**Team**: 05, Eugen Deutsch, Ralf von der Reith

**Aufgabenaufteilung**:

1. Ralf: Entwurf Initialisierungs- und Beendigungsphase, Koordinator – Koordinator Implementation
2. Eugen: Entwurf Arbeitsphase, GGT – GGT & Starter Implementation

**Quellenangaben**: -/-

**Bearbeitungszeitraum**:

* + - 1. 30.04.2017 – 3 Stunden
      2. 02.05.2017 – 4 Stunden
      3. 03.05.2017 – 4 Stunden
      4. 06.05.2017 – 2 Stunden
      5. 07.05.2017 – 6 Stunden
      6. 08.05.2017 – 4 Stunden
      7. 09.05.2017 – 5 Stunden
* 28 Stunden

**Aktueller Stand**:

* Der Entwurf ist fertig
* Die Implementation ist komplett

**Änderungen des Entwurfs**: -

# Koordinator

*Interne Datenstrukturen*

*Korrekturflag:*

Werte: true/false

Gibt Auskunft, ob bei inkorrekter Terminierungsmeldung eine Information darüber per sendy an den Sender zurückgeschickt werden soll

*Clients:*

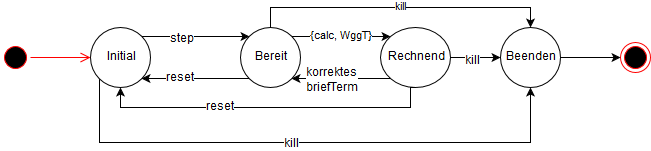
Liste von Tupeln: [Tupel1, Tupel2, ..., TupelN]

Tupelformat: {Clientname, CurrentMi}

Clientname: Name eines ggT-Prozesses, mit dem dieser beim Namensdienst registriert ist

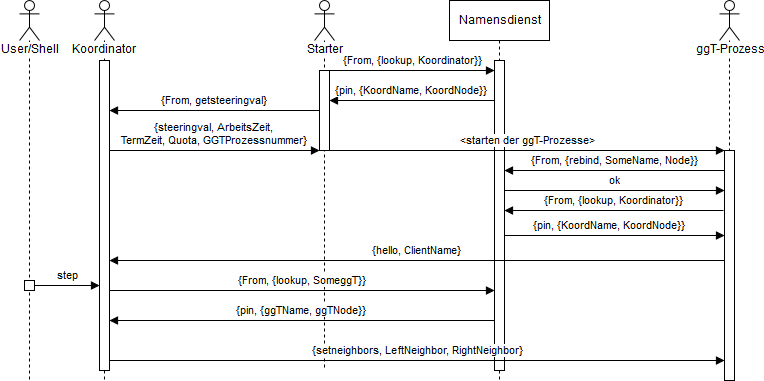
CurrentMi: Das von diesem Client zuletzt gemeldete Mi

*Verhalten/Interner Ablauf:*



Nach Start ist der Koordinator in der Initialisierungsphase. Er wartet auf Anfragen von Startern und antwortet ihnen mit den steuernden Werten (Schnittstellen::getsteeringval). Außerdem können sich in dieser Phase ggT-Prozesse bei dem Koordinator registrieren (Schnittstellen::hello).

Wird die Initialisierungsphase des Koordinators beendet (Schnittstellen::step), bildet er aus allen registrierten ggT-Prozessen einen zufällig angeordneten Ring und informiert die Prozesse über ihre Nachbarn. Nun ist der Koordinator in der Bereitschaftsphase.

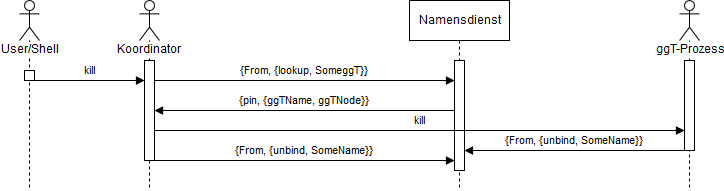


Nun kann eine Berechnung gestartet werden (Schnittstellen::calc). Der Koordinator geht in die Arbeitsphase über, berechnet einige Startwerte und gibt diese an die ggT-Prozesse weiter. Anschließend wählt er per Zufall 20% (und mindestens 2) der ggT-Prozesse aus und sendet ihnen ein ebenfalls zuvor berechnetes Mi zu, um so die Berechnung zu starten.

Der Koordinator erhält Informationen über neu berechnete Mis durch die ggT-Prozesse und schreibt diese in eine Datei. Nach einer erfolgreichen Terminierungsabstimmung durch die ggT-Prozesse wird der Koordinator ebenfalls darüber informiert und schreibt auch dies in eine Datei. Bei gesetztem Korrektur-Flag sendet er per sendy sein minimal bekanntes Mi, sofern es geringer ist als das bei Terminierung gemeldete Mi.

Zu jedem Zeitpunkt kann der Koordinator über "kill" in die Beendigungsphase versetzt werden.

In der Beendigungsphase sendet der Koordinator einen kill-Befehl an alle gemeldeten ggT-Prozesse und beendet sich anschließend selbst.



# GGT

Die ggts werden vom Starter gestartet (ein Starter mehrere ggts starten kann – config:ggtProzessnummer) und erhalten von diesem ihre config Werte (Arbeitszeit, termzeit, quota). Danach registrieren sie sich auf der Erlang Node und melden sich beim Namensservice (rebind) sowie beim Koordinator (hello) an, wobei sie vom ersteren über lookup an letzteren kommen.

# C:\Users\Steven\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Arbeitsphase.png

Nachdem der Koordinator die ggts mit Werten versorgt und 20 % per sendy zum Starten der Berechnung gebracht hat, wird nach folgendem Algorithmus vorgegangen:

{Eine Nachricht <y> ist eingetroffen}

If y < Mi then

Mi := mod(Mi – 1, y) + 1;

Send #Mi to all neighbours;

Fi

Für eine Berechnung braucht ein ggt eine bestimmte Zeit (config::ArbeitsZeit) und er wird seine Berechnung ausführen, sofern das erhaltene Y kleiner, als das aktuelle Mi ist und damit ein neues Mi errechnen. Dieses wird dann per sendy an den linken und rechten Nachbaren geschickt, woraus dann die Rekursion besteht und zusätzlich wird der Koordinator über briefterm informiert. Fertig ist dies sobald ein ggt-Prozess seit X Sekunden (aus config) kein neues Mi errechnet und somit den if Block nicht betritt und eine Abstimmung über den Nachrichtendienst startet (multicast). Die anderen ggt-Prozesse werden mit voteYes zustimmen, falls ihre letzte erhaltene Zahl X / 2 Sekunden her ist und ansonsten durchs Ignorieren ablehnen.

Hat der Initiator der Wahl genug Stimmen erhalten, sendet er dem Koordinator mit briefterm sein Mi zu, der das wiederrum mit seinem minimalen Mi abgleicht und gegebenenfalls (je nach Flag) mit sandy eingreift.

Falls der Initiator nicht genug Stimmen hat, startet er seine nächste Umfrage erst nachdem er zwischenzeitlich ein neues Y erhalten hat und wartet ansonsten auf Nachrichten.

Ansonsten kann ein ggt-Prozess Auskunft über sein Mi (tellmi) geben, oder ob dieser noch vorhanden ist (pingGGT). Über kill kann ein ggt-Prozess jederzeit beendet werden. Die Referenz zum Koordinator holt er sich vom Nachrichtendienst (lookup). Der Name eines ggts ist ein Atom, dass sich aus der Praktikumsgruppe (bei uns 1), der Teamnummer (5), der Nummer des ggt-Prozesses und der Nummer des Starters zusammensetzt.

# Schnittstellen (Koordinator)

### {From, getsteeringval}

**From**: PID des Senders

**Funktion**: Gibt die für den Starter relevanten Parameter zurück.

**Response**: {steeringval, **ArbeitsZeit**, **TermZeit**, **Quota**, **GGTProzessnummer**}

**ArbeitsZeit**: simulierte Verzögerungszeit für Berechnungen in Sekunden

**TermZeit**: Wartezeit bis zur Initiierung einer Terminierungs-Wahl in Sekunden

**Quota**: Absolute Anzahl benötiger Zustimmungen für erfolgreiche Terminierungs-Wahl

**GGTProzessnummer**: Anzahl der zu starteten ggT-Prozesse

### {hello, Clientname}

**Clientname**: Name des Clients, bei dem sich der Sender beim Namensdienst angemeldet hat

**Funktion**: Registriert den Absender über seinen Clientnamen als ggT-Prozess beim Koordinator.

### {briefme, {Clientname, CMi, CZeit}}

**Clientname**: Name des Senders, mit dem er beim Nameserver gemeldet ist

**CMi**: sein neues Mi

**CZeit**: Zeit, zu der die Berechnung des neuen Mis stattfand.

**Funktion**: Infomieren des Koordinators über sein neues Mi.

### {From, briefterm, {Clientname, CMi, CZeit}}

**From**: PID des Senders

**Clientname**: Name des Senders, mit dem er beim Nameserver gemeldet ist

**CMi**: das Mi, mit dem die Berechnung terminiert wurde

**CZeit**: Zeit, zu der die Berechnung terminiert wurde

**Funktion**: Informiert den Koordinator über die Terminierung eines ggT-Prozesses mit dem Ergebnis und dem Endzeitpunkt. Bei gesetztem Korrektur-Flag wird im Falle einer inkorrekten Terminierungsmeldung eine Antwort geschickt.

**Response**: (Nur bei gesetztem Korrektur-Flag!) {sendy, **Y**}

**Y**: aktuell bekanntes, minimales Mi

*Nachrichten zur Steuerung des Koordinators*

### reset

**Funktion**: Zurücksetzen des Koordinators in den Initialzustand. Außerdem wird allen zu diesem Zeitpunkt registrierten ggT-Prozessen das kill-Kommando gesendet.

### step

**Funktion**: Beendet die Initialphase des Koordinators und bildet einen Ring aus allen registrierten ggT-Prozessen. Wartet auf den Start einer ggT-Berechnung.

### prompt

**Funktion**: Fragt bei allen registrierten ggT-Prozessen das aktuelle Mi ab und schreibt es in eine Logfile.

### nudge

**Funktion**: Fragt den aktuellen Zustand aller registrierten ggT-Prozesse ab und schreibt diesen in eine Logfile

### toggle

**Funktion**: Verändert das Flag zur Korrektur bei falschen Terminierungsmeldungen.

wahr -> falsch

falsch -> wahr

### kill

**Funktion**: Beendet den Koordinator und sendet das kill-Kommando an alle ggt-Prozesse.

### {calc, Wggt}

**WggT**: Wunsch-ggT für die nächste Berechnung

**Funktion**: Veranlasst den Koordinator, eine neue Berechnung mit dem Wunsch-ggT zu starten.

# Schnittstellen (Starter)

### {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer}

**ArbeitsZeit**: simulierte Verzögerungszeit für Berechnungen in Sekunden

**TermZeit**: Wartezeit bis zur Initiierung einer Terminierungs-Wahl in Sekunden

**Quota**: Absolute Anzahl benötiger Zustimmungen für erfolgreiche Terminierungs-Wahl

**GGTProzessnummer**: Anzahl der zu starteten ggT-Prozesse

**Funktion**: Informiert den Starter über die steuernden Werte. Daraufhin startet der Starter die ggT-Prozesse und terminiert.

# Schnittstellen (GGT)

### {setneighbors, LeftN, RIghtN }

**LeftN**, **RightN**: Die Namen (Nicht PID) des rechten und linken Nachbaren.

**Funktion**: Registriert beim jeweiligen ggT-Prozess seinen rechten und linken Nachbaren, an die er später sein Ergebnis senden wird.

### {setpm, MiNeu}

**MiNeu**: Die neue Zahl zur Berechnung.

**Funktion**: Setzt die Zahl für den ggT-Prozess. Im Gegensatz zu {sendy, Y} wird hier explizit die Zahl gesetzt und nicht berechnet und ist somit zur Initialisierung gedacht.

### {sendy, Y}

**Y**: Der Y Wert (siehe Algorithmus).

**Funktion**: Setzt den Y Wert zur rekursiven Berechnung. Sofern Y kleiner ist, als das Mi des ggT-Prozesses, rechnet er ein neues Mi aus, setzt es bei sich und sendet es an seine Nachbaren als neues Y weiter.

### {From, {vote, Initiator}}

**From**: Die PID des Absenders.

**Initiator**: Der Name (nicht PID) des Absenders.

**Funktion**: Fragt beim ggT-Prozess danach, ob die Berechnung beendet werden sollte. Sofern dieser nun seit mindestens X/2 Sekunden keine neue Zahl zur Berechnung erhalten hat, sendet er {voteYes, Name} als Antwort (Name ist sein eigener Name, nicht die PID), oder ignoriert es anderenfalls.

### {voteYes, Name}

**Name**: Der Name des Absenders (nicht PID).

**Funktion**: Nachdem ein ggT-Prozess über multicast nach einer Abstimmung gefragt hat, erhält er von den anderen Prozessen eventuell diese Antwort. Sofern er genug bzgl. seiner Quote erhalten hat, schickt er dem Koordinator ein {briefterm, {Clientname, CMi, CZeit}}.

### {From, tellmi}

**From**: Die PID des Absenders.

**Funktion**: Sendet dem Absender (From – PID) das aktuelle Mi.

### {From, pingGGT}

**From**: Die PID des Absenders.

**Funktion**: Sendet dem Absender (From – PID) ein {pongGGT, GGTname} (GGTname ist keine PID), um so zu signalisieren, dass der ggT noch vorhanden ist.

### kill

**Funktion**: Der ggT-Prozess wird beendet, unabhängig seines momentanen Berechnungsstandes.