Team: 9, David Asmuth, Vladimir Malikov

Aufgabenaufteilung:

1. Entwurf, Vladimir Malikov.
2. Entwurf, David Asmuth.

Quellenangaben: http://users.informatik.haw-hamburg.de/~klauck/VerteilteSysteme/aufg1-rmi.html

Begründung für Codeübernahme: Es wurde kein Code übernommen.

Bearbeitungszeitraum:

28.03.2015: 1 Stunde
29.03.2015: 5 Stunden
30.03.2015: 2 Stunden
31.03.2015: 3 Stunden

Aktueller Stand: Entwurf fertig. Die Implementation noch nicht angefangen.

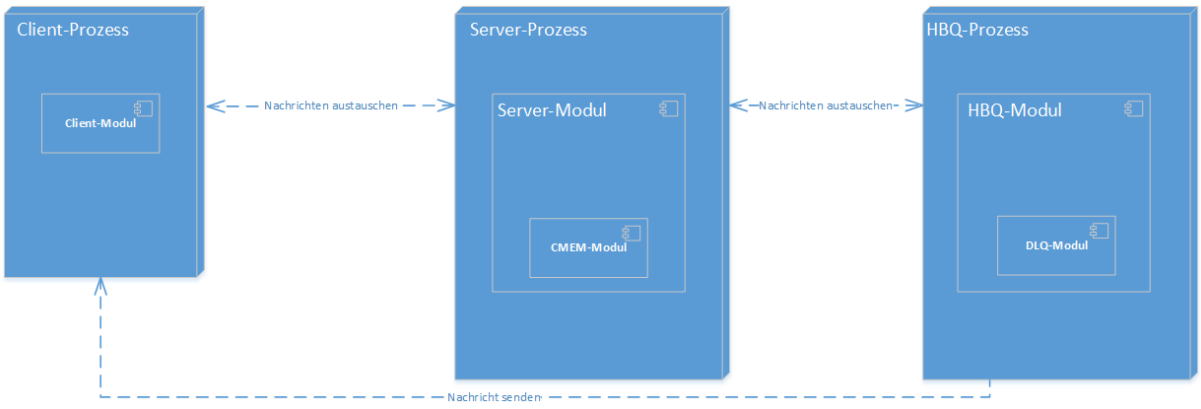
Änderungen im Entwurf: …

Entwurf:

Aufgabe:

Die Aufgabe besteht darin ein System zu entwickeln, was den Austausch von Nachrichten zwischen den Lesern(Client) und Redakteur(Client) ermöglich. Dieser erfolgt über ein Server, der auch die Verwaltung der Nachrichten übernimmt. Alle Nachrichten kriegen eine eindeutige Nummer zugewiesen. Es soll außerdem gewährleistet sein, dass die Möglichkeit besteht Einzelteile durch Fremd-Implementierung zu ersetzten. Das wird durch die Einhaltung der Schnittstellen gewährleistet.

Aufbau:



Der Client-Prozess, Server-Prozess und HBQ-Prozess sind selbständig laufende Einheiten. Bei der Terminierung des Server-Prozess, soll auf den richtigen Verlauf geachtet werden, indem erst der HBQ-Prozess beendet wird und anschließend das Server-Modul selbst.

Client:

Der Client übernimmt abwechselnd die Rolle von Leser und Redakteur. Der Client soll die Nachrichten, die von ihm(als Redakteur) zu ihm (als Leser) beim Schreiben ins Log kenntlich machen. Außerdem hat jeder Client eine Lebenszeit nach der er terminiert. Diese Zeit wird im client.cfg angegeben.

Client/Leser:

Der Client/Leser fragt einzeln nach jeder Nachricht bei dem Server ab. Aus der Antwort kann der Client/Leser entnehmen, ob es weitere Nachrichten für ihn gibt. Ist das der Fall, so schickt der Client/Leser noch eine Anfrage an Server. Gibt es keinen neuen Nachrichten, so wird wechselt der Client/Leser in die Redakteurs Rolle.

Der Client/Leser überprüft jede eingehende Nachricht, ob sie von ihm stammt. Ist der Fall, so soll beim Schreiben ins Log diese Nachricht markiert werden.

Client/Redakteur:

Der Client/Redakteur sendet einzeln die Nachrichten an den Server. Bevor er das, aber tut, fragt er wieder einzeln für jede Nachricht beim Server nach einer eindeutigen Nummer. Die Nachrichten werden in bestimmten Zeitabstand gesendet. Diese ist im client.cfg anzugeben. Die Wartezeit ist zwischen der Anfrage des Nummer und Absendet, sowie nach dem Absendet und Anfrage nächster Nummer abzuwarten.

Die Nachricht soll Client Name (sein Rechnernamen(z.B.: lab18), die Praktikumsgruppe (z.B.: 3) und die Teamnummer (z.B.: 9).

0-client@lab18309: 9te_Nachricht. Sendezeit: 1.04 18:01:30,769|(9);

Für die 5te Nachricht wird nur eine neue Nachrichtennummer angefragt ohne die Nachricht selber zu senden. Dies wird, aber in dem Log vermerkt:

14te_Nachricht um 1.04 18:02:15,769| vergessen zu senden ******

Der Client/Redakteur speichert die Nummer aller abgesendeten Nachrichten ab und teilt dann diese dem Client/Leser, bei der Wechsel von Client/Redakteur zu Client/Leser, mit. Der Wechsel findet nach 5ter Nachricht statt.

Server:

Der Server besteht aus zwei Prozessen, die über eine Schnittstelle kommunizieren. Server selbst und die ADTs hbq/dlq. Der Server-Prozess enthält daneben noch eine ADT cmem.

Server-Prozess:

Der Server-Prozess empfängt die Nachrichten von Client, sowie verwaltet Nummernausgabe und Leser. Der Server-Modul soll robust gegen falschen Nachrichten sein. Der Empfang solcher Nachrichten, soll zu keiner Störung führen. Dabei wird folgende Schnittstelle implementiert:

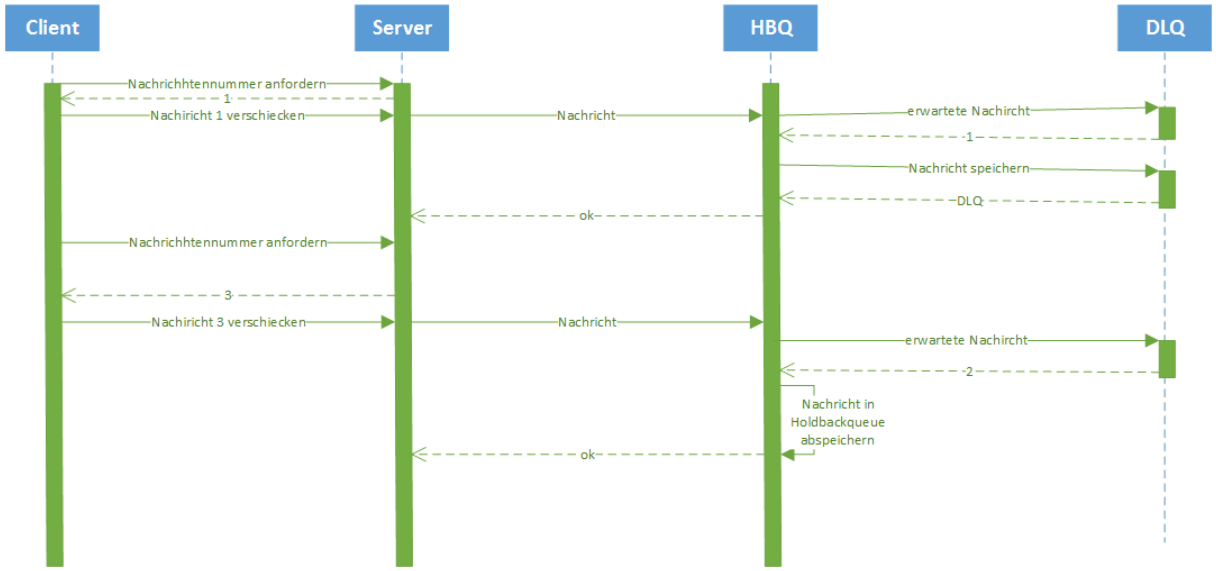
```
/* Abfragen einer Nachricht */
receive {self(), getmessages}

/* Empfangen einer Nachricht */
receive {dropmessage,[INNr,Msg,TSclientout]},

 /* Abfragen der eindeutigen Nachrichtennummer */
receive {self(),getmsgid}
```

Bei jeder Empfangenen und übertragen der Nachricht an Holdbackqueue wird die Empfangszeit hinten rechts an die Nachricht angefügt.

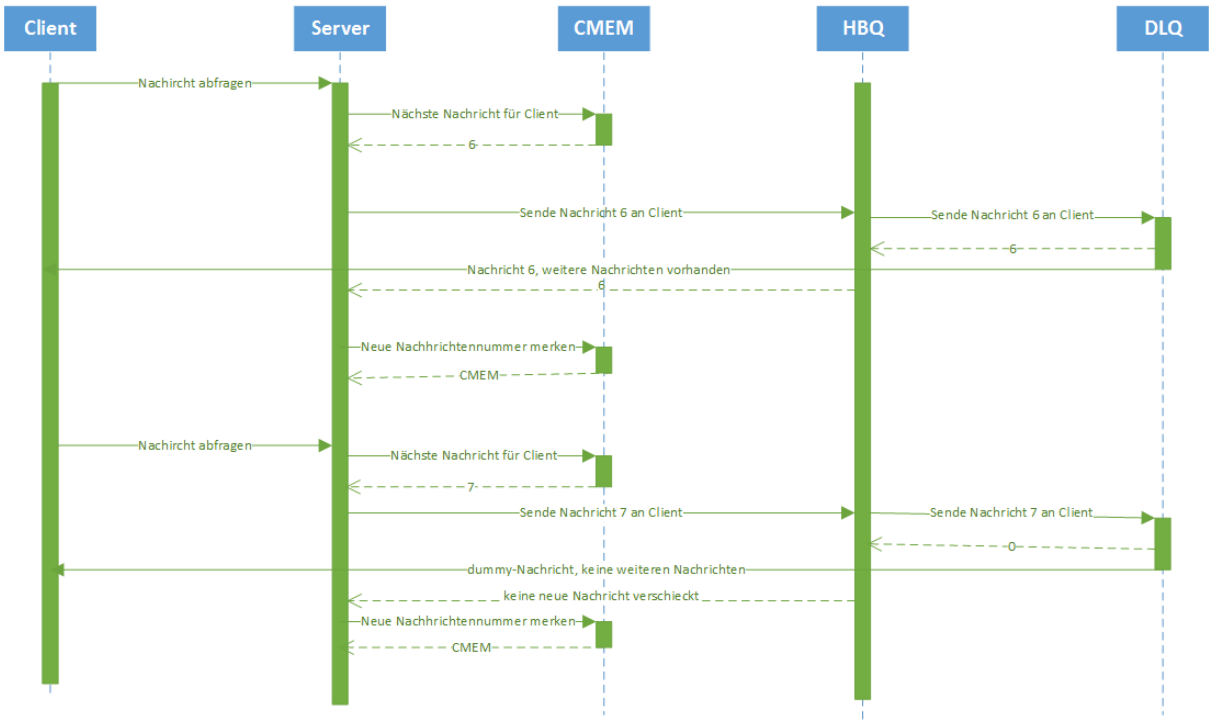
Fordert ein Client eine Nachricht an, so soll erstmals in cmem überprüft werden, ob er schon bekannt ist. Ist das der Fall, so wird bei cmem abgefragt, welche letzte Nachrichtsnummer der Client bekommen hat. Diese wird Holdbackqueue mitgeteilt, die dann senden passender Nachricht regelt. Ist der Client nicht bekannt, so wird Holdbackqueue eine 1 übergeben. Gibt es danach keine weiteren Nachrichten, die Client kriegen könnte, so wird ihm es in einem Flag in der Antwort mitgeteilt. Gibt es überhaupt keine Nachrichten, die versendet werden können, so schickt der Server eine nicht leere dummy-Nachricht.

Fordert ein Client eine eindeutige Nummer, so wird diese ohne weiteres vergeben.

Kriegt der Server innerhalb von gegeben Wartezeit keine neue Anfragen, so terminiert er. Diese Wartezeit ist im server.cfg anzugeben.



Hier kann man sehen, wie ein Client in der Rolle des Redakteurs eine Nachricht an Server schickt. Entsprechend der Vorgaben fordert der Client eine eindeutige Nummer an und versendet danach die Nachricht. Der Server hat dabei die Aufgabe die Nummern zu vergeben und die Nachrichten an den HBQ-Prozess weiterzuleiten. Im dem HBQ-Prozess wird dann die Entscheidung getroffen, ob die Nachricht in die Holdbackqueue soll, oder direkt in die Deliveryqueue geleitet wird. Der erste dargestellte Fall ist eine Nachricht mit der Nummer 1. Diese Nachricht wird von DLQ erwartet und daher direkt in die DLQ geleitet. Im zweiten dargestellten Fall wird aber die Nummer 2 erwartet, also wird die Nachricht in der Holdbackqueue gehalten.



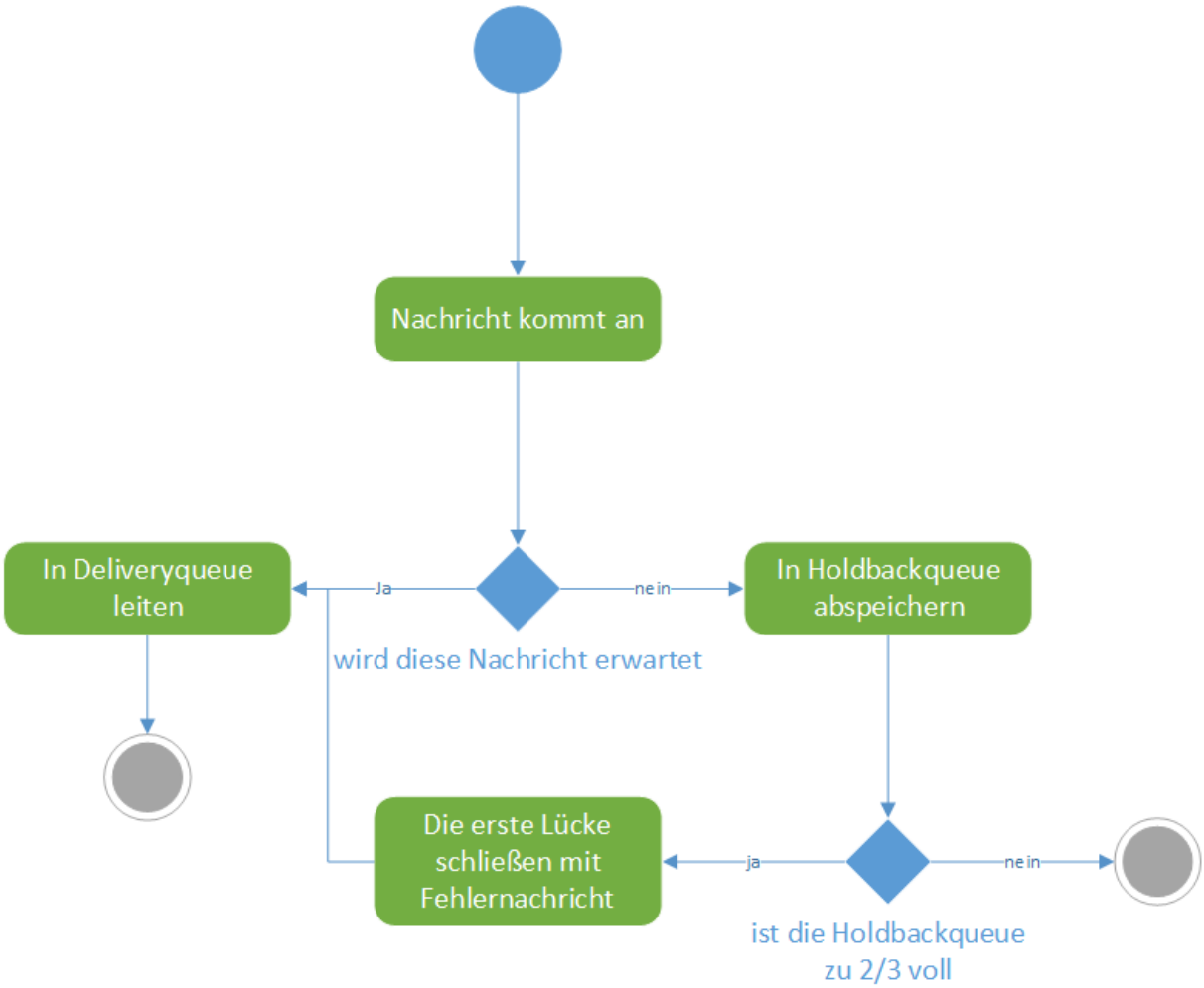
Hier kann man sehen wie der Client in der Leser-Rolle die Nachrichten von den Server anfordert. Dabei prüft Server in der CMEM ob der Client schon bekannt ist. Das ist diesmal der Fall. Der Server kriegt die passende Nachrichtennummer (enthält die Nummer die der Client noch nicht gelesen hat [6]) und leitet die Anfrage weiter an den HBQ-Prozess. Die leitet es wiederum an DLQ weiter. Die DLQ entscheidet dann ob die Nachricht versendet werden kann. Im ersten Fall ist es so und der Client erhält die Nachricht mit der Nummer 6. Der Server erhält ebenfalls eine Nachricht vom HBQ-Prozess, mit der höchsten Nummer der gesendeten Nachrichten (6). Der Server updatet im CMEM entsprechend mit der Nummer 7. Im zweiten Fall hat die DLQ keine weiteren Nachrichten und sendet eine dummy-Nachricht. Der Server erhält diesmal, als versendete Nachrichtnummer 0. Damit wird dem Server mitgeteilt, dass es keine normale Nachricht versendet wurde.

In beiden Sequenzdiagrammen wird der Ablauf der Kommunikation zwischen einem Client und Server dargestellt. Es kann aber sein, dass es mehrere Client gibt's. Dadurch kann es vorkommen, dass der Server zuerst eine Anfrage von Client A und dann von Client B kriegt. Es soll sichergestellt werden, dass die Clients trotzdem die Richtigen Antworten kriegen.

HBQ-Prozess:

Der HBQ-Prozess verwaltet die Nachrichten und schickt diese an den Client.

Bekommt der HBQ-Prozess die Aufforderung eine Nachricht zu schicken, so wird diese weiter an das DLQ-Modul delegiert. Diese Prüft, ob die Nachricht mit der Nummer vorhanden ist und schickt diese raus. Ist das nicht der Fall, aber die nächste Nummer in DLQ ist größer, als angegebene, so wird diese abgeschickt. In der Antwort für der Client wir durch ein Flag mitgeteilt, ob es weitere Nachrichten für ihn gibt.Die neue Nummer wird dann auch noch in der Antwort für Server-Modul eingetragen. Ist die angegeben Nummer größer, als die größte in DLQ, so kriegt der Client eine nicht leere dummy-Nachricht mit den Flag *true,* also keine weiter Nachrichten für den Client.



Bekommt der HBQ-Modul eine neue Nachricht, so prüft es zuerst, ob das die nächste von DLQ zu erwartende Nummer ist. Ist das der Fall, so wird die Nachricht direkt in die DLQ eingetragen. Ist das nicht der Fall, kommt die Nachricht in die HBQ. Ist die HBQ zu 2/3 voll (es richtet sich nach DLQ), so wird die erste Lücke durch eine Fehlernachricht geschlossen. Die Nachrichten werden dann bis zu nächster Lücke in die DLQ übertragen.

CMEM-Modul:

Ist ein Teil von Server-Prozess und speichert ab, welcher Client welche Nachricht zuletzt bekommen hat. CMEM vergisst den Client nach vorgegebener Zeit. Diese Zeit wird im server.cfg angegeben. Der CMEM wird über eigene Schnittstelle angesprochen.

Schnittstellen:

Server-Prozess:

```
/* Abfragen einer Nachricht */
Server ! {self(), getmessages}
receive {reply,[NNr,Msg,TSclientout,TShbqin,TSdlqin,TSdlqout],Terminated}

/* Senden einer Nachricht */
Server ! {dropmessage,[INNr,Msg,TSclientout]},
```

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

Server-Prozess/CMEM-Modul:

/* Initialisieren des CMEM */
initCMEM(RemTime,Datei)

/* Speichern/Aktualisieren eines Clients in dem CMEM */
updateClient(CMEM,ClientID,NNr,Datei)

/* Abfrage welche Nachrichtennummer der Client als nächstes erhalten darf */
getClientNNr(CMEM,ClientID)

HBQ-Prozess:

/* Initialisieren der HBQ */
HBQ ! {self(), {request,initHBQ}}
receive {reply, ok}

/ Speichern einer Nachricht in der HBQ */*
HBQ ! {self(), {request,pushHBQ,[NNr,Msg,TSclientout]}}
receive {reply, ok}

/ Abfrage einer Nachricht */*
HBQ ! {self(), {request,deliverMSG,NNr,ToClient}}
receive {reply, SendNNr}

/ Terminierung der HBQ */*
HBQ ! {self(), {request,dellHBQ}}
receive {reply, ok}

HBQ-Prozess/DLQ-Modul:

/* Initialisieren der DLQ */
initDLQ(Size,Datei)

/* Abfrage welche Nachrichtennummer in der DLQ gespeichert werden kann */
expectedNr(Queue)

/* Speichern einer Nachricht in der DLQ */
push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei)

/* Ausliefern einer Nachricht an einen Leser-Client */
deliverMSG(MSGNr,ClientPID,Queue,Datei)