



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Referat

von Igor Arkhipov, BAI-5

Koordination von Teilaktivitäten im verteilten
System

Igor Arkhipov

„Koordination von Teilaktivitäten im verteilten
System“

Referat eingereicht im Rahmen der Vorlesung Verteilte Systeme

im Studiengang Angewandte Informatik (AI)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. C. Klauck

Abgegeben am 16.11.2016

Inhaltsverzeichnis

1	Einleitung	5
2	Komponentenübersicht	6
3	Phasenübersicht	7
3.1	Initialisierungsphase.....	7
3.2	Arbeitsphase	8
3.3	Beendigungsphase	8
4	Komponentendetails	9
4.1	Manuelles Steuerungsmodul	9
4.1.1	Methoden	9
4.2	Starter.....	10
4.2.1	Methoden	11
4.3	Koordinator	11
4.3.1	Methoden	12
4.4	Externe Komponenten	12
5	Schnittstellen.....	13
5.1	Der Zustand „initial“	13
5.1.1	Steuerungsmodul.....	13
5.1.2	Starter	13
5.2	Der Zustand „bereit“	14
5.2.1	Steuerungsmodul.....	14
5.2.2	Starter	15

5.2.3	Koordinator	15
5.3	Der Zustand „beenden“	16
5.3.1	Steuerungsmodul	16
5.3.2	Koordinator	16
6	Appendix	17
6.1	Sequenzdiagramme.....	17
6.1.1	Initialisierungsphase (Voraussetzung: Module sind gestartet)	17
6.1.2	Arbeitsphase	17
6.1.3	Beendigungsphase	18
6.2	Ablauf des Programms (Bearbeitungsschritte)	18
7	Referenzen	22

1 Einleitung

Im Rahmen dieses Referats wird ein Ansatz für die Implementation des Koordinators und zugehörigen Komponenten vorgestellt, in dem ein verteilter Algorithmus wird davon gesteuert. Die gesamte Anwendung enthält noch ein Startermodul und ein Steuerungsmodul für manuelle Kontrolle über verteilten Prozesse und basiert auf Programmiersprache Erlang.

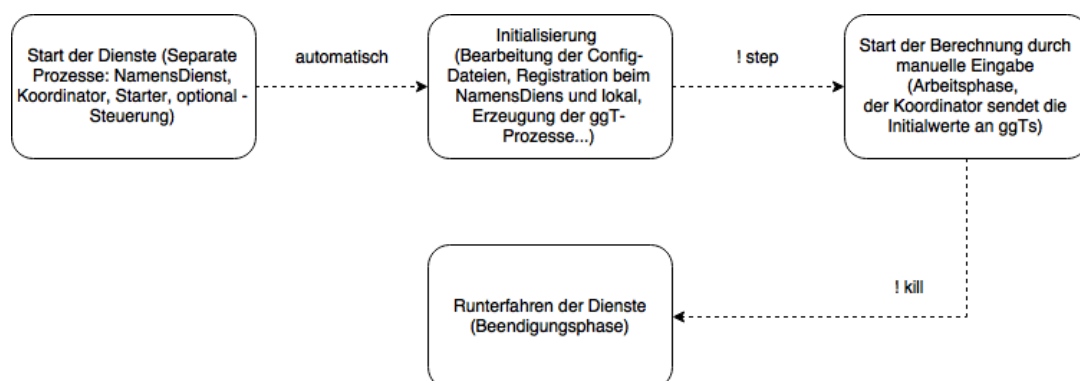
Der Algorithmus stellt die nebenläufige Berechnung des ggT (des größten gemeinsamen Teilers) von mehreren Zahlen (entsprechend der Anzahl der Arbeitsprozesse) dar und wird als Thema des anderen Referats in diesem Dokument nur oberflächlich referenziert.

Der Koordinator ordnet die verteilten ggT-Prozesse in einem Kreis an, in dem sie jeweils nur ihre Nachbarn kennen und sich mit ihnen über Ergebnisse der Berechnungen austauschen können.

Das System soll eine längere Zeit für mehrere Berechnungen zur Verfügung stehen. Die Terminierung passiert nur dann, wenn der zugehörige Befehl von dem Steuerungssystem ankommt.

Die Kommunikation zwischen den Komponenten (Prozessen) läuft über einen externen Namensdienst. Die Anzahl der ggT-Prozesse kann beliebig sein.

Der Ablauf des gesamten Prozesses für die ggT-Berechnung sieht wie folgt aus:



2 Komponentenübersicht

Der externe Namensdienst ordnet den PIDs die registrierten Prozessnamen zu.

Der Starter-Prozess ist für Starten der ggT-Prozesse zuständig.

Der Koordinator bietet eine Schnittstelle zwischen dem System und einem manuellen Steuerungsmodul an.

Die ggT-Prozessen verwirklichen die eigentliche Berechnung des ggT.

Das Steuerungsmodul vermittelt die Nachrichten zwischen User und Koordinator.

Die Kommunikation zwischen den Komponenten wird mit Message-Passing von Erlang/OTP implementiert.

3 Phasenübersicht

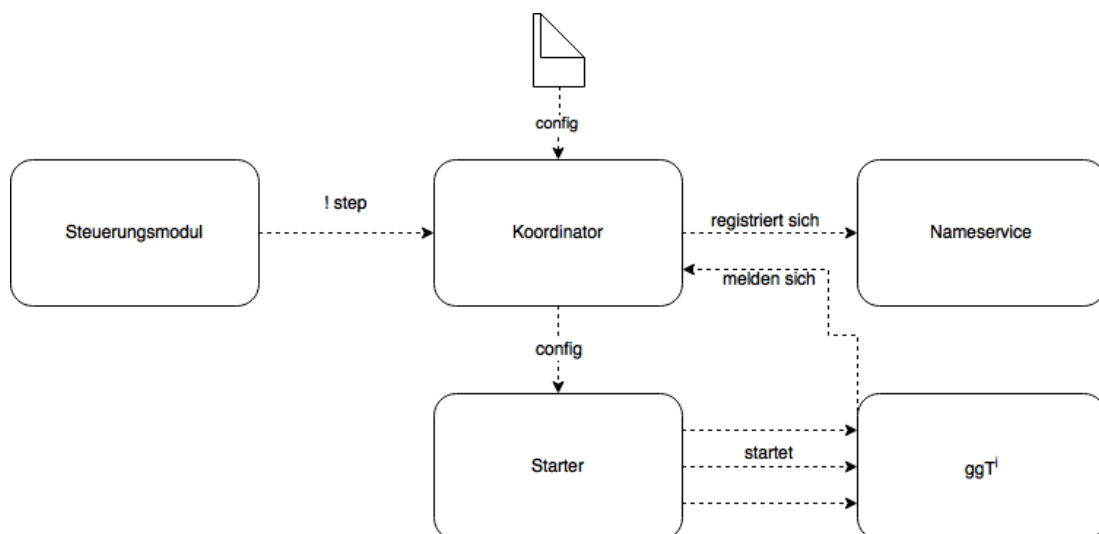
Das System geht die folgenden Phasen durch: die Initialisierungsphase, die Arbeitsphase und die Beendigungsphase.

Zwischen den Phasen wird mit Hilfe von Steuerungsmodul gewechselt (Versand der entsprechenden Nachrichten als manuelle Befehle).

3.1 Initialisierungsphase

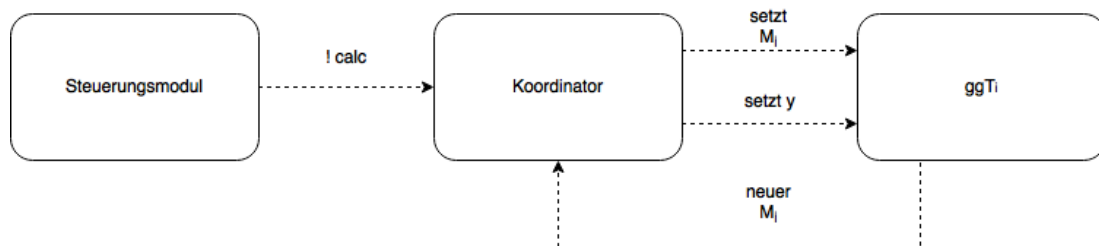
- Zuerst werden der Namensdienst und der Koordinator gestartet.
- Dann werden die Starter Prozesse gestartet.
- Die Starter erfragen beim Koordinator die steuernden Werte für die ggT-Prozesse, um damit die gewünschte Anzahl der Prozesse zu starten und sich im Anschluss selbständig zu beenden. Zu steuernden Werten gehören die Anzahl der zu startende ggT-Prozesse, eine ggT-Verzögerungszeit (Laufzeit einzelner Berechnung) und auch eine ggT-Terminierungszeit (Zeit ohne eingehende Nachrichten bis die Terminierungsanfrage an dem rechten Nachbar gestellt wird).
- Der Koordinator sammelt die ggT-Prozesse zu einem zufällig aufgebauten Ring, indem er jedem ggT-Prozess seinen linken und rechten Nachbar zuordnet.

Wenn der Koordinator in den Zustand "bereit" manuell eingestellt wird, gibt er sowohl keine Steuerwerte an Starter mehr, als auch registriert keine ggT-Prozesse mehr.



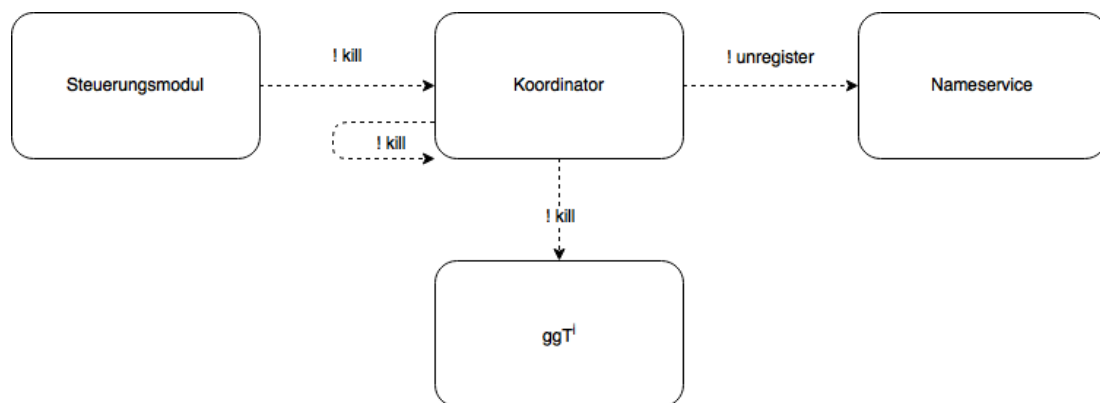
3.2 Arbeitsphase

- Es werden keine Konfigurationsdaten von Koordinator mehr gesendet.
- Es werden keine ggT-Prozesse beim Koordinator mehr registriert.
- Am Anfang der Arbeitsphase wird vom Steuerungsmodul an Koordinator der gewünschte ggT zur Kontrolle der verteilten Berechnung übergeben.
- Es werden initiale M_i Werte mit Hilfe von werkzeug Datei berechnet und vom Koordinator jedem ggT-Prozess mitgeteilt.
- Nachdem alle ggT-Prozesse ihren M-Wert erhalten haben, schickt der Koordinator an per Zufall ausgewählte 20% der ggT-Prozesse (min. 2) eine Nachricht mit einem y-Startwert, um die Berechnung zu starten.
- Immer wenn sich der M_i -Wert eines ggT-Prozesses ändert, informiert er den Koordinator darüber.
- Wenn die Berechnung beendet und das Endergebnis der Berechnung wird dem Koordinator gemeldet.



3.3 Beendigungsphase

- Der Koordinator empfängt vom Steuerungsmodul einen Terminierungsbefehl und schickt jedem ggT-Prozess eine Nachricht „kill“.
- Nach dem Empfang dieser Nachricht werden alle Prozessaktivitäten gestoppt. Die jeweiligen Prozesse melden sich beim Namensdienst ab und beenden sich.
- Am Ende terminiert der Koordinator.



4 Komponentendetails

4.1 Manuelles Steuerungsmodul

Dieses Modul ermöglicht dem Benutzer einen Einfluss auf das System, z.B. kann explizit die Berechnung des ggTs gestartet werden oder terminiert werden.

4.1.1 Methoden

/ Sendet den Reset Befehl an den Koordinator und bekommt seine Antwort zurück */*

`reset() : void -> atom`

/ Sendet den Step Befehl an den Koordinator und bekommt seine Antwort zurück */*

`step() : void -> atom`

/ Sendet den Prompt Befehl an den Koordinator und bekommt seine Antwort zurück */*

`prompt() : void -> atom`

/* Sendet den Nudge Befehl an den Koordinator und bekommt seine Antwort zurück */

nudge() : void -> atom

/* Sendet den Toggle Befehl an den Koordinator und bekommt seine Antwort zurück */

toggle() : void -> atom

/* Sendet den Calc Befehl an den Koordinator und bekommt seine Antwort zurück */

calc(WggT) : Int -> atom */* WggT ist ein gewünschter ggT Wert als erwartetes Ergebnis von ggT-Prozessen */*

4.2 Starter

Der Starter Prozess meldet sich beim Koordinator und bekommt von ihm die initialen steuernden Werte, z.B. wie viele ggT-Prozesse gestartet werden sollen. Die weiteren Daten werden aus der Konfigurationsdatei **ggt.cfg** ausgelesen:

- die Nummer der Praktikumsgruppe
- die Nummer des Teams
- die Erlang-Node und der Name des Namensdienstes
- der Name des Koordinators

Dann startet er die gewünschte Anzahl an ggT-Prozessen mit den zugehörigen Daten:

- der Verzögerungszeit (simulierte Verzögerungszeit zur Berechnung in Sekunden)
- der Terminierungszeit (die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung angekündigt wird)
- der Startnummer dieses Prozesses (wievielte gestartete ggT-Prozess er ist)
- seiner eindeutigen Starternummer

- der Praktikumsgruppennummer
- der Teamnummer sowie den benötigten Kontaktdaten für den Namensdienst und den Koordinator
- der Abstimmungsquote als konkrete Anzahl (wie viele Nachbarn zur Terminierungsanfrage vom Initiator stimmen müssen).

4.2.1 Methoden

// Startet den Starter Prozess und bekommt seine Antwort zurück

Start(Starternummer) : Int -> atom

/* Starternummer gilt als ID des konkreten Starter-Prozess */

4.3 Koordinator

Der Koordinator verwaltet das Starten und Runterfahren des Berechnungssystems und steuert die Berechnung. Alle ggT- und Starter-Prozesse melden sich zunächst bei ihm, damit sie selbständig die Aufgabe mit ausreichenden Daten weiterbearbeiten können. Die manuelle Steuerung ist mittels spezielles Steuerungsmodul möglich und simuliert diesbezüglich den Einfluss von Benutzer.

Initialisiert wird der Koordinator durch eine `koordinator.cfg` Datei. Die jeweiligen Parameter sind:

- Die Arbeitszeit (simuliert einen Arbeitsaufwand) der ggT-Prozesse in Sekunden
- Terminierungszeit (wann muss eine Terminierungsanfrage gestartet werden, falls keine neuen Nachrichten von Nachbarn ankommen) der ggT-Prozesse in Sekunden
- Die Anzahl der ggT-Prozesse pro Starter
- die Erlang-Node und der Name des Namensdienstes
- der Name des Koordinators

- die Quote (die Anzahl an Bestimmungen von den ggT-Prozessen) als prozentualer Wert
- der Flag „korrigieren“, der bestimmt, inwieweit der Koordinator bei Terminierungsmeldungen korrigierend eingreifen soll (z.B. wenn das Ergebnis dem voraussichtlichen noch widerspricht).

4.3.1 Methoden

/* Startet den Koordinator Prozess und bekommt seine Antwort zurück */

Start() : void -> atom

4.4 Externe Komponenten

Diese Komponente stellen nützliche Funktionen für das Loggen, das Auslesen der Konfigurationsdatei und auch Prozessnamenregistrierung und Adressierung bereit.

Die Komponenten bestehen aus dem Erlang Modul “werkzeug.erl” und “nameservice.beam”.

Wie schon oben genannt, es gibt noch ggT-Prozess(e), die nach Anfrage von Koordinator einen ggT-Wert mittels des Satzes von Euklid berechnen.

5 Schnittstellen

5.1 Der Zustand „initial“

5.1.1 Steuerungsmodul

/ Wechseln den Zustand des Koordinators in Initialmodus */*

Koordinator ! step

step

Diese Nachricht schaltet den Koordinator in die **Initialisierungsphase** um, womit der Koordinator den ggT-Ring aufbaut.

Sobald die Phase ist aktiv, sendet er die erfolgreiche Benachrichtigung zurück und wartet auf den Befehl zum Starten der Berechnung.

Als *self()* wird die Erlang-Node vom Steuerungsmodul übergeben.

5.1.2 Starter

/ Abfragen die steuernden Werte */*

Koordinator ! {self(), getsteeringval}

Starter ! {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer}

getsteeringval

Fragt beim Koordinator eine aktuelle Konfiguration für ggT-Prozessen Kette.

self() stellt die Rückrufadresse des Starters dar.

Als Rückgabewert erhält er eine für ihn aktuelle *steeringval* (Konfiguration) zugestellt.

Die *Arbeitszeit* ist die simulierte Verzögerungszeit zur Berechnung in Sekunden.

Die *TermZeit* ist die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung initiiert wird.

Quota ist die konkrete Anzahl an benötigten Zustimmungen zu einer Terminierungsabstimmung und *GGTProzessnummer* ist die Anzahl der zu startenden ggT-Prozesse.

5.2 Der Zustand „bereit“

5.2.1 Steuerungsmodul

```
/* Start neuer Berechnung */  
Koordinator ! {calc, WggT}
```

```
/* Rückkehr in initialen Zustand */  
Koordinator ! reset
```

```
/* Abfragen aktuelles  $M_i$  bei ggT Prozessen */  
Koordinator ! prompt
```

```
/* Abfragen aktuelles Lebenszustandes bei ggT Prozessen */  
Koordinator ! nudge
```

```
/* Umschalten des Flags für manuelle Korrektur bei Terminierungsmeldungen, wenn das  
Ergebnis sich von der vorgegebenen Zahl unterscheidet */  
Koordinator ! toggle
```

calc

Nach dem Empfang dieses Befehls startet der Koordinator eine neue ggT-Berechnung mit Wunsch-ggT *WggT*.

reset

Nach dem Empfang dieses Befehls sendet der Koordinator allen ggT-Prozessen das *kill*-Kommando und bringt sich selbst in den initialen Zustand, indem sich Starter wieder melden können.

prompt

Nach dem Empfang dieses Befehls erfragt der Koordinator bei allen ggT-Prozessen per *tellmi* deren aktuelles M_i ab und zeigt dies im Log an.

nudge

Nach dem Empfang dieses Befehls erfragt der Koordinator bei allen ggT-Prozessen per *pingGGT* deren Lebenszustand ab und zeigt dies im Log an.

toggle

Nach dem Empfang dieses Befehls verändert der Koordinator den Flag zur Korrektur bei falschen Terminierungsmeldungen.

5.2.2 Starter

```
/* Abfragen die steuernden Werte */  
Koordinator ! {self(), getsteeringval}  
Starter ! {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer}
```

getsteeringval

Fragt beim Koordinator eine aktuelle Konfiguration für ggT-Prozessen Kette.

self() stellt die Rückrufadresse des Starters dar.

Als Rückgabewert erhält er eine für ihn aktuelle *steeringval* (Konfiguration) zugestellt.

Die *Arbeitszeit* ist die simulierte Verzögerungszeit zur Berechnung in Sekunden;

Die *TermZeit* ist die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung initiiert wird;

Quota ist die konkrete Anzahl an benötigten Zustimmungen zu einer Terminierungsabstimmung und *GGTProzessnummer* ist die Anzahl der zu startenden ggT-Prozesse.

5.2.3 Koordinator

```
/* Setzt den Namen des linken und rechten Nachbarn */  
ggT-Prozess ! {setneighbors, LeftN, RightN}
```

```
/* Setzt das  $M_i$  im GGT-Prozess neu */  
ggT-Prozess ! {setpm, MiNeu}
```

```
/* Startet die Berechnung (oder stoßt die terminierte Berechnung weiter an) */  
ggT-Prozess ! {sendy,Y}
```

```
/* Abfragen aktuelles Wertes von  $M_i$  */  
ggT-Prozess ! {self(), tellmi}  
{mi, Mi} // neuer  $M_i$ -Wert
```

```
/* Prüfung der Lebendigkeit des Rings */  
ggT-Prozess ! {self(), pingGGT}  
{pongGGT, GGTname} // ggT-Prozess läuft noch
```

setneighbors

Die (lokal auf deren Node registrierten und im Namensdienst registrierten) Namen (keine PID) des linken und rechten Nachbarn werden gesetzt.

setpm

Die zu bearbeitende Zahl für eine neue Berechnung wird gesetzt.

sendy

Der rekursive Aufruf der ggT Berechnung.

tellmi

Gibt das aktuelle Mi zurück. Wird genutzt, um bei einem Berechnungsstillstand die Mi-Situation im Ring anzuzeigen.

pingGGT

Wird genutzt, um auf manuelle Anforderung hin die Lebendigkeit des Rings zu prüfen.

5.3 Der Zustand „beenden“

5.3.1 Steuerungsmodul

```
/* Beenden des Koordinator Prozesses */  
Koordinator ! kill
```

kill

Nach dem Empfang dieses Befehls wird der Koordinator beendet und sendet an alle ggT-Prozessen das *kill*-Kommando.

5.3.2 Koordinator

```
/* Beenden des ggT Prozesses */  
GGT-Prozess ! kill
```

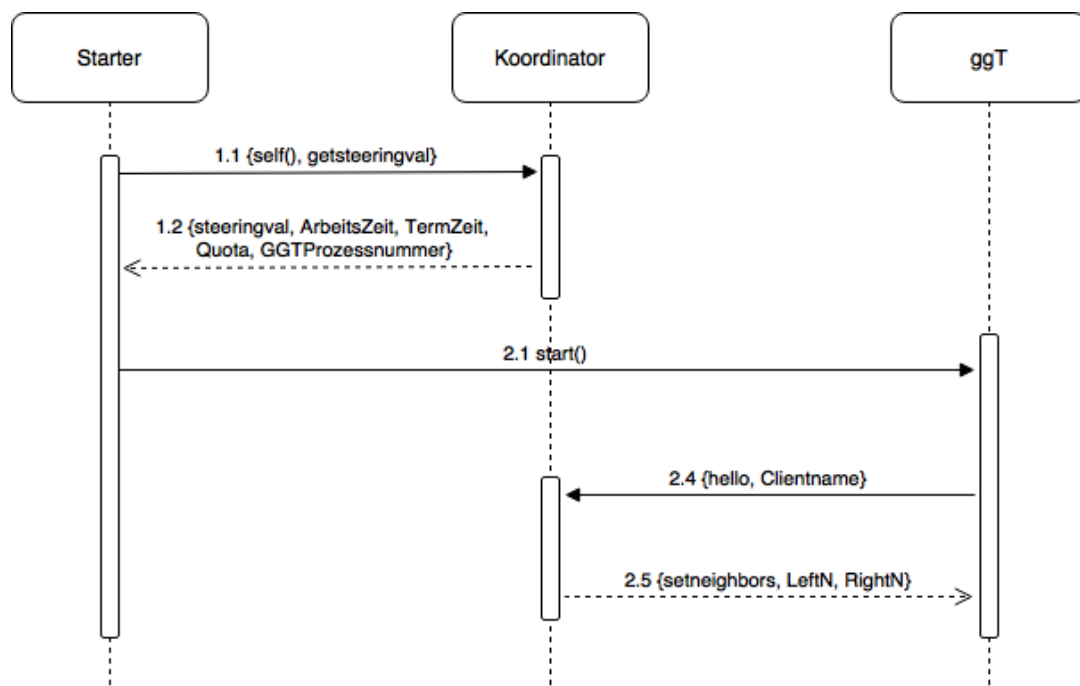
kill

Anfrage an den ggT-Prozess, damit er beenden werden kann.

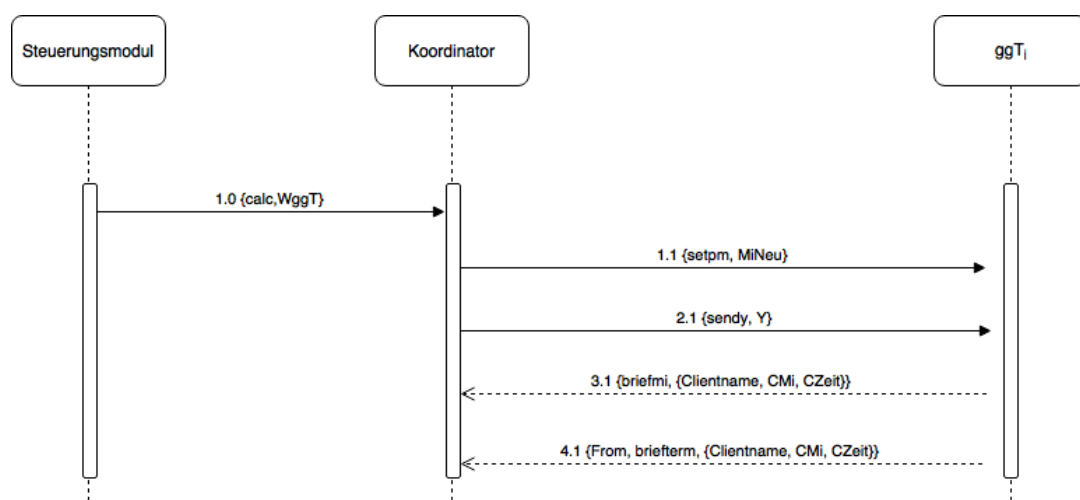
6 Appendix

6.1 Sequenzdiagramme

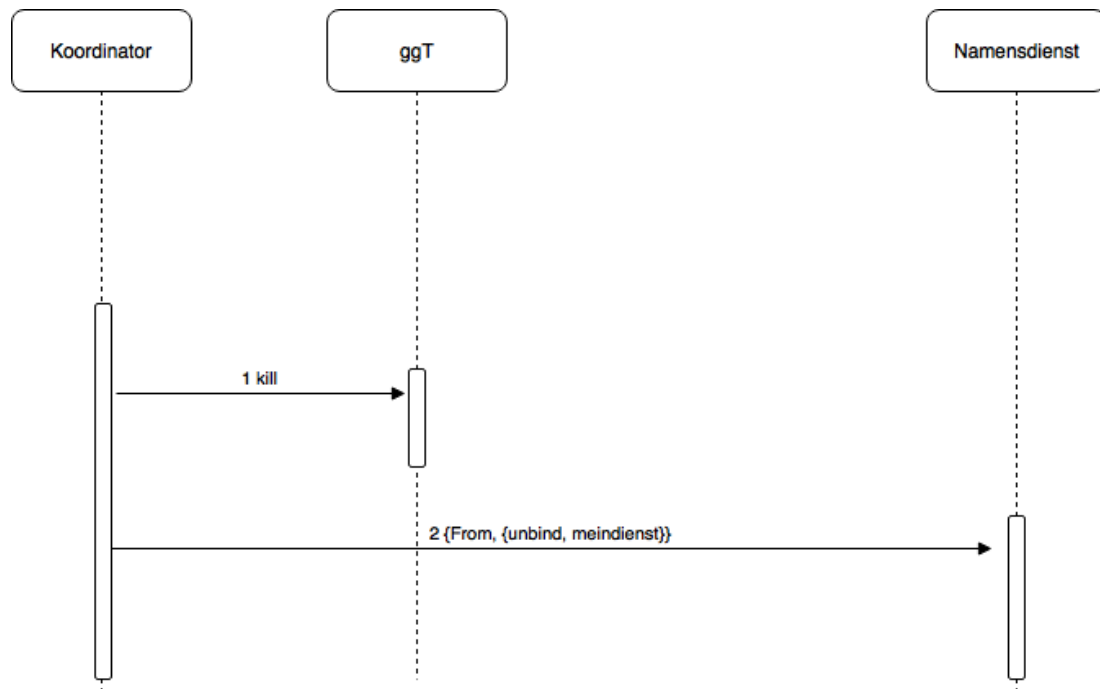
6.1.1 Initialisierungsphase (Voraussetzung: Module sind gestartet)



6.1.2 Arbeitsphase



6.1.3 Beendigungsphase



6.2 Ablauf des Programms (Bearbeitungsschritte)

Initialisierungsphase

1. /* Initialisieren des Namensservice */

```

global:register_name(nameservice,NServerPid)
% passiert automatisch in nameservice.beam

```

2. /* Initialisieren des Koordinators */

```

register(KoName,KoPID)
Nameservice ! {self(),{rebind,KoName,KoNode}}
receive ok -> io:format("..rebind.done.\n")

```

+ Log:

- Koordinator-ko@Brummpa-KLC Startzeit: 24.02 14:39:32,453 | mit PID <0.37.0>
- koordinator.cfg geöffnet...
- koordinator.cfg gelesen...
- Nameservice gebunden...
- lokal registriert...
- beim Namensdienst registriert.

Zustand “initial”

3. /* Initialisieren des Starter */

```

register(StarterName,StarterPID)
Nameservice ! {self(),{rebind,StarterName,StarterNode}}
receive ok -> io:format("..rebind.done.\n")
Koordinator ! {self(), getsteeringval}
receive {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessAnzahl}

```

% ArbeitsZeit -> die simulierte Verzögerungszeit zur Berechnung in Sekunden.

% TermZeit -> die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung initiiert wird.

% Quota -> die konkrete Anzahl an benötigten Zustimmungen zu einer Terminierungsabstimmung.

% GGTProzessAnzahl -> die Anzahl der zu startenden ggT-Prozesse.

GGT-Prozess ! start(ArbeitsZeit, TermZeit, Quota, GGTProcNr, StarterNr, GruppeNr, TeamNr, Nameservice, Koordinator)

% die angegebene Anzahl von GGTS starten

+ Log:

- Starter_5-ggt@KI-VS-KLC Startzeit: 24.02 14:40:14,222 | mit PID <0.40.0>
- ggt.cfg geöffnet...
- ggt.cfg gelesen...
- Nameservice gebunden...
- Koordinator chef (chef) gebunden.
- getsteeringval: 2 Arbeitszeit ggT; 42 Wartezeit Terminierung ggT; 7 Abstimmungsquote ggT; 9-te GGT Prozess.

4. /* Initialisieren des ggT-Prozess */

```

...
Koordinator ! {hello, Clientname}
% Clientname -> Name ist der lokal registrierte Name, keine PID!
...

```

+ Log:

- ...
- beim Koordinator gemeldet.
- ...

5. /* manuelle Steuerung / Starten der Berechnung */

```

Koordinator ! step

```

% Der Koordinator beendet die Initialphase und bildet den Ring. Er wartet nun auf den Start einer ggT-Berechnung.

Zustand "bereit"

6. /* manuelle Steuerung */

```
Koordinator ! {calc,WggT}
% eine ggT-Berechnung wird per manuellem Befehl (calc) gestartet.
% WggT -> Wunsch-ggT
```

7. /* Koordinator */

```
GGT ! {setpm, MiNeu}
GGT ! {sendy, Y}
```

8. /* ggT */

```
Koordinator ! {briefmi,{Clientname,CMi,CZeit}}
% Ein ggT-Prozess mit Namen Clientname (keine PID!) informiert über sein neues
Mi CMi um CZeit Uhr.
```

9. /* ggT */

```
Koordinator ! {From, briefterm, {Clientname, NewMi, CZeit}}
% Terminierungsnachricht des GGT-Prozesses an den Koordinator mit dem
entsprechenden Wert
```

10. /* Koordinator */

```
ggT-Prozess ! {From, tellmi}
From ! {mi, Mi}
% Aktuelles Mi abfragen
```

Zustand "beenden"

11. /* Terminierung des Koordinator */

```
receive kill
ggT-Prozess ! {self(), kill}
...
receive ok -> io:format("..unbind..done.\n")
```

+ Log

- Allen ggT-Prozessen ein 'kill' gesendet.
- Downtime: 24.02 14:44:43,094 | vom Koordinator chef

12. /* Terminierung des GGT */

```
receive {self(), kill}
...
```

13. /* Ausserdem */

```
/* GGT */
receive {From,pingGGT}
From ! {pongGGT,GGTname}
% Sendet ein pongGGT an From (ist PID). Wird vom Koordinator z.B. genutzt, um
auf manuelle Anforderung hin die Lebendigkeit des Rings zu prüfen.
```

```
/* manuelle Steuerung */
Koordinator ! reset
% Der Koordinator sendet allen ggT-Prozessen das kill-Kommando und bringt sich
selbst in den initialen Zustand, indem sich Starter wieder melden können.
```

```
Koordinator ! prompt
% Der Koordinator erfragt bei allen ggT-Prozessen per tellmi deren aktuelles Mi ab
und zeigt dies im Log an.
```

```
Koordinator ! nudge
% Der Koordinator erfragt bei allen ggT-Prozessen per pingGGT deren Lebenszustand
ab und zeigt dies im Log an.
```

```
Koordinator ! toggle
% Der Koordinator verändert den Flag zur Korrektur bei falschen
Terminierungsmeldungen.
```

7 Referenzen

Aufgabenstellung:


- <http://users.informatik.haw-hamburg.de/~klauck/VerteilteSysteme/aufg2.html>

Erklärung zur schriftlichen Ausarbeitung des Referates

Erklärung zur schriftlichen Ausarbeitung des Referates

Hiermit versichere ich, dass ich diese schriftliche Ausarbeitung meines Referates selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie die aus fremden Quellen (dazu zählen auch Internetquellen) direkt oder indirekt übernommenen Gedanken oder Wortlaute als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Hamburg, den 16.11.2016

 Igor Arkhipov