**Team**: 6, Mert Siginc, Michael Müller

**Aufgabenaufteilung**:

1. Client.erl und Server.erl
2. Hbq.erl, Dlq.erl und CMEM.elr

**Quellenangaben**:

* <http://erlang.org/doc/apps/stdlib/index.html>
* <http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html>

**Bearbeitungszeitraum**: <Datum und Dauer der Bearbeitung an der Aufgabe von allen Teammitgliedern>

**Aktueller Stand**:

Entwurf fertig, einzelne Teile der Komponenten fertig, einzelne Tests erstellt und „laufen“.

**Änderungen des Entwurfs**: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. des Vorentwurfs vorgenommen worden.>

**Entwurf**:

**Client:**

**Beschreibung:**

Der Client ist dafür da, die angegebene Anzahl an Clients zu starten und diese in den Rollen Redakteur und Leser hin und her zu wechseln bis dessen Lebenszeit verstreicht. In dieser Zeit tauscht der Client mit dem Server Nachrichten aus. Vor jeder Nachricht die der Client verschickt, fragt es vorher nach der nächsten Nachrichtenummer und wartet nach einer verschickten Nachricht eine gewisse Zeit. Hat der Client 5 Nachrichten verschickt, fragt er noch einmal nach einer Nachrichtenummer, vergisst aber diese Nachricht zu senden. Hiernach wechselt der Client zum Leser und fragt den Server, ob es Nachrichten zu lesen gibt. Dies macht er solange bis alle Nachrichten auf dem Server gelesen wurden und der Client wieder in die Redakteur Rolle übergeht.

**Server:**

**Beschreibung:**

Der Server agiert als Schnittstelle für die Clients und verwaltet die eingehenden Nachrichten. Der Server arbeitet mit der HBQ und der CMEM zusammen, um die Anfragen vom Client zu bewerkstelligen. Der Server hat zudem die Aufgabe, eindeutige Nachrichtennummern zu verteilen. Der Server terminiert sobald der letzte Kontakt mit einem Client länger ist als die Wartezeit.

**Schnittstellen:**

**Abfragen der eindeutigen Nachrichtennummer:**   
Server ! {self(),getmsgid}  
receive {nid, Number}  
Diese Schnittstelle gibt die nächste Nachrichtennummer zurück. Diese braucht der Server, um die Nachrichten in der richtigen Reihenfolge verwalten zu können.

**Senden einer Nachricht:**Server ! {dropmessage,[INNr,Msg,TSclientout]}Sobald ein eine Nachricht beim Server ankommt, wird diese zur Speicherung an die HBQ weitergeleitet.

**Abfragen einer Nachricht:**Server ! {self(), getmessages}  
*receive {reply,[NNr,Msg,TSclientout,TShbqin,TSdlqin,TSdlqout],Terminated}* }Sobald der Client Lesen möchte, ruft der Server aus der CMEM die nächste Nachrichtennummer für diesen Client ab und beauftragt die HBQ, diese Nachricht an den Client zu verschicken. Ist keine neue Nachricht vorhanden, wird eine leere Dummy-Nachricht an den Client geschickt. Hiernach wird die CMEM aktualisiert, so dass bei der nächsten Abfrage auf die nächste Nachricht verwiesen wird.

**HBQ:**

**Beschreibung:**

Die HBQ wird zur Verwaltung der einkommenden Nachrichten der Clients genutzt.  
Nachrichten werden entweder in der HBQ oder der DLQ gespeichert. Will der Client lesen, so wird die DLQ von der HBQ beauftragt eine Nachricht an den Client zu schicken.  
Nur der HBQ greift auf die HBQ zu.

**Schnittstellen:**

**Initialisieren der HBQ:**   
HBQ ! {self(), {request,initHBQ}}  
receive {reply, ok}   
Beim Starten des Servers wird die HBQ über diese Schnittstelle initialisiert und steht dem Server hiernach zu Verfügung.

**Terminierung der HBQ:**HBQ ! {self(), {request,dellHBQ}}  
receive {reply, ok} Sobald der Server terminiert, wird diese Schnittstelle vom Server angesprochen um die HBQ gleichen falls zu terminieren.

**Speichern einer Nachricht:**HBQ ! {self(), {request,pushHBQ,[NNr,Msg,TSclientout]}}  
receive {reply, ok} Sobald eine Nachricht vom Client über den Server erhalten wurde, wird geprüft, ob die Nachrichtennummer der eingegangenen Nachricht, mit der erwarteten Nummer der DLQ übereinstimmt. Ist dies der Fall, wird die Nachricht in der DLQ gespeichert, falls nicht wird die Nachricht in der HBQ gespeichert. Sind in der HBQ mehr Nachrichten gespeichert als 2/3 der maximalen Anzahl der DLQ, so wird die Lücke geschlossen und eine Fehlernachricht in der DLQ eingetragen, welche die größte fehlende Nachrichtennummer beinhaltet.

**Abfrage einer Nachricht:**HBQ ! {self(), {request,deliverMSG,NNr,ToClient}}  
receive {reply, SendNNr}Die HBQ leitet die Anfrage an die DLQ weiter, wo diese behandelt und beantwortet wird. Als Rückmeldung erhält die HBQ die gesendete Nachrichtennummer, die die HBQ wiederum an den Server weitergeleitet wird.

**CMEM:**

**Beschreibung:**

Die CMEM wird als Speicher genutzt, um die Clients zu verwalten.  
Es merkt sich für jeden Client, welche Nachricht bereits an welchen Client geschickt wurde. Nur der Server greift auf die CMEM zu.

**Funktionen:**

**delCMEM(CMEM):** Sobald der Server terminiert, wird diese Funktion vom Server aufgerufen um die CMEM gleichen falls zu terminieren.

**initCMEM(RemTime,Datei):**Beim Starten des Servers wird die CMEM initialisiert und steht dem Server hiernach zu Verfügung.

**updateClient(CMEM,ClientID,NNr,Datei):** Sobald ein Client aus dem Server liest, wird die CMEM für diesen Client, zur letzten empfangenen Nachrichtennummer aktualisiert, so dass der Client bei der nächsten Abfrage die nächste Nachricht erhält.

**getClientNNr(CMEM,ClientID):** Sobald der Client vom Server lesen möchte, benötigt der Server Informationen darüber, welche Nachrichten der Client bisher erhalten hat. Durch das Abfragen der Nachrichtennummer aus der CMEM, erhält er diese Information. Ist der Client unbekannt wird eine 1 zurückgegeben. Die CMEM erinnert sich nur für eine gewisse Zeit an Clients, hiernach wird der Client vergessen und ist für die CMEM wieder unbekannt.

**DLQ:**

**Beschreibung:**

Die DLQ wird zur Speicherung der Nachrichten genutzt, die von den Clients gelesen werden können. Sobald ein Client vom Server lesen möchte, wird die DLQ vom Server über die HBQ beauftragt, die nächste Nachricht an den Client zu übermitteln. Die DLQ darf nur von der HBQ angesprochen werden.

**Funktionen:**

**delDLQ(Queue):** Sobald die HBQ terminiert, wird diese Funktion von der HBQ aufgerufen um die DLQ gleichen falls zu terminieren.

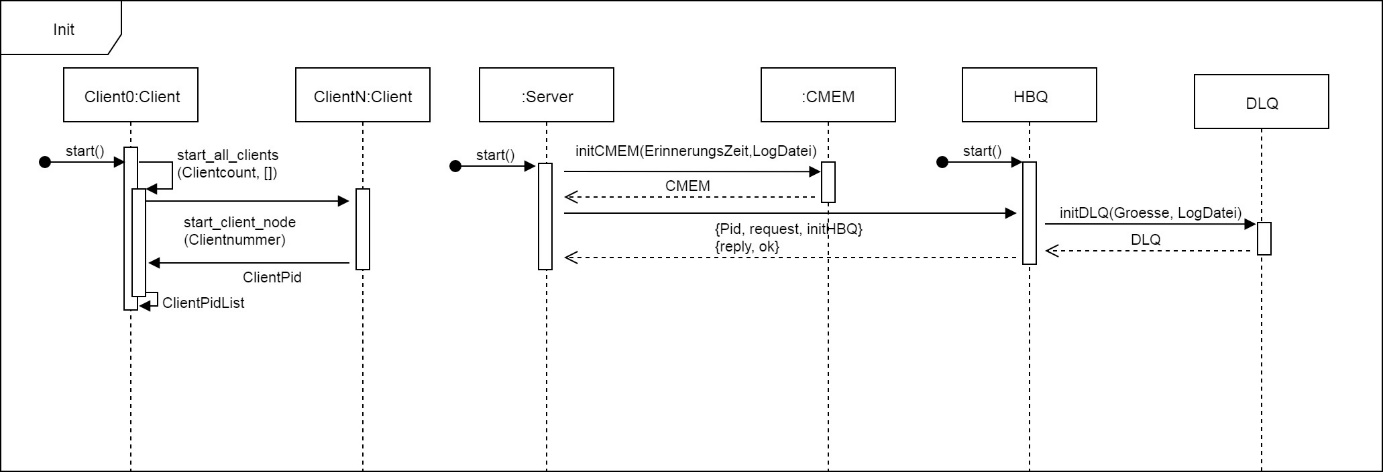
**initDLQ(Size,Datei):** Beim initialisieren der HBQ wird die DLQ initialisiert und steht der HBQ hiernach zu Verfügung.

**push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei):**Die von der HBQ übergebene Nachricht wird in die DLQ gespeichert. Wenn die DLQ voll ist, wird die letzte Nachricht verworfen.

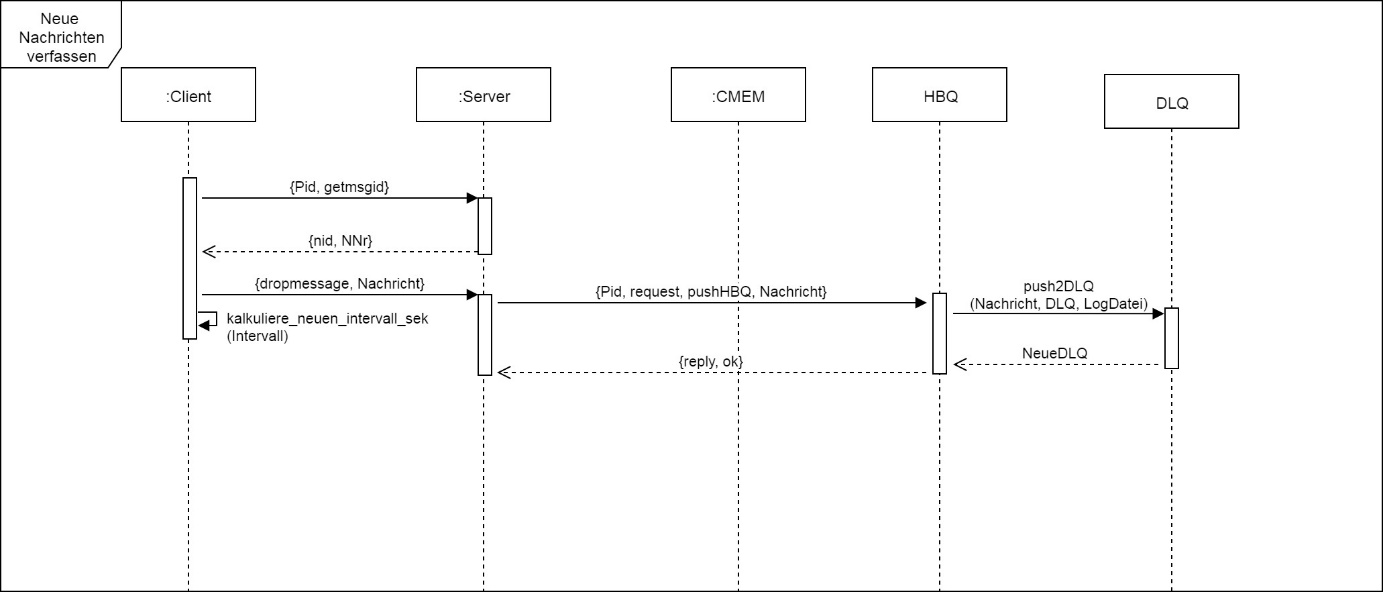
**deliverMSG(MSGNr,ClientPID,Queue,Datei):**Diese Funktion übermittelt eine Nachricht an den Client. Ist die Nachricht mit der übergebenen Nachrichtennummer vorhanden, wird diese vom DLQ an den Client verschickt. Falls die Nummer nicht vorhanden ist, wird die nächst größere in der DLQ vorhandene Nachricht gesendet.

**expectedNr(Queue):** Fragt die DLQ welche Nachricht als nächstes in der DLQ gespeichert werden kann. Ist die DLQ leer gibt dies 1 zurück.

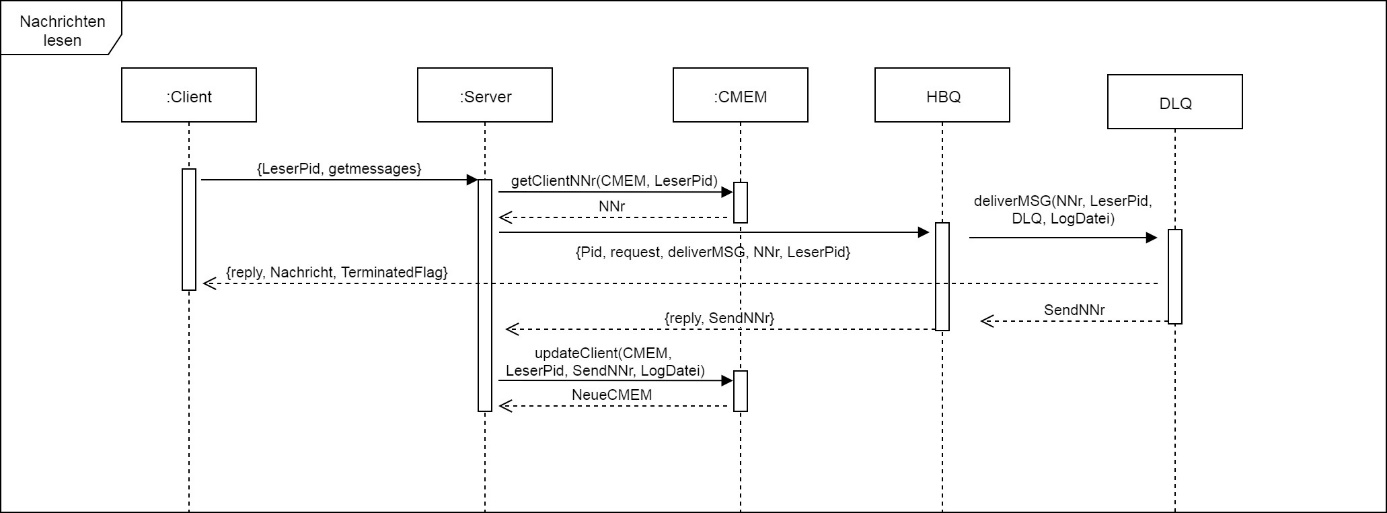
**UML Sequenzdiagramme**



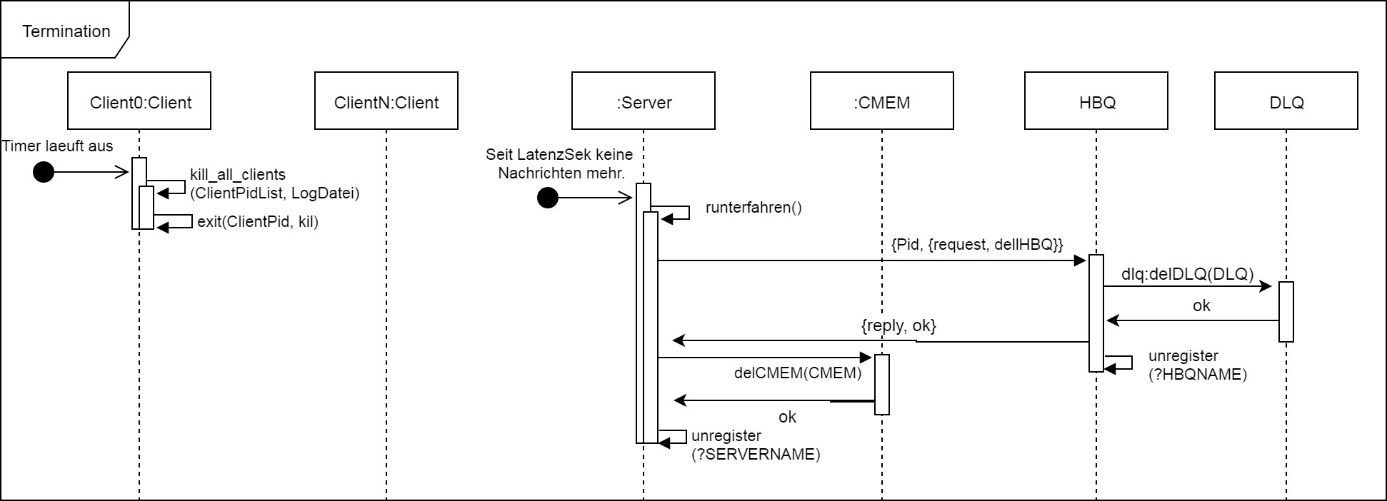
Sequenzdiagramm zur Initierungphase



Sequenzdiagramm zum Redakteur / für das erstellen und erfassen neuer Nachrichten im System



Sequenzdiagramm zum Leser / für das Lesen der vom System erfassten Nachrichten



Sequenzdiagramm zur Termination des Systems