

DezSys06 – JMS Chat

23.11.2014

STEFAN GEYER, MATHIAS RITTER
4AHIT

Inhaltsverzeichnis

Aufgabenstellung	2
Zeitaufzeichnung	3
Schätzung	3
Tatsächlich benötigte Zeit	3
Arbeitsaufteilung	3
Designüberlegungen	4
Sender- / Empfängerebene	4
Clientebene	4
Kommandoverarbeitungsebene.....	4
UML-Klassendiagramm	5
Implementierung.....	6
Aufbau der Überprüfung der Konsolenargumente & Start des Programms.	6
Aufbau der Nachrichten & Konfiguration.....	6
Aufbauen einer Verbindung zum Broker.....	6
Initialisierung des Receivers	7
Initialisierung des Senders.....	7
Git.....	8
Repositoryname	8
Log.....	8
Testfall.....	9
Screenshot Mac.....	9
Screenshot PC.....	10

DezSys06 – JMS Chat

Aufgabenstellung

Implementieren Sie eine Chatapplikation mit Hilfe des Java Message Service. Verwenden Sie Apache ActiveMQ (<http://activemq.apache.org>) als Message Broker Ihrer Applikation. Das Programm soll folgende Funktionen beinhalten:

Benutzer meldet sich mit einem Benutzernamen und dem Namen des Chatrooms an.

Beispiel für einen Aufruf:

```
vsdbchat <ip_message_broker> <benutzername> <chatroom>
```

Der Benutzer kann in dem Chatroom (JMS Topic) Nachrichten an alle Teilnehmer eine Nachricht senden und empfangen.

Die Nachricht erscheint in folgendem Format:

```
<benutzername> [<ip_des_benutzers>]: <Nachricht>
```

Zusätzlich zu dem Chatroom kann jedem Benutzer eine Nachricht in einem persönlichen Postfach (JMS Queue) hinterlassen werden. Der Name des Postfachs ist die IP Adresse des Benutzers (Eindeutigkeit).

Nachricht an das Postfach senden:

```
MAIL <ip_des_benutzers> <nachricht>
```

Eignes Postfach abfragen:

```
MAILBOX
```

Der Chatraum wird mit dem Schlüsselwort EXIT verlassen. Der Benutzer verlässt den Chatraum, die anderen Teilnehmer sind davon nicht betroffen.

Gruppenarbeit: Die Arbeit ist in einer 2er-Gruppe zu lösen und über das Netzwerk zu testen!

Abnahmen, die nur auf localhost basieren sind unzulässig und werden mit 6 Minuspunkten benotet!

Zeitaufzeichnung

Schätzung

Beschreibung	Zeitaufwand
UML-Diagramm entwerfen	1h
Implementiert	5h
Testen	2h
Protokollierung	1h
Gesamt	9h

Tatsächlich benötigte Zeit

Beschreibung	Zeitaufwand	Name
UML-Diagramm entwerfen	30min	Ritter
Implementierung	3h 20min	Geyer
Implementierung	3h 45min	Ritter
Testen	1h 40min	Geyer
Testen	1h 25min	Ritter
UML-Redesign	30min	Geyer
Protokollierung	30min	Geyer
Protokollierung	45min	Ritter
	12h 45min	

Summe Geyer: 6h

Summe Ritter: 6h 45min

Arbeitsaufteilung

	Stefan Geyer	Mathias Ritter
Package	data, io	net
Zusätzliche Klassen	net.NetworkController, net.Networking	Main
Tests (Package)	net	data, io
Zusätzliches	Projekt aufgesetzt	UML - Klassendiagramm

Designüberlegungen

Mithilfe eines Apache ActiveMQ-Servers soll ein Chatroom basierter Chat entwickelt werden. Dabei soll der User auch die Möglichkeit haben, eine private E-Mail an einen weiteren User zu senden und seine eigenen Mails abzufragen. Weil mit ActiveMQ gearbeitet wird, muss sich um die Implementierung des Servers nicht gekümmert werden.

Der Chat wird auf drei Ebenen aufgeteilt.

- Sender- / Empfängerebene
- Clientebene
- Kommandoverarbeitungsebene

Sender- / Empfängerebene

Diese Ebene kommuniziert direkt mit dem Server. D.h., dass alle Pakete die vom Server kommen und die an den Server gesendet werden, werden in dieser Ebene verwaltet und übertragen.

Clientebene

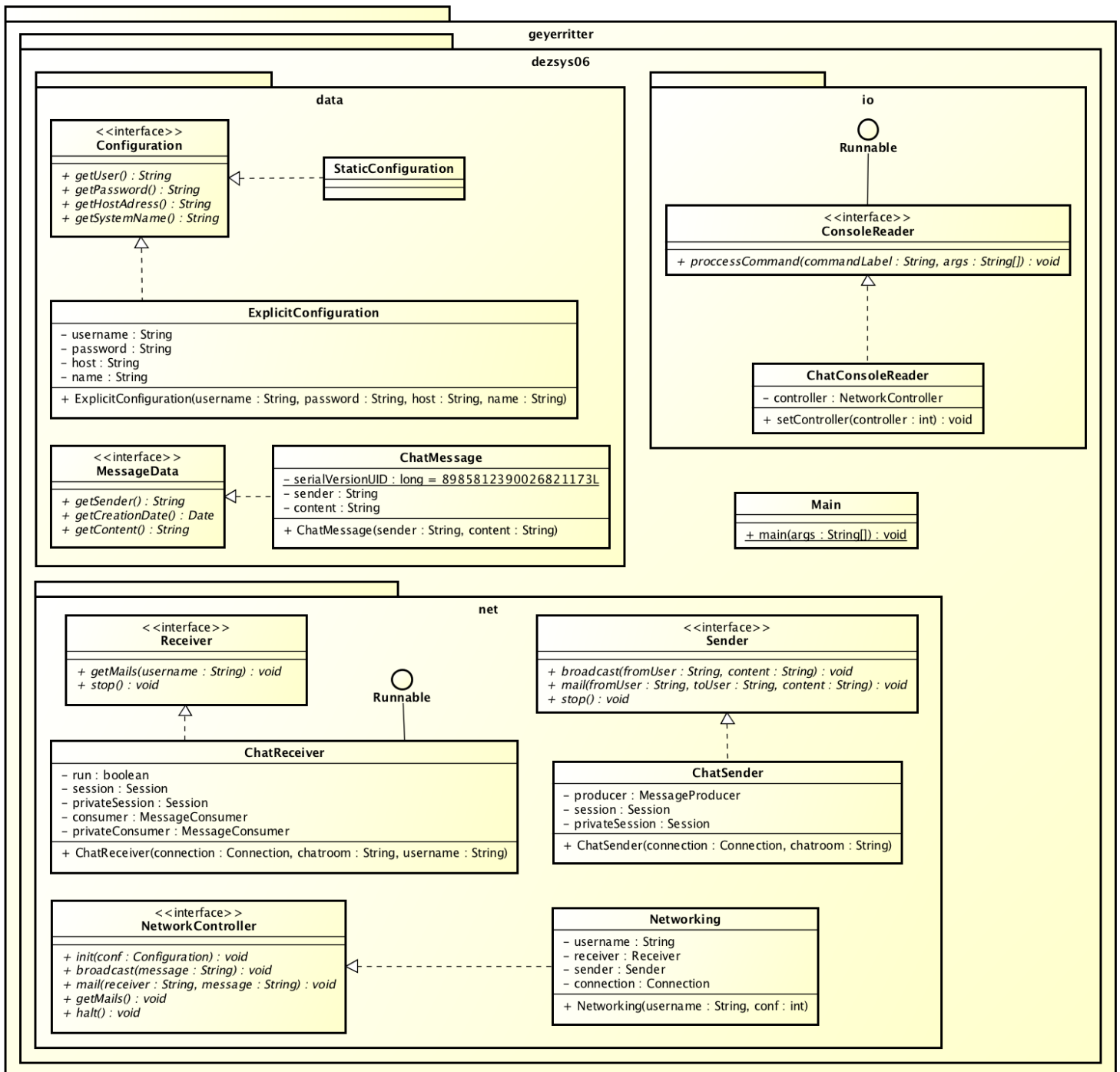
Diese Ebene bietet die Funktionen die zum Senden und zum Empfangen benötigt werden. Die Funktionalität wird über die Sender- / Empfängerebene abgehandelt.

Kommandoverarbeitungsebene

Diese Ebene verarbeitet die Eingaben des Users und wertet diese aus. Je nach Kommando werden die Funktionen aus der Clientebene angewandt.

UML-Klassendiagramm

pkg



Implementierung

Aufbau der Überprüfung der Konsolenargumente & Start des Programms.

Das Programm wird in der Klasse Main in `tgm.geyerritter.dezsys06` gestartet. Der Benutzer gibt alle Eingaben in der Konsole ein. Diese Eingaben werden vom `ChatConsoleReader` überprüft. Der `ChatConsoleReader` implementiert das Interface `ChatConsole`. Bei falscher Eingabe wird eine entsprechende Meldung ausgegeben. Zur Abfrage aller verfügbaren Kommandos kann man „help“ eingeben.

Aufbau der Nachrichten & Konfiguration

Der Aufbau der Chat-Nachrichten sowie Klassen zur Speicherung der Konfiguration wurden im Package `tgm.geyerritter.dezsys06.data` angelegt.

Eine Chatnachricht (`Chatmessage`) enthält den Absender, das Erstellungsdatum und den Inhalt. Die Chatnachricht implementiert das Interface `MessageData`.

Die Konfiguration wird durch das Interface `Configuration` vorgegeben. Zwei Klassen implementieren dieses Interface. Eine dieser Klassen enthält die Standard-Konfiguration für ActiveMQ, die andere enthält die Konfiguration wie vom User eingegeben.

Aufbauen einer Verbindung zum Broker

Für das Verwalten der Verbindungen wurde das Package `tgm.geyerritter.dezsys06.net` angelegt.

In diesem Package wurde ein Interface `NetworkController` angelegt, welcher den zentralen Punkt der Verbindungsverwaltung darstellt. Die Klasse `Networking` implementiert dieses Interface.

Die Initialisierung der Connections erfolgt in der Methode `init`. Zuerst wird eine neue Connection initialisiert und gestartet.

```
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(
    conf.getUser(), conf.getPassword(),
    conf.getHostAddress());
this.connection = connectionFactory.createConnection();
this.connection.start();
```

Danach werden Receiver und Sender initialisiert. Der Receiver muss gestartet werden, da das Empfangen von Nachrichten in einem eigenen Thread abläuft.

```
this.reciever = new ChatReceiver(connection, conf.getSystemName(),
    this.username);
new Thread(this.reciever).start();
this.sender = new ChatSender(connection, conf.getSystemName());
```

Initialisierung des Receivers

Der Receiver initialisiert im Konstruktor 2 Sessions: Eine zum Empfangen von Chatnachrichten im Chatraum (Topic) und eine zum Empfangen von Privatnachrichten (Queue). Die Queues/Topics werden als Destination angegeben. Zum Empfangen der Nachrichten wird ein MessageConsumer initialisiert:

```
this.consumer = session.createConsumer(destination);
```

Das Empfangen von Messages des Chatraums findet in der run-Methode statt. Die receive-Methode blockiert so lange, bis eine Nachricht empfangen wird.

```
ObjectMessage message = (ObjectMessage) consumer.receive();
```

Für das Empfangen von Privatnachrichten wurde die Methode getMails erstellt. Das Empfangen der Nachrichten wird durch die Methode receiveNoWait ausgelöst, welche nicht blockiert und nur zu diesem Zeitpunkt verfügbare Nachrichten abfragt.

```
ObjectMessage message = (ObjectMessage) privateConsumer.receiveNoWait();
```

Initialisierung des Senders

Der Receiver initialisiert 2 Sessions, Destinations und MessageProducer: Je 2 zum Senden von Chatnachrichten im Chatraum (Topic) und je 2 zum Senden von Privatnachrichten (Queue). Zum Senden der Nachrichten im Chatraum (Topic) wird ein MessageProducer initialisiert:

```
this.producer = session.createProducer(destination);  
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
```

Zum Senden der Nachrichten in einer Queue (Privatnachricht) wird jeweils ein neuer MessageProducer für diese Queue initialisiert

```
MessageProducer privateProducer =  
    privateSession.createProducer(privateDestination);  
privateProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
```


Git

Repositoryname

<https://github.com/mritter-tgm/DezSys06>

Log

Siehe die sich im Repo befindende Datei: log.txt


Testfall

Die Tests wurden mit JUnit durchgeführt. Eine genaue Beschreibung der Testfälle findet man in den Javadocs. Alle Testfälle (22) wurden erfolgreich durchlaufen und es wurde eine Test- Coverage von ca. 88,6% erreicht.

Zusätzlich wurde das Programm mit zwei verschiedenen Computern (im selben Netz) erfolgreich getestet:

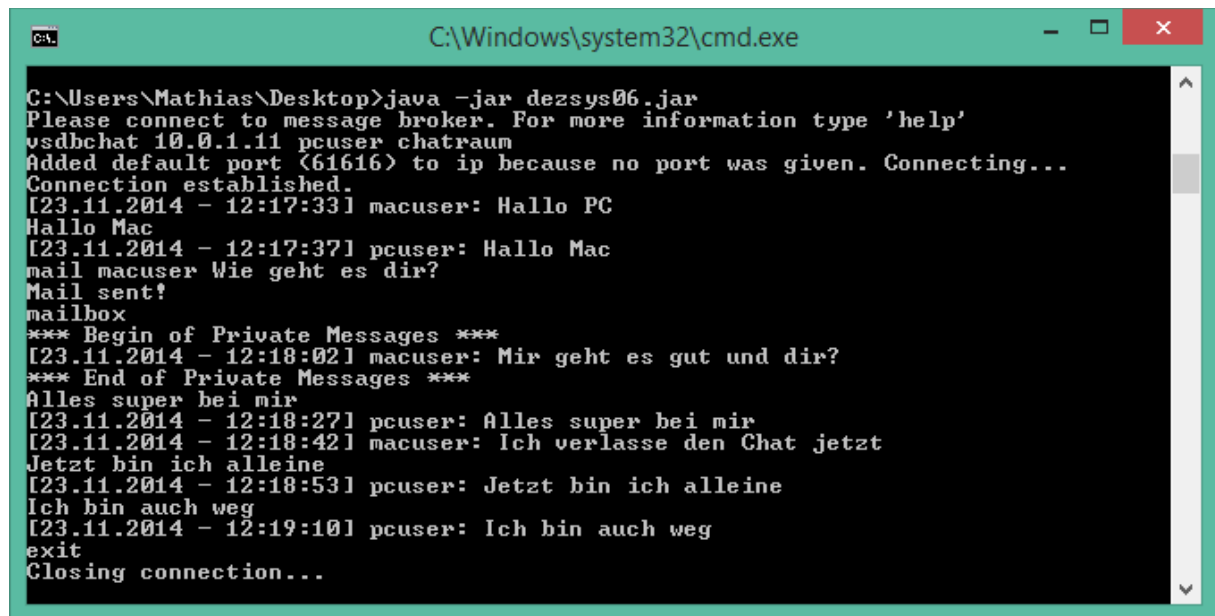
1. Starten von ActiveMQ am Mac
2. Starten des Programms auf dem Mac & dem PC
3. Auf beiden Computern wurde eine Verbindung mit dem Message Broker auf dem Mac aufgebaut
4. Beide Computer konnten Nachrichten im Chatraum senden & empfangen
5. Beide Computer konnten sich gegenseitig Privat-Nachrichten (Mails) senden und diese abfragen
6. Das Programm wurde mit „exit“ beendet.

Screenshot Mac



```
Laptop-Mathias:Desktop Mathias$ java -jar dezsyst06.jar
Please connect to message broker. For more information type 'help'
vsdbchat 127.0.0.1 macuser chatraum
Added default port (61616) to ip because no port was given. Connecting...
Connection established.
Hallo PC
[23.11.2014 - 12:17:33] macuser: Hallo PC
[23.11.2014 - 12:17:37] pcuser: Hallo Mac
mailbox
*** Begin of Private Messages ***
[23.11.2014 - 12:17:43] pcuser: Wie geht es dir?
*** End of Private Messages ***
mail pcuser Mir geht es gut und dir?
Mail sent!
[23.11.2014 - 12:18:27] pcuser: Alles super bei mir
Ich verlasse den Chat jetzt
[23.11.2014 - 12:18:42] macuser: Ich verlasse den Chat jetzt
exit
Closing connection...
Laptop-Mathias:Desktop Mathias$
```

Screenshot PC



```
C:\Windows\system32\cmd.exe

C:\Users\Mathias\Desktop>java -jar dezsyst06.jar
Please connect to message broker. For more information type 'help'
vsdbchat 10.0.1.11 pcuser chatraum
Added default port (61616) to ip because no port was given. Connecting...
Connection established.
[23.11.2014 - 12:17:33] macuser: Hallo PC
Hallo Mac
[23.11.2014 - 12:17:37] pcuser: Hallo Mac
mail macuser Wie geht es dir?
Mail sent!
mailbox
*** Begin of Private Messages ***
[23.11.2014 - 12:18:02] macuser: Mir geht es gut und dir?
*** End of Private Messages ***
Alles super bei mir
[23.11.2014 - 12:18:27] pcuser: Alles super bei mir
[23.11.2014 - 12:18:42] macuser: Ich verlasse den Chat jetzt
Jetzt bin ich alleine
[23.11.2014 - 12:18:53] pcuser: Jetzt bin ich alleine
Ich bin auch weg
[23.11.2014 - 12:19:10] pcuser: Ich bin auch weg
exit
Closing connection...
```