



# Hochschule für Angewandte Wissenschaften München

## Fakultät Informatik und Mathematik

### **Datenkommunikation** **(Bachelorstudiengang Wirtschaftsinformatik)**

Studienarbeit im Wintersemester 2019/2020

München, September 2019  
Prof. Dr. Peter Mandl, LBA Björn Rottmüller, M.Sc.

### **Inhalt**

<b>1</b>	<b>Einführung.....</b>	<b>2</b>
<b>2</b>	<b>Aufgabenstellung.....</b>	<b>2</b>
2.1	Überblick.....	2
2.2	Teilaufgabe 1.....	2
2.3	Teilaufgabe 2.....	3
2.4	Teilaufgabe 3.....	4
<b>3</b>	<b>Hinweise zur Ausführung und Bewertung .....</b>	<b>7</b>
3.1	Programmierung und Test.....	7
3.2	Abgabe der Studienarbeit.....	7
3.3	Präsentation und Kolloquium .....	8
3.4	Bewertungssystem.....	8
3.5	E-Mail-Adressen der Dozenten .....	8
3.6	Web-Links zum Einstieg.....	9
	<b>Anhang 1: Einstieg in den Sourcecode des Chat-Projekts .....</b>	<b>10</b>
	<b>Anhang 2: Protokollspezifikation SimpleChat .....</b>	<b>11</b>

# 1 Einführung

Im Fach Datenkommunikation ist eine Studienarbeit begleitend zur Vorlesung anzufertigen. Diese wird in kleinen Teams (bestehend aus maximal vier Studierenden) erstellt. Gemäß Studien- und Prüfungsordnung geht die Prüfungsleistung in die Gesamtnote des Moduls Datenkommunikation ein.

Die Studienarbeit befasst sich mit der Konzeption, der Implementierung und der Erprobung von Kommunikationssoftware und dient der Sammlung von Erfahrungen in der Entwicklung und im Test verteilter Anwendungen. Spezielle Aspekte der Entwicklung und des Einsatzes sowie Problemstellungen des Designs von verteilten Anwendungen sollen vertieft werden. Die Schwerpunkte der Studienarbeit liegen in der Protokollkonzeption und -implementierung sowie im Test und der Leistungsbewertung einer selbst entwickelten Kommunikationssoftware.

## 2 Aufgabenstellung

### 2.1 Überblick

Konkretes Ziel der Studienarbeit in diesem Semester ist es, die Programmierung von Kommunikationsanwendungen auf Basis von Sockets zu erlernen und Erfahrungen mit Transportprotokollen zu sammeln. Als Programmiersprache verwenden wir Java mit den dort vorhandenen Java-Socket-Klassen.

Im Team wird eine Erweiterung zu einer verteilten Chat-Anwendung entwickelt und getestet. Ein Chat-Server bedient in der vorhandenen Anwendung mehrere Chat-Clients nebenläufig. Sendet ein Client eine Nachricht, wird diese an alle angemeldeten Clients verteilt. Die Chat-Anwendung wird bereitgestellt und ist um eine neue Funktionalität zu erweitern.

Die Studienarbeit ist in drei Teilaufgaben untergliedert:

### 2.2 Teilaufgabe 1

Es wird ein Java-Projekt bereitgestellt, das den Sourcecode von einigen Basis-Interfaces und Basis-Klassen sowie auch Klassen und Interfaces einer Chat-Anwendung (SimpleChat) enthält. Ebenso werden eine Server-GUI zum Starten des Servers und eine Client-GUI sowie eine Benchmarking-GUI ausgeliefert.

Die erste Teilaufgabe sieht eine Einarbeitung in den vorhandenen Sourcecode vor, um die vorgegebenen Programmteile zu verstehen. Eine Protokollspezifikation für das SimpleChat-Protokoll ist im Anhang dieses Dokuments zu finden. Die SimpleChat-Lösung soll als Powerpoint-Foliensatz mit Grafiken (Architekturskizzen, Klassenmodelle, Ablaufdiagramme, Sequenzdiagramme, Zustandsautomaten, ...) verständlich dokumentiert werden. Dabei steht die Client-/Server-Kommunikation und die client- und serverseitige Implementierung der Kommunikationsbausteine (Chat-Protokoll, Threadmodell, interne Datenstrukturen für die Benutzerverwaltung, Chat-Nachrichtenverteilung) im Vordergrund. Die Benchmarking-Anwendung und auch die Statistik-Bausteine müssen nur angewendet werden können, die Benchmarking-Sourcen müssen nicht analysiert werden.

## 2.3 Teilaufgabe 2

Die Chat-Lösung ist um einen AuditLog-Server zu erweitern. Der AuditLog-Server existiert noch nicht und ist vollständig neu in Java zu entwickeln. Der neue Server hat die Aufgabe, alle Login-, Logout- und ChatMessage-Nachrichten auf einem externen Speichermedium (z. B. in einer Datei) zu protokollieren.

Bei jeder ankommenden Nachricht von den Chat-Clients (Login-Request, Logout-Request und ChatMessage-Request) wird vom Chat-Server eine Audit-Nachricht an den neu zu entwickelnden AuditLog-Server gesendet. Dabei wird ein unidirektionales Anwendungsprotokoll (AuditLog-Protokoll) ohne Bestätigungsnachricht verwendet.

Bitte beachten Sie:

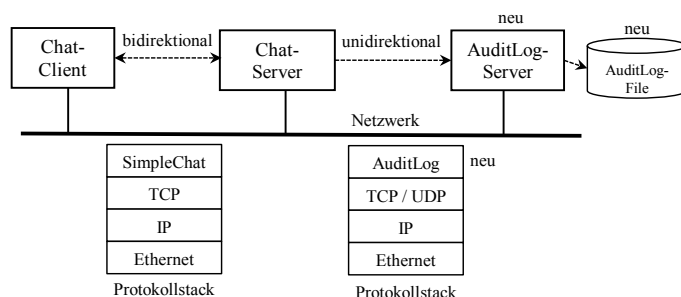
- Der Chat-Server ist bereits für die Nutzung des AuditLogs an den entsprechenden Stellen vorbereitet und **muss nicht verändert werden**.
- Der Nachrichtenaufbau zur Kommunikation zwischen Chat-Server und AuditLog-Server ist in einer vordefinierten Objektklasse (AuditLogPDU) **zwingend vorgegeben**.

Als Transportprotokoll zur Kommunikation zwischen dem Chat-Server und dem AuditLog-Server ist zunächst UDP und zum Vergleich danach TCP zu verwenden. Der AuditLog-Server bietet seine Dienste standardmäßig über den TCP- bzw. den UDP-Port 40001 an. Eine Veränderung ist über den Startdialog des Chat-Servers möglich.

Beim Start des Chat-Servers wird per Dialog eingestellt, ob auch eine TCP-Verbindung oder eine UDP-Kommunikation zum AuditLog-Server eingerichtet werden soll. Beim Anhalten des Chat-Servers wird die Kommunikationsbeziehung abgebaut. Der AuditLog-Server muss danach neue Verbindungsaufbauwünsche entgegennehmen können, also bei Nutzung von TCP sofort wieder auf einen ankommenden Verbindungsaufbauwunsch warten. Der Chat-Server muss nicht verändert werden, er ist für die Kommunikation mit dem AuditLog-Server vorbereitet.

Wie der AuditLog-Server gestartet und beendet wird, über eine grafische Oberfläche oder über die Kommandozeile, ist dem Team überlassen.

Abb. 1 zeigt das Zusammenspiel der Komponenten und die Protokollstacks des vorhandenen SimpleChat- und des neuen AuditLog-Protokolls. Der Chat-Server agiert aus Sicht des AuditLog-Servers in der Rolle des Clients und aus Chat-Sicht in der Rolle des Servers.



**Abb. 1:** Zusammenspiel der Systemkomponenten der neuen Lösung

In der AuditLogPDU-Klasse sind folgende Protokollfelder vorhanden und dürfen nicht verändert werden:

- Typ des Protokoll-Datensatzes (LoginRequest, ChatRequest, LogoutRequest)
- Identifikation des Chat-Client-Threads
- Identifikation des verarbeitenden WorkerThreads im Chat-Server
- Benutzername des Chatters
- Zeitstempel der Erzeugung der AuditLog-Nachricht
- Inhalt der Chat-Nachricht (nur bei ChatMessage-Requests)

Die an den AuditLog-Server übertragenen Nachrichten sollen jeweils in einem Datensatz in das persistente AuditLog geschrieben werden.

Darüber hinaus ist ein Administrationsprogramm zu schreiben, welches das AuditLog ausliest und aufbereitete statistische Informationen (Anzahl Clients, Anzahl Datensätze je Typ, ...) zum Inhalt ausgibt.

## 2.4 Teilaufgabe 3

### Round Trip Time (RTT)

In dieser Teilaufgabe ist die entwickelte AuditLog-Lösung aus Teilaufgabe 2 zu testen. Weiterhin es ist eine Leistungsbewertung durchzuführen. Die TCP- und die UDP-basierte AuditLog-Protokoll-Implementierung sind zu vergleichen.

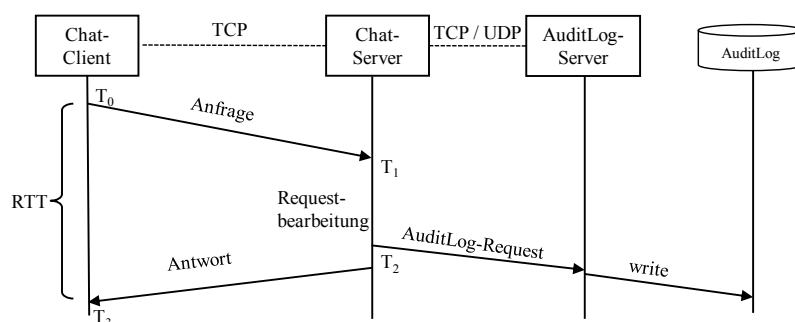
Bei den Messungen dient die sog. Round Trip Time (RTT) als Metrik (siehe auch Abb. 2). Dies ist aus Client-Sicht die Zeit vom Absenden eines ChatMessage-Requests bis zur Ankunft der zugehörigen ChatMessage-Response:

$$RTT = T_3 - T_0$$

Die Latenzzeit für die reine Kommunikation (wir nennen sie  $T_k$ ) lässt sich ebenfalls ermitteln, indem man noch die Bearbeitungszeit für den ChatMessage-Request im Server abzieht. Damit ergibt sich gemäß Skizze aus Abb. 2:

$$T_k = (T_3 - T_0) - (T_2 - T_1)$$

$T_k$  beinhaltet die Bearbeitungszeit in allen Protokollinstanzen im Client und im Server sowie die Zeit im Netzwerk benötigte Zeit.



**Abb. 2:** RTT als Metrik

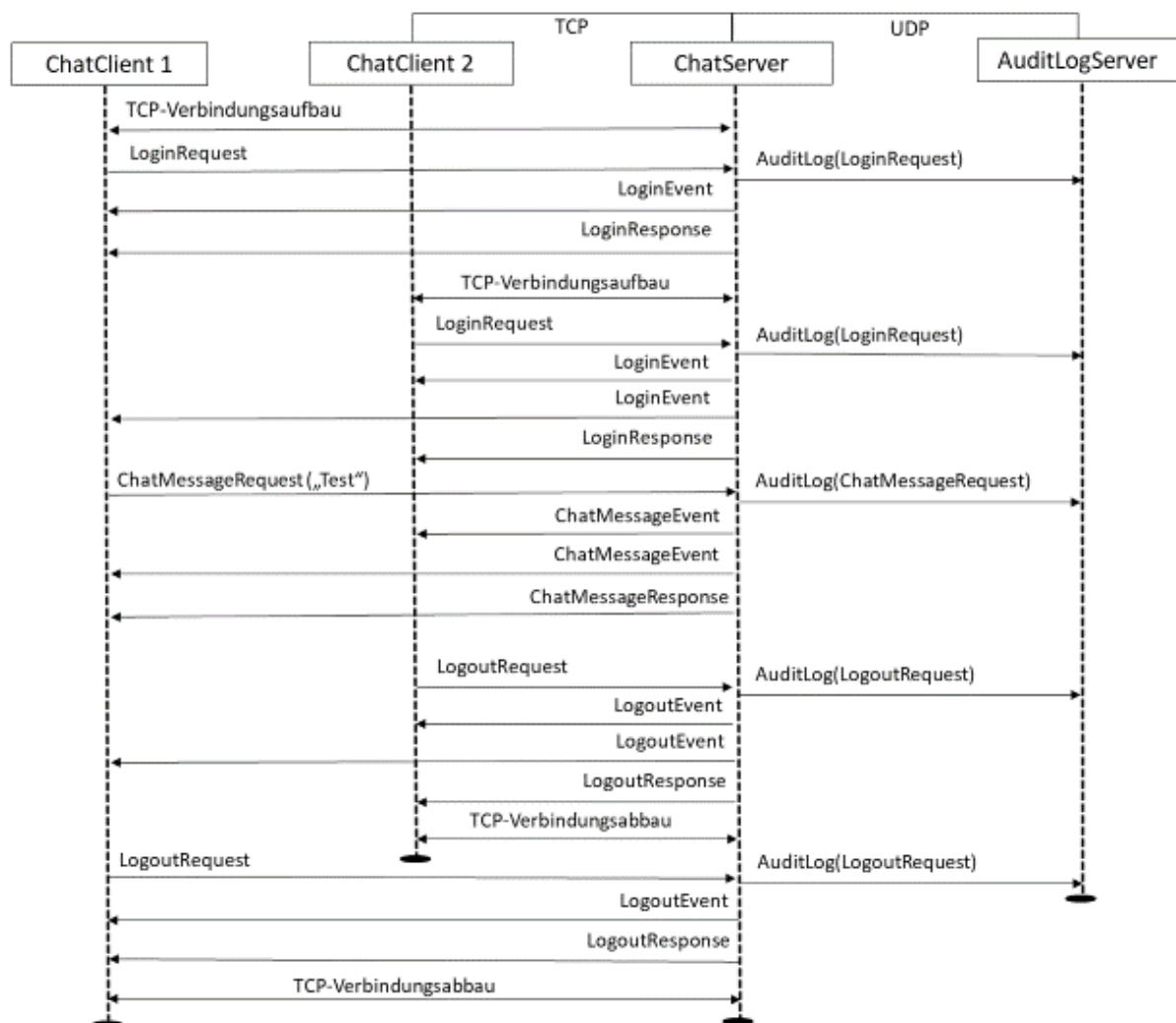
Die Benchmarks sind in einem kleinen, abgeschlossenen Netzwerk durchzuführen, das aus drei Raspberry Pi (Raspi) und einem Switch besteht. Eine Dokumentation des Testnetzwerks finden Sie in einem eigenen Dokument.

---

## Analyse des Nachrichtenverkehrs

Vor der Ausführung der Benchmarks ist im Testnetzwerk eine Analyse des Nachrichtenverkehrs über einen Nachrichten-Mitschnitt mit dem frei verfügbaren Tool *Wireshark* bzw. *tshark* (Open Source Paket Analyser) durchzuführen und zu dokumentieren.

Der Nachrichtenmitschnitt beschränkt sich auf das Szenario gemäß Abb. 3 mit zwei Chat-Clients. Beide melden sich beim Chat-Server an. Für das AuditLog verwenden Sie die UDP-Variante. Ein Chat-Client sendet eine Chat-Nachricht, die beide Chat-Clients empfangen. Anschließend melden sich beide Chat-Clients wieder ab. Finden Sie im Mitschnitt die Nachrichten der Chat-Anwendung (Anwendungsschicht) für den Login-, Chat-, Logout-Vorgang sowie die zugehörigen AuditLog-Nachrichten und interpretieren Sie die Kommunikation der unteren Schichten über die Protokolle TCP, UDP, IPv4 und Ethernet.



**Abb. 3:** Szenario für zu analysierenden Nachrichtenverkehr

## Dokumentation der Benchmarking-Ergebnisse

Die Ergebnisse des Benchmarks sind zu dokumentieren. Neben einer tabellarischen Darstellung der Ergebnisse der Lasttests ist eine grafische Darstellung, wie in Abb. 4

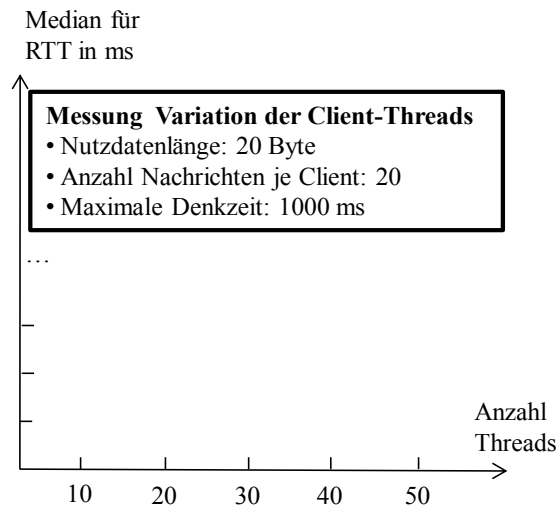
vorgegeben, zu erarbeiten. Die Entwicklung der durchschnittlichen RTT bei Veränderung der Anzahl an Threads von 10 auf 50 in 10er-Schritten, bei konstanter Nachrichtenlänge von 20 Byte und 20 Nachrichten pro Client sowie einer maximalen Denkzeit von 1000 ms soll jeweils mit TCP-basiertem und mit UDP-basiertem AuditLog-Server aufgezeigt werden.

Die Messungen sind mindestens fünf Mal zu wiederholen. Um statistisch vernünftige Aussagen zu erhalten, sind die RTT-Mittelwerte und die Mittelwerte die 50%-Percentile (Median) über die Wiederholungen zu berechnen.

Insgesamt sind je Implementierung (TCP, UDP) 50 Testläufe durchzuführen:  $2 * 5 * 5 = 50$ , mit 2 Implementierungsvarianten, 5 Messpunkten und jeweils 5 Wiederholungen je Messpunkt.

Der bereits vorhandene Benchmarking-Client (Klasse `BenchmarkingClient`) kann mit entsprechender GUI gestartet werden.

Mit Hilfe der ebenfalls vorgegebenen Klasse `SharedClientStatistics` werden Messwerte erfasst und in eine Protokolldatei mit dem Namen `Benchmarking-ChatApp-Protokolldatei` geschrieben. Am Ende jeder Einzelmessung wird vom Benchmarking-Client ein Datensatz in dieser Protokolldatei erzeugt, der u.a. die Clientanzahl, die durchschnittliche RTT, den Median (0,5-Quantil), die Anzahl gesendeter Chat-Requests und die Anzahl gesendeter Chat-Message-Events enthält. Die Feldinhalte der Protokolldatei sind im Ordner `Documentation` beschrieben. Die Messdaten können mit MS Excel oder einem anderen Tabellenkalkulationsprogramm nachbearbeitet und ausgewertet werden.



**Abb. 4:** Grafische Darstellung der Messergebnisse

Bei den einzelnen Tests ist jeweils zu protokollieren, wie viele Nachrichten verlorengehen.

Die Entwicklung der Heap-Größe und der CPU-Auslastung während der Tests sollen zur Laufzeit mit dem Tool *JConsole* beobachtet und dokumentiert werden, um ein Gefühl für den Ressourcenverbrauch zu bekommen.

Hinweis: Alle Lasttests sollen mit den gleichen Einstellungen (Parametern) vorgenommen werden.

## 3 Hinweise zur Ausführung und Bewertung

### 3.1 Programmierung und Test

Die Programmierung erfolgt in Java mit Eclipse als Entwicklungswerkzeug. Wahlweise kann auch eine andere Entwicklungsumgebung verwendet werden. Als Java-Version soll 1.8 verwendet werden (verfügbar zum Download, siehe Web-Links).

Zur Versionsverwaltung wird jeder Gruppe angeraten, ein git-Repository anzulegen.

Für die Test- und Diagnoseunterstützung wird als Trace-/Logging-Funktionalität der Apache-Standard *Log4j* (siehe Packages *org.apache.log4j*, *org.apache.common.logging.\**) verwendet. Trace-/Logging-Ausgaben sollten sinnvoll eingesetzt werden. Für die Entwicklung des AuditLog-Servers kann das Package *Apache Commons IO* verwendet werden.

Vor den Lasttests sollten Sie Funktionstests durchführen, um eine ausreichende Stabilität sicherzustellen. Testen Sie auch Ausfallsituationen wie den Absturz eines Servers, eines Clients oder den Ausfall einer Netzverbindung. Der Benchmarking-Client kann auch für die Funktionstests verwendet werden. Sie können die Chat-Anwendung auch auf einem einzigen Rechner unter Verwendung der IP-Loopback-Funktionalität testen.

Hinweis: Reduzieren Sie die Log-Ausgaben bei der Durchführung der Benchmarks auf ein Minimum, um unnötigen Overhead zu vermeiden. Dies kann die Chat-Anwendung in den bereitgestellten Konfigurationsdateien *log4j.client.properties* und *log4j.server.properties* eingestellt werden, z.B. über die Anweisung *log4j.rootLogger=ERROR, RollingFileAppender*. Für den AuditLog-Server legen Sie bitte eigene Property-Dateien an.

### 3.2 Abgabe der Studienarbeit

Die Ergebnisse der Studienarbeit mit allen Programmen einschließlich der Dokumentation sind termingerecht abzugeben. Arbeiten Sie sich also zügig in die Thematik ein. Ab der 2. Oktoberwoche sind für alle Arbeiten insgesamt maximal 12 Wochen einkalkuliert. Der kalkulierte Aufwand je Teilaufgabe soll als grober Anhalt dienen:

- Teilaufgabe 1: 3 Wochen (überwiegend für das Erlernen und Dokumentieren der vorhandenen Lösung)
- Teilaufgabe 2: 5 Wochen
- Teilaufgabe 3: 4 Wochen

Die Studienarbeit erfordert ein kontinuierliches Arbeiten an der Lösung. Jedes Team sollte sich einen Projektplan erstellen, der zu jeder Teilaufgabe einen Meilenstein definiert. Eine Parallelisierung der Bearbeitung ist durchaus sinnvoll. Der Projektplan ist mit abzugeben.

**Der Abgabetermin der Studienarbeit wird in den Übungsstunden festgelegt.**

Zu beachten ist noch folgendes:

- Der Sourcecode ist ausreichend nach Java-Standard zu dokumentieren. Die gesamte Programmdokumentation sollte im Sourcecode untergebracht werden.

- Der komplette Sourcecode und die Dokumentation sind in elektronischer Form im ZIP-Format als vollständiges Eclipse-Projekt an den zuständigen Dozenten zu übergeben.
- Die Abgabe erfolgt per E-Mail direkt an den zuständigen Dozenten unter Angabe des Gruppennamens und der Gruppenteilnehmer. Eine Bestätigung der Abgabe erfolgt ebenso per E-Mail.

### 3.3 Präsentation und Kolloquium

Jede Gruppe muss ihre Ergebnisse präsentieren. Hierzu sind Präsentationsfolien vorzubereiten und zwei Wochen vor dem Präsentationstermin abzugeben. Die Abgabe erfolgt per E-Mail direkt an den zuständigen Dozenten unter Angabe des Gruppennamens und der Gruppenteilnehmer.

Wie Sie präsentieren, ist Ihnen freigestellt. Anfangs sollten auf alle Fälle das Team und die Aufgabenverteilung vorgestellt werden.

Folgende Hinweise sollten noch beachtet werden:

- Die Präsentation sollte aus 30 Powerpoint-Folien bestehen (ca. 10 Folien für Teilaufgabe 1, 10 Folien für Teilaufgabe 2 und 10 Folien für Teilaufgabe 3)
- **Die Präsentationstermine werden im Laufe des Semesters bekanntgegeben.**
- Als Präsentationsgrundlagen sind vor allem die lauffähige Chat-Anwendung und die vorbereiteten Powerpoint-Folien zu nutzen.
- Jedes Teammitglied sollte auf Fragen zu allen Teilaufgaben und auch auf Verständnisfragen zur Lösung vorbereitet sein. Während der Präsentation werden Überblicksfragen und Fragen zur eigenen Lösung an alle Teilnehmer gestellt (Kolloquium).

### 3.4 Bewertungssystem

Alle Teilaufgaben müssen im Team in der vorgegebenen Reihenfolge bearbeitet werden. Für die Bewertung der Studienarbeit werden individuell Punkte je Teilaufgabe vergeben. Insgesamt kann jeder Studierende während der Studienarbeit maximal 100 Punkte sammeln:

- Teilaufgabe 1: max. 20 Punkte
- Teilaufgabe 2: max. 30 Punkte
- Teilaufgabe 3: max. 20 Punkte
- Abschlusspräsentation und mögliche Fragen zu allen Teilaufgaben (auch zu Teilaufgabe 1): max. 30 Punkte

Als Mindestpunktzahl zum Bestehen der Studienarbeit sind 50 Punkte festgelegt. Die Abschlusspräsentation ist in jedem Fall erforderlich.

### 3.5 E-Mail-Adressen der Dozenten

- Prof. Dr. Peter Mandl: mandl@hm.edu
  - LBA Björn Rottmüller: bjoern.rottmueller@hm.edu
-



### 3.6 Web-Links zum Einstieg

- <http://www.eclipse.org>: Homepage von Eclipse
- <https://www.wireshark.org/download.html>: Wireshark Download
- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>: Java Home Page von Oracle
- <http://www.prof-mandl.de>: Webseite von Prof. Mandl mit Material zur Studienarbeit
- <https://commons.apache.org/proper/commons-io>: Information für die Audit-Log-Datei Generierung

## Anhang 1: Einstieg in den Sourcecode des Chat-Projekts

Die vorhandene Chat-Anwendung stellt einige Java-Interfaces und -Basisklassen bereit. Die Package-Struktur des Chat-Projekts sieht wie folgt aus:

### **Package edu.hm.dako.chat.benchmarking**

Dieses Package beinhaltet Interfaces und Klassen für das Benchmarking. Das Package enthält im Wesentlichen eine grafische Oberfläche zur Parametrisierung und Auswertung von Benchmark-Läufen sowie eine Simulation zur Vervielfältigung von Chat-Clients.

### **Package edu.hm.dako.chat.client**

Dieses Java-Package beinhaltet die Interfaces und Klassen des Chat-Clients. Neben einer Chat-GUI ist die clientseitige Logik implementiert.

### **Package edu.hm.dako.chat.common**

Dieses Java-Package enthält gemeinsam nutzbare (wiederverwendbare) Klassen, die vom Server und auch vom Client verwendet werden, z.B. verwenden sowohl Server- als auch Client die Klasse `ChatPDU` als Datenobjekt, um eine Chat-Nachricht zu erstellen. Eine andere Klasse (`SharedClientStatistics`) dient der Sammlung von Statistikdaten zur Ermittlung von Round Trip Times (RTT) innerhalb eines Testlaufs für die Kommunikation zwischen Client-Threads und dem Server. Die Daten werden in einem Array gesammelt, das einen Eintrag für jeden Client-Thread enthält. Nach jedem Testlauf erfolgt ein Eintrag eines Protokollsatzes in die Datei namens `Benchmarking-ChatApp-Protokolldatei`.

### **Package edu.hm.dako.chat.connection**

Dieses Package implementiert die eigentliche Client- und Serverlogik für das Verbindungsmanagement und die Datenübertragung.

### **Package edu.hm.dako.chat.server**

In diesem Package befindet sich die Sourcen für den Serverteil inkl. einer Server-GUI.

### **Package edu.hm.dako.chat.tcp**

In diesem Package liegen alle Klassen für die TCP-Implementierung sowohl für den Client als auch für den Server.

### **Package edu.hm.dako.chat.udp**

In diesem Package liegen alle Klassen für die UDP-Implementierung sowohl für den Client als auch für den Server.

## Anhang 2: Protokollspezifikation SimpleChat

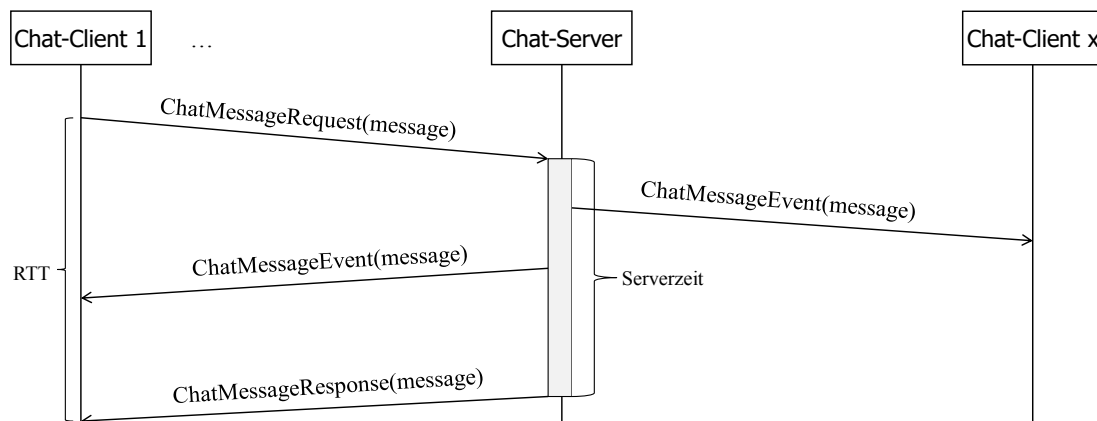
### Protocol Data Unit

Der Nachrichtenaufbau der Chat-Protokolle ist in der Klasse `ChatPDU` beschrieben. Das in der Java-Klasse vorgegebene Nachrichtenformat dient als Vorgabe. Zur Kommunikation zwischen Chat-Client und Chat-Server wird ein serialisierter Java-Objektstrom verwendet.

### Protokollverlauf

Der einfache Nachrichtenfluss im SimpleChat-Protokoll zwischen dem Chat-Clients und dem Chat-Server ist in Abb. 5 skizziert. Ein Client sendet einen `ChatMessageRequest` an den Server. Der Server verteilt den Request in einer `ChatMessageEvent`-Nachricht an alle angemeldeten Clients (hier stellvertretend Client x) und auch an den Absender des `ChatMessage`-Requests. Erst danach wird der Request mit einer `ChatMessageResponse` bestätigt und der Client kann die nächste Nachricht senden.

Die Chat-Clients registrieren sich vor der Datenaustauschphase beim Chat-Server (Login-Vorgang) und melden sich am Ende ab (Logout-Vorgang). Der Login- und der Logout-Vorgang laufen vom Protokollverlauf her ähnlich wie die Übertragung der `ChatMessages` ab.



**Abb. 5:** Kommunikation zwischen Chat-Client und Chat-Server im SimpleChat

### Übersicht über Protokollzustände

Das SimpleChat-Protokoll ist im Folgenden kurz anhand der Zustandsautomaten sowohl client- als auch serverseitig beschrieben. Im Protokoll sind vier Zustände vorgesehen, die sich aufgrund ankommender Nachrichten oder aufgrund von internen Funktionsaufrufen (z.B.

login-Aufruf im Client) entsprechend verändern. Die Zustandsübergänge mit Hinweisen auf erforderliche Aktionen sind in tabellarischer Form spezifiziert. Der Server verwaltet für jeden Client eine Instanz des Protokollautomaten. An Zuständen sind definiert:

- UNREGISTERED: Client ist nicht im Server angemeldet
- REGISTERING: Client ist im Anmeldevorgang
- REGISTERED: Client ist im Server angemeldet
- UNREGISTERING: Client meldet sich vom Server ab (Abmeldevorgang)

Im Folgenden sind die erforderlichen Feldbelegungen je Nachrichtentyp erläutert. Die Nachrichtentypen und die Bedeutung der Felder sind der vorgegebenen Klasse `ChatPDU` zu entnehmen.

### Clientseitiger Protokollautomat

SimpleChat-Zustandsautomat für Clientseite					
Incoming Event   Client State	UNREGISTERED	REGISTERING	REGISTERED	UNREGISTERING	
login	send (LOGIN_REQUEST_PDU); New State = REGISTERING				
LOGIN_RESPONSE		New State = REGISTERED			
logout			send (LOGOUT_REQUEST_PDU); New State = UNREGISTERING		
LOGOUT_RESPONSE				New State = UNREGISTERED	
tell			send (CHAT_MESSAGE_REQUEST_PDU); Block chat		
CHAT_MESSAGE_RESPONSE			Unblock chat		
CHAT_MESSAGE_EVENT		Show new message	Show new message	Show new message	
LOGIN_EVENT		Show new login-list	Show new login-list	Show new login-list	

Serverseitiger Protokollautomat

SimpleChat-Zustandsautomat für Serverseite (für jeden Client ein Zustandsautomat)				
Incoming Event   Corresponding Client State	UNREGISTERED	REGISTERING	REGISTERED	UNREGISTERING
LOGIN_REQUEST	create login-list entry; New State = REGISTERING; send (LOGIN_EVENT_PDU) to all active clients; send (LOGIN_RESPONSE_PDU); New State = REGISTERED			
LOGOUT_REQUEST			New State = UNREGISTERING; send (LOGOUT_EVENT_PDU) to all active clients; send (LOGOUT_RESPONSE_PDU); New State = UNREGISTERED; delete login_list_entry;	
CHAT_MESSAGE_REQUEST			send (CHAT_MESSAGE_EVENT_PDU) to all active and registered clients; send (CHAT_MESSAGE_RESPONSE_PDU);	

## ChatPDU-Feldnutzung

Chat-Protokoll (SimpleChat), Feldbelegung								
PduType	Used by Chat-Protokoll	Used by	userName	eventUserName	clientThreadName	serverThreadName	sequenceNumber	message
LOGIN_REQUEST	SimpleChat	Client	Client (unique id)	not used	Client-Threadname	not used	not used	not used
LOGIN_RESPONSE	SimpleChat	Server	Copy of Login_Request	not used	Copy of Login_Request	Current Worker-Threadname	not used	not used
LOGOUT_REQUEST	SimpleChat	Client	Client (unique id)	not used	Client (unique id)	not used	not used	not used
LOGOUT_RESPONSE	SimpleChat	Server	Copy of Logout_Request	not used	Copy of corresponding Logout_Request	Current Worker-Threadname	not used	not used
CHAT_MESSAGE_REQUEST	SimpleChat	Client	Client (unique id)	not used	Client (unique id)	not used	unique number of message	Client
CHAT_MESSAGE_RESPONSE	SimpleChat	Server	Copy of Login_Request	not used	Copy of Chat_Message_Request	Current Worker-Threadname	Copy of Chat_Message_Request	Copy of Chat_Message_Request
CHAT_MESSAGE_EVENT	SimpleChat	Server	Copy of Chat_Message_Request	Copy of Chat_Message_Request	Copy of Chat_Message_Request	Current Worker-Threadname	not used	Copy of Chat_Message_Request
LOGIN_EVENT	SimpleChat	Server	Copy of Login_Request	Copy of Login_Request	Copy of Login_Request	Current Worker-Threadname	not used	not used
LOGOUT_EVENT	SimpleChat	Server	Copy of Logout_Request	Copy of Logout_Request	Copy of Logout_Request	Current Worker-Threadname	not used	not used
PduType	clients	serverTime	clientStatus	errorCode	numberOfReceivedChatMessages, numberOfSentEvents, numberOfRetries			
LOGIN_REQUEST	not used	not used	REGISTERING	not used	not used			
			REGISTERED or UNREGISTERED ( when error)	Corresponding Error Code				
LOGIN_RESPONSE	not used	not used			not used			
LOGOUT_REQUEST	not used	not used	UNREGISTERING	not used	not used			
LOGOUT_RESPONSE	not used	not used	UNREGISTERED	not used	for statistical usage in benchmarks only			
CHAT_MESSAGE_REQUEST	not used	not used	REGISTERED	not used	not used			
CHAT_MESSAGE_RESPONSE	not used	Request duration	REGISTERED	not used	for statistical usage in benchmarks only			
CHAT_MESSAGE_EVENT	not used	not used	REGISTERED	not used	not used			
LOGIN_EVENT	not used	not used	REGISTERING	not used	not used			
LOGOUT_EVENT	not used	not used	UNREGISTERING	not used	not used			

Client: Feld wird vom Client gesetzt  
 Server: Feld wird von Server gesetzt  
 not used: Feld wird nicht benutzt