Team: 09, Alexander Sawadski, Wlad Timotin

Aufgabenaufteilung:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>,
   <Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>,
   <Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden>

Quellenangaben:

-

Begründung für Codeübernahme:

-

Bearbeitungszeitraum:

13.10.2014	4 Std
14.10.2014	11 Std
15.10.2014	30 min

Insgesamt: 15 Std 30 min

Aktueller Stand:

Entwurf fertig zur Übergabe,
Implementierung noch nicht angefangen

Änderungen im Entwurf:

Kapitel 3.1: Holdbackqueue geändert

Kapitel 3.2: Deliveryqueue geändert

Inhalt

Aufgabe 1: Message of the Day

1. Verteilungssicht

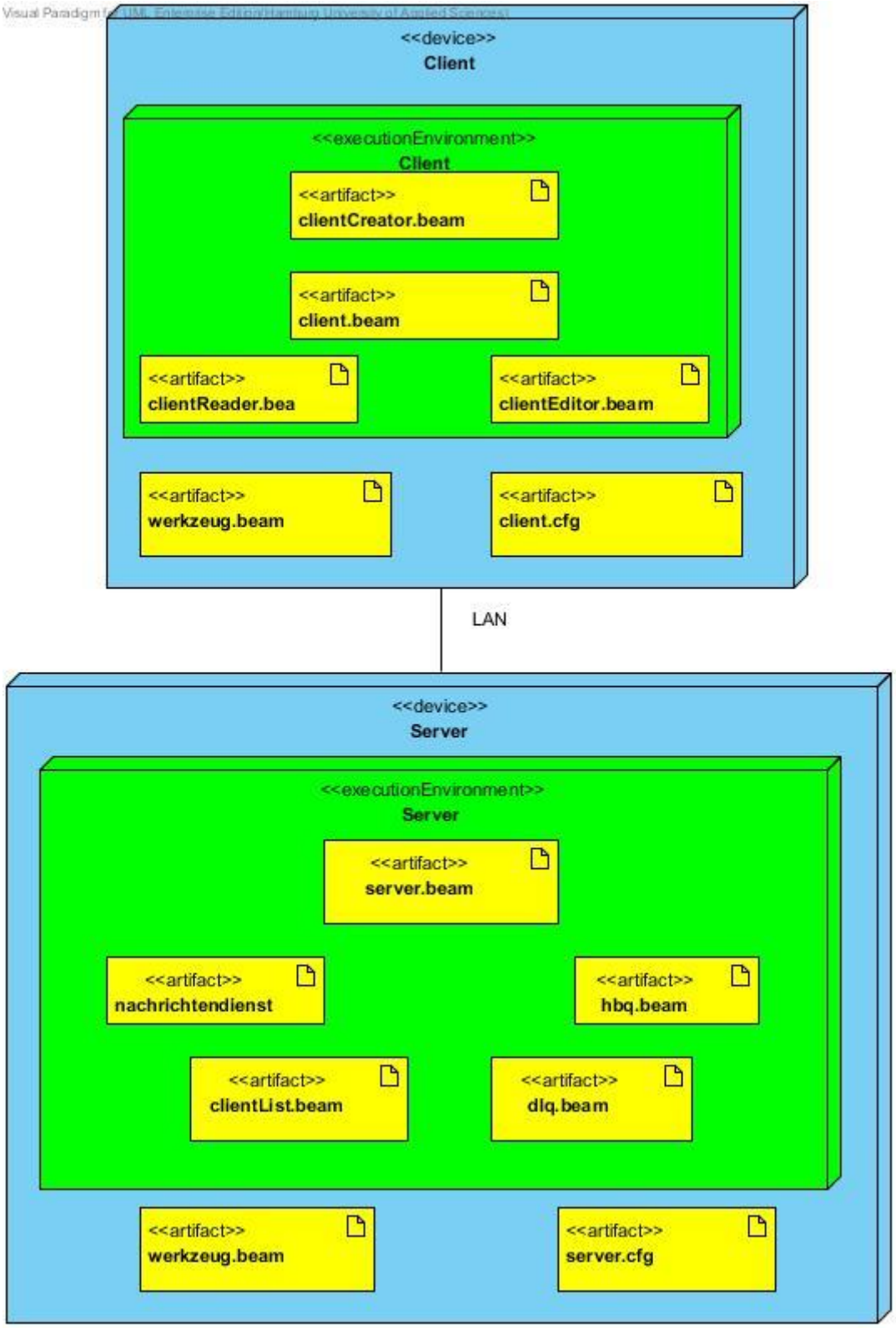

Abbildung 1 Verteilungssicht "Message of the day"

Beschreibung:
Die Abbildung 1 Verteilungssicht „Message of the day“ beschreibt die technische Infrastruktur unserer Anwendung.
Das System besteht aus zwei logischen Teilen:

- einem oder mehreren Clients
- einem Server.

1.1 CLIENT

Der Client besteht aus 2 Prozessen. Auf einem Prozess werden die Arbeiten des Leser- und Redakteur -Clients ausgeführt, der andere Prozess ist für das Loggen zuständig. Das Loggen ist schon im „werkzeug.beam“ implementiert (*logging(Datei,Inhalt)*) und wird dort als Prozess gestartet.

1.1.1 ClientCreator

Beim Start des Artefakts „clientCreator.beam“ wird als Parameter die aktuelle Node des Servers übergeben. Die anderen steuernden Werte werden aus der „client.cfg“ ausgelesen, wie zum Beispiel die Anzahl der zu startenden Clients, welche hier Initialisiert werden.

1.1.2 Client

Das Artefakt „client.beam“ ist für das Zusammenspiel zwischen Lese- und Redakteur –Client verantwortlich. Diese werden abwechselnd aufgerufen.

1.1.3 ClientReader

Das Artefakt „clientReader.beam“ übernimmt die Rolle des Lese-Clients.

1.1.4 ClientEditor

Das Artefakt „clientEditor.beam“ übernimmt die Rolle des Redakteur-Clients.

1.1.5 ClientConfig

Der Client benötigt eine „client.cfg“ mit folgendem Inhalt:

```
{clients, 1}.
{lifetime, 90}.
{servername, wk}.
{sendeintervall, 2}.
{praktikumsgruppe, 3}.
{rechnername, lab1}.
{teamnummer, 9}.
```

Definition:

clients	*Die Anzahl der zu startenden Clients*
lifetime	*Die Lebenszeit eines Clients in Sekunden*
servername	*Der Name des Servers mit dem man sich verbinden will*
sendeintervall	*Der Zeitabstand zwischen den Senden der Nachrichten*
praktikumsgruppe	*Die Nummer der Praktikumsgruppe*
rechnername	*Der Name des Client Rechners*
teamnummer	*Die Nummer des Teams*

1.2 Server

Der Server besteht aus 2 Prozessen. Auf den ersten Prozess werden die Services für die Clients angeboten und ist für die Verwaltung der Queues, sowie die Clientliste zuständig. Der zweite Prozess bearbeitet das Loggen (was im Diagramm nicht zu sehen ist, da es schon im „werkzeug.beam“ vorhanden ist).

1.2.1 Server

Das Artefakt „server.beam“ startet den Server, wobei die steuernden Werte aus der „server.cfg“ ausgelesen werden und bietet die Schnittstellen für die Clients an. Dieser arbeitet mit dem Nachrichtendienst, Nummerndienst und der Holdbackqueue.

1.2.2 Nachrichtendienst

Das Artefakt „nachrichtendienst.beam“ ist für das Versenden der unbekannten Nachrichten an die Leseclients verantwortlich. Dabei bekommt er die jeweiligen Nachrichtennummern von der ClientList und die dazugehörige Nachricht von der DLQ.

1.2.3 ClientList

Die „clientlist.beam“ bietet Methoden zur Verwaltung der bekannten Clients (*siehe Aufgabenstellung: 5te Punkt*) an, mit der entsprechenden Nummer der zuletzt empfangenden Nachricht. Für jeden dieser Clients wird ein Timer gestartet, der nach Ablauf den Client aus der Liste löscht. Der Timer kann zurückgesetzt werden, wenn der LeseClient eine Nachricht anfordert.

1.2.4 HoldbackQueue (HBQ)

Das Artefakt „holdbackqueue.beam“ speichert alle gesendeten Nachrichten von Clients, die noch nicht versendet werden dürfen.
Eine Nachricht wird von der HBQ in die DLQ transferiert, falls dessen Nachrichtennummer mit der größte Nachrichtennummer von der DLQ eine Differenz von 1 besitzt.
Falls die Anzahl an Nachrichten in der HBQ mehr als 50% der maximalen Größe der DLQ ist, wird eine Fehlernachricht erstellt. Die Nachrichtennummer der Fehlernachricht ist gleich der kleinsten Nachrichtennummer der HBQ – 1.

Annahmen:

- Wenn eine Nachricht mit der Nachrichtennummer kommt, welche kleiner gleich der maximalen Nachrichtennummer der DLQ ist, wird diese nicht in die HBQ aufgenommen.
- Es wird nur dann transferiert, wenn der Server eine Nachricht vom Client erhält.

1.2.5 DeliveryQueue (DLQ)

Das Artefakt „deliveryqueue.beam“ speichert alle zu sendenden Nachrichten von Clients.

1.2.6 ServerConfig

Der Server benötigt eine „server.cfg“ mit folgendem Inhalt:

```
{latency, 42}.
{clientlifetime,2}.
{servername, wk}.
{dlqlimit, 50}.
```

Definition:

latency	*Die Lebenszeit des Servers (bei Anfragen wird aktualisiert), in Sekunden*
clientlifetime	*Die Zeit wie lange der Server sich einen Client merkt, in Sekunden*
servername	*Der Name des Servers*
dlqlimit	*Die maximale Anzahl an Nachrichten in der Deliveryqueue*

2. Schnittstellen

Im Folgenden wird die Schnittstelle des Servers für einen Client beschrieben. Da eine gemeinsame Vorführung stattfindet, ist sie unbedingt einzuhalten!

```
/* Abfragen aller Nachrichten */
Server ! {getmessages, self()},
receive {reply,Number,Nachricht,Terminated} ->

/* Senden einer Nachricht */
Server ! {dropmessage, {Nachricht, Number}},

/* Abfragen der eindeutigen Nachrichtennummer */
Server ! {getmsgid, self()}
receive {nid, Number} ->
```

2.1 getmessages

Fragt beim Server eine aktuelle Textzeile ab. self() stellt die Rückrufadresse des Lese-Clients dar. Als Rückgabewert erhält er eine für ihn aktuelle Textzeile zugestellt (Nachrichteninhalt) und deren eindeutige Nummer (Number). Mit der Variablen Terminated signailiert der Server, ob noch für ihn aktuelle Nachrichten vorhanden sind. Terminated == false bedeutet, es gibt noch weitere aktuelle Nachrichten, Terminated == true bedeutet, dass es keine aktuellen Nachrichten mehr gibt, d.h. weiter Aufrufe von getmessages sind nicht notwendig.

2.2 dropmessage

Sendet dem Server eine Textzeile (Nachricht), die den Namen des aufrufenden Clients und seine aktuelle Systemzeit sowie ggf. irgendeinen Text beinhaltet (z.B.), zudem die zugeordnete (globale) Nummer der Textzeile (Number).

2.3 getmsgid

Fragt beim Server die aktuelle Nachrichtenummer ab. self() stellt die Rückrufadresse des Lese-Clients dar. Als Rückgabewert erhält er die aktuelle und eindeutige Nachrichtennummer (Number).

3. ADTs

3.1 Holdbackqueue

Signatur	Beschreibung
createHBQ() : HBQ	liefert eine **Leere Queue** zurück. Return: **HBQ** ist eine Liste
dropmessage(HBQ,DLQ, {Nachricht, ID}, MaxDLQ) : {NewHBQ, NewDLQ}	Fügt eine Nachricht in die Holdbackqueue und Prüft ob ein Transfer von Nachrichten möglich ist. Falls das der Fall ist wird in die DLQ transferiert. Parameter: **HBQ** ist eine Liste von Nachrichten **DLQ** ist eine Liste von Nachrichten mit fester Größe **Nachricht** ist ein String **ID** ist ein Integer **MaxDLQ** ist ein Integer, zeigt die maximale Größe der DLQ Return: **NewHBQ** ist die aktuelle HBQ **NewDLQ** ist die aktuelle DLQ

3.2 Deliveryqueue

createDLQ(MaxSize) : DLQ	Erstellt eine leere DLQ. Die MaxSize gibt die größe der DLQ an. Parameter: **MaxSize** ist ein Integer Return: **DLQ** ist eine Liste
add(DLQ, {Nachricht, ID}) : NewDLQ	Fügt eine Nachricht in die Deliveryqueue hinten dran und löscht vordere Nachricht wenn die DLQ voll ist. Parameter: **DLQ** ist eine Sortierte Liste von Nachrichten **Nachricht** ist ein String **ID** ist ein Integer Return: **NewDLQ** ist die aktuelle DLQ Liste
getMaxID(DLQ) : ID	Liefert die größte Nachrichtennummer aus der Deliveryqueue. Parameter: **DLQ** ist eine Sortierte Liste von Nachrichten Return: **ID** ist ein Integer, zeigt größte Nachrichtennummer aus DLQ
getNext(NachrichtenNr) : {NewNachrichtenNr, Nachricht, Terminated}	Liefert die nächstgrößere Nachrichtennummer mit der dazugehörigen Nachricht und einen Flag

	ob die größte Nachrichtennummer zurückgeliefert wurde. Parameter: **NachrichtenNr** ist ein Integer Return: **NewNachrichtenNr** ist ein Integer **Nachricht** ist ein String **Terminated** ist ein Wahrheitswert

4. Sequenzdiagramme

4.1 Client-Initialisierung

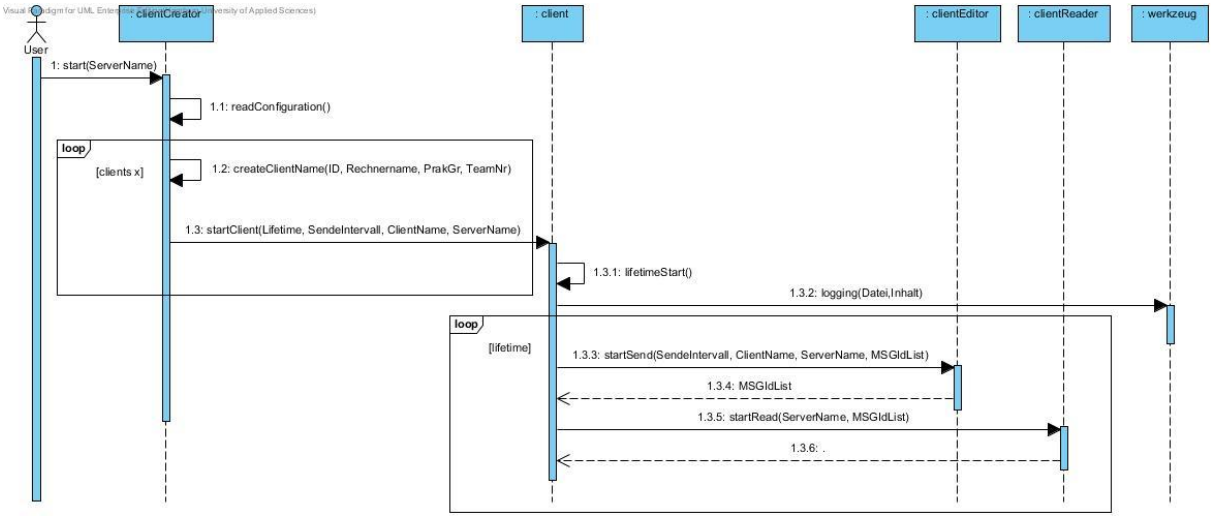

Abbildung 2 SeqDiagramm Client-Initialisierung

Beschreibung:

1:start(ServerName) :

Beim Start des Clients wird die aktuelle Node des Servers als Parameter übergeben.

1.1: readConfiguration():

Die steuernden Werte werden aus der Datei „*client.cfg*“ ausgelesen, wie zum Beispiel die Anzahl der Clients (*clients*), die Lebenszeit eines Clients (*lifetime*), den Rechnername, die Praktikumsgruppe, die Teamnummer sowie den Zeitabstand bei dem der Redakteur-Client seine Nachrichten sendet (*sendeintervall*). Die Anzahl der Clients bestimmt die Durchläufe der Schleife „*clients x*“.

1.2:createClientName(ID, Rechnername, PrakGr, TeamNr):

Der Name des Clients wird erstellt (ClientName). Der Name des Clients sieht folgendermaßen aus: „<Nummer>-client“.

1.3:startClient(Lifetime, SendeIntervall, ClientName, ServerName):

Bei der Initialisierung jedes Clients werden die Lifetime, das Sendeintervall, der Name des Clients und der Servername übergeben.

1.3.1: lifetimeStart():

Zunächst wird ein Timer für die Lifetime gestartet. Wenn der Timer abgelaufen ist, wird der Client terminiert.

1.3.2: logging(Datei, Inhalt):

Beim Logging wird protokolliert in welcher Zeit der Client gestartet wird.

(Datei) : Die Logdatei vom Client.

(Inhalt): *siehe 5.2 GUI* Client: Client start

loop [lifetime]:

Die Schleife „lifetime“ läuft solange die Lifetime eines Clients nicht abgelaufen ist. Hierbei wird zwischen dem Redakteur- und Lese-Client gewechselt.

1.3.3: startSend(SendeIntervall, ClientName, ServerName,MSGIdList):

Zuerst wird der Redakteur-Client aufgerufen, hier erhalten wir die Liste aller Nachrichtennummern des Clients ***1.3.4: MSGIdList***.

1.3.5:startRead(ServerName, MSGIdList):

Mit der Liste aller Nachrichtennummern und den Servernamen wird der Lese-Client aufgerufen.

4.2 Client-Redakteur

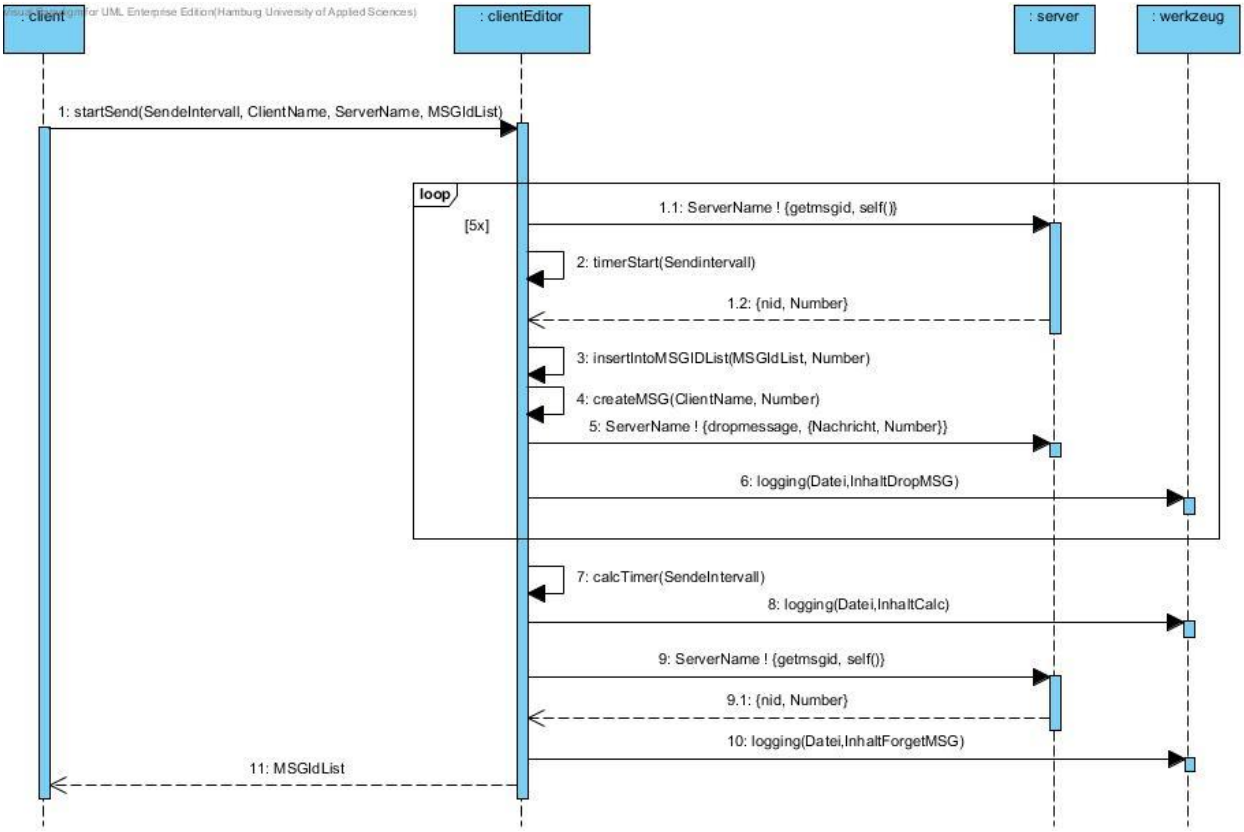

Abbildung 3 SeqDiagramm Client-Redakteur

Beschreibung:

1: startSend(SendeIntervall, ClientName, ServerName, MSGIdList) :

Der Redakteur-Client (*clientEditor*) wird aufgerufen.

loop 5x:

Die Schleife „5 x“ wird 5-mal durchlaufen.

1.1: Server ! {getmsgid, self()}:

Abfragen einer eindeutigen Nachrichtennummer beim Server und erhalten der Antwort vom Server ***1.2: {nid, Number}***.

2: timerStart(Sendintervall):

SendeIntervalltimer wird gestartet. Nach Ablauf des Timers wird die Nachricht versendet.

3:insertIntoMSGIDList(MSGIdList, Number):

Die erhalte Nachrichtennummer wird in der „*MSGIdList*“ Liste gespeichert, dies dient zur Erkennung der eigenen Nachrichten für den Lese-Client.

4: createMSG(ClientName, Number):

Aus dem ClientNamen und der Nachrichtennummer wird eine neue Nachricht erstellt.

5: Server ! {dropmessage, {Nachricht, Number}}:

Nachricht wird an den Server gesendet,

6:logging(Datei,InhaltDropMSG):

Das senden der Nachricht wird in der Log-Datei protokolliert.

7:calcTimer(SendeIntervall):

Nach den 5 Durchläufen wird das Sendeintervall neu berechnet.

8: logging(Datei,InhaltCalc):

Das neue Sendeintervall wird geloggt.

9: Server ! {getmsgid, self()}:

Eine neue Nachrichtennummer wird beim Server abgefragt, zu der keine Nachricht versendet wird. Die Antwort des Servers wird abgefangen ***9.1: {nid, Number}.***

10: logging(Datei,InhaltForgetMSG):

Das Vergessen der Nachricht wird protokolliert.

11:MSGIdList:

Zum Schluss wird die Liste aller abgefragten Nachrichtennummern zurückgeliefert.

4.3 Lese-Client

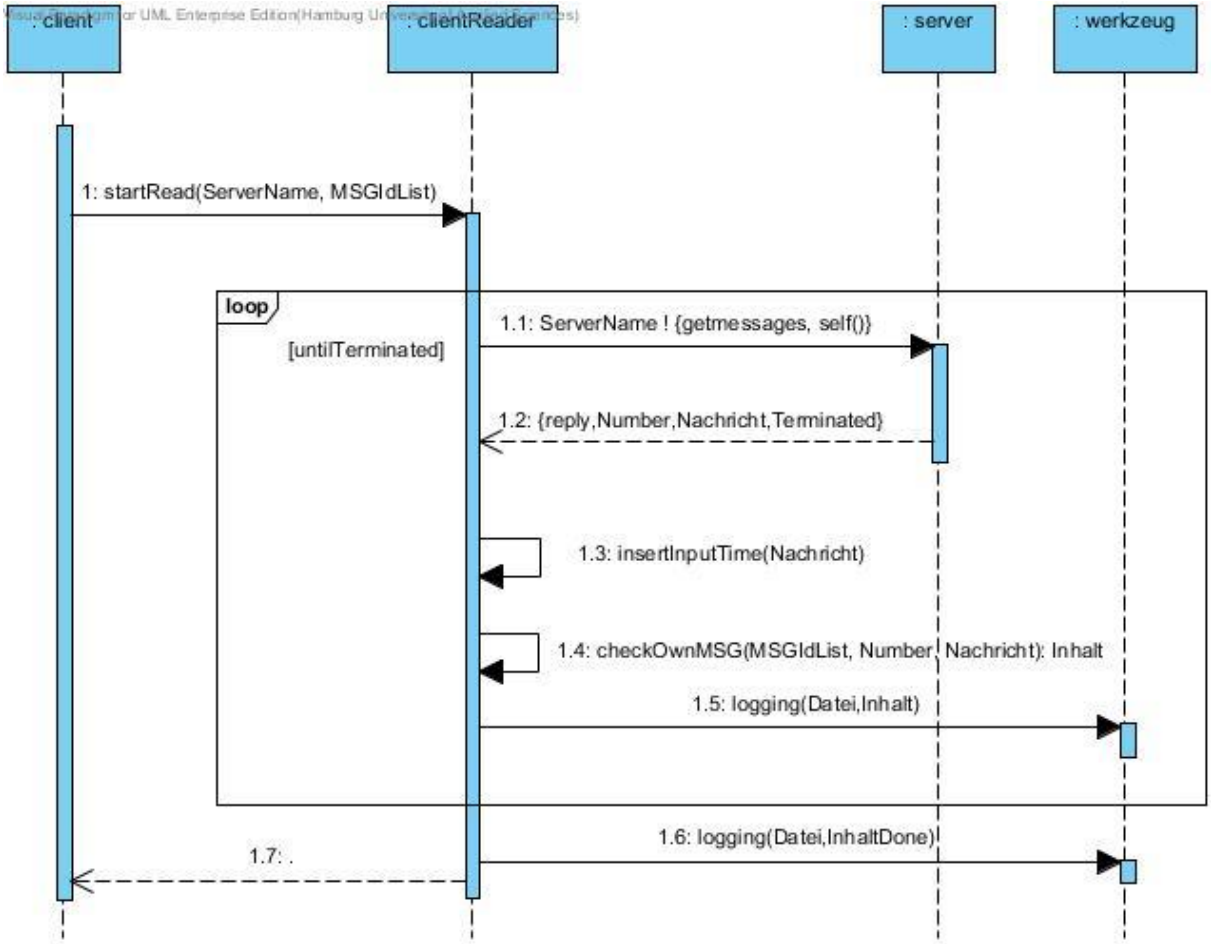

Abbildung 4 SeqDiagramm Lese-Client

Beschreibung:

1:startRead(ServerName, MSGIdList):

Der Lese-Client bekommt die MSGIdList und den Servernamen.

Loop untilTerminated:

Die Schleife „*untilTerminated*“ läuft solange bis der Server eine Antwort mit **Terminated = true** zurückliefert **1.2:{reply,Number,Nachricht,Terminated}**.

1.1: {getmessages, self()}:

Ansonsten wird immer wieder eine neue Anfrage einer Nachricht an den Server gestellt.

1.3:insertInputTime(Nachricht):

fügt zur Nachricht die Empfangszeit hinzu.

1.4:checkOwnMSG(MSGIdList, Number, Nachricht):

Nach dem Empfangen einer Nachricht wird geprüft ob es sich um eine eigene Nachricht handelt, in diesem Fall werden „*******“ an die Nachricht angehängt.

1.5:logging(Datei,Inhalt):

Die Nachrichten werden in die Log-Datei geschrieben.

1.6:logging(Datei,InhaltDone):

Zum Schluss wird geloggt das Client fertig mit Lesen ist.

4.4 Server

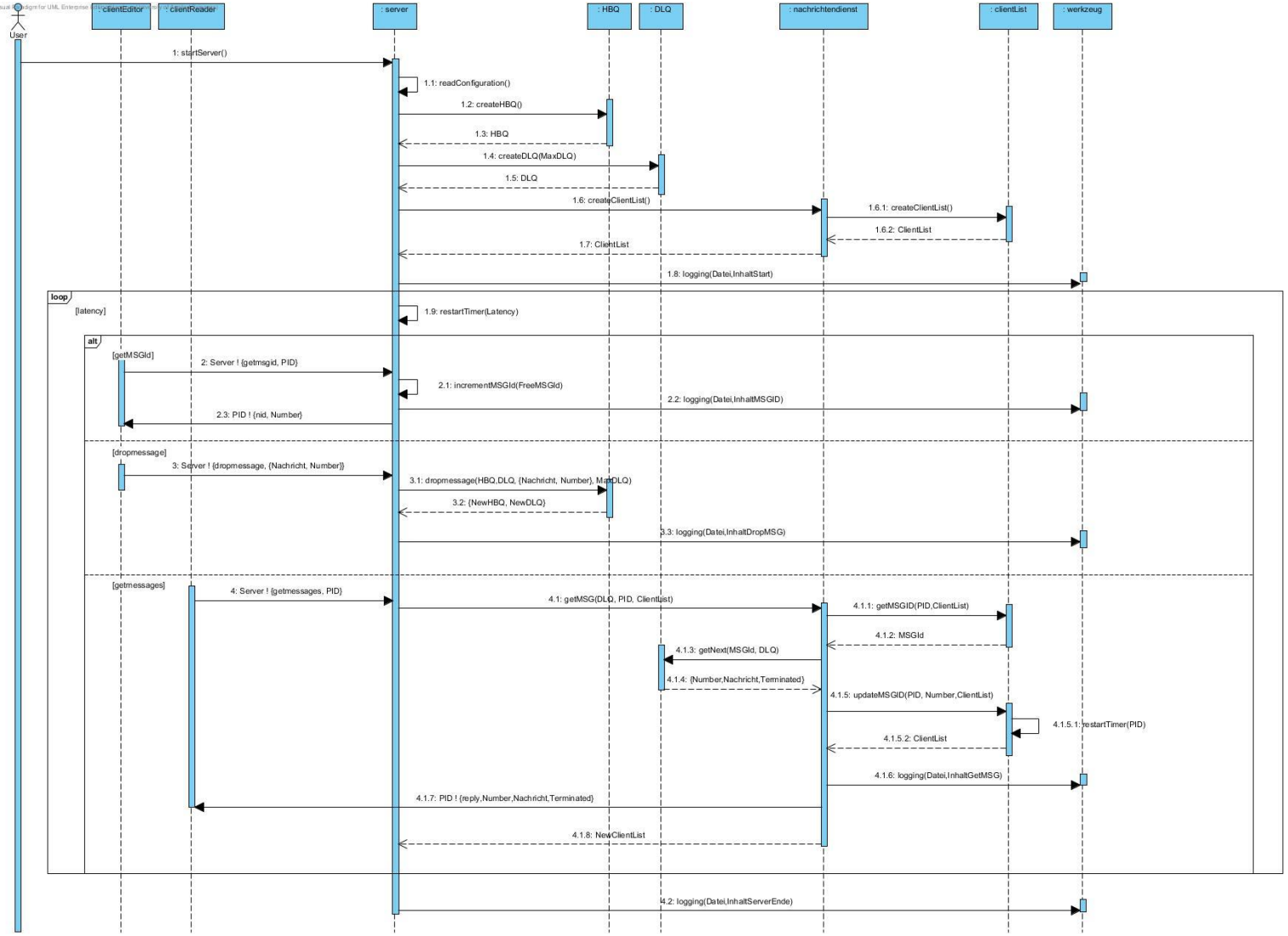

Abbildung 5 SeqDiagramm Server

Beschreibung:

1:startServer():

Der User startet den Server.

1.1:readConfiguration():

Die steuernden Werte werden aus der „*server.cfg*“ gelesen**.**

1.2:createHBQ():

Erstellt die HBQ.

1.3:HBQ:

Liefert die HBQ zurück.

1.4:createDLQ(MaxDLQ):

Erstellt die DLQ.

1.5:DLQ:

Liefert die DLQ zurück.

1.6:createClientList()

Ruft im Nachrichtendienst die Methode zur Erstellung der ClientList auf.

1.6.1:createClientList()

Ruft in der ClientList die Methode zur Erstellung der ClientList auf.

1.6.2: ClientList:

Liefert ClientList von ClientList an Nachrichtendienst.

1.7: ClientList:

Liefert ClientList von Nachrichtendienst an Server.

1.8:logging(Datei,InhaltStart):

Der Start des Servers wird in einer Log-Datei protokolliert.

loop Latency:

Der Server terminiert sich wenn die Differenz von der aktuellen Systemzeit und die Zeit der letzten Abfrage eines Clients länger als seine Wartezeit „*latency*“ beträgt.

1.9:restartTimer(Latency):

Nach jeder Abfrage eines Clients wird der Timer neugestartet.

Es gibt **3 Alternativen** den Server zu kontaktieren:

2. Server ! {getmsgid, PID}:

Abfragen der eindeutigen Nachrichtennummer.

2.1:incrementMSGId(FreeMSGId):

Die im Server gespeicherte freie Nachrichtennummer, wird incrementiert und an den Client zurückgesendet

2.2:logging(Datei,InhaltMSGID):

Loggt das Senden der Nachrichtennummer an der Client.

2.3:Pid ! {nid, Number}:

Antwort vom Server.

3:Server ! {dropmessage, {Nachricht, Number}}:

3.1:dropmessage(HBQ, DLQ, {Nachricht, Number}, MaxDLQ):

Speichern der Nachricht in die Holdbackqueue.

3.2:{NewHBQ, NewDLQ}:

Liefert die aktuelle Holdbackqueue und Deliveryqueue zurück.

3.3:logging(Datei,InhaltDropMSG):

Loggt, dass der Server eine Nachricht bekommen hat.

4:Server ! {getmessages, PID}:

4.1:getMSGID(DLQ, PID, ClientList):

Nachrichtendienst Anfordern eine Nachricht an einen Client zu senden.

4.1.1: getMSGID(PID, ClientList):

Welche Nachrichtnummer hat der Client zuletzt bekommen.

4.1.2:MSGId:

Wenn der Client noch nicht in der Liste vorhanden ist, wird „0“ zurückgeliefert.

4.1.3:getNext(MSGID, DLQ):

Holt aus der DLQ die Nächste unbekannte Nachricht.

4.1.4:{Number,Nachricht,Terminated}:

Liefert die Nachricht, Nachrichtennummer und ein Flag ob es die letzte Nachricht für eine Client ist zurück.

4.1.5:updateMSGID(PID,Number,ClientList):

Aktualisiert die ClientList.

4.1.5.1:restartTimer(PID):

Hier wird der Timer für den Client gestartet.

4.1.5.2:ClientList:

Liefert aktuelle ClientList zurück.

4.1.6:logging(Datei,InhaltGetMSG):

Loggt, dass der Server eine Nachricht an Client schickt

4.1.7:PID ! {reply,Number,Nachricht,Terminated}:

Sendet an Client die Nachricht

4.1.8: NewClientList:

Liefert den Server die aktuelle ClientList zurück

4.2:logging(Datei,InhaltServerEnde):

Loggt das sich der Server beendet hat.

5. GUI

5.1 Struktur:

LogDatei Client	'client_'**<ClientID><Clienthost>**'.log'
LogDatei Server	'Server_'**<ClientID><Clienthost>**'.log'
ClientName	**<ClientID>**'-client@'**<ClientHost>**'-'**<CPID>**'-Gruppe'**<GruppenNr>**'-'**<TeamNr>**
ClientID	**<Zahl>**
ClientHost	Aus Der Config-Datei (**rechnername**)
CPID	ProzessID des Clients
SPID	ProzessID des Servers
GruppenNr	Aus Der Config-Datei (**praktikumsgruppe**)
TeamNr	Aus Der Config-Datei (**teamnummer**)
Systemzeit	DD.MM hh:mm:ss,ms'|'
NachrichtenNr	**<Zahl>**
Textzeile1	**<ClientName>**': '**<NachrichtenNr>**'te_Nachricht. C Out: '**<Systemzeit>**
Textzeile2	**<Textzeile1>**'('**<NachrichtenNr>**'); HBQ In: '**<Systemzeit>**
Textzeile3	**<Textzeile2>**' DLQ In: '**<Systemzeit>**.
FehlerNachricht	***Fehlernachricht fuer Nachrichten **<NachrichtenNr>** bis **<NachrichtenNr>** um **<Systemzeit>**.
NeuerWert	**<Zahl>**
AlterWert	**<Zahl>**
NachrichtenList	'['' **<NachrichtenNr>** [','**<NachrichtenNr>**]* ']'
Zahl	[**1-9**][**0-9**]*

5.2 Client

Für jeden Client wird eine *.log-Datei erstellt.

Client start
<ClientName> Start: **<Systemzeit>**.
Client sendet Nachricht an Server
<Textzeile1> gesendet
Vergessene Nachricht vom Client
<NachrichtenNr>te_Nachricht um **<Systemzeit>** vergessen zu senden ******
Client bekommt eigene Nachricht vom Server
<Textzeile3>******; C In: **<Systemzeit>**
Client bekommt eine fremde Nachricht vom Server
<Textzeile3>|<FehlerNachricht> ; C In: **<Systemzeit>**
Client hat alle Nachrichten vom Server bekommen
..getmessages..Done...
Client hat neues Sendeintervall berechnet
Neues Sendeintervall: **<NeuerWert>** Sekunden (**<alterWert>**)

5.3 Server

Nur eine *.log-Datei wird erstellt.

Server start
`Server Startzeit: <Systemzeit> mit PID <SPID>`
Server sendet Nachrichtennummer an Client
`Server: Nachrichtennummer <NachrichtenNr> an <CPID> gesendet`
Server bekommt eine Nachricht vom Client
`<Textzeile2>-dropmessage`
Nachrichten werden von der HBQ in die DLQ transferiert
`QVerwaltung>>> Nachrichten <NachrichtenList> von HBQ in DLQ transferiert.`
Server schickt Client eine Nachricht mit Terminiert false
`<Textzeile3>|<FehlerNachricht>(<NachrichtenNr>)-getmessages von <CPID>-false`
Server schickt Client eine Nachricht mit Terminiert true
`<Textzeile3>|<FehlerNachricht>(<NachrichtenNr>)-getmessages von <CPID>-true`
Client wird von Server vergessen
`Client <CPID> wird vergessen! **********`
Fehlernachricht wird erzeugt
`<FehlerNachricht>`
Nachrichten werden von DLQ gelöscht
`QVerwaltung>>> Nachrichten <NachrichtenList> von DLQ geloescht.`
Server beendet sich
`Downtime: <Systemzeit> vom Nachrichtenserver <SPID>; Anzahl Restnachrichten in der HBQ:<Zahl>`

Abbildungsverzeichnis