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

Aufgabenteilung:

Quellenangaben:

Bearbeitungszeitraum:

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

Aktueller Stand:

Änderung des Entwurfs:

- Komponentendiagramm erweitert

Inhaltsverzeichnis

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 Kontextsicht

2.2 Bausteinsicht

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

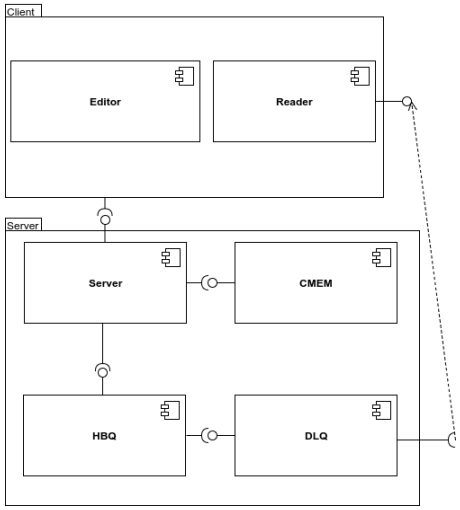

Abbildung 1: Komponentendiagramm der Message Of The Day App

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.

2.3 Laufzeitsicht

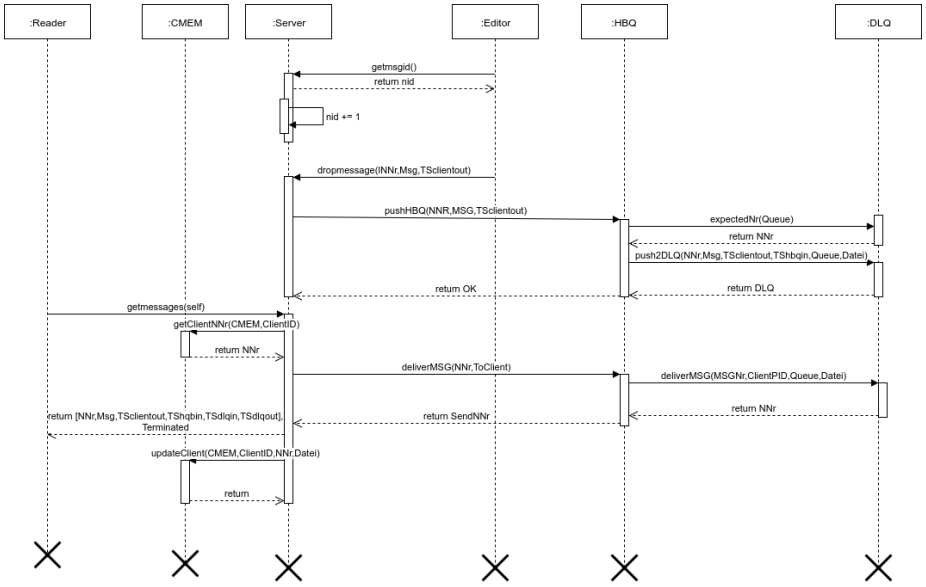

Abbildung 2: Sequenzdiagramm bei fehlerfreiem Nachrichtenaustausch

2.4 Verteilungssicht

2.5 Konfigurationsparameter

2.6 Benutzungsschnittstelle

3 Subsysteme und Komponenten

3.1 Client

3.1.1 Aufgabe und Verantwortung des Clients

Der Client ist ein einzelner Prozess, der sich jedoch in zwei Rollen aufteilt. Im Endlosdurchlauf einer Schleife wechselt er zwischen Leser und Redakteur - Rolle.

Der Client wird in drei Module aufgespalten: Client, Reader und Writer. Client ruft in einer Endlosschleife wechselweise die soeben beschriebene Funktionalität in den Modulen Reader und Writer auf.

3.1.2 Entwurfsentscheidungen

3.1.3 Außensicht

3.1.4 Innensicht

3.1.5 Konfigurationsparameter

3.1.6 Komponente Leserclient

Aufgabe und Verantwortung des Leseclients Der Leseclient liest solange Nachrichten vom Server, bis keine weiteren Nachrichten mehr vorhanden sind.

Entwurfsentscheidungen

Außensicht

Innensicht

Konfigurationsparameter

3.1.7 Komponente Redakteurclient

Aufgabe und Verantwortung des Redakteurclients Der Redakteurclient sendet fünf Nachrichten an den Server, und fragt danach einmal die nächste NNr ab, um eine Lücke in den fortlaufend nummerierten Nachrichten zu produzieren.

Entwurfsentscheidungen

Außensicht

Innensicht

Konfigurationsparameter

3.2 Server

3.2.1 Aufgabe und Verantwortung des Servers

Der Server hat die Aufgaben

1. Sich anfragende Clients mit ProzessID und Zeitstempel zu merken
2. Clients die ihnen zugehörigen Nachrichten zurückzugeben

Für die erste Aufgabe wird das Modul CMEM verwendet. Zum Senden an die Clients leitet der Server die Nachrichten zunächst an die HBQ weiter. Erscheint eine Anfrage vom Client, erfolgt über die HBQ der Aufruf an die DLQ, die Nachricht zu übermitteln.

3.2.2 Entwurfsentscheidungen

3.2.3 Außensicht

3.2.4 Innensicht

3.2.5 Konfigurationsparameter

3.2.6 Benutzungsschnittstelle

3.2.7 Komponente CMEM

Aufgabe und Verantwortung der CMEM Dieses Modul hat die Aufgabe sich anfragende Clients mittels einer ProzessID und eines Zeitstempels in einer Map-Datenstruktur zu merken, und diese für den Server bereitzustellen.

Entwurfsentscheidungen

Außensicht

Innensicht

Konfigurationsparameter

3.2.8 Komponente HBQ

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.

Entwurfsentscheidungen

Außensicht

Innensicht

Konfigurationsparameter

3.3 DLQ-Modul

3.3.1 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.

3.3.2 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
```

3.3.3 Entwurfsentscheidungen

Die Deliveryqueue wird als Erlang 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,TSdlqout]:
    [Integer X String X 3-Tupel X 3-Tupel X 3-Tupel X 3-Tupel]

/* DLQ Format */
/* Nachrichtenliste ist eine Liste aus Nachrichten 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.
Das kann passieren, wenn die Größe der Liste gleich ihrer Kapazität ist und eine Neue Nachricht hinzugefügt werden soll.

Konfigurationsparameter

- Kapazität der DLQ