

Entwurf VSP3

Team: 10 (Antoni Romanski, Benjamin Schröder)

Aufgabenaufteilung

1. Der Entwurf wurde von beiden Teammitgliedern gemeinsam erstellt.
2. Die Implementation wurde wie folgt aufgeteilt:

Datei	Bearbeitet von
koordinator.erl	Antoni
starter.erl	Benjamin
ggt.erl	Benjamin

Quellenangaben: Folien und Skript, sowie Vorlesungsmitschriften von Prof. Dr. Christoph Klauck.

Bearbeitungszeitraum:

- Gemeinsam: 12.05. (3 Stunden), 13.05. (1 Stunde), 15.05. (3 Stunden), 16.05. (1 Stunde), 18.05 (2 Stunden 30 Minuten), 24.05. (30 Minuten), 30.05. (2 Stunden), 31.05. (10 Stunden)
- Antoni: 21.05 (1 Stunde) , 22.05 (2 Stunden), 23.05 (4 Stunden), 27.05 (4 Stunden), 28.05 (6 Stunden), 29.05 (4 Stunden)
- Benjamin: 18.05 (45 Minuten), 19.05. (4 Stunden 30 Minuten), 22.05. (2 Stunden 15 Minuten), 23.05. (2 Stunden), 24.05. (3 Stunden 30 Minuten), 25.05. (1 Stunde 30 Minuten), 28.05. (30 Minuten), 30.05. (2 Stunden)

Projektstruktur

Datei	Beschreibung
koordinator.erl	koordiniert den globalen Ablauf des Systems
koordinator.cfg	Konfigurationsdatei für den Koordinator
starter.erl	startet die benötigten ggT-Prozesse auf dem lokalen Rechner
ggt.erl	führt die eigentliche ggT-Berechnung durch
ggt.cfg	Konfigurationsdatei für den Starter und die davon gestarteten ggT-Prozesse

Projektbeschreibung

In diesem Projekt wird eine verteilte Version des ggT-Algorithmus, basierend auf dem Satz von Euklid, implementiert. Es gibt vier wesentliche Komponenten, die miteinander interagieren: Einen Koordinator, der den globalen Ablauf des Systems verwaltet; einen Starter pro Rechner, der das verteilte Starten der eigentlichen Berechnungsprozesse ermöglicht; die eigentlichen ggT-Prozesse, die die Berechnung durchführen; und den Namensdienst, der die Kommunikation zwischen den Komponenten ermöglicht. Der Namensdienst ist hier schon vorgegeben und muss nicht implementiert werden.

Beschreibung der Komponenten

Koordinator

Der Koordinator ist die Zentrale Verwaltungseinheit, welche den verteilten Algorithmus verwaltet und überwacht. Er liest seine Konfigurationswerte aus der Datei **koordinator.cfg** aus. Der Ablauf wird in einer Datei **Koordinator@<Rechnername>.log** protokolliert und lautet wie folgt:

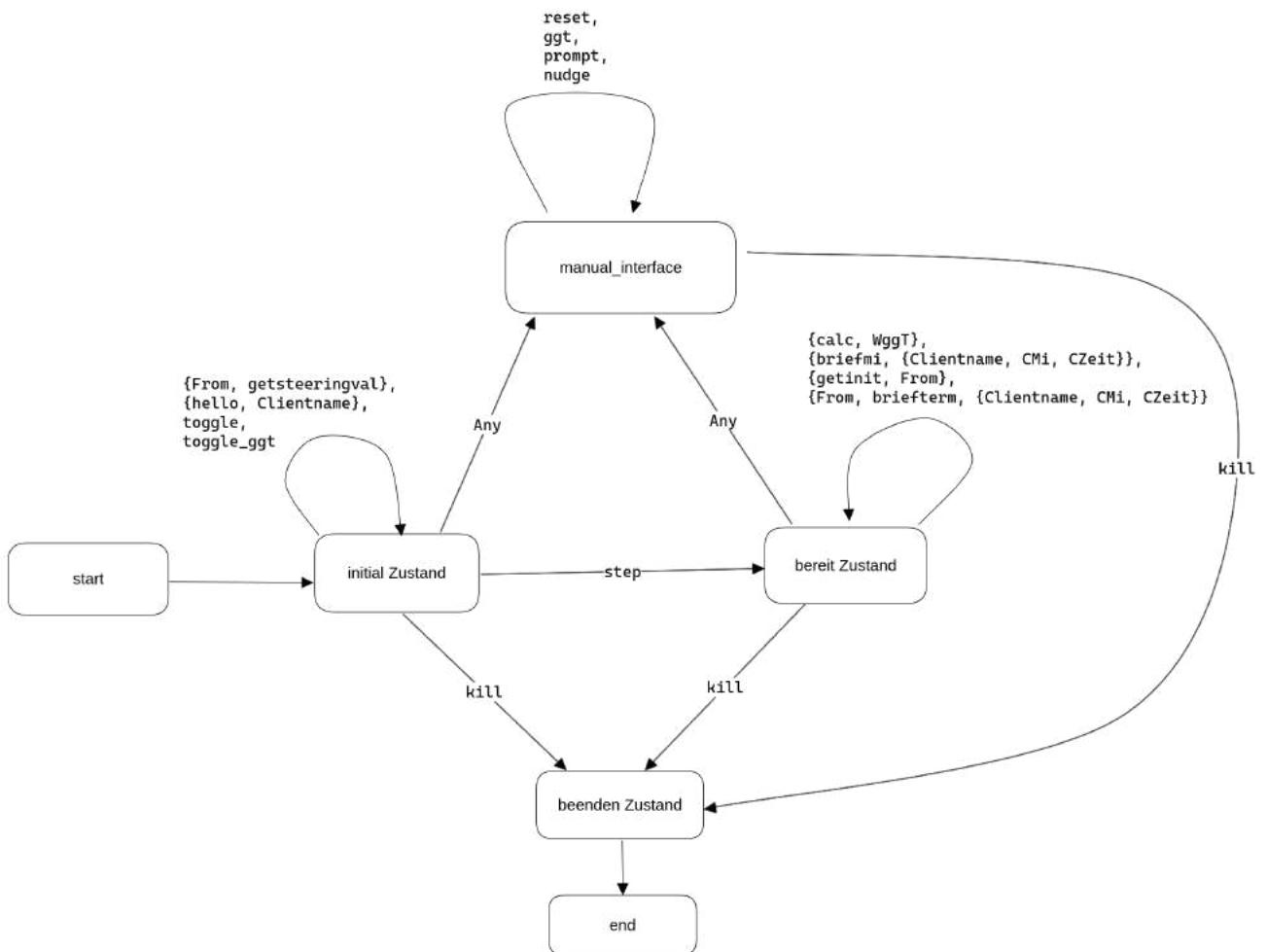
1. Auslesen der Konfigurationsdatei: Die Parameter in der Konfigurationsdatei können nicht erweitert werden und beinhalten folgende Felder:
 - **arbeitszeit**: Länge des simulierten Arbeitsaufwands
 - **termzeit**: die Terminierungszeit, welche ein ggT-Prozess wartet, ob er eine Zahl **y** bekommt
 - **ggtprozessanzahl**: Anzahl der zu startenden ggT-Prozesse
 - **nameserviceode**: Erlang-Node des Namensdienstes
 - **koordinatorname**: der Name des Koordinators
 - **korrigieren**: ein Flag, ob der Koordinator bei der Berechnung des ggT, durch eine korrigierende Antwort beim Erhalten von Terminierungsmeldungen von ggT-Prozessen, eingreifen soll
2. Registrieren: Nachdem der Koordinator die Konfiguration ausgelesen hat, registriert er sich beim Namensdienst und der lokalen Erlang-Node, um von Startern und ggT-Prozessen Nachrichten erhalten zu können.
3. "initial" Zustand: nach dem Registrieren gibt der Koordinator den Startern Auskunft durch die **{From, getsteeringval}** Schnittstelle bei welcher er **{steeringval, {AZMin, AZMax}, TermZeit, GGTProzessnummer}** zurück schickt.
4. "bereit" Zustand: durch **step** wechselt der Koordinator in den "bereit" Zustand, in welchem er keine Auskunft mehr gibt und durch **{calc, WggT}**, das Starten der Berechnung des ggT ermöglicht.
5. Ring an ggT-Prozessen: für die Berechnung baut der Koordinator einen Ring an ggT-Prozessen auf, indem diese über **{hello, Clientname}** sich beim Koordinator melden und dieser sie über ihre Nachbarn informiert (**{setneighbors, LeftN, RightN}**). Dies leitet die Arbeitsphase ein. (NB hier schaut der Koordinator über den den Namensdienst die Erlang-Node des ggT-Prozesses nach und merkt sich diese mit dem Clientnamen in **GGTClients**.).
- Der Aufbau des ggT-Rings funktioniert wie folgt:
 - a. Liste der ggT-Prozesse durchmischen
 - b. die Liste durchgehen und immer zwei ggT-Clients nehmen und die beiden als Nachbarn bei sich jeweils eintragen
 - c. wenn wir ggT-Clients besucht haben packen wir sie in eine neue "Out" Liste
 - d. beim letzten Element in der Liste nehmen wir das Letzte Element in der Out Liste (welches in der ursprünglichen ggT-Clients Liste der Erste war) und tragen ihn als Nachbar ein.
6. Start der Berechnung: Der Koordinator startet die ggT-Berechnungen, indem er den ggT-Prozessen ihre jeweiligen Startwerte mitteilt. Er wählt zufällig einen Teil der ggT-Prozesse aus, die mit der Berechnung beginnen sollen. Er informiert sie über ihre Auswahl und sie erfragen dann den initialen Wert für die Berechnung.
7. Überwachung der Terminierung: Der Koordinator erhält von den ggT-Prozessen Informationen über deren Terminierung. Falls ein ggT-Prozess ein falsches Endergebnis meldet, kann der Koordinator korrigierend eingreifen. Er kann auch eine Terminierungsabstimmung starten, wenn ein ggT-Prozess für eine bestimmte Zeit keine Zahl erhält.
8. Beendigungsphase: Während der Beendigungsphase, informiert der Koordinator die ggT-Prozesse über die Beendigung und meldet sich anschließend bei der Erlang-Node und dem Namensdienst ab.

9. Während des Algorithmus sind folgende Schnittstellen des Koordinators verfügbar und können manuell über eine Erlang-Shell und Message-Passing angesteuert werden:

- a. **reset**: sendet allen ggT-Prozessen kill und bringt sich selbst in den initialen Zustand, während welchem Starter Anfragen machen können
- b. **step**: versetzt den Koordinator in den "bereit" Zustand (wie in 4. beschrieben)
- c. **{calc, WggT}**: wie in 4. beschrieben startet der Koordinator die Berechnung des ggT für die Zahl, welche in **WggT** übergeben wird
- d. **prompt**: schickt an alle ggT-Prozesse **{self(), tellmi}**, um deren aktuelles Mi zu bekommen und log dieses auf der Konsole und Log-Datei
- e. **nudge**: sendet an alle ggT-Prozesse **{self(), pingGGT}**, um deren Lebenszustand abzufragen und log dieses auf Konsole und in der Log-Datei
- f. **toggle**: ändert den Flag zur Korrektur bei der Terminierungsabstimmung wie in 7. beschrieben
- g. **toggle_ggt**: ändert das Korrektur-Flag bei allen ggT-Prozessen durch versenden von **{self(), toggle}**
- h. **kill**: sendet **kill** an alle ggT-Prozesse und beendet diese dadurch

Koordinator Zustände

Der Koordinator hat 3 verschiedene Zustände in welchen er sich befinden kann (manual_interface dient zum vereinfachen des initial Zustands und bereit Zustands indem es Funtionalität welche in beiden besteht in einen weiteren Zustand zieht.



Starter

Der Starter wird genutzt, um es dem Koordinator zu ermöglichen die benötigten ggT-Prozesse auf entfernten Rechnern starten zu können. Die hierfür wichtigen Einstellungen lassen sich in der Konfigurationsdatei **ggt.cfg** anpassen. Der Ablauf wird in einer Datei **ggtSTARTER_<StarterNum>@<Rechnername>.log** protokolliert und lautet wie folgt:

1. Aus der Konfigurationsdatei werden die folgenden Parameter ausgelesen:
 - nameservicenode
 - koordinatormame
 - praktikumsgruppe
 - teamnummer
2. Mithilfe der ausgelesenen Nameservice-Node wird der Namensdienst ausfindig gemacht (Details dazu im Abschnitt zum Namensdienst).
3. Beim jetzt bekannten Namensdienst werden die Kontaktinformationen für den Koordinator angefragt (Senden von **{self(), lookup, <KoordinatorName>}}** an den Namensdienst).
4. Die Kontaktdaten des Koordinators werden aus der Antwort vom Namensdienst (**{pin, {<ProzessName>, <Node>}}**) ausgelesen.
5. Beim jetzt bekannten Koordinator werden die steuernden Informationen angefragt (Senden von **{self(), getsteeringval}** an den Koordinator).
6. Die steuernden Informationen werden aus der Antwort vom Koordinator (**{steeringval, {<AZMin>, <AZMax>, <TermZeit>, <Anzahl>}}**) ausgelesen.
7. Nun werden **<Anzahl>** ggT-Prozesse mit den folgenden Informationen auf der aktuellen Erlang-Node gestartet:
 - seine simulierte Arbeitszeit (zufällig ermittelt aus dem Intervall **{<AZMin>, <AZMax>}**)
 - die Terminierungszeit (**TermZeit**)
 - eine eindeutige Nummer für den momentanen ggT-Prozess
 - die eindeutige Nummer des Starters; wird beim Starten des Starters übergeben (**StarterNum**)
 - die Praktikumsgruppe (**praktikumsgruppe**)
 - die Teamnummer (**teamnummer**)
 - die Kontaktdaten für den Namensdienst
 - die Kontaktdaten für den Koordinator

ggT-Prozess

Der ggT-Prozess führt die eigentliche Berechnung des ggT durch. Die hierfür wichtigen Einstellungen lassen sich ebenfalls in der Konfigurationsdatei **ggt.cfg** anpassen, werden dem ggT-Prozess allerdings nur indirekt durch seinen Starter übermittelt. Der Ablauf wird in einer Datei **GGTP_<GGTName>@<Rechnername>.log** protokolliert und lautet wie folgt:

1. Der Name des aktuellen Prozesses (**GGTName**) ist ein Atom bestehend aus Zahlen und wird gebildet aus den beim Starten übergebenen Parametern:
 - die Praktikumsgruppe
 - die Teamnummer
 - die eindeutigen Nummer des momentanen ggT-Prozesses
 - die eindeutigen Nummer des Starters
2. Mit dem gebildeten Namen registriert sich der Prozess lokal auf der Erlang-Node (**register(<GGTName>, self())**) und beim Namensdienst (**{self(), rebind, <GGTName>, node()}**)
3. Der Prozess meldet sich beim Koordinator an (**{hello, <GGTName>}**).
4. Anschließend erhält er als Antwort vom Koordinator Informationen zu seinen Nachbarn (**{setneighbors, <LeftN>, <RightN>}**) und merkt sich diese.
5. Wird eine neue ggT-Berechnung vorbereitet, erhält der Prozess eine neue Zahl **Mi** vom Koordinator (**{setpm, <MiNeu>}**). Dies kann zu jeder Zeit passieren.
6. Sobald die Berechnung dann gestartet wird, erhalten mindestens zwei ggT-Prozesse eine Aufforderung vom Koordinator, um mit der Berechnung zu beginnen (**{calc, start}**). Sollte der aktuelle Prozess diese Nachricht erhalten, so muss er beim Koordinator den initialen Wert anfragen (**{getinit, self()}**). Die Antwort vom Koordinator erfolgt dann mittels **{sendY, <Y>}**-Nachricht.
7. Nun wartet der Prozess auf die Zusendung von einer neuen Zahl mittels **{sendY, <Y>}**-Nachricht. Wenn er eine solche erhält, wird der eigentliche ggT-Algorithmus ausgeführt:

```
if y < Mi then
    Mi := mod(Mi-1, y)+1;
fi
```
8. Sollte sich hierbei seine Zahl **Mi** geändert haben, werden folgende Dinge ausgeführt:
 - a. Mithilfe einer Anfrage an den Namensdienst (**{self(), twocast, tell, <MiNeu>}**) werden zufällig zwei ggT-Prozesse ausgewählt und über das geänderte **Mi** informiert. Dabei kümmert sich der Namensdienst auch um die Zustellung der Nachrichten mittels **{sendY, <MiNeu>}**-Nachricht.
 - b. Der Koordinator wird ebenfalls über die Änderung informiert. Dabei wird außerdem der Name des ggT-Prozesses, sowie die aktuelle Systemzeit übertragen (**{briefmi, {<GGTName>, <MiNeu>, <Zeit>}}**).
9. Sobald diese Berechnung durchgeführt wurde, wartet der Prozess die übergebene Arbeitszeit ab, um eine umfangreichere Berechnung zu simulieren (**timer:sleep(<Arbeitszeit>)**).
10. Neben den Schnittstellen für die eigentliche Berechnung, müssen außerdem die folgenden weiteren Schnittstellen vom ggT-Prozess bereitgestellt werden:
 - a. **{From, toggle}**: Wechselt das Flag zum Korrigieren. Ist dieses gesetzt, wird die Terminierungsabstimmung angepasst (siehe Punkt 11c).
 - b. **{From, tellmi}**: Sendet das aktuelle **Mi** per **{mi, <Mi>}**-Nachricht an den Prozess **From** zurück.
 - c. **{From, pingGGT}**: Sendet ein **{pongGGT, <GGTName>}** an **From** zurück, um anzugeben, dass dieser ggT-Prozess noch am Leben ist.

11. Sollte der Prozess beim Warten auf eine neue Nachricht (`{sendY, <Y>}` oder `{setpm, <MiNeu>}`) die übergebene Terminierungszeit überschreiten, so startet er die sogenannte Terminierungsabstimmung. Diese läuft ab wie folgt:
- Es wird eine Terminierungsanfrage (`{self(), {vote, <GGTName>, <Mi>}}`) an beide Nachbarn (`LeftN` und `RightN`) geschickt.
 - Als Empfängerprozess dieser Anfrage wird die übertragene Zahl (`MiIn`) eingelesen und mit dem eigenen `Mi` verglichen. Stimmen beide Werte überein, wird eine Zustimmung (`{voteYes, <GGTName>}`) an den anfragenden Prozess zurückgeschickt.
 - Ist das Flag zum Korrigieren gesetzt (siehe Punkt 10a), so wird der anfragende Prozess über das eigene `Mi` informiert, wenn dieses kleiner sein sollte, als das übertragene `MiIn` (`{sendY, <Mi>}`).
 - Erhält der anfragende Prozess von beiden Nachbarn diese Zustimmung, so gilt die Terminierungsabstimmung als positiv und der Koordinator wird informiert. Hierbei werden der Name des aktuellen Prozesses, der errechnete ggT (`Mi`), sowie die aktuelle Systemzeit mitgesendet (`{self(), briefterm, {<GGTName>, <Mi>, <Zeit>}}`).
 - Erhält der anfragende Prozess jedoch **Terminierungszeit** lang keine Antwort oder eine neue `{sendY, <Y>}`- oder `{setpm, <MiNeu>}`-Nachricht während er wartet, so wird die momentane Terminierungsabstimmung beendet.
12. Der ggT-Prozess zählt seine erfolgreich gemeldeten Terminierungsmeldungen und notiert diese in der angegebenen Log-Datei.
13. Der ggT-Prozess kann mithilfe einer **kill**-Nachricht vom Koordinator beendet werden. Sollte diese eintreffen, muss sich der Prozess per `{self(), {unbind, <GGTName>}}` beim Namendienst abmelden und dann selbst beenden.

Namensdienst

Der bereitgestellte Namensdienst ermöglicht es den anderen Komponenten, eine Verbindung zwischen sich aufzubauen. Er ist unter dem Prozessnamen **nameservice** auf einer eigenen Erlang-Node erreichbar, welche in den Konfigurationsdateien der anderen Komponenten angegeben werden muss.

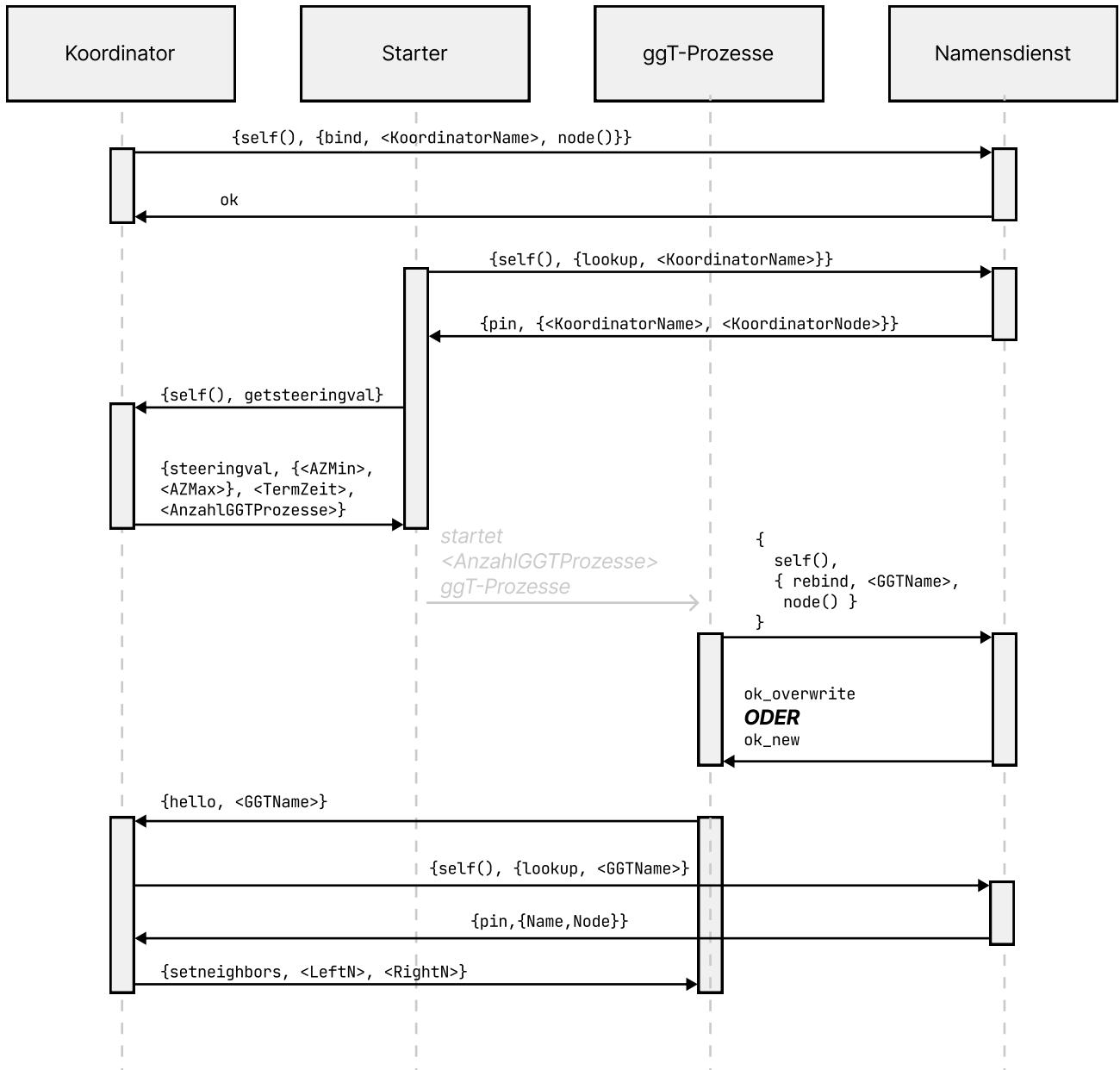
Zu diesem Zweck muss sich jeder Prozess zunächst beim Namensdienst mit **rebind** oder **bind** registrieren und vor seiner Beendigung mit **unbind** abmelden.

Außerdem informiert ein ggT-Prozess über den Namensdienst zwei weitere, zufällig ausgewählten ggT-Prozesse, indem er `{self(), {twocast, tell, <MiNeu>}}` an den Namensdienst schickt, welcher `{sendy, <Y>}` weiterschickt.

Ablauf/Interprozesskommunikation

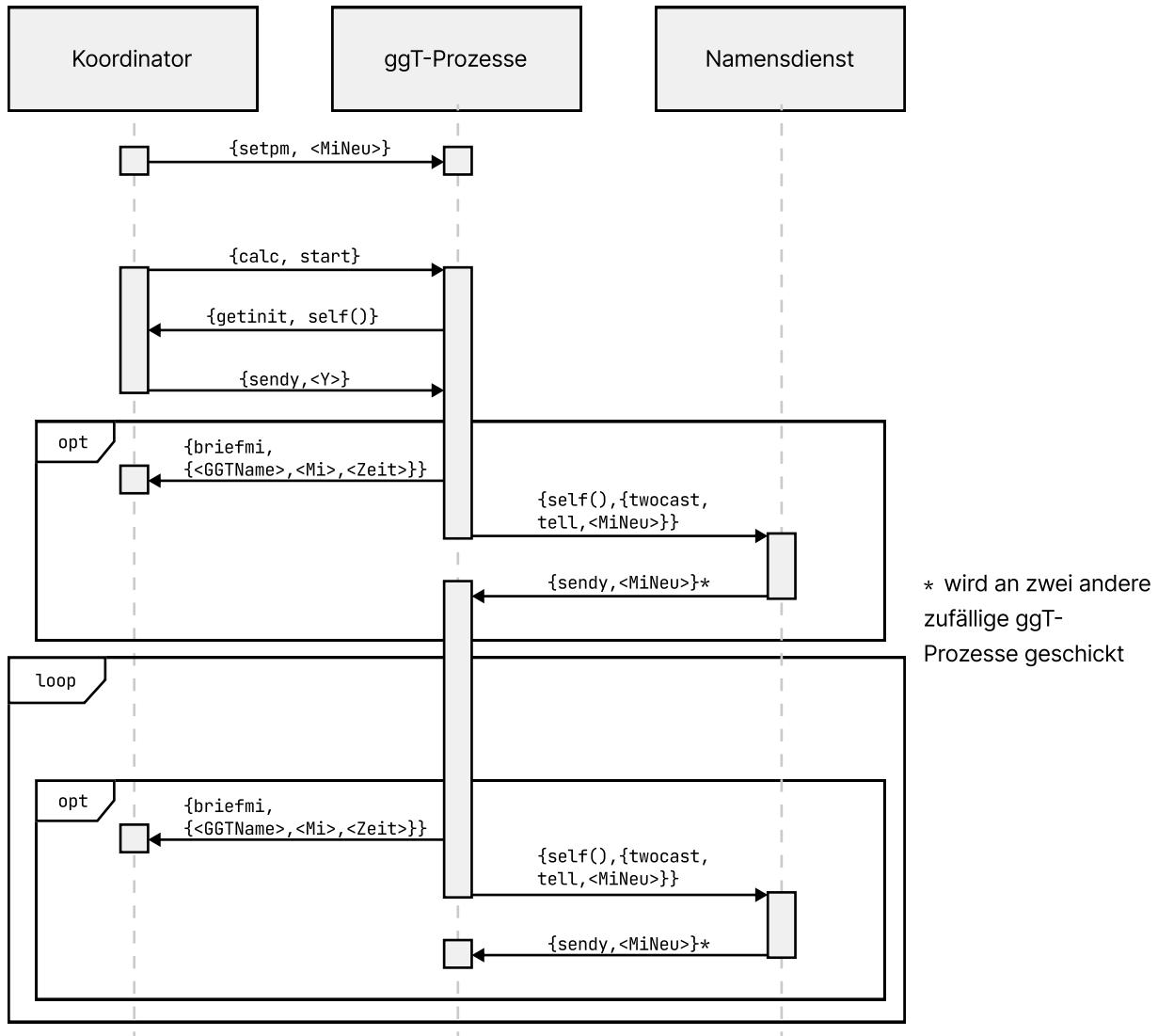
Der globale Ablauf des Systems wird in drei Phasen unterteilt. Anschließend ist dargestellt, wie die Prozesse in diesen Phasen miteinander interagieren und welche Nachrichten dabei versendet werden:

Initialisierungsphase

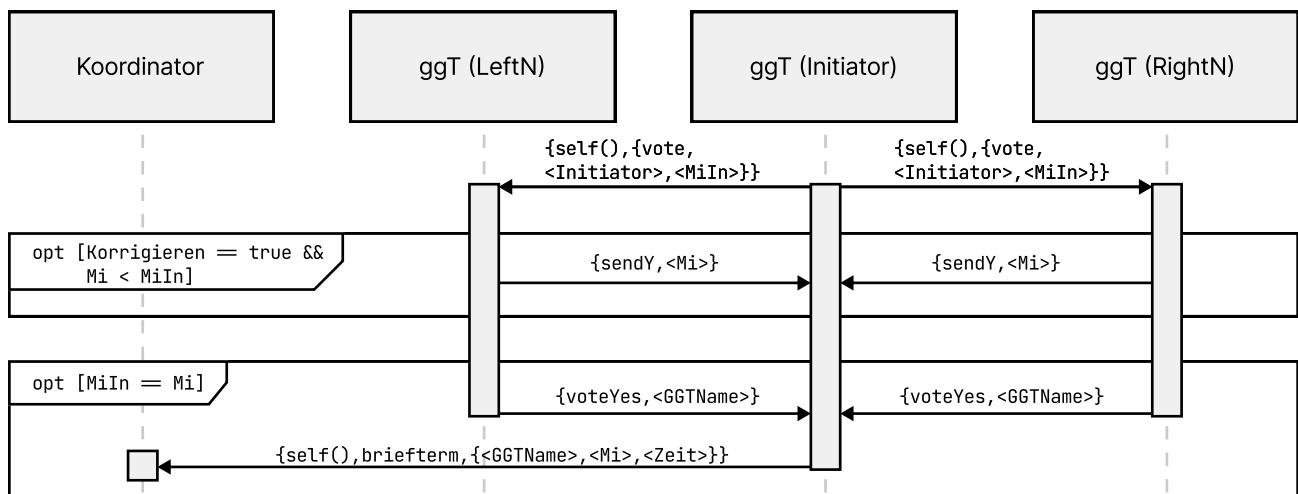


Arbeitsphase

Im Kern verläuft die Arbeitsphase wie folgt:



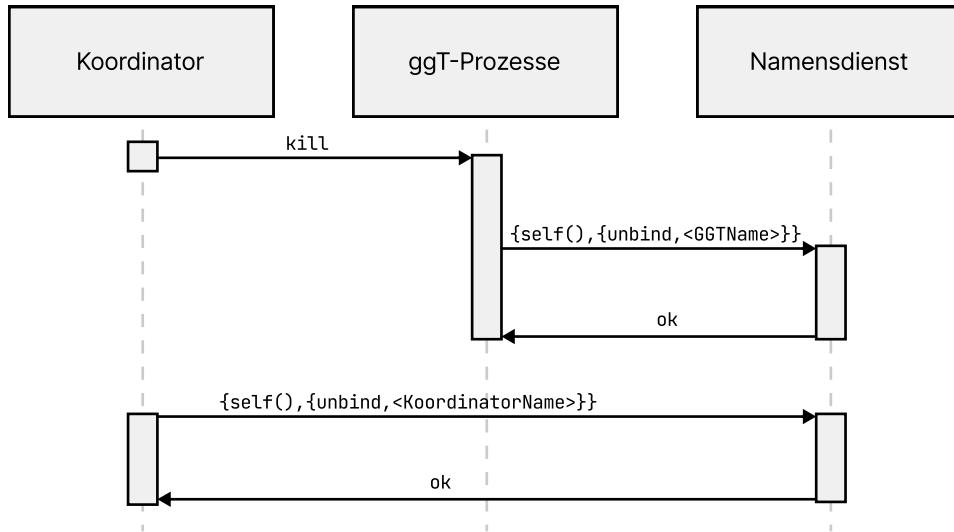
Beim Ablauf der Terminierungszeit des ggT-Prozesses, d.h. sollte er derweil keine neue Nachricht erhalten, wird die Terminierungsabstimmung mit folgendem Ablauf gestartet:



Die Terminierungsabstimmung wird abgebrochen, sollte in der Zwischenzeit ein neues **sendy/setpm** eintreffen oder sollte der Initiator nach erneutem Abwarten der Terminierungszeit kein **voteYes** von beiden Nachbarn erhalten haben.

Beendigungsphase

Die Beendigungsphase wird eingeleitet, sobald alle ggT-Prozesse erfolgreich ein Berechnungsergebnis gemeldet haben. Sie kann allerdings auch manuell durch den Nutzer ausgelöst werden (**kill**-Kommando im Koordinator).



Analyse

Parameter	Einfluss
arbeitszeit	Bei einer geringen Arbeitszeit kommt das System sehr schnell und mit geringer Fehlerwahrscheinlichkeit zu einem Ergebnis. Bei einer längeren Arbeitszeit dauert die Berechnung sehr viel länger, und es gibt eine höhrere Wahrscheinlichkeit, dass das System stehen bleibt, ohne auf ein Ergebnis zu kommen.
termzeit	Eine geringe Terminierungszeit führt zu vielen unnötigen Terminierungsabstimmungen, die zu keinem Ergebnis führen. Bei einer großen Terminierungszeit passiert dies sehr selten.
ggtprozessanzahl	Mit sehr wenigen ggT-Prozessen bleibt das System oft stehen, ohne ein Ergebnis berechnet zu haben. Mit zunehmender Prozessanzahl wird dieses Risiko immer geringer.
korrigieren (Koordinator)	Hier konnten wir keinen Einfluss feststellen.
korrigieren (ggT-Prozesse)	Ist das Korrigieren-Flag der ggT-Prozesse nicht gesetzt, kann es dazu kommen, dass ein ggT-Prozess in einer Endlosschleife bei der Terminierungsabstimmung landet. Ist das Flag gesetzt, passiert dies nicht.