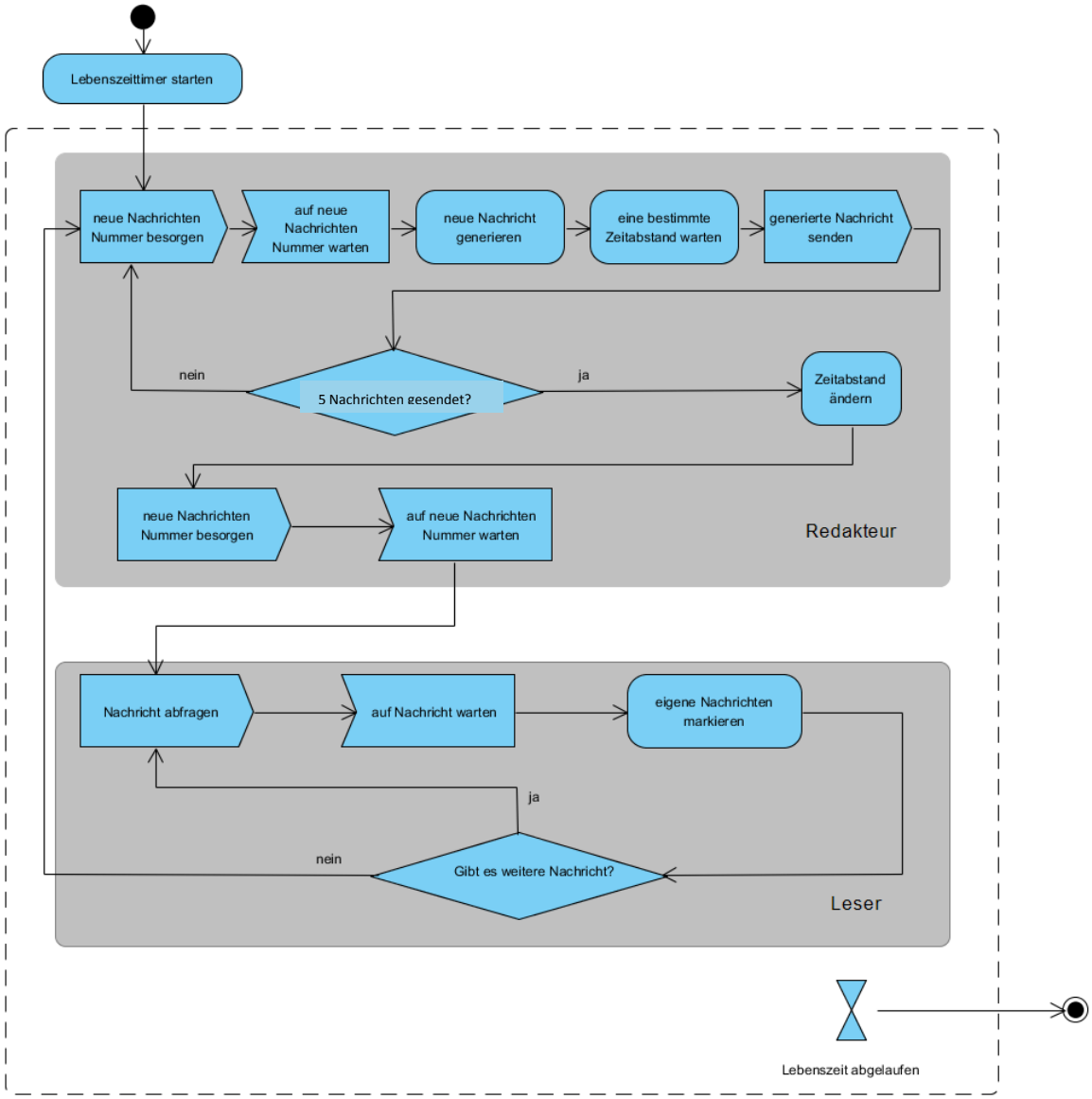
Team: 12, Diana Topko, Maxim Rjabenko
Aufgabenaufteilung: Gemeinsamer Entwurf
Quellenangaben: Folien aus der Vorlesung
Begründung für Codeübernahme: keine Codeübernahme
Bearbeitungszeitraum: 26.03.2015 – 29.03.2015 etwa 15 Stunden
Aktueller Stand: Entwurfsskizze
Änderungen im Entwurf:
Entwurf:

Der Client in der Grobbeschreibung:
sendet Nachrichten an den Server und fragt in bestimmten Abständen die aktuellen Nachrichten ab.

- Der Client hat 2 logische Rollen: Redakteur- und Lese-Client. Lese-Client und Redakteur-Client kennen sich nicht und werden sequentiell ausgeführt. Der Client startet als Redakteur-Client.
- Der Programmablauf kann daher in einem einzelnen Prozess erfolgen. Dieser Prozess beendet sich nach Ablauf der in der Konfiguration angegebenen Lebenszeit.

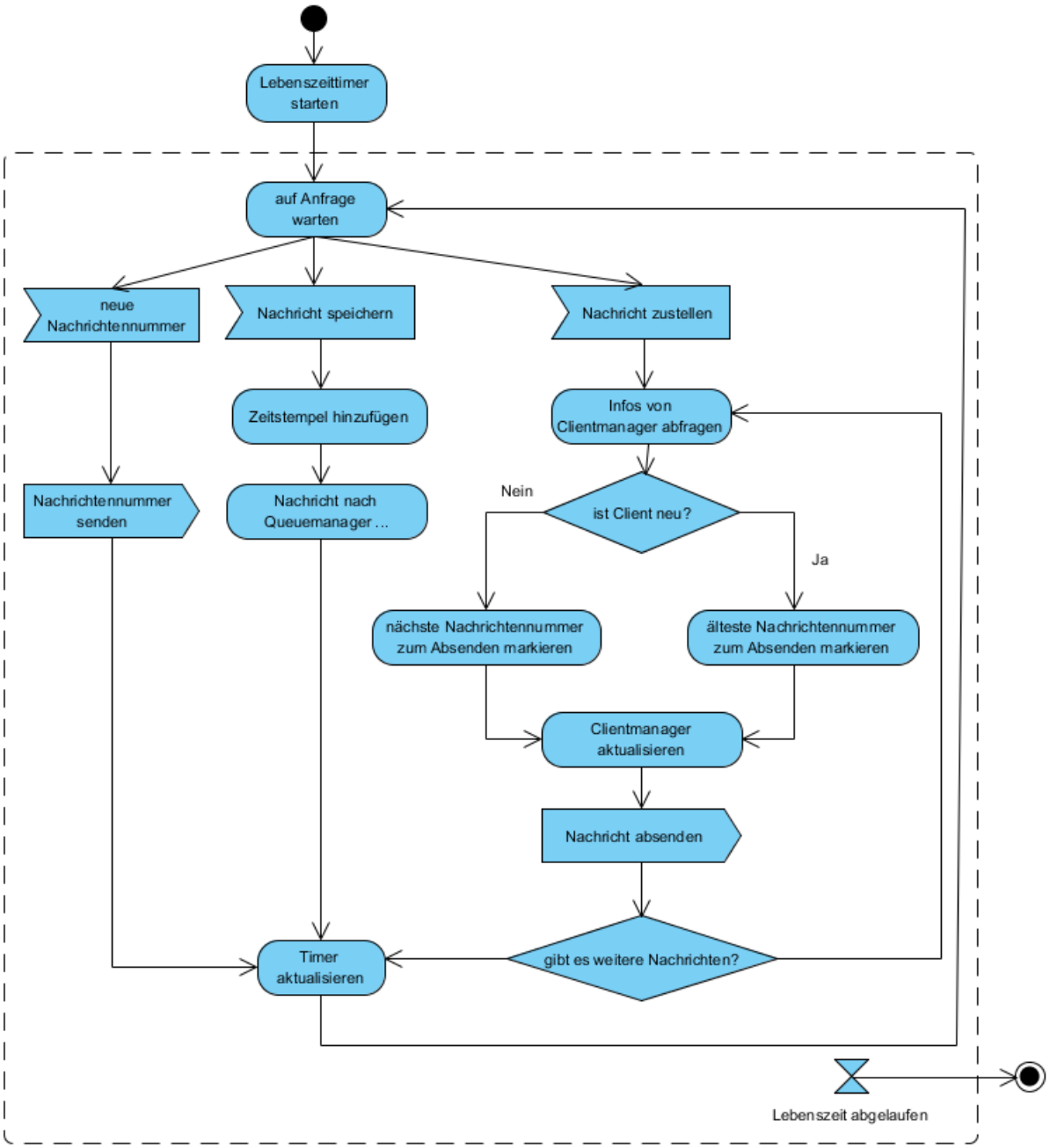


Der Server in der Grobbeschreibung:
verwaltet Nachrichten, die ihm von unterschiedlichen Clients zugesendet werden und liefert diese in richtiger Reihenfolge den Clients zurück.

Die Serverkomponente:
Der Server wird durch zwei Prozesse realisiert.

1. Empfang von Nachrichten, Zwischenspeicherung von Nachrichten in einer eigenen Datenstruktur und Zurücklieferung von Nachrichten in richtiger Reihenfolge.
2. Verwaltung eine Liste von registrierten Clients mit zugehörigen Informationen (ClientPid, letzte gesendete Nachrichtennummer, letzte Zugriffszeit) – das CMEM.

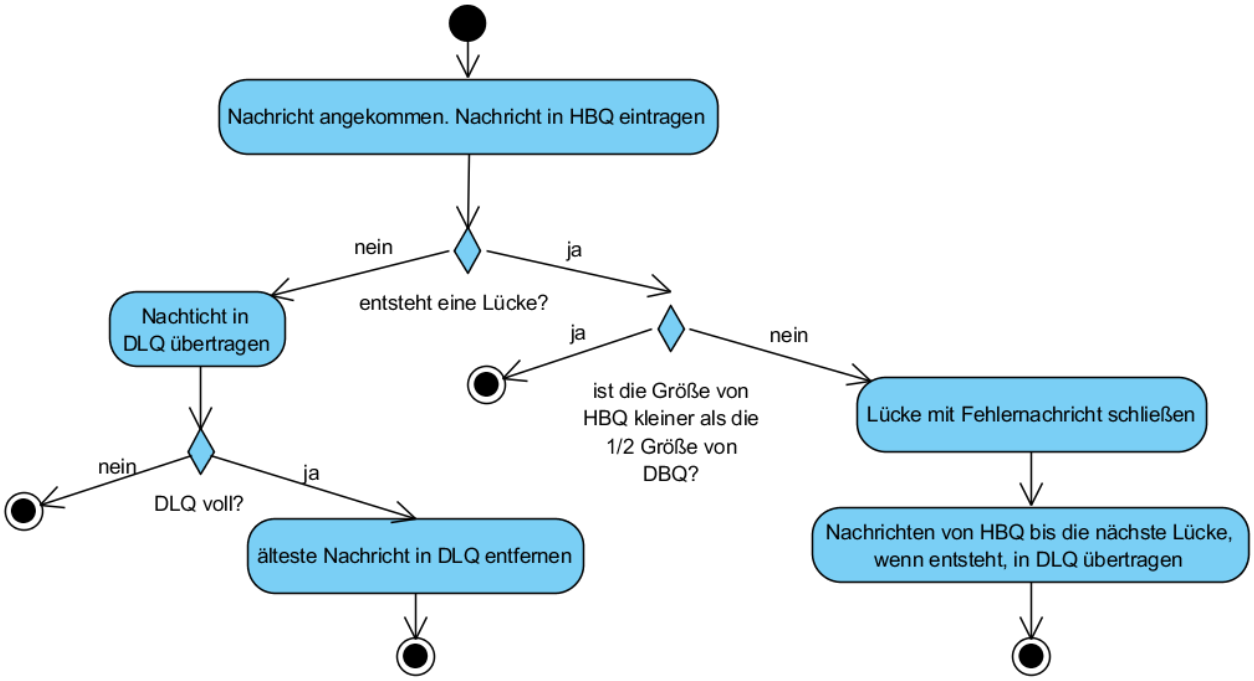
Die Prozesse beenden sich, wenn in einer vorgegebenen Zeit keine Nachrichtenabfrage erfolgt.



ADT‘s in der Grobbeschreibung:

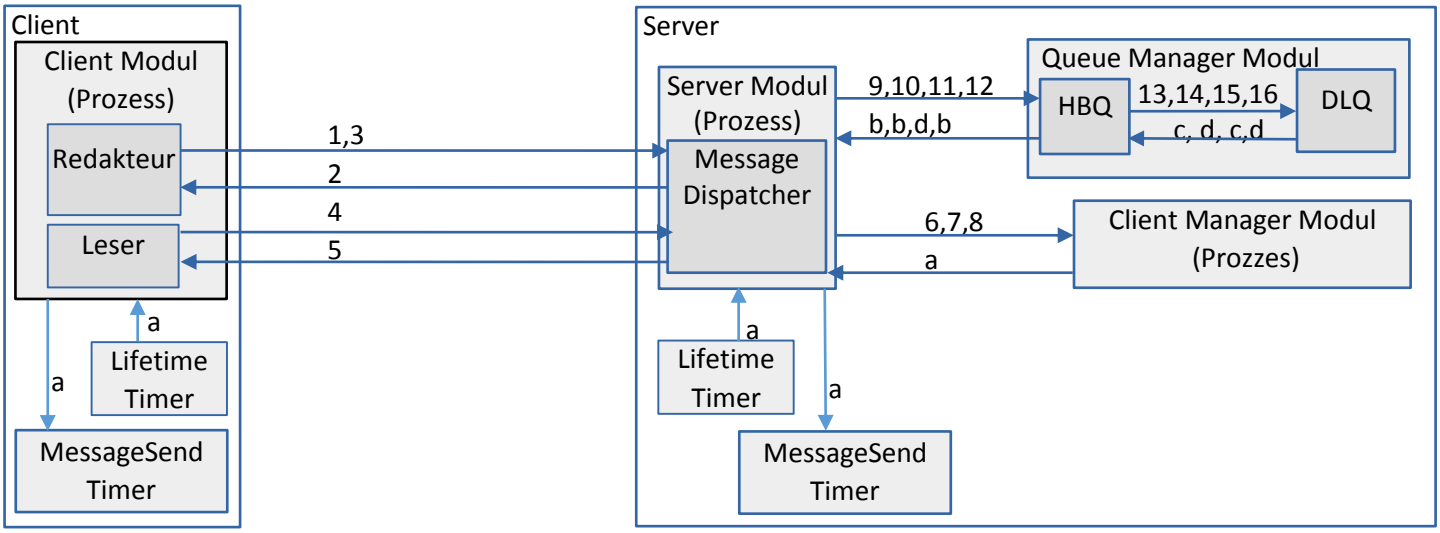
- **CMEM** verwaltet eine Liste von registrierten Clients mit zugehörigen Informationen (ClientPid, letzte gesendete Nachrichtennummer, letzte Zugriffszeit). Meldet sich ein Lese-Client eine gewisse Zeit nicht, dann vergisst der Server diesen Lese-Client. Jeder unbekannte Leser-Client bekommt alle Nachrichten. Jeder bekannte Lese-Client bekommt nicht vorher an ihn gelieferte Nachrichten. Bei einer erneuten Abfrage (nach dem Vergessen) wird er wie ein unbekannter Lese-Client behandelt.
- **Die Holdbackqueue (HBQ)** verwaltet die Nachrichten, die angekommen sind und überträgt diese in die Deliveryqueue. HBQ achtet auf die Reihenfolge der zu sendenden Nachrichten und schließt Lücken, wenn solche entstanden sind. HBQ wird durch eine Liste von Erlang repräsentiert. HBQ enthält keine Duplikate.
- **Deliveryqueue (DLQ)** verwaltet die Nachrichten, die an Clients ausgeliefert werden können. DLQ enthält keine Duplikate. DLQ hat eine Kapazität. Die Größe der DLQ ist vorgegeben.

HBQ-DLQ-Algorithmus



Beträgt die Nachrichtenmenge in der Holdbackqueue mehr als die Hälfte der vorgegebene Deliveryqueuegroße, dann entsteht eine **Lücke**. Diese Lücke wird mit genau einer Fehlernachricht geschlossen. Es werden keine weiteren Lücken innerhalb der Holdbackqueue gefüllt und die wird weiter bearbeitet

Kommunikationsdiagramm und Schnitstelle



a. <<create>>
b. {reply, ok}
c. {queue, Queue}
d. {reply, SendNNr}

Schnittstelle des Servers:

- Abfragen der eindeutigen Nachrichtennummer
  Fragt beim Server die aktuelle Nachrichtenummer ab. self() stellt die Rückrufadresse des Leser-Clients dar. Als Rückgabewert erhält er die aktuelle und eindeutige Nachrichtennummer (Number).
  1. {self(),getmsgid}
  2. receive {nid, Number}

- Senden einer Nachricht
  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)
  3. {dropmessage,[INNr,Msg,TSclientout]}

- Abfragen einer Nachricht
  Fragt beim Server eine aktuelle Textzeile ab. self() stellt die Rückrufadresse des Leser-Clients dar. Als Rückgabewert erhält er eine für ihn aktuelle Textzeile 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.
  4. {self(), getmessages}
  5. receive {reply,[NNr,Msg,TSclientout,TShbqin,TSdlqin,TSdlqout],Terminated}

Schnittstelle des CMEM: Das CMEM darf nur vom Server aus angesprochen werden!

- Initialisieren des 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.
  6. initCMEM(RemTime,Datei)

- Speichern/Aktualisieren eines Clients in dem CMEM speichert bzw. aktualisiert im CMEM den Client ClientID und die an ihn gesendete Nachrichtenummer NNr. Datei kann für ein logging genutzt werden.
  7. updateClient(CMEM,ClientID,NNr,Datei)

- Abfrage welche Nachrichtennummer der Client als nächstes erhalten darf gibt die als nächstes vom Client erwartete Nachrichtennummer des Clients ClientID aus CMEM zurück. Ist der Client unbekannt wird 1 zurück gegeben.
  8. getClientNNr(CMEM,ClientID)

Schnittstelle der HBQ:

- Initialisieren der HBQ
  initialisiert die HBQ und die DLQ. Bei Erfolg wird ein ok gesendet.
  9. {self(), {request,initHBQ}}

- Speichern einer Nachricht in der HBQ
  fügt eine Nachricht Msg (Textzeile) mit Nummer NNr und dem Sende-Zeitstempel TSclientout (mit erlang:now() erstellt) in die HBQ ein. Bei Erfolg wird ein ok gesendet.
  10. {self(), {request,pushHBQ,[NNr,Msg,TSclientout]}}

- Abfrage einer Nachricht
  beauftragt die HBQ über die DLQ die Nachricht mit der Nummer NNr (falls nicht verfügbar die nächst höhere Nachrichtennummer) an den Client ToClient (als PID) auszuliefern. Bei Erfolg wird die tatsächlich gesendete Nachrichtennummer SendNNr gesendet.
  11. {self(), {request,deliverMSG,NNr,ToClient}}

- Terminierung der HBQ
  Bei Erfolg wird ein ok gesendet.
  12. {self(), {request,dellHBQ}}

Schnittstelle der DLQ: Die DLQ darf nur von der HBQ aus angesprochen werden!

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

- Abfrage welche Nachrichtennummer in der DLQ gespeichert werden kann

liefert die Nachrichtennummer, die als nächstes in der DLQ gespeichert werden kann. Bei leerer DLQ ist dies 1.

14. expectedNr(Queue)

- Speichern einer Nachricht in der DLQ
  Die Nachricht [NNr,Msg,TSclientout,TShbqin] wird in der DLQ Queue gespeichert und einen Eingangszeitstempel wird ihr angefügt (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.

  15. push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei)

- Ausliefern einer Nachricht an einen Leser-Client
  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.

  16. deliverMSG(MSGNr,ClientPID,Queue,Datei)