Team: TEAM 01, Falco Winkler (FW), Daniel Schruhl (DS)

Aufgabenteilung:

- DLQ (DS)
- CMEM (DS)
- HBQ (FW)
- Server (DS)

Quellenangaben:

Bearbeitungszeitraum:

- 26.03.2017 (FW,DS)
- 03.04.2017 (DS)
- 04.04.2017 (DS)
- 05.04.2017 (FW,DS)

Aktueller Stand:

- DLQ fertig und getestet
- CMEM fertig und getestet
- HBQ angefangen
- Server fertig
- Client

Änderung des Entwurfs:

- Formatierung angepasst
- Komponentendiagramm erweitert
- Detailbeschreibungen für jedes Modul und Paket

Inhaltsverzeichnis

1 Einführung und Ziele **3**
- 1.1 Randbedingungen . . . 3
- 1.2 Kontextbegrenzung . . . 3

2 Gesamtsystem **4**
- 2.1 Bausteinsicht . . . 4
- 2.2 Laufzeitsicht . . . 5

3 Subsysteme und Komponenten **6**
- 3.1 Server-Modul . . . 6
  - 3.1.1 Aufgabe und Verantwortung . . . 6
  - 3.1.2 Schnittstelle . . . 6
  - 3.1.3 Entwurfsentscheidungen . . . 7
  - 3.1.4 Konfigurationsparameter . . . 7
- 3.2 CMEM-Modul . . . 8
  - 3.2.1 Aufgabe und Verantwortung . . . 8
  - 3.2.2 Schnittstelle . . . 8
  - 3.2.3 Entwurfsentscheidungen . . . 8
  - 3.2.4 Konfigurationsparameter . . . 9
- 3.3 Komponente HBQ . . . 10
  - 3.3.1 Aufgabe und Verantwortung der HBQ . . . 10
  - 3.3.2 Entwurfsentscheidungen . . . 10
  - 3.3.3 Außensicht . . . 10
  - 3.3.4 Innensicht . . . 10
  - 3.3.5 Konfigurationsparameter . . . 10
- 3.4 DLQ-Modul . . . 11
  - 3.4.1 Aufgabe und Verantwortung . . . 11
  - 3.4.2 Schnittstelle . . . 11
  - 3.4.3 Entwurfsentscheidungen . . . 12
  - 3.4.4 Konfigurationsparameter . . . 12

1 Einführung und Ziele

Es soll eine Message of the Day Anwendung erstellt werden. Dabei werden von verschiedenen Clients an einen Server verschiedene Nachrichten des Tages gesendet. Die Clients rufen vom Server alle Nachrichten ab, so dass jeder Client alle Nachrichten in einer festen Reihenfolge hat.

1.1 Randbedingungen

Es soll eine Client/Server-Architektur implementiert werden. Der Server verwaltet dabei die ihm von den Clients gesendeten Nachrichten. Das beinhaltet eine feste Numerierung der Nachrichten.
Die Clients rufen dabei in bestimmten Abständen die Nachrichten ab. Falls ein Client dem Server schon bekannt ist, bekommt der nur die ihm noch unbekannten (neuen) Nachrichten.
Der Server muss sich also die Clients merken. Es soll mit einer Holdbackqueue und einer Deliveryqueue gearbeitet werden, um die korrekte Auslieferung in einer bestimmten Reihenfolge der Nachrichten zu garantieren.

1.2 Kontextbegrenzung

Das System soll in Erlang umgesetzt werden. Es muss auf Computern mit Linux Betriebssystem lauffähig sein.

2 Gesamtsystem

2.1 Bausteinsicht

Das Softwareprodukt besteht aus mehreren Modulen und Paketen (Abbildung 1). Diese Pakete setzen sich zusammen aus dem Client-Paket und dem Server-Paket.

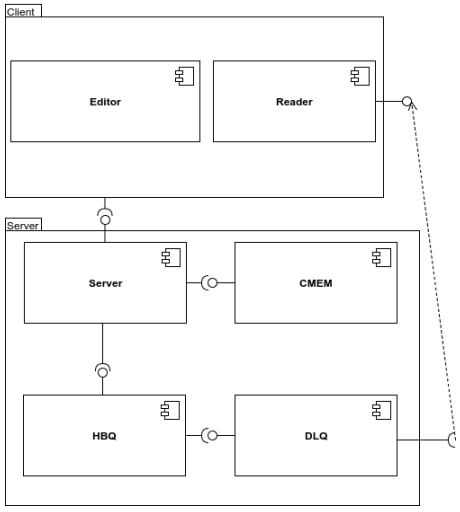

Abbildung 1: Komponentendiagramm der Message Of The Day App

Das Server Paket beinhaltet das Server-Modul und alle vom Server-Modul verwendeten Datenstrukturen.
Das Server-Modul ist für alle Funktionalitäten des Servers zuständig. Dazu gehört das Nummerieren und Verwalten der Nachrichten und die Verwaltung der Clients.
Das Server-Modul benutzt daher per Schnittstelle das HBQ-Modul und das CMEM-Modul. Das Server-Modul stellt eine Schnittstelle für die Clients bereit.

Das CMEM-Modul ist für die Speicherung der Clients und ihrer aktuellen zu erwartenden Nachrichten Nummer zuständig. Das CMEM-Modul soll als lokale ADT realisiert werden. Diese wird nur vom Server angesprochen.

Das HBQ-Modul ist für die Holdbackqueue zuständig und regelt die Sortierung der einkommenden Nachrichten in die Deliveryqueue und der damit verbundenen Fehlerbehandlung. Dabei ist das HBQ-Modul als entfernte ADT realisiert. Diese wird nur von dem Server-Modul verwendet.

Das DLQ-Modul realsiert die Deliveryqueue, die für die Auslieferung der Nachrichten in Reihenfolge an die Clients zuständig ist. Die Schnittstelle des DLQ-Moduls wird nur vom HBQ-Modul konsumiert.

Das Client-Paket beinhaltet die Client-Module. Diese sind zum einen der Lese Client (Reader-Modul) und der Redakteur Client (Editor-Modul). Beide Clients sind als ein Prozess implementiert und verwenden die vom Server bereitgestellte Schnittstelle. Der Redakteur Client ist für das Schicken von Nachrichten zuständig und der Lese Client für das Lesen von Nachrichten.

2.2 Laufzeitsicht

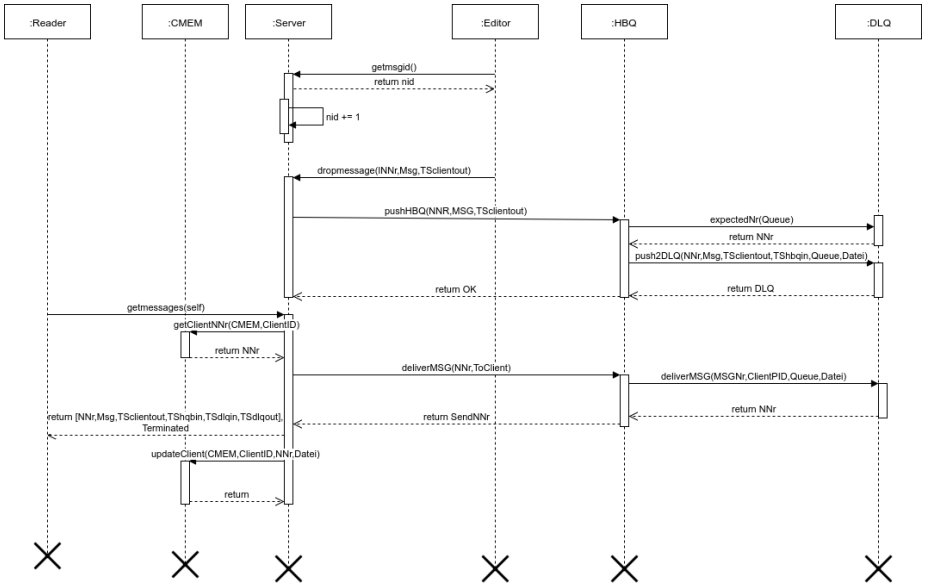

Abbildung 2: Sequenzdiagramm bei fehlerfreiem Nachrichtenaustausch

3 Subsysteme und Komponenten

3.1 Server-Modul

3.1.1 Aufgabe und Verantwortung

Der Server hat die Aufgabe Nachrichten von Clients entgegen zu nehmen, diese zu verarbeiten und an seine Subprozesse bzw. an seine an ihn verbundenen Module weiter zu schicken. Der Server ist außerdem die Zentrale Stelle für die Koordinierung der nächst höchsten freien Nachrichtennummer, die von den Clients zum Senden verwendet werden soll.
Wenn der Server für eine bestimmte Zeit von keinem Client mehr angesprochen wird, soll er sich und seine Subprozesse herunterfahren.

3.1.2 Schnittstelle

```
/* 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}

/* Nur fuer interne Prozesse: Einleiten des Herunterfahrens */
Server ! terminate

/* Nur fuer interne Prozesse: Terminierungsprozess erfolgreich, fahre runter*/
Server ! {reply,ok}
```

getmessages: Fragt beim Server eine aktuelle Textzeile ab. self() stellt die Rückrufadresse des Leser-Clients dar. Als Rückgabewert erhält er eine für ihn aktuelle Textzeile (Zeichenkette) zugestellt (Msg) und deren eindeutige Nummer (NNr). Zudem erhält er die Zeitstempel explizit (erstellt durch erlang:now(), TSclientout, TShbqin, TSdlqin, TSdlqout). 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. weitere Aufrufe von getmessages sind nicht notwendig.

dropmessage: Sendet dem Server eine Textzeile (Msg), die den Namen des aufrufenden Clients und seine aktuelle Systemzeit sowie ggf. irgendeinen Text beinhaltet, zudem die zugeordnete (globale) Nummer der Textzeile (INNr) und seine Sendezeit (erstellt mit erlang:now(), TSclientout).

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

terminate: Nur von interne Prozesse zu verwenden. Startet den Terminierungsprozess.

reply, ok: Nur von interne Prozesse zu verwenden. Signalisiert, dass alle Submodule und Prozesse des Servers heruntergefahren wurden und der Server nun selber herunterfahren kann.

3.1.3 Entwurfsentscheidungen

Der Server bearbeitet Anfragen per message passing. Die NNr ist ein Zählerwert, der nach jedem Herausgeben an einen Client inkrementiert wird.

3.1.4 Konfigurationsparameter

1. Latenzzeit bestimmt die maximale Zeit die der Server ungenutzt läuft
2. Lebenszeit des Clients bestimmt die Zeit, für die sich der Server einen Client merkt
3. Name definiert den Namen des Servers
4. Größe der Deliveryqueue

3.2 CMEM-Modul

3.2.1 Aufgabe und Verantwortung

Dieses Modul hat die Aufgabe sich anfragende Clients, ihre letzte bekommene Nachrichtennummer und den letzten Zeitpunkt ihrer Anfrage zu merken. Clients, die sich seit einiger Zeit (Konfigurationsparameter) nicht gemeldet haben, sollen aus dem CMEM-Modul gelöscht werden können.
Die gespeicherten Daten sollen über die Schnittstelle des Moduls vom Server aus abrufbar sein.

3.2.2 Schnittstelle

```
/* Initialisieren des CMEM */
initCMEM(RemTime, Datei): Integer X Atom -> CMem

/* Speichern/Aktualisieren eines Clients in dem CMEM */
updateClient(CMEM, ClientID, NNr, Datei): Cmem X PID X Integer X Atom -> CMem

/* Abfrage welche Nachrichtennummer der Client als naechstes erhalten darf */
getClientNNr(CMEM, ClientID) : Cmem X PID -> Integer
```

initCMEM(RemTime,Datei): initialisiert den CMEM. RemTime gibt dabei die Zeit an, nach der die Clients vergessen werden Bei Erfolg wird ein leeres CMEM zurück geliefert. Datei kann für ein logging genutzt werden.

updateClient(CMEM,ClientID,NNr,Datei): speichert bzw. aktualisiert im CMEM den Client ClientID und die an ihn gesendete Nachrichtenummer NNr. Datei kann für ein logging genutzt werden.

getClientNNr(CMEM,ClientID): gibt die als nächstes vom Client erwartete Nachrichtennummer des Clients ClientID aus CMEM zurück. Ist der Client unbekannt wird 1 zurück gegeben.

3.2.3 Entwurfsentscheidungen

Die CMEM wird mit Hilfe einer Liste realisiert. An erster Stelle der CMEM Liste steht die Liste der Clients. Die Elemente der Client Liste sind Tupel, die aus der Client Prozess ID, der zuletzt erhaltenen NNr und einem Timestamp der letzten Aktion in Millisekunden bestehen.
An zweiter Stelle der CMEM Liste steht die maximale Zeit, für die ein Client gemerkt wird in Millisekunden.

```
/* Client Tupel Format */
Client := {ClientPID, NNr, ClientTS}:
    {PID X Integer X Integer}

/* CMEM Format */
```

```
/* ClientList ist Liste bestehend aus Clients */
/* RemTime ist Zeit in Millisekunden fuer die Clients gemerkt werden */
CMEM := [ClientList, RemTime]: [List X Integer]
```

Falls beim Aktualisieren eines Clients der Client noch nicht im CMEM steht, wird er hinzugefügt. Ansonsten wird er einfach mit den angegebenen Parametern in der CMEM Client Liste aktualisiert.

Wenn die nächste Nachrichtennummer für einen Client abgerufen wird (`getClientNNr`), wird ein Check gemacht, ob der Client bekannt ist. Ein Client ist bekannt, wenn er in der CMEM Liste steht und wenn die Summe seines Timestamps mit der RemTime größer gleich der aktuellen Zeit (als Timestamp) ist.

Für bekannte Clients wird die resultierende nächste Nachrichtennummer inkrementiert. Für unbekannte Clients wird 1 als nächste Nachrichtennummer zurück gegeben.

3.2.4 Konfigurationsparameter

1. Zeit nach der die Clients vergessen werden in Millisekunden (RemTime)

3.3 Komponente HBQ

3.3.1 Aufgabe und Verantwortung der HBQ

Von Redakteur-Clients gesendete Nachrichten werden hier zwischengespeichert. Gibt es keine Lücke in der fortlaufenden Nummerierung der Nachrichten, werden Sie an die DLQ weitergeleitet. Bei einer Überfüllung der HBQ (2/3 der Maximalen Kapazität der DLQ) wird die älteste Lücke in den Nachrichten mit einer künstlichen Nachricht geschlossen, und es werden erneut alle Lückenlosen Nachrichten an die DLQ weitergeleitet. Die HBQ stellt nur ein Interface für den Server bereit.

3.3.2 Entwurfsentscheidungen

Es wird die Aktuelle NNr gespeichert. Jede Nachricht, die der aktuellen erwarteten NNr entspricht, wird direkt weitergeleitet. Wenn noch auf alte Nachrichten gewartet wird, wird die nachricht in die HBQ eingefügt. Wenn die Nachrichtennummer größer als der erwarteten Nachrichtennummer ist, müssen alle bereits empfangenennachrichten ab der erwarteten Nachrichtennummer durchgegangen werden. Die Nachrichten, die aufeinanderfolgende Sequenznummern haben oder eine Dummy Nachricht sind werden an die DLQ ausgeliefert.
Die Dummynachrichten schließen Lücken bestehend aus einer oder mehrerer Sequenznummern. Eine Dummynachricht wird durch den String Fehlernachrichtïm Inhalt identifiziert. Ihre NNr wird auf das Ende des Intervalls, welches Sie abdeckt festgelegt.
Die Liste wird nach jedem Einfügen Sortiert, um die Reihenfolge zu erhalten. Wenn die Liste überfüllt ist, wird rekursiv die erste Lücke zwischen Sequenznummern gesucht. Dann wird eine entsprechende Dummy - Nachricht für diese Lücke eingefügt, und die Funktion zum Senden aller aufeinanderfolgenden Sequenznummern wird erneut aufgerufen. So werden mindestens alle Nachrichten bis zum Abschluss der Lücke aus der HBQ entfernt.

3.3.3 Außensicht

3.3.4 Innensicht

3.3.5 Konfigurationsparameter

- Die Maximalgröße der DLQ wird verwendet um die HBQ bei kritischer Überfüllung zu leeren.

3.4 DLQ-Modul

3.4.1 Aufgabe und Verantwortung

Die Deliveryqueue hat die Aufgabe Nachrichten an Clients zuzustellen und stellt eine Datenstruktur dar, die eine maximale Menge (Kapazität) an Nachrichten hält. Sie verhält sich dabei wie von einer Warteschlange zu erwarten beim Einfügen und Herausholen von Nachrichten (FIFO). Nur das HBQ-Modul darf auf das DLQ-Modul zugreifen. Das DLQ-Modul sendet über eine Schnittstelle der Clients die Nachrichten an die Clients.

3.4.2 Schnittstelle

```
/* Initialisieren der DLQ */
initDLQ(Size,Datei): Integer X Atom -> DQueue

/* Abfrage welche Nachrichtennummer in der DLQ gespeichert werden kann */
expectedNr(Queue) : DQueue -> Integer

/* Speichern einer Nachricht in der DLQ */
push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei) :
    MSG_list X DQueue X Atom -> DQueue

/* Ausliefern einer Nachricht an einen Leser-Client */
deliverMSG(MSGNr,ClientPID,Queue,Datei):
    Integer X PID X DQueue X Atom -> Integer
```

initDLQ(Size,Datei): initialisiert die DLQ mit Kapazität Size. Bei Erfolg wird eine leere DLQ zurück geliefert. Datei kann für ein logging genutzt werden.

expectedNr(Queue): liefert die Nachrichtennummer, die als nächstes in der DLQ gespeichert werden kann. Bei leerer DLQ ist dies 1.

push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei): speichert die Nachricht [NNr,Msg,TSclientout,TShbqin] in der DLQ Queue und fügt ihr einen Eingangszeitstempel an (einmal an die Nachricht Msg und als expliziten Zeitstempel TSdlqin mit erlang:now() an die Liste an. Bei Erfolg wird die modifizierte DLQ zurück geliefert. Datei kann für ein logging genutzt werden.

deliverMSG(MSGNr,ClientPID,Queue,Datei): sendet die Nachricht MSGNr an den Leser-Client ClientPID. Dabei wird ein Ausgangszeitstempel TSdlqout mit erlang:now() an das Ende der Nachrichtenliste angefügt. Sollte die Nachrichtennummer nicht mehr vorhanden sein, wird die nächst größere in der DLQ vorhandene Nachricht gesendet. Bei Erfolg wird die tatsächlich gesendete Nachrichtennummer zurück geliefert. Datei kann für ein logging genutzt werden.

3.4.3 Entwurfsentscheidungen

Die Deliveryqueue wird als Liste realisiert. Das erste Element der Liste ist dabei eine Liste der Nachrichten und das zweite Element der Liste gibt die Kapazität der DLQ an.

```
/* Nachrichten Format */
/* minimal 3 Elemente, pro Station kommt eins hinzu; maximal 6 Elemente */
MSG_List := [NNr,Msg,TSclientout,TShbqin,TSdlqin]:
    [Integer X String X 3-Tupel X 3-Tupel X 3-Tupel]

/* DLQ Format */
/* Nachrichtenliste ist eine Liste aus MSG_List und hat */
/* maximal Kapazitaet Anzahl an Elementen */
DLQ := [Nachrichtenliste, Kapazitaet]: [List X Integer]
```

Neue Nachrichten werden am Anfang der Liste angefügt. Das vereinfacht die Nachfrage nach der nächsten zu speichernden Nachrichtennummer in der DLQ. Demnach werden Nachrichten am Ende der Liste entnommen. Wenn eine Nachricht in die DLQ eingetragen wird, wird ans Ende der Nachricht der aktuelle Zeitstempel zum Zeitpunkt des Einfügens hinzugefügt.
Nachrichten werden der DLQ entnommen, wenn die Größe der Liste gleich ihrer Kapazität ist und eine neue Nachricht hinzugefügt werden soll.

Beim Senden einer Nachricht durch Angabe der Nachrichtennummer wird in der DLQ nach der Nachricht mit der Nachrichtennummer gesucht. Wenn diese gefunden wird, wird die Nachricht an den Client (mit der ClientPID) gesendet und die Nachrichtennummer zurück gegeben.
Falls die Nachricht nicht gefunden werden konnte, wird die nächste Nachricht mit der nächst größeren Nachrichtennummer an den Client gesendet. Diese wird mit einer Hilfsfunktion ermittelt (via reduce/foldl).
Beim Senden einer Nachricht wird die Nachricht zusammen mit einem boolean Flag an den Client gesendet. Dieser boolean Flag signalisiert, ob noch weitere Nachrichten folgen.
Das wird mit einer weiteren Hilfsfunktion `remainingMessagesExist(MsgNr,DLQ))` ermittelt.
Diese Hilfsfunktion holt sich den Kopf der DLQ Nachrichtenliste (aktuellste Nachricht mit höchster Nachrichtennummer) und vergleicht, ob diese Nachrichtennummer größer ist als die übergebene `MsgNr`.

3.4.4 Konfigurationsparameter

- Kapazität der DLQ als ganze Zahl