Verteilte Systeme Praktikum WS2014
Teilnehmer: Michael Schmidt, Eugen Winter

1. Praktikumsaufgabe: Client/Server-Anwendung „Verteilte Nachrichten-Queue“

Für die Implementierung einer RMI Server/Client-Anwendung soll Java RMI verwendet werden.
Es gilt das folgende Interface zu implementieren:

MessageService.java

<< interface >> **MessageService extends Remote**
+ nextMessage(String clientID): String (throws RemoteException) + newMessage(String clientID, String message): void (throws RemoteException)

Programmaufbau

Klassen:

Message.java

Message
- df: DateFormat - clientsRequestsList: HashMap<String, Date> - messageIDCounter: int - nextMessage: Message - message: String - clientID: String - messageID: int - msgResendTime: int - date: Date
+ Message(String clientID, String message, int msgResendTime) + getValidMessage(String clientID): Message + getNextMessage: Message + getMessage: String + getClientID: String + getMessageID: int + getMsgResendTime: int + getDate: Date + setNextMessage(Message): void + toString: String

Die Message-Klasse kapselt die Informationen, die mit der newMessage- Funktion übermittelt werden. Sie speichert neben dem reinen Message-String und der Client-ID auch eine fortlaufend vergebene Message-ID, die Uhrzeit und die Gedächtniszeit *t* in der Variablen *msgResendTime*. Außerdem kennt sie für die spätere Verwendung in einer Delivery-Queue nach FIFO Prinzip eine Referenz auf ihr nächstes Message-Objekt.

Um die korrekte Funktionsweise der Gedächtniszeit *t* zu realisieren, besitzt jedes Message-Objekt zusätzlich eine HashMap in welcher als Key/Value Paare jeweils die Client-ID und die Zeit gespeichert werden, sobald ein Client die nextMessage-Funktion benutzt.

Beim Abruf einer Message heißt dies, dass unbekannte Clients die Nachricht erhalten und gleichzeitig mit Client-ID und Uhrzeit im Message-Objekt hinterlegt werden.
Ein erneutes Abrufen desselben Clients für führt durch Bildung der Differenz der hinterlegten und der aktuellen Zeit dazu, dass diese Nachricht nicht noch einmal an denselben Client gesendet wird, solange die Gedächtniszeit des letzten Abrufs noch nicht überschritten ist. Gleichzeitig wird der erneute Abruf

Verteilte Systeme Praktikum WS2014
Teilnehmer: Michael Schmidt, Eugen Winter

DeliveryQueue.java

DeliveryQueue
- firstMessage: Message - currentSize: int - MAXSIZE: int
+ DeliveryQueue(int maxSize) + addMessage(Message message): void + getMessage(String clientID): Message + toString: String

Die Delivery-Queue wird mit einer maximalen Größe vorinitialisiert. Sie besitzt zwei Funktionen zum Ablegen und Herausholen der Message-Objekte aus der Delivery-Queue. Dabei ist der Speicher nach dem FIFO-Prinzip aufgebaut. Ist die maximale Kapazität erreicht, wird das älteste Message-Objekt aus der Liste entfernt.
Die getMessage-Funktion greift zurück auf die getValidMessage-Funktion eines Message-Objekts, um sicherzustellen, dass nur neue, nicht bereits abgerufene Nachrichten an Clients zurückgegebenen werden und dabei nicht innerhalb der Gedächtniszeit *t* des jeweiligen Clients liegen.

MessageServiceServer.java

MessageServiceServer extends UnicastRemoteObject implements MessageService
- SERIALVERSIONUID: long - LOGSEND: File - LOGRECV: File - stringBuffer: StringBuilder - messageService: MessageService - deliveryQueue: DeliveryQueue - registry: Registry - serverName: String - msgResendTime: int - deliveryQueueSize: int - textServerName: JTextField - textQueueSize: JTextField - textMsgResendTime: JTextField - textOutputArea: JTextArea - buttonStartRegistry: JButton - buttonStart: JButton - buttonStop: JButton - buttonFindIp: JButton - buttonShowQueue: JButton
+ MessageServiceServer() + nextMessage(String clientID): String + newMessage(String clientID, String message): void - createAndShowGUI(): void - showDeliveryQueue(): void - findIpAddre(): void - writeLog(String clientID, String message, File logfile): void - startRegistry(): void - startServer(): void - stopServer(): void

Der Server implementiert das MessageService-Interface und konkretisiert somit die beiden Funktionen nextMessage und newMessage. Da die Logik hierfür bereits in der DeliveryQueue-, sowie der Message-Klasse implementiert worden sind, werden die Remote-Aufrufe von Clients direkt weiter delegiert und zwar an ein statisches DeliveryQueue-Objekt, welches die Nachrichten der Clients als Message-Objekte verpackt entgegennimmt, bzw. als solche zurückliefert.
Die At-most-once Fehlersemantik wird mit Hilfe der Gedächtniszeit *t* bei der Implementierung der Message-Klasse realisiert. Demnach verschickt er höchstens einmal dieselbe Nachricht wieder.
Weiterhin besitzt der Server eine GUI, eine Funktion zum Ermitteln der lokalen IP-Adressen, Logging-Funktionalität, sowie die Möglichkeit die Java RMI Registry direkt aus der GUI zu starten und danach den Server an der RMI Registry an (bind) oder ab (unbind) zu melden.

Verteilte Systeme Praktikum WS2014
Teilnehmer: Michael Schmidt, Eugen Winter

MessageServiceClient.java

MessageServiceClient
- LOGFILE: File - stringBuffer: StringBuilder - messageService: MessageService - registry: Registry - messageToSend: String - serverAddr: String - serverName: String - clientID: String - serverTimeout: int - isResending: boolean - textServerAddr: JTextField - textServerName: JTextField - textClientID: JTextField - textServerTimeout: JTextField - textOutputArea: JTextArea - textInputArea: JTextArea - buttonConnect: JButton - buttonDisconnect: JButton - buttonSend: JButton - buttonRecv: JButton
- createAndShowGUI: void - writeLog(String message): void - connect: void - disconnect: void - testConnection: boolean - resendMessage: void - sendMessage: void - receiveMessage: void

Der Client besitzt eine GUI, über die man die Adresse und den Namen des Remote-Servers angeben kann. Ebenso seinen Namen, bzw. die Client-ID und den Wert für den Server-Timeout, falls es zu Problemen bei der Verbindung oder einem Verbindungsabbruch kommen sollte.
Nachrichten werden im unteren Teil des Fensters getippt und mit einem Klick auf den Send-Button oder via Enter-Taste an den Server geschickt.
Mit Hilfe des Receive-Buttons lassen sich alle neuen Nachrichten und Nachrichten außerhalb der Gedächtniszeit *t* abrufen. Diese erscheinen dann in der Mitte im Textfeld.
Die Maybe-Fehlersemantik für das Empfangen von Nachrichten wird durch den Aufruf der receiveMessage-Funktion dahingehend umgesetzt, dass sie versucht einmalig alle neuen Messages vom Server abzuholen. Messages die „null“ sind, also nicht existieren, bleiben dabei unberücksichtigt. Im Fehlerfall wird der Client gewarnt, aber es wird kein erneutes Request gemacht.
Die At-least-once Fehlersemantik in der sendMessage-Funktion ermittelt über die testConnection-Funktion, ob der Server noch in der RMI Registry gelistet ist. Ist dieses nicht der Fall wird die resendMessage-Funktion aufgerufen, um noch einmal einen Request an den Server zu stellen und ihm die Nachricht zukommen zu lassen. Dabei wird die Ausfallzeit *s* benutzt, um einen vom Client vordefinierten Zeitintervall für das erneute Senden der Nachricht einzuhalten. In dieser Zeit versucht der Client sekündlich den Server erneut in der RMI Registry zu finden. Ist die Ausfallzeit *s* abgelaufen, wird die Verbindung getrennt, bzw. die GUI für eine neue Verbindungsaufnahme wieder zurückgesetzt.

Protokollierung

Wireshark:

Hier folgt die Identifizierung und Protokollierung der Kommunikation des Programmes mittels Wireshark.