**Entwurf**

**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]

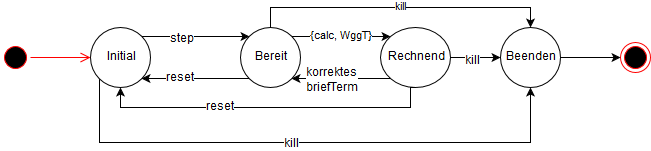
Tupelformat: {Clientname, CurrentMi}

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

CurrentMi: Das von diesem Client zuletzt gemeldete Mi

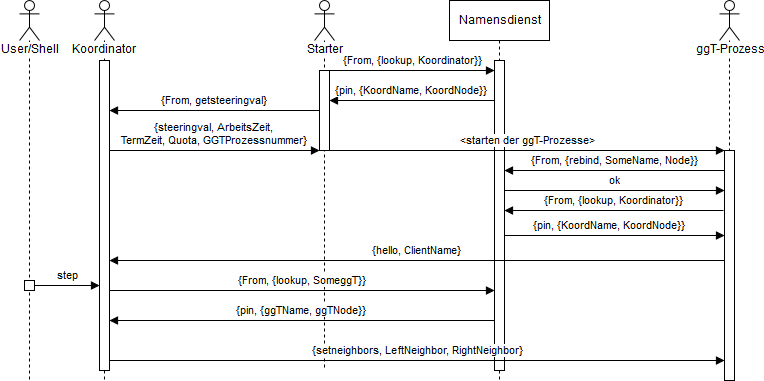
*Verhalten/Interner Ablauf:*

Der Koordinator arbeitet in mehreren Phasen:

* Initialisierungsphase (Zustand: Initial)
* Bereitschaftsphase (Zustand: Bereit)
* Arbeitsphase (Zustand: Rechnend)
* Beendigungsphase (Zustand: Beenden)

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.

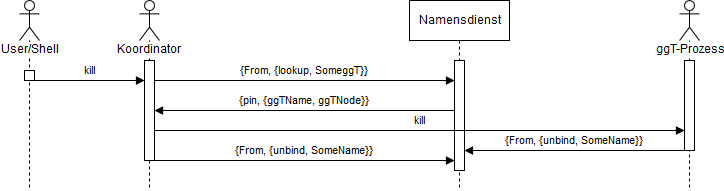


Nun kann eine Berechnung gestartet werden (Schnittstellen::calc). Der Koordinator geht in die Arbeitsphase über, berechnet einige Startwerte und gibt diese an die ggT-Prozesse weiter. Anschließend wählt er per Zufall 20% der ggT-Prozesse aus und sendet ihnen ein ebenfalls zuvor berechnetes Mi zu, um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



*Schnittstellen:*

*Nachrichten zur Kommunikation mit den anderen Prozessen.*

**getsteeringval**

Request

Format: {From, getsteeringval}

Parameter:

From: PID des Senders

Gibt die für Starter relevanten Parameter zurück.

Response

Format: {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer}

Parameter:

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**

Request

Format: {hello, Clientname}

Parameter:

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

registriert den Absender über seinen Clientnamen als ggT-Prozess beim Koordinator.

**briefme**

Request

Format: {briefme, {Clientname, CMi, CZeit}}

Parameter:

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.

Infomieren des Koordinators über sein neues Mi.

**briefterm**

Request

Format: {From, briefterm, {Clientname, CMi, CZeit}}

Parameter:

From: PID des Absenders

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

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

Bedingung: nur bei gesetztem Korrektur-Flag

Format: {sendy, Y}

Parameter:

Y: aktuell bekanntes, minimales Mi

---

*Nachrichten zur Steuerung des Koordinators*

**reset**

Request

Format: reset

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

**step**

Request

Format: step

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

**prompt**

Request

Format: prompt

Fragt bei allen registrierten ggT-Prozessen das aktuelle Mi ab und schreibt es in eine logfile

**nudge**

Request

Format: nudge

Fragt den aktuellen Zustand aller registrierten ggT-Prozesse ab und schreibt diesen in eine logfile

**toggle**

Request

Format: toggle

Verändert das Flag zur Korrektur bei falschen Terminierungsmeldungen.

wahr -> falsch

falsch -> wahr

**kill**

Request

Format: kill

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

**calc**

Request

Format: {calc, WggT}

Parameter:

WggT: Wunsch-ggT für die nächste Berechnung.

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

**Starter**

*Verhalten:*

Der Starter erfragt bei dem Namensdienst die PID des Koordinators. Anschließend erfragt er über getsteeringval (Koordinator::Schnittstellen::getsteeringval) die steuernden Werte und startet entsprechend dieser die ggT-Prozesse. Danach terminiert der Starter.

*Schnittstellen:*

**steeringval**

Request

Format: {steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer}

Parameter:

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.

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