Entwurf VS - Praktikum 1

Team:
- Praktikumsgruppe 2 - Team 2
- Lukas Lühr & Florian Stäps

Aufgabenaufteilung:
1. Entwurf gemeinsam erarbeitet
2. Unabhängig von einander umgesetzt und anschließend zusammengeführt

Quellenangaben:
- Folien Klauck,
- Klauck HAW - Seite - http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html

Bearbeitungszeitraum: 30 Stunden bisher

Aktueller Stand:
- Alles fertig

Änderungen des Entwurfs: Bisher keine Änderungen

Entwurf

1. Sequenzdiagramm
    a. Siehe unten
2. Services
    a. Message Handler
        i. Verwaltet für den Server die eigehenden Nachrichten und leitet diese an die Services weiter
    b. Number Service
        i. Ist für die Zuteilung von Nachrichten Nummern zuständig
        ii. Leitet diese an den Client direkt weiter
    c. CMEM
        i. Verwaltet die Clientliste
    d. Controller
        i. Verarbeitet die „getmessages“ - Nachrichten des Message Handlers
        ii. Interagiert mit der CMEM und hält diese aktuell
        iii. Kommuniziert mit der HBQ zur Nachrichtenauslieferung
    e. Trash Can
        i. Dient als Entsorgungsstelle für nicht wichtige Antworten
    f. HBQ
        i. Dient als Zwischenspeicher für out of order Nachrichten
        ii. Leitet die Nachrichten, die in der richtigen Reihenfolge ankommen, an die DLQ weiter
3. Passive Komponenten
    a. DLQ
        i. Speichert alle zu sendenden Nachrichten
            1. Es wird vorher definierte Menge an Nachrichten gespeichert
                a. Wird diese überschritten, werden alte Nachrichten verworfen

Details zur Implementierung

1. HBQ
    a. Von der Struktur ähnelt die HBQ der DLQ stark
    b. Ist nach Nachrichtennummer sortiert
    c. Hat eine empfangene Nachricht eine geringe Nachrichten Nummer als die höchste der DLQ, so wird die empfangene Nachricht verworfen
    d. Sollte die empfangene Nachricht direkt in die DLQ übertragen werden können, so geschieht dies
        i. Zudem wird überprüft, ob der Inhalt der HBQ bis zur nächsten Lücke an die DLQ übertragen werden kann
        ii. Ansonsten wird die Nachricht in die HBG einsortiert und geprüft, ob die Lücke übersprungen werden kann
2. DLQ
    a. Interne Daten Struktur
        i. Die DLQ ist ein Tupel aus Größe und einer Liste:
        { Size , List}
        ii. Die Liste ist eine Ansammlung von Eintrags - Tupel:

        [Eintrag1, Eintrag2, . . ., EintragN]

iii. Ein Eintrag hat die Struktur:

{Nummer, Nachricht}

iv. Die Struktur einer Nachricht ist in den Anforderungen definiert.

b. Umsetzung der einzelnen Funktionen

i. Initialisierung der DLQ

1. Es wird die Struktur der DLQ wie definiert mit einer leeren Liste deklariert

ii. Löschen der DLQ

1. Da es sich um eine rein passive Struktur handelt, sind keinerlei Aufräummaßnahmen notwendig

iii. Ermitteln der erwarteten Nachrichtennummer

1. Es muss lediglich die Nummer des ersten Tupels der List zurückgegeben werden
2. Handelt es sich um eine leere Liste, so ist eins zurück zu geben

iv. Ablegen von Nachrichten

1. Aus der Nachricht wird ein Eintrags Tupel generiert, dieses wird vorne in die Liste eingetragen. Anschließend wird die Liste auf Länge gekürzt falls notwendig
2. Ausliefern einer Nachricht an den Client
    a. Es wird geprüft, ob die angefragte Nachricht in der Liste enthalten ist, falls ja wird die gefundene Nachricht an den Client ausgeliefert. Ansonsten die letzte Nachricht
    b. Die Nummer der ausgelieferten Nachricht wird zurückgegeben
    c. Die gesamte Liste muss beim Suchen, durchlaufen werden, da auch Lücken vorhanden sein können

3. CMEM

a. Die CMEM ist als eigener Prozess geplant, welcher intern eine Datenstruktur hält und die Schnittstellen kommunizieren mittels Nachrichtenaustausch mit der Datenstruktur

b. delCMEM

i. Dient dazu die CMEM zu beenden, es wird ein Kill request An diese Gesendet.

ii. Der Status Wird zurückgesendet, und anschließend von der Schnittstelle zurückgegeben

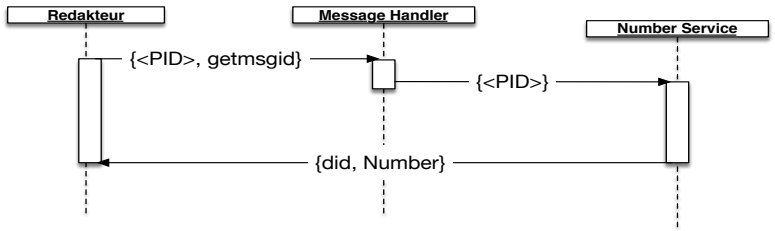
1. send : {request,kill,self())
2. Response : {ok}

c. updateClient(CMEM,ClientID,NNr,Datei)
    i. Speichert bzw. aktualisiert im CMEM den Client (ClientID) und die an ihn gesendete Nachrichtenummer (NNr)
    ii. Liefert ein ok
        1. send {request,update,ClientID,NNr,self(),Datei},
        2. Response {ok}

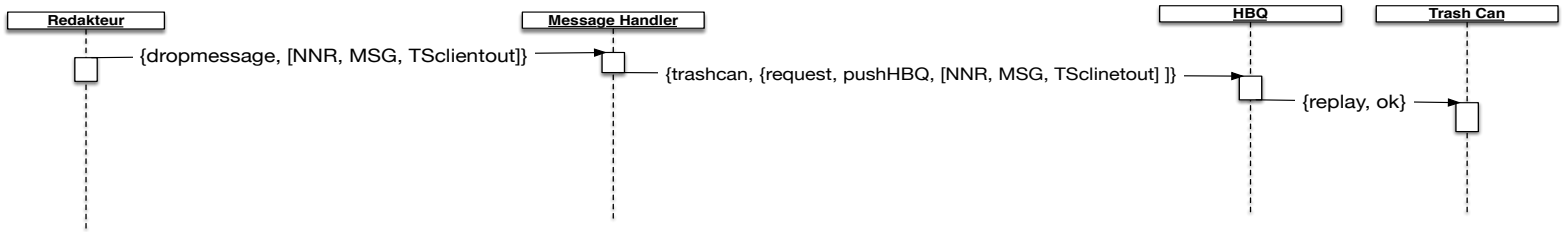
d. getClientNNr(CMEM,ClientID)
    i. Gibt die als nächstes vom Client erwartete Nachrichtennummer des Clients ClientID aus CMEM zurück
    ii. Ist der Client unbekannt wird 1 zurückgegeben
        1. send {request,get,ClientID,self()}
        2. Response {N}
e. Internes des CMEM
    i. Intern verwendet die CMEM eine Liste von Tupel mit der Struktur
    ii. Tupel -> {ClientID,UpdateTime,Value}
    iii. Es werden inten lediglich 2 Funktionen benötigt:
        1. findCNum
            a. Um die Daten zu einem Client zu finden
        2. updateCNum
            a. Um die Daten zu einem Client zu aktualisieren

Sequenzdiagramme

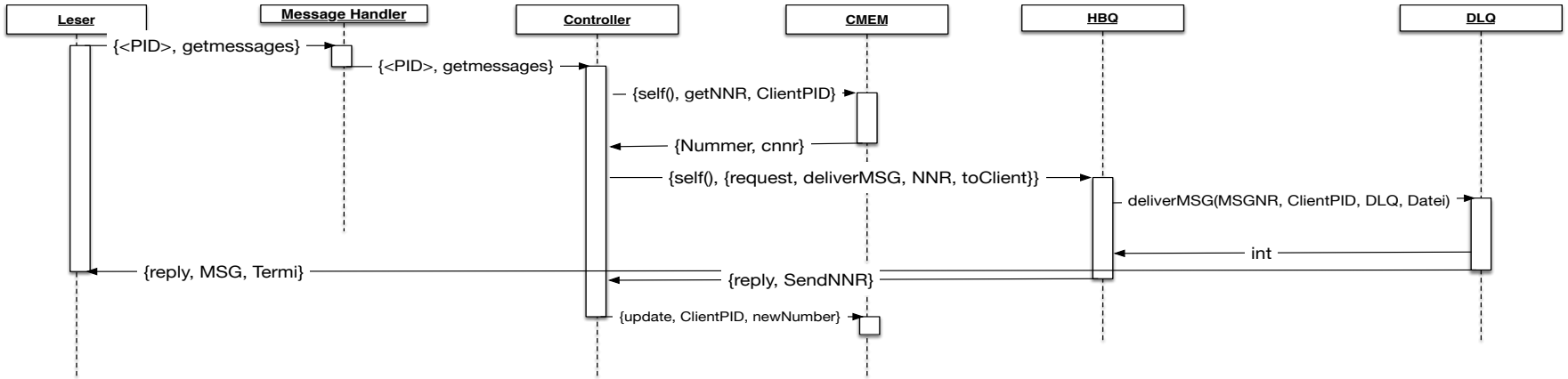
Anwendungsfall 1



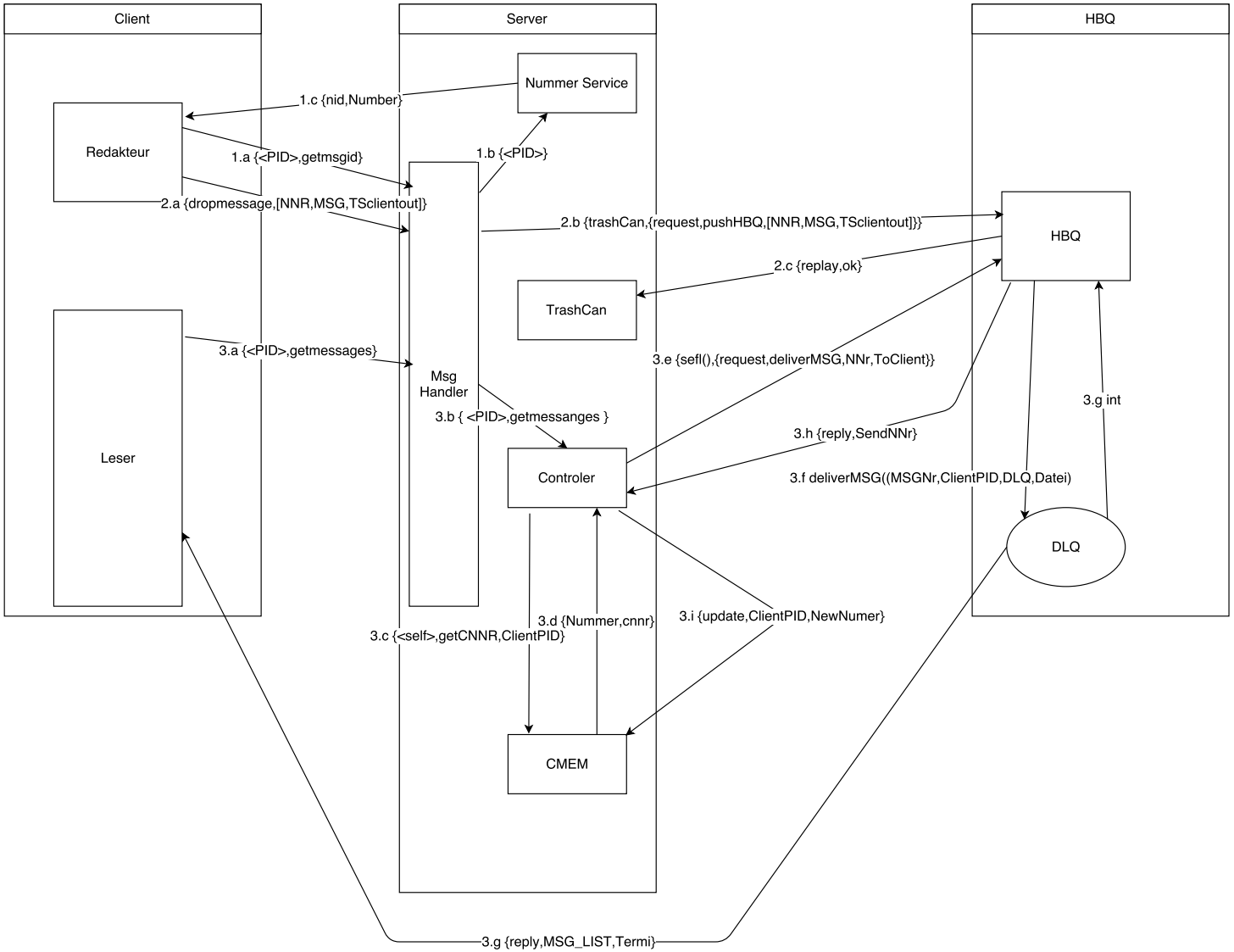
Anwendungsfall 2



Anwendungsfall 3



Übersicht



Anmerkung zu 2.
Die HBQ ist nach Nachrichten nummern sortiert.

Hat die Nachricht eine geringer nummer als die nachricht mit der höchsten nummer aus der DLQ, so wird die neue nachricht verworfen.

Sollte die empfangene Nachricht direkt in die DLQ übertragen werden können, so geschieht dies, und es wird geprüft ob die HBQ angehängt werden kann.

ansonsten wird die nachricht sortiert in die HBQ eingefügt, und geprüft ob die vorhandene Lücke übersprungen werden kann.