ENTWURF

ENTWURF zur aufgabe 2 aus der vorlesungsreihe “VERTEILTE SYSTEME”

TEAM 1: MICHAEL STRUTZKE, iGOR arkhipov

Aufgabenaufteilung

Alle Aufgaben wurden gemeinsam entworfen und bearbeitet.

## Quellenangaben

Aufgabenstellung, Vorstellung der Aufgabe in der Vorlesung.

## Bearbeitungszeitraum

|  |  |
| --- | --- |
| Datum | Dauer in Stunden |
| 21.10.16 | 3 |
| 22.10.16 | 7 |
| 25.10.16 | 4 |
| 26.10.16 | 4 |
| 27.10.16 | 5 |
| 28.10.16 | 6 |
| Gesamt | 29 |

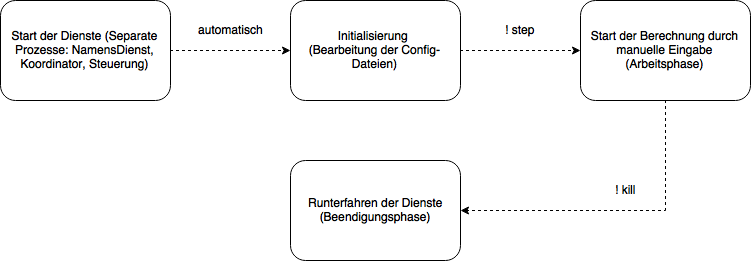
## Aktueller Stand

Der Entwurf ist fertig. Die Implementation wird eingesetzt.

## Änderungen im Entwurf

Noch keine.

## Details zum Entwurf:



Es soll sowohl ein verteilter Algorithmus, als auch ein Koordinator dafür implementiert werden. Als Hilfsmodule sollen noch ein Startermodul und ein Steuerungsmodul für manuelle Kontrolle der gesamten Anwendung implementiert werden.

Der Algorithmus stellt die nebenläufige Berechung des ggT (des größten gemeinsamen Teilers) von mehreren Zahlen (entsprechend der Anzahl der Arbeitsprozesse) dar. 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.

Der verteilte Algorithmus basiert auf dem Satz von Euklid und funktioniert folgenderweise:

* Jeder Prozess bekommt seine eigene verwaltende Zahl Mi (als Variable).
* Der ggT aller Mi wird wie folgt berechnet:

{Eine Nachricht <y> ist eingetroffen}

if y < Mi

then Mi := mod(Mi-1,y)+1;

send #Mi to all neighbours;

fi

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.

## Komponentenübersicht

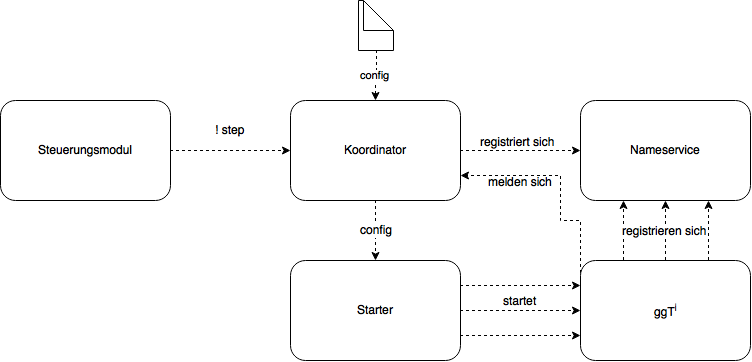
* Der externe Namensdienst ordnet den PIDs die registrierten Prozessnamen zu.
* Der Starter-Prozess ist für Starten der ggT-Prozesse zuständig.
* Der Koordinator bietet eine Schnittstelle zwischen dem System und einem manuellen Steuerungsmodul an.
* Die GGT-Prozessen verwirklichen die eigentliche Berechnung des ggT.
* Das Steuerungsmodul vermittelt die Nachrichten zwischen User und Koordinator.
* Jede Komponente erstellt ausführliche Logdateien. Die Kommunikation wird mit Message- Passing von Erlang/OTP implementiert.

## 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).

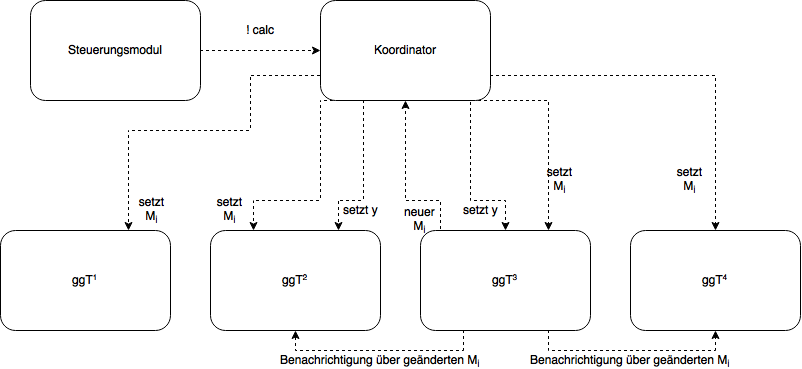
### Initialisierungsphase

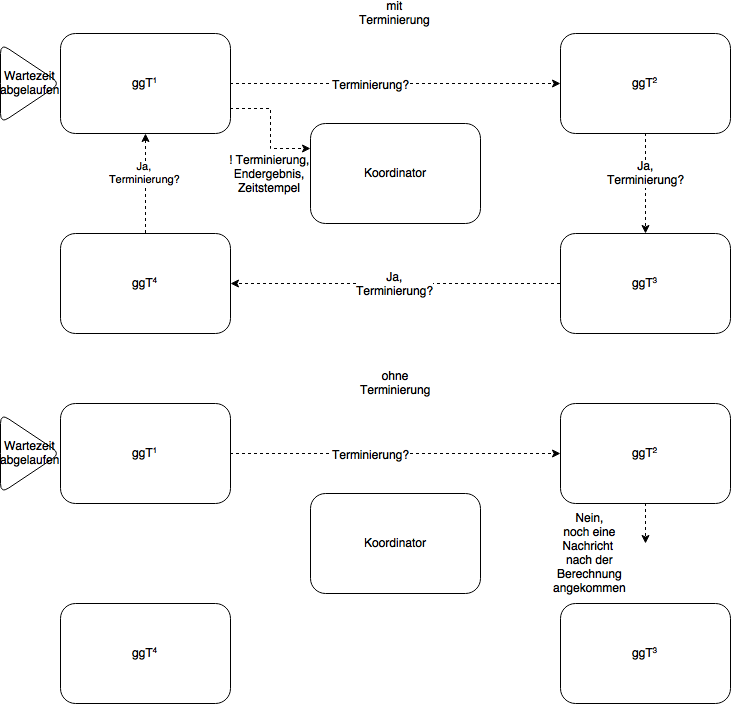
* Zuerst werden der Namensdienst und der Koordinator gestartet.
* Dann werden die Starter Prozesse gestartet.
* Die Starter erfragen beim Koordinator die steuernden Werte für die ggT-Prozesse, um damit die gewünschte Anzahl der Prozesse zu starten und sich im Anschluss selbständig zu beenden. Zu steuernden Werten gehören die Anzahl der zu startende ggT-Prozesse, eine ggT-Verzögerungszeit (Laufzeit einzelner Berechung) und auch eine ggT-Terminierungszeit (Zeit ohne eingehende Nachrichten bis die Terminierungsanfrage an dem rechten Nachbar gestellt wird).
* Jeder gestartete ggT-Prozess meldet sich beim Koordinator und beim Namensdienst mit seinem lokal registrierten Namen.
* 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.



### Arbeitsphase

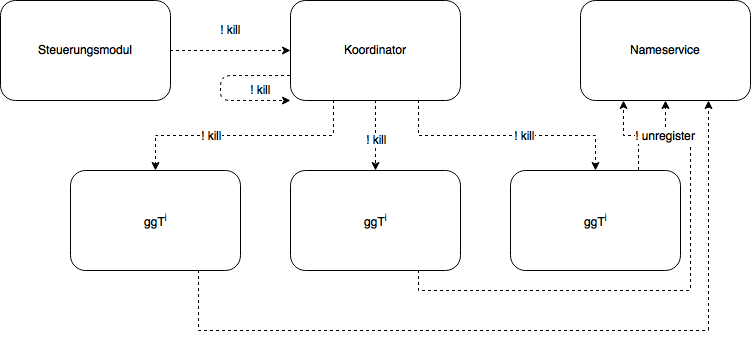
* Es werden keine Konfigurationsdaten von Koordinator mehr gesendet.
* Es werden keine ggT-Prozesse beim Koordinator mehr registriert.
* Am Anfang der Arbeitsphase wird vom Steuerungsmodul an Koordinator der gewünschte ggT zur Kontrolle der verteilten Berechnung übergeben.
* Es werden initiale Mi Werte 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 Mi-Wert eines ggT-Prozesses ändert, informiert er den Koordinator darüber und schickt diesen Wert auch an seinen linken und rechten Nachbarn.
* Wenn einer der ggT-Prozesse während der Wartezeit (vorkonfigurierte) keine Nachricht bekommt, startet er eine Terminierungsabstimmung. Dazu schickt er eine mit seinem eigenen registrierten Namen (als Initiator) Nachricht an seinen rechten Nachbarn.
* Wenn sein Nachbar Wartezeit/2 Millisekunden seit dem Empfang der Terminierungsabstimmungsanfrage gewartet hat, stimmt er damit zu und schickt diese Nachricht weiter.
* Kommt diese Nachricht beim ursprünglichen Versender wieder an, gilt die Berechnung als beendet und das Endergebnis der Berechnung wird dem Koordinator gemeldet.
* Alternativ: ein ggT-Prozess (nicht Initiator) erhält die Nachricht zur Terminierungsabstimmung, aber bekommt in den letzten Wartezeit/2 Sekunden noch eine Zahl. Dann wird die Terminierungsnachricht ignoriert.





### Beendigungsphase

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



## Komponentendetails

### 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.

### **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 \*/

### 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 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).

### **Methoden:**

### **// Startet den Starter Prozess und bekommt seine Antwort zurück**

### **Start(Starternummer) : Int -> atom** /\*Starternummer gilt als ID des konkreten Starter-Prozess \*/

### 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, fall 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, die für die Abstimmung der Terminierungsanfrage notwendig ist) als prozentualer Wert
  + der Flag „korrigieren“, der bestimmt, inwieweit der Koordinator bei Terminierungsmeldungen korrigierend eingreifen soll (z.B. wenn das Ergebnis dem voraussichlichen noch widerspricht)

### **Methoden:**

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

### **Start() : void -> atom**

### ggt-Prozess

Der ggT-Prozess registriert sich beim Namensdienst und Koordinator. Danach werden die notwendigen Konfigurationsdaten abgefragt. Der Name wird in folgender Form eingesetzt:  <PraktikumsgruppenID><TeamID><Nummer des ggT-Prozess> <Nummer des Starters>

Nach Anfrage von Koordinator berechnet der Prozess einen ggT-Wert mittels des Satzes von Euklid und wartet die vorgegebene Arbeitszeit. Sobald der Prozess fertig ist, schickt er das Ergebnis an den Koordinator zurück.

Wenn aber der Prozess für eine Wartezeit gewartet hat und keine neue Zahl von seinen Nachbarn währenddessen erhalten hat, startet er eine Abstimmungsumfrage, indem er eine spezielle Nachricht an seinen rechten Nachbarn sendet. Wenn beim rechten Nachbarn jetzt keine Nachricht für Wartezeit/2 ankommt, sendet er die Terminierungsanfrage vom Initiator weiter nach rechts. Ansonsten wird die Anfrage ignoriert. Erst dann, wenn solche Nachricht eine Runde macht und wieder beim Initiator von dem linken Nachbarn eingegangen wird, wird die Nachricht mit dem errechneten Wert an den Koordinator gesendet. Als zusätzliche Voraussetzung muss noch die Anzahl der „Zustimmungen“ gleich oder sogar größer als die Quote sein.

Für die Terminierung des Prozesses reicht eine Nachricht „kill“ vom Koordinator. In diesem Fall meldet sich der Prozess beim Namensdienst ab und terminiert sich danach.

### **Methoden:**

### **// Startet den ggT Prozess und bekommt seine Antwort zurück**

### **Start(Arbeitszeit, Termzeit, Name) : Int x Int x String -> atom**

### /\*Arbeitszeit gilt als Arbeitsaufwand in Sekunden; Termzeit ist eine Zeit bis die Terminierungsanfrage generiert wird; Name ist ein Name des ggT Prozesses \*/

### 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”.

# Schnittstellen

## der Zustand „initial“

### Steuerungsmodul

### **/\* Wechseln den Zustand des Koordinators in Initialmodus \*/**

### **Koordinator ! {self(), step}**

### **{reply, ok} // Initialisierung war erfolgreich**

### **step**

Diese Nachricht schaltet den Koordinator in die **Initialisierungphase** 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.

### 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; TermZeit ist die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung initiiert wird; Quota ist die konkrete Anzahl an benotwendigten Zustimmungen zu einer Terminierungsabstimmung und GGTProzessnummer ist die Anzahl der zu startenden ggT-Prozesse.

### ggT-Prozess

### **/\* Registrierung des Prozesses beim Koordinator \*/**

### **Koordinator ! {hello, Clientname}**

### **{ hello, Clientname** }

### Ein ggT-Prozess meldet sich beim Koordinator mit Namen Clientname an (Name ist der lokal beim nameservice registrierte Name).

## Der Zustand „bereit“

### Steuerungsmodul

### **/\* Start neuer Berechnung \*/**

### **Koordinator ! {self(), calc, WggT}**

### **{reply, ok} // Start war erfolgreich**

### **{reply, fail} // Die Berechnung wurde nicht gestartet**

### **/\* Rückkehr in initialen Zustand \*/**

### **Koordinator ! {self(), reset}**

### **{reply, ok} // Die Zurücksetzung war erfolgreich**

### **/\* Abfragen aktuelles Mi bei ggT Prozessen \*/**

### **Koordinator ! {self(), prompt}**

### **{reply, ok} // Die Anfrage wurde bestätigt**

### **/\* Abfragen aktuelles Lebenszustandes bei ggT Prozessen \*/**

### **Koordinator ! {self(), nudge}**

### **{reply, ok} // Die Anfrage war erfolgreich**

### **/\* Umschalten des Flags für manuelle Korrektur bei Terminierungsmeldunge, wenn das Ergebnis sich von der vorgegebenen Zahl unterscheidet \*/**

### **Koordinator ! {self(), toggle}**

### **{reply, ok} // Start war erfolgreich**

### **calc**

Nach dem Empfang dieses Befehls startet der Koordinator eine neue ggT-Berechnung mit Wunsch-ggT WggT. Als self() wird die Erlang-Node vom Steuerungsmodul übergeben.

### **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 Mi 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.

### 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; TermZeit ist die Wartezeit in Sekunden, bis eine Wahl für eine Terminierung initiiert wird; Quota ist die konkrete Anzahl an benotwendigten Zustimmungen zu einer Terminierungsabstimmung und GGTProzessnummer ist die Anzahl der zu startenden ggT-Prozesse.

### Koordinator

### **/\* Setzt den Namen des linken und rechten Nachbarn \*/**

### **ggT-Prozess ! {setneighbors, LeftN, RightN}**

### **/\* Setzt das Mi 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 Mi \*/**

### **ggT-Prozess ! {self(), tellmi}**

### **{mi, Mi} // neuer Mi-Wert**

### **/\* Prüfung der Lebendigkeit des Rings \*/**

### **ggT-Prozess ! {self(), pingGGT}**

### **{pongGGT, GGTname} // ggT-Prozess läuft noch**

### **setneighbors**

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

### **setpm**

die zu berabeitende 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.

### ggT-Prozess

### **/\* Führt die Berechnung rekursiv \*/**

### **ggT-Prozess ! {sendy,Y}**

### **/\* Abfragen aktuelles Wertes von Mi \*/**

### **Koordinator ! {briefmi, {Clientname, CMi, CZeit}}**

### **{mi, Mi} // neuer Mi-Wert**

### **/\* Initiierung der Terminierung beim aktuellen Berechnung \*/**

### **ggT-Prozess ! {self(), {vote, Initiator}}**

### **/\* Abstimmung mit der Terminierungsanfrage \*/**

### **ggT-Prozess ! {voteYes, Name}**

### **/\* Information über das Ergebnis der Berechnung \*/**

### **Koordinator ! {self(), briefterm, {Clientname, CMi, CZeit}}**

### **sendy**

der rekursive Aufruf der ggT Berechnung.

### **briefmi**

Ein ggT-Prozess mit Namen Clientname (keine PID!) wird nach diesem Aufruf den Koordinator über sein neues Mi mit dem zugehörigen Zeitstempel informieren.

### **vote**

Wahlnachricht für die Terminierung der aktuellen Berechnung; Initiator ist der Initiator dieser Wahl (Name des ggT-Prozesses, keine PID!) und From (ist PID) ist sein Absender.

### **voteYes**

erhaltenes Abstimmungsergebnis, wobei Name der Name des Absenders ist (keine PID!).

### **briefterm**

Ein ggT-Prozess mit Namen Clientname (keine PID!) und Absender PID informiert über die Terminierung der Berechnung mit Ergebnis CMi um CZeit Uhr.

## Der Zustand „beenden“

### Steuerungsmodul

### **/\* Beenden des Koordinator Prozesses \*/**

### **Koordinator ! {self(), kill}**

### **{reply, ok} // Wechsel in den „beenden“ Zustand war erfolgreich**

### **kill**

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

### Koordinator

### **/\* Beenden des ggT Prozesses \*/**

### **GGT-Prozess ! {kill}**

### **kill**

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

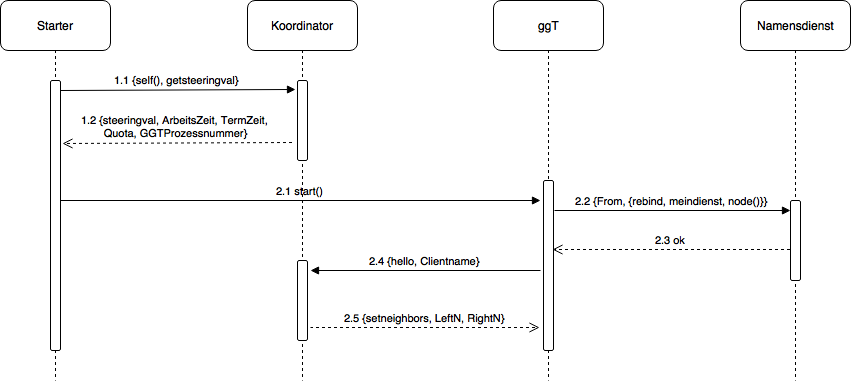
# Logs

## Koordinator + ggT-Prozess

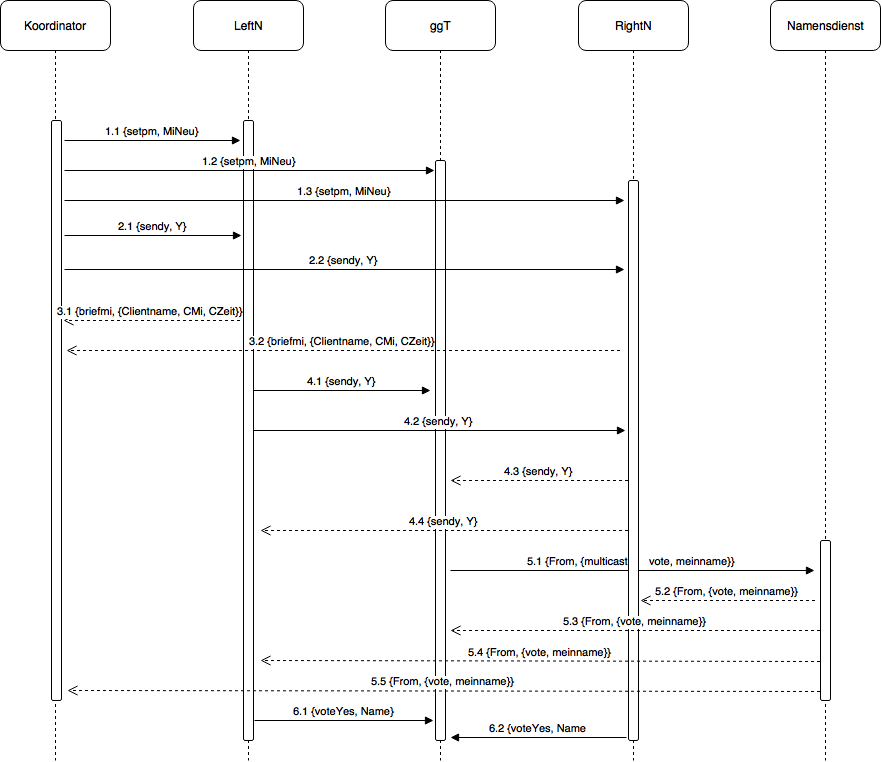
<Noch fehlt>

# Sequenzdiagramme

## Initialisierungsphase



## Arbeitsphase



## Beendigungsphase

