Team: 15, Adrian Helberg

Aufgabenaufteilung: 1er Team, keine Aufteilung

Quellenangaben:

- Aufgabenstellung: http://users.informatik.haw-hamburg.de/~klauck/VerteilteSysteme/aufg1.html
- Entity-Relationship-Modell und Sequenzdiagramm erstellt mit „Draw.io“ : https://www.draw.io/
- Erstellen eines Entwurfs: http://users.informatik.haw-hamburg.de/~klauck/VerteilteSysteme/Entwurf.pdf

Bearbeitungszeitraum: 05.10.2018 – 11.10.2018

Aktueller Stand: Überarbeiteter Entwurf

Änderungen des Entwurfs: Version 2

Entwurf: Ab Seite 2

Inhalt

Rahmenbedingungen

Zu implementieren sind die Server-Komponenten CMEM, HBQ und DLQ

Architektur

Die Aufgabe wird in einer Client-Server-Architektur in einem verteilten System umgesetzt. Dabei bietet der Server Dienste an, die der Client auf Wunsch anfordern kann.
Die Server-Komponente ist in folgende Sub-Komponenten unterteilt:

- HBQ
  - Hold-Back-Queue
  - Hält Nachrichten, die nicht ausgeliefert werden
  - Wird als lokaler Erlang Node ausgeführt
  - Läuft als Prozess, der mittels erlang:spawn/2 gestartet wird
- DLQ
  - Deliveryqueue
  - Hält Nachrichten, die an den Leser ausgeliefert werden können
  - Wird als globaler Erlang Node ausgeführt
  - Läuft als Prozess, der mittels erlang:spawn/2 gestartet wird
- CMEM
  - Client Memory
  - Gedächtnis für die Leser
  - Wird als lokaler Erlang Node ausgeführt
  - Läuft als Prozess, der mittels erlang:spawn/2 gestartet wird

Rahmenwerke und Bibliotheken

Entwickelt wird in der Entwicklungsumgebung IntelliJ IDEA (Version 2018.2.4) mit dem Plugin

- Erlang
  - von Sergey Ignatov
  - http://ignatov.github.io/intellij-erlang/
  - code completion
  - syntax and error highlighting
  - code inspections
  - code navigation

Um den Erlang-eigenen Observer, Debugger und ein erweitertes Kommandozeilenprogramm nutzen zu können, werden die Nodes mit dem Schlüsselwort werl gestartet.

Schnittstellen

Die einzelnen Komponenten werden mittels erlang:register/2 entweder im lokalen oder globalen Namensraum registriert. Die Kommunikation im verteilten System wird von der Erlang-Umgebung übernommen. Server-Komponenten und deren Dienste werden über den Namen, der beim Erstellen der Node gewählt wird, und dem Funktionsnamen angesprochen.

Aufrufen einer Funktion „test()“ der Komponente „hbq“ der Node „hbq@host“: *hbq:test().*

Modularisierung

Paketstruktur

Jeder Node wird in einer eigenen Datei „<Node>.erl“ realisiert

Beschreibung

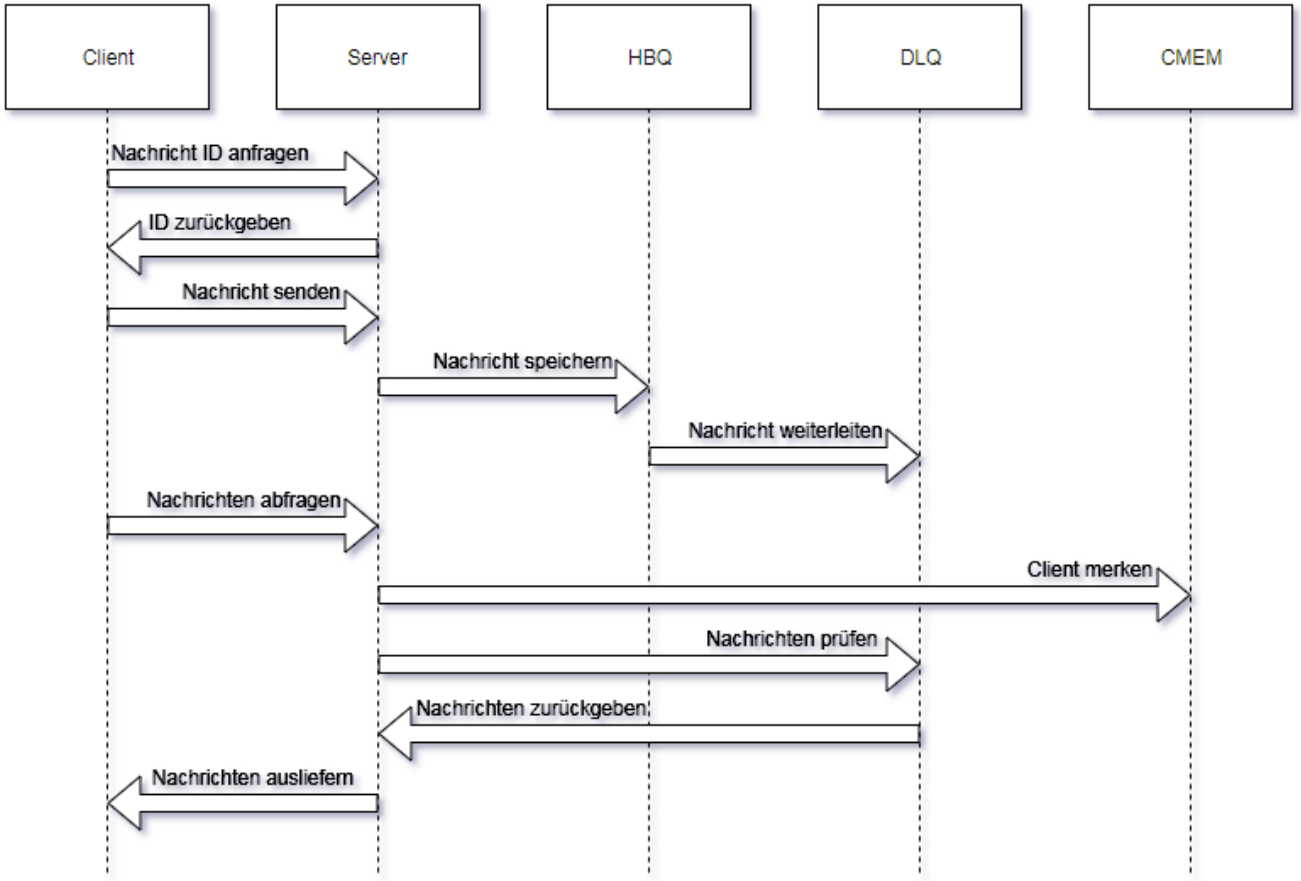
Die in Klammern stehenden Namen sind die Funktionsnamen der Dienste

- HBQ (hbq.erl)
    - Initialisierung (initHBQ)
        - Prozess spawnen und Prozess-ID zurückgeben
    - Terminierung (dellHBQ)
        - Prozess terminieren (exit)
    - Speichern einer Nachricht (pushHBQ)
        - Schreiben der Nachricht in interne Liste
    - Abfrage einer Nachricht (deliverMSG)
        - Zurückgeben einer Nachricht über die ID
    - Logging (listDLQ, listHBQ)
        - Schreiben des aktuellen Stands in eine Logging-Datei
- DLQ (dlq.erl)
    - Initialisierung (initDLQ)
        - Prozess spawnen und Prozess-ID zurückgeben
    - Terminierung (delDLQ)
        - Prozess terminieren (exit)
    - Speichern einer Nachricht (push2DLQ)
        - Schreiben der Nachricht in interne Liste
    - Abfrage welche Nachrichtennummer in der DLQ gespeichert werden kann (expectedNr)
        - Zurückgeben einer ID
    - Ausliefern einer Nachricht an einen Leser-Client (deliverMSG)
        - Zurückgeben einer Nachricht
    - Abfrage einer Liste aller Nachrichtennummern (listDLQ)
        - Gibt eine Liste der Nachrichtennummern zurück
    - Abfragen der Größe der DLQ (lengthDLQ)
        - Gibt die Größe der DLQ zurück

- CMEM (cmem.erl)
    - Initialisierung (initCMEM)
        - Prozess spawnen und Prozess-ID zurückgeben
    - Terminierung (delCMEM)
        - Prozess terminieren (exit)
    - Speichern/Aktualisieren eines Clients (updateClient)
        - Gibt den gespeicherten/aktualisierten Client zurück (PID)
    - Abfrage welche Nachrichtennummer der Client als nächstes erhalten darf (getClientNNr)
        - Gibt die ID zurück
    - Abfrage aller Leser (listCMEM)
        - gibt eine Liste der Leser-PIDs zurück
    - Abfragen der Größe des CMEM (lengthCMEM)
        - Gibt die Größe des CMEM zurück

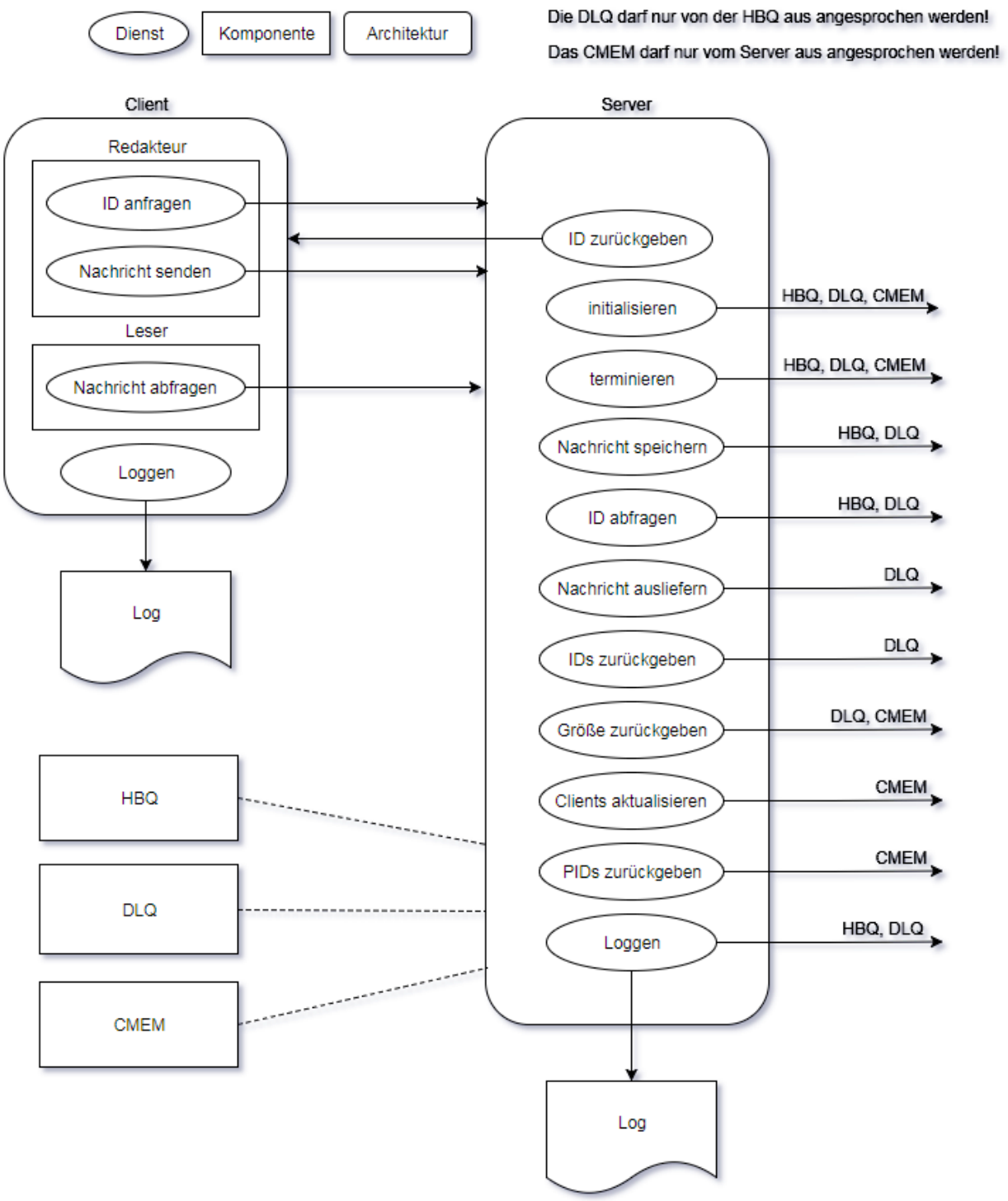
Kommunikation

Sequenzdiagramm



Datenstrukturen

Entity-Relationship-Modell



Schnittstellenbeschreibungen

Dateischnittstellen

Ausgaben werden in Log-Dateien geschrieben

- HBQ: HB-DLQ<Node>.log

API-Schnittstellen

Um auf die Funktionen der Module zugreifen zu können, müssen diese exportiert werden

- z.B. „-export([fact/1]).“ , um eine Funktion fact, die einen Parameter entgegennimmt, zu exportieren

Weitere funktionale Anforderungen

- Server
  - Basis-Strukturen Liste (lists) und Tupel (tuple) werden verwendet
  - Nachrichtenformat
    - MSG_List := [NNr,Msg,TSclientout,TShbqin,TSdlqin,TSdlqout]:
      - [Integer X String X 3-Tupel X 3-Tupel X 3-Tupel X 3-Tupel]
  - DLQ halt maximal ?Xdlq viele Nachrichten
  - Übertragungszeiten werden mit **erlang:timestamp()** getrackt
  - Sind keine Nachrichten beim Server vorhanden, wird eine Dummy-Nachricht versendet
  - Leser ohne Anfragen werden nach ?Xleser Sekunden beim Server abgemeldet
  - Besteht zwischen HBQ und DLQ zu 2/3 der Nachrichten eine Inkonsistenz, wird diese mit genau einer Fehlernachricht geschlossen
  - Nach einer gewissen Wartezeit ohne Anfragen, terminiert der Server
  - Die ADTs (HBQ, DLQ, CMEM) müssen austauschbar sein
  - Die HBQ wird als globaler ADT implementiert, DLQ und CMEM als lokaler
  - Konfigurationen wie Variablen (z.B. ?Xleser) werden in einer server.cfg – Datei angegeben
  - Ausgaben werden in Dateien Server<Node>.log und HB-DLQ<Node>.log geschrieben

User Stories

- Epics
  - Lesern fragen die „Nachrichten des Tages“ ab
  - Redakteure verfassen die „Nachrichten des Tages“
  - Server verwaltet Anfragen für die „Nachrichten des Tages“