Benutzer den Chat verlässt, teilt der Server dies allen übrigen Teilnehmern

im Chat mit einer *userlist*-Nachricht mit. Im Inhalt dieser an den jeweiligen Chatraum adressierten Nachricht befinden sich die Benutzernamen aller Teilnehmer, durch das vom Protokoll genutzte Trennzeichen jeweils separiert.

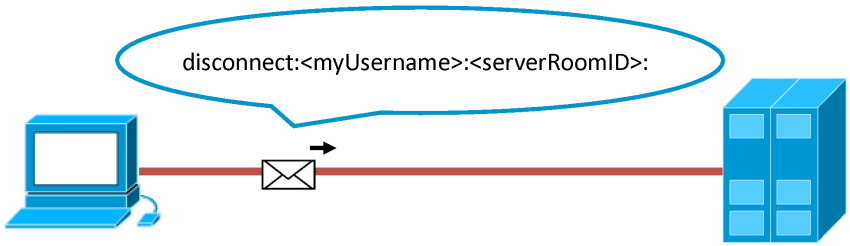


Ein Chatraum wird durch Versenden einer an den jeweiligen Chatraum adressierte *leavechat*-Nachricht verlassen. Dem Benutzer steht es offen einen Abschiedstext anzugeben, der nach dem Verlassen im Chat angegeben wird.



4.1.5 Ausloggen

Das Ausloggen wird mit einer *disconnect*-Nachricht bewerkstelligt. Wenn der Server diese durch eine weitere *disconnect*-Nachricht dem Inhalt „ok“. bestätigt, wurde der Benutzer erfolgreich ausgeloggt.



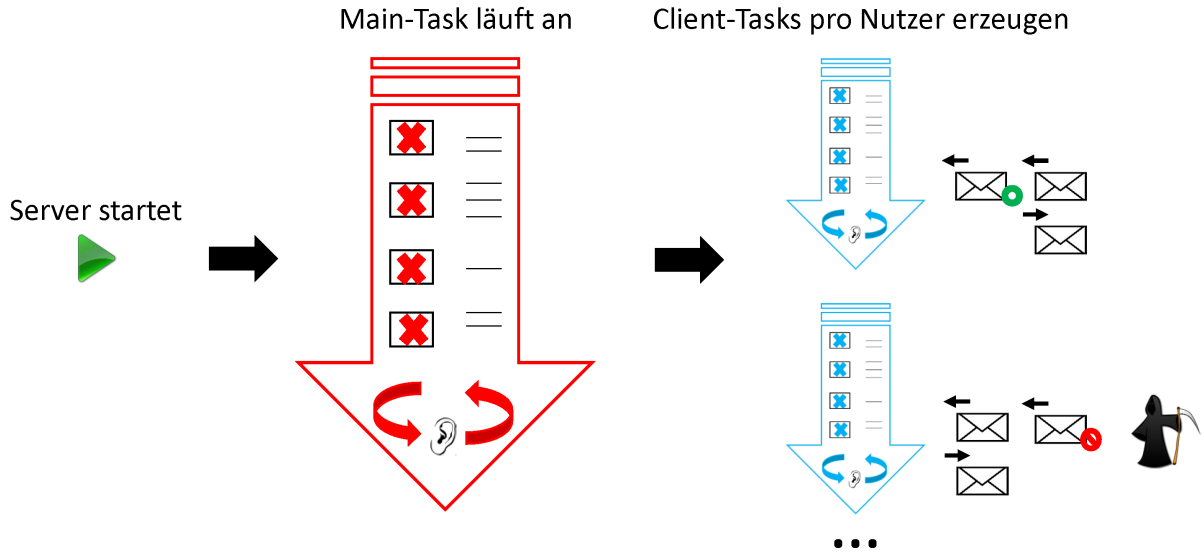
4.2 Implementierungslogik

In diesem Abschnitt soll beschrieben werden, wie das Programm von der Logik her arbeitet. Auf die konkreten Datenstrukturen, die zur Informationshaltung und -verwaltung

genutzt werden wird nicht näher eingegangen.

Während des Elaboration Tasks erzeugt und konfiguriert der Server seinen Kommunikationssocket für eingehende Verbindungsanfragen auf der Transportschicht. Hierzu zählt beispielsweise die Initialisierung von IP-Adresse und Port, sowie das Binden des Socket an diese. Daraufhin erzeugt und startet er einen weiteren Task, folgend Main-Task genannt.

Der Main-Task hat die Aufgabe auf eingehende Verbindungsanfragen zu lauschen und eine Verbindung zwischen Client und Server auf der Transportschicht herzustellen. Zum Lauschen auf den Socket benutzt er eine blockierende Subroutine, die zurückkehrt, sobald Aktivität anliegt. Daraufhin erzeugt der Main-Task für diesen Client einen eigenen Socket und Task. Als letzte Aktion startet der Main-Task diesen Client-Task und fährt nach dessen Aktivierungsphase mit einem erneuten Lauschen auf den Server-Socket fort. Solch ein Client-Task wird für jeden Nutzer bzw. Client erzeugt.



Ein Client-Task setzt in seinen Handlungsroutinen das Kommunikationsprotokoll um. Dies bedeutet, dass er auf seinem Socket auf eingehende Nachrichten horcht und sobald solch eine eintrifft, je nach Nachrichtentyp handelt. Dies beinhaltet ebenso den vom Protokoll vorgesehenen Verbindungsaufbau auf der Anwendungsschicht mittels einer *connect*-Nachricht. Analog zum Server kehrt er nach solch einer Routine zum erneuten Horchen zurück. Sobald der Wunsch auf Verbindungsabbau von einem der beiden Seiten gestellt wird, typischerweise vom Client, wird der Client-Task verschiedene Verwaltungsakte durchführen. Dies beinhaltet z.B. dass er den Client aus dessen Chaträumen entfernt und den Socket schließt. Ist dies alles vollbracht, endet der Client-Task. Bei erneutem

Verbindungsaufbau des Nutzers legt der Main-Task einen frischen Client-Task an, welcher analog verfährt.

Die Clientseite des Chatprogramms arbeitet sehr ähnlich. Da diese aber zu jedem Zeitpunkt nur auf einen Socket lauschen muss, wird nur ein Task zum Abfragen der Socketverbindung verwendet, der Main-Task entfällt.

4.3 Klassendiagramme

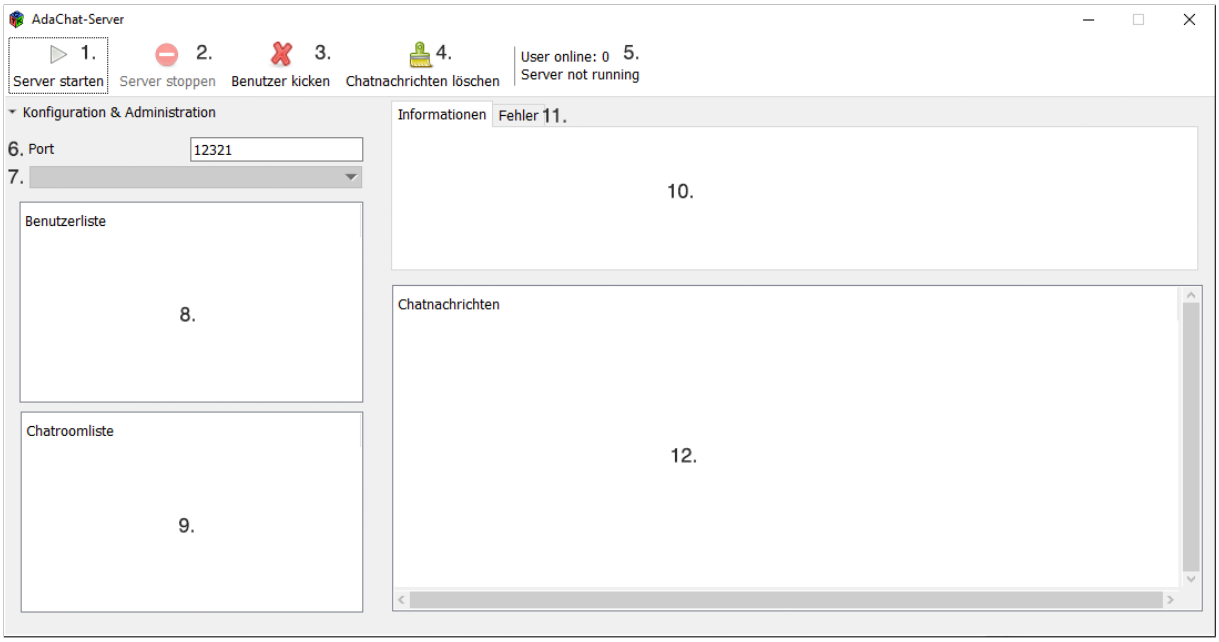
Die Klassendiagramme können aufgrund ihrer Größe dem beigelegten Verzeichnis „Klassendiagramme“ entnommen werden.

5 Benutzerhandbuch

Im Folgenden soll die Benutzung der Oberflächen erklärt werden, um dem Benutzer einen leichten Einstieg zu gewähren.

5.1 Server

Die Server-GUI dient als Oberfläche dazu, die Steuerung der Serverlogik einfach zu gestalten und somit eine effiziente Verwaltung zu gewährleisten.



1. Mit diesem Button lässt sich der Server starten. Diese Schaltfläche ist nur aktiviert, wenn der Server nicht läuft.
2. Mit diesem Button lässt sich der Server stoppen. Diese Schaltfläche ist nur aktiviert, wenn der Server läuft.
3. Mit dieser Schaltfläche ist es möglich einen Benutzer aus dem Chat zu entfernen("kicken"). Der User, der entfernt werden soll, wird aus dem Drop-down-Menü unter *7.* ausgewählt.
4. Mit der Schaltfläche ist es möglich das Fenster mit den Chatnachrichten zu leeren.
5. Hier wird der aktuelle Status des Servers angezeigt und die Anzahl der verbundenen Benutzer.
6. In diesem Textfeld lässt sich der Port einstellen. Diese Einstellung lässt sich nicht zur Laufzeit verändern, sondern nur vor dem Start des Servers.

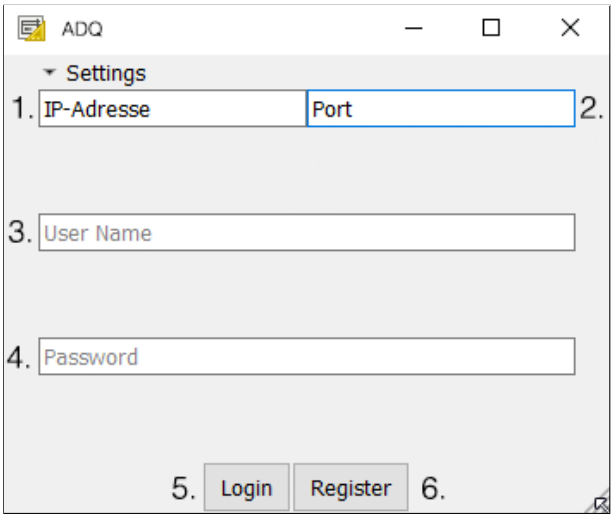
7. In diesem Drop-down-Menü befinden sich alle Nutzer, die aktuell online sind. Wenn hier ein Benutzer ausgewählt wird, kann er über Schaltfläche *3.* entfernt werden.
8. Hier befindet sich die Benutzerliste. Dort erhalten Sie folgende Informationen über einen Benutzer:
   - Benutzernamen
   - IP-Adresse
   - Seine Kontakte
   - Die Chaträume in denen er sich befindet
9. Hier finden Sie die Chaträume und die Benutzer, die sich in dem jeweiligen Raum befinden.
10. In dem Feld unter dem Reiter *Informationen* befinden sich Nachrichten des Servers über Ereignisse und Zustandsänderungen.
11. In dem Feld unter dem Reiter *Fehler* finden sich Fehlermeldungen des Servers.
12. In diesem Feld befinden sich die Nachrichten der Benutzer in den einzelnen Chaträumen.

5.2 Client

Nach dem Server soll nun die Benutzeroberfläche der Clientanwendung beschrieben werden. Der Client besteht aus drei verschiedenen Oberflächen: Eine Loginoberfläche, ein Chatfenster und eine Übersichtsdarstellung.

5.2.1 Loginoberfläche

Die Loginoberfläche dient dazu einen vorhanden Benutzer beim Server anzumelden oder, falls der Benutzer noch nicht vorhanden ist, diesen zu registrieren.



1. In diesem Feld wird die IP-Adresse des Rechners angegeben, auf dem der Server läuft.
2. Hier wird der Port angegeben, auf dem der Server lauscht. Der Standardport ist 20000.
3. Hier wird der Benutzername eingegeben.
4. In dieses Feld wird das Passwort eingegeben
5. Mit einem Druck auf "Login"wird der Benutzer, falls die Benutzerdaten korrekt sind, beim Sever angemeldet.
6. Mit einem Druck auf "Register"wird ein Benutzer mit den eingegebenen Benutzerdaten beim Server registriert.

5.2.2 Chatoberfläche

Die Chatoberfläche ist das Fenster, in dem Chatnachrichten zwischen den Benutzern ausgetauscht werden.

1. In diesem Textfeld befindet sich eine Übersicht der bisher ausgetauschten Chatnachrichten.
2. Hier befinden sich die Teilnehmer des Chats.
3. In diesem Textfeld werden die Chatnachrichten eingegeben, die an die anderen Teilnehmer gesendet werden sollen.
4. Über diesen Button lassen sich Smileys einfügen.
5. Über diese Schaltfläche lassen sich Spiele mit den anderen Chatteilnehmern starten.
6. Hierüber kann man sich vergangene Konversationen ansehen.
7. Über diesen Button ist es möglich einen weiteren Teilnehmer zum Chat hinzuzufügen.

6 Hindernisse

Im Folgenden sind die Problemstellungen aufgeführt, die in den Augen des Projektteams gravierend waren und die Entwicklung merkbar gehemmt haben.

6.1 Dokumentation der Sprache Ada

Das schwerwiegenste Problem, dass das Team über das ganze Projekt begleitet hat, war die unzureichende bis nicht vorhandene Dokumentation der Sprache Ada bzw. ihrer API.

Das Verhalten von bspw. Unterprogrammroutinen musste sich häufig allein durch die Bezeichnung und den Paramtern oder aus dem Kontext eines Programmierbeispiels von einem beliebigen Blog hergeleitet werden. Durch darauf folgendes mühsames und zeitintensives experimentieren konnte die fehlende Dokumentation dann meist kompensiert werden.

Es gibt zwar das Ada Reference Manual, welches sich jedoch meist nicht als sonderlich hilfreich erwies.

Jedoch muss man erwähnen, dass die Code-Qualität trotz alledem unmittelbar darunter gelitten hat. Denn dadurch dass die Bibliotheksroutinen nicht ausreichend dokumentiert sind, kann unser Programm nur beschränkt auf z.B. Fehlersituationen oder anderweitiges situationsbedingtes Verhalten reagieren.

Abgesehen davon wird aus diesem Grund eine Beschäftigung mit der Spache über den Kurs hinaus kaum Anhang finden.

6.2 Zirkuläre Abhängigkeiten

Beim Aufbau von Datenstrukturen, wo die einen Bestandteil der anderen sind, traten des Öfteren zirkuläre Abhängigkeitsprobleme auf. Es wurden zwei Methoden verwendet, um diese Probleme zu beseitigen. Zum einen ist es möglich, alle Datentypen in einer gemeinsamen Spezifikationsdatei (.ads) zu definieren. Somit ist es nicht notwendig, andere Spezifikationen einzubinden, es existieren keine Abhängigkeiten nach außen. Da dies allerdings zu einer sehr unübersichtlichen Projektstruktur führt, wurde wenn möglich die umgekehrte Methodik verwendet und unabhängige Datenstrukturen jeweils in ihrem eigenen File beschrieben, welches dann von den zwei nutzenden Strukturen referenziert wird.

Ada 2005 bietet zur Beseitigung von zirkulären Abhängigkeiten die „limited with“-Klausel an. Da dies aber nur eine unvollständige Sicht auf das somit eingebundene Paket oder den somit eingebundenen Typen liefert, konnte dies nicht immer verwendet werden (siehe

`http://www.adaic.org/resources/add_content/standards/05rat/html/Rat-1-3-3.html`).

Rekursive Datenstrukturen - zum Beispiel ein Benutzer, der seine Kontakte in einer Liste aus Benutzern speichert - können in Ada nur definiert werden, in dem der Datentyp zuvor als unvollständiger Typ deklariert wurde. Dieses Prinzip ist zwar aus anderen Programmiersprachen wie C bekannt, zerstückelt den Quellcode aber dennoch und ist somit ein weiterer Nachteil dieser Sprache.

Außerdem war verwunderlich, dass das Einbinden von Spezifikationsdateien in eine Bodydatei (.adb) ein anderes Folgeverhalten bezüglich der zirkulären Abhängigkeiten aufweisen kann, als wenn diese in die dem Body zugehörige Spezifikationsdatei eingebunden werden. Manche zirkuläre Abhängigkeit konnte nur so aufgelöst werden.

6.3 Arbeiten mit GtkAda

GtkAda ist eine Ada-Implementierung des erfolgreichen GTK+ Toolkits, mit dem sich grafische Oberflächen entwickeln lassen. Es ist neben Qt die Bibliothek zum entwickeln von GUIs. Während der Entwicklung traten einige Dinge besonders negativ in den Vordergrund.

1. Dokumentation
   Wie auch schon Ada im allgemeinen ist die Dokumentations- und Informationslage zu GtkAda dürftig. Es gibt keine offizielle Dokumentation und wenige Beispiele, die leider auch meistens nur einen kleinen Umfang haben und nur einzelne Komponenten abdecken, nicht jedoch mehrere Komponenten im Zusammenspiel. Falls man nach Lösungen zu bestimmten Fragestellungen sucht, findet man meistens keine GtkAda spezifische Antwort, sondern muss oftmals Lösungen aus anderen Programmiersprachen (Python oder C) auf Ada übertragen. Dies lässt sich jedoch oftmals nicht ganz einfach umsetzen, da GtkAda manche Dinge anders handhabt bzw. benennt (Beispielsweise bei der Nutzung von *Gtk_Tree_Model* und *Gtk_Tree_Store*).

2. Glade
   Glade ist ein Tool um sich grafische Oberflächen schnell zu erstellen und diese in eine *.glade* Datei zu exportieren. Dabei handelt es sich um eine XML-strukturierte Datei, die die Oberfläche und Voreinstellungen enthält. Das Problem an Glade ist leider, das es nicht sonderlich stabil läuft und Portierungen für Windows und Mac OS X zu Darstellungsfehlern und Performanceeinbrüchen neigen.

3. GtkAda & Tasks
   Leider ist es mit GtkAda auch nicht möglich aus einem laufenden Task heraus

dynamisch neue Objekte zu erzeugen und diese der Oberfläche hinzuzufügen. Dies verhindert beispielsweise einen einzelnen Task pro Konversationsfenster zu erstellen.