Team: 9, David Asmuth, Vladimir Malikov

Aufgabenaufteilung:

1. Entwurf, Vladimir Malikov.
2. Entwurf, David Asmuth.

Quellenangaben: http://users.informatik.haw-hamburg.de/~klauck/VerteilteSysteme/aufg1-rmi.html

Begründung für Codeübernahme: Es wurde kein Code übernommen.

Bearbeitungszeitraum:

23.04.2015:	1 Stunde
27.04.2015	6 Stunden
29.04.2015	5 Stunden

Aktueller Stand: Entwurf fertig. Die Implementation noch nicht angefangen.

Änderungen im Entwurf: …

1. Einleitung

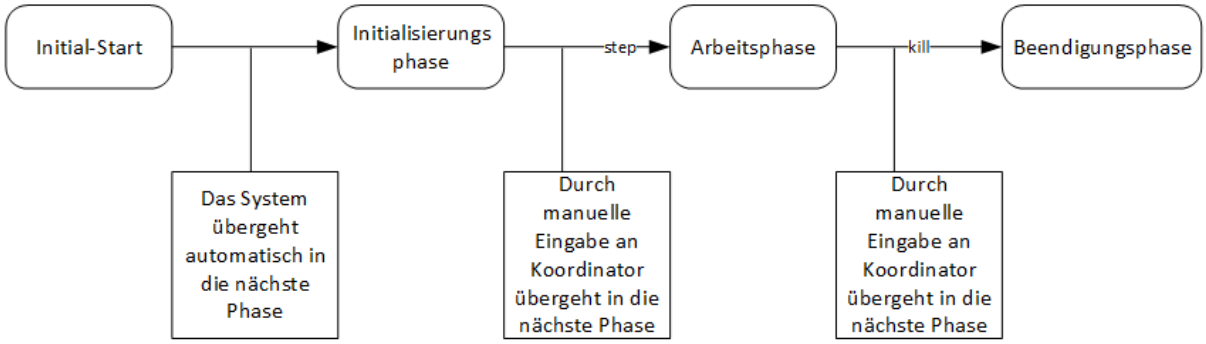
Ziel

Ziel ist es einen Verteilten Algorithmus und dessen Koordinierung zu implementieren. Als Algorithmus wird der angepasste Satz von Euklid verwendet.

2. Rahmenbedingungen

2.1 Der allgemeine Ablauf

Um einen Überblick zu bekommen ist im folgenden Diagramm der allgemeine Ablauf des ganzen Systems dargestellt. Hierbei ist anzumerken, dass der Koordinator, der Namensdienst, sowie die Starter und GGTs eigene Prozesse darstellen:



2.2 Verwendete Libraries

Es wird ausschließlich die Library „werkzeug.erl“ sowie „nameservice.beam“ von Professor Klauck verwendet.

3. Modularisierung

3.1 Koordinator-Komponente

3.1.1 Verantwortlichkeit

Der Koordinator ist dafür Zuständig die Starter und ggT Prozesse mit Initialisierungs Informationen zu versorgen, sowie die korrekte Beendigung sicher zu stellen. Er ordnet er die ggT-Prozesse als Ringsystem an, indem er ihnen die Nachbar-ggT-Prozesse mittteilt. Zudem führt er Buch über die vom Algorithmus erarbeiteten Werte.

3.1.2 Außensicht

Der Koordinator liefert auf Anfrage Initialwerte für die GT-Prozesse. Und liefert für ggT-Prozesse eine Möglichkeit die Mi Werte zu übermitteln.

Zusätzlich bietet der Koordinator noch die Befehle reset, step, promt, nudge, toggle und kill als Benutzereingabe an.

3.1.3 Innensicht

Der Koordinator Prozess besteht aus der koordinator.erl und koordinatorSteuerung.erl. Während die koordinator.erl die oben genannten Aufgaben erledigt, implementiert die

Steuerung die Möglichkeit die Benutzerbefehle aus zu führen. Die realisiert er indem er entsprechende Befehle an den koordinator.erl schickt.

3.1.4 Entwurfsentscheidungen

Die Benutzersteuerung ist separiert, da während des Receive-Loops keine Benutzereingaben möglich sind.

3.2 Starter-Komponente

3.2.1 Verantwortlichkeit

Übernimmt die Aufgabe als „Gegenstelle“ um auf einem (entfernten) Rechner ggT-Prozesse zu starten.

3.2.2 Außensicht

Der Starter bietet nach Außen keine Schnittstellen an, verwendet aber die des ggT Prozesses und des Koordinators.

3.2.3 Innensicht

Die Kompoente besteht nur aus der starter.erl. Diese holt beim start die Initialisierungswerte vom Koordinator und liest die Konfiguration für die ggt-Prozesse ein. Mit diesen Werten startet er anschließend den/die ggt-Prozesse

3.2.4 Entwurfsentscheidungen

Es wurde die Aufgabenstellung als Entwurfsentscheidung berücksichtigt.

3.3 ggT-Komponente

3.3.1 Verantwortlichkeit

Die ggt-Komponente führt den eigentlichen Algorithmus aus. Dabei Koordiniert er sich mit seinen Nachbar-Prozessen, indem er Ergebnisse von ihnen entgegen nimmt oder mitteilt. Führ seine Beendigung wird ein Voting mit allen weiteren ggt-Prozessen gestartet.

3.3.2 Außensicht

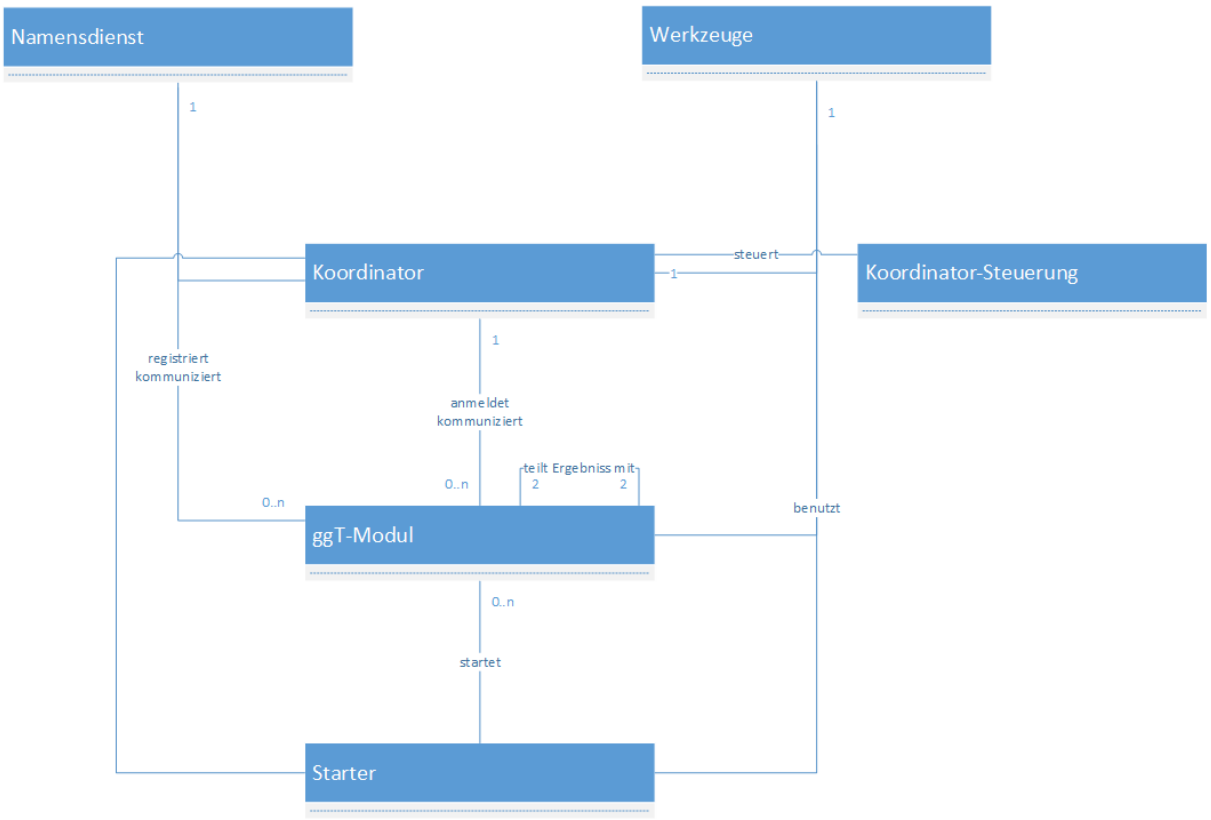
Die Komponente biete eine Schnittstelle für andere ggt-Prozesse an um Werte auszutauschen. Für den Koordinator eine Schnittstelle um den initialen Mi-Wert sowie Nachbarn zu setzen. Gibt auf Anfrage den aktuellen Mi aus und empfängt Voting-Ergebnisse. Zusätzlich antwortet die Komponente auf Ping anfragen mit pong.

3.3.3 Innensicht

Die Komponente besteht lediglich aus der ggT.erl und implementiert alle in der Außensicht angegebenen Schnittstellen/Services

3.3.4 Entwurfsentscheidungen

Es wurde die Aufgabenstellung als Entwurfsentscheidung berücksichtigt.



4 Klassenbeschreibung

Koordinator.erl

readConfig(File)

Definition

Liest Die benötigten steuernden Werte sind aus der Datei koordinator.cfg

Precondition

Config muss vorhanden sein.

Postcondition

Korrekt ausgelesene Konfiguration.

Return

{ nameservicenode, nameservicename, ggtprozessnummer, arbeitszeit, termzeit, koordinatorname, korrigieren, quote }

loopInitialize()

Definition

- In der Schleife für die Initialisierung wartet der Koordinator auf eingehende Anfragen von ggT- oder Starter-Prozessen, sowie Steuer-Kommandos vom koordinatorSteuerung.erl. Es werden ausschließlich Pakete aus der Initialisierungsphase bearbeitet.
- Erhält der Koordinator eine Anfrage vom Starter-Prozess: {From:,getsteeringval}, so antwortet der Koordinator mit den zuvor aus der

Config geladenen werten: { steeringval, ArbeitsZeit, TermZeit, Quota, GGTProzessnummer }

- Registriert dich ein ggT mit {hello,Clientname}, so antwortet der Koordinator nicht, sondern merkt sich den ggT in einer Liste.
- Bei dem Steuer-Kommando step beendet der Koordinator die Schleife und geht in den loopReady() über. toggle Ändert das Flag zur Behandlung von zu niedrigen Mi Ergebnissen.

Precondition

Eine korrekt geladene Config.

Postcondition

Alle Starter wurden gestartet, alle ggTs haben sich beim Koordinator angemeldet (hello).

Return

Die Liste aller teilnehmenden ggT-Prozessen.

loopReady()

Definition

- In der Schleife für den eigentlichen Bearbeitungsprozess wartet der Server auf Mi-Aktualisierung der ggT-Prozesse.
- Aktualisiert ein ggT seinen Mi mit der Nachricht {briefmi,{Clientname,CMi,CZeit}} so aktualisiert der Koordinator seinen Eintrag für diesen ggT-Prozess.
- Ist eine Abstimmung unter den ggTs erfolgreich, so erhält der Koordinator ein: {From,briefterm,{Clientname,CMi,CZeit}}.
  Der Koordinator überprüft nun ob dieses CMi, das kleinste Mi aller bekannten Mi‘s ist. Ist die CMi kleiner oder gleich, ist die Berechnung erfolgreich und der loop wird verlassen. Ist die CMi größer als eine gespeicherte Mi, so ist wird je nach toggle-Flag entweder eine Fehlermeldung geloggt oder dem ggT-Prozess wird ein {sendy,Y} gesendet. Wobei Y die kleinste bekannte Mi ist.

Das Steuerkommando…

- reset sendet ann alle den kill Befehl und beendet die Schleife und kehrt zurück zur loopInitialize()
- prompt: Geht durch die Liste aller angemeldeter ggT-Prozesse und fragt alle mit {From,tellmi} nach der aktuellen Mi; Antwort Form von: {mi,Mi}. Die Antworten werden in das Log geschrieben.
- Nudge: Geht durch die Liste aller registrierten ggT-Prozesse und sendet ein Ping an diese: {From,pingGGT}. Es wird geloggt von welchen eine/keine Antwort erhalten wurde ({pongGGT,GGTname})
- {calc,WggT} : Erzeugt Mi‘s mittels werkzeug:bestimme_mis/2 und versendet sie an alle registrierten ggT-Prozesse mittels {setpm,MiNeu}

- kill: geht durch die Liste aller registrierten ggT-Prozesse und sendet ein kill. Anschließend verlässt er die Schleife und beendet damit sich selber.

Precondition

Alle ggTs haben sich beim Namensdienst registriert, sowie bei Koordinator gemeldet (Liste alle teilnehmenden ggT-Prozesse).

Postcondition

Mi-Wert wurde berechnet.

Return

Mi-Wert.

ggT.erl

start()

Definition

- Registriert sich an der Erlang-Node(register(Name)).
- Registriert sich beim Namensdienst({From,{rebind,meindienst,node()}).
- Meldet sich beim Koordinator({hello,Clientname}).
- Wartet auf die Setzung der Namchbarn ({setneighbors,LeftN,RightN}).
- Wartet auf initialen Mi-Wert({setpm,MiNeu}).

Precondition

Alle benötigten Start-Parameter wurden übergeben.

Postconditon

Nachbarn wurden gesetzt.
Mi-Wert wurde gesetzt.

Return

Void

loop()

Definition

- Der ggT-Prozess erwartet {sendy,Y}, wobei Y der Mi-Wert von anderem Prozess ist. Erhält er diesen, so wird die Berechnung ausgeführt(calc(Y,Mi)). Entsprechend dem Ergebnis wird eine Nachricht an die Nachbarn gesendet ({sendy,Y}) und der Koordinator informiert({briefmi,{Clientname,CMi,CZeit}}). Des Weiteren wird bei jeden sendy und setpm der Timeout zurückgesetzt.
- Um die Berechnung zu simulieren wird nach der Erhalt von sendy eine bestimmte Zeit abgewartet.
- Erhält der ggT-Prozess eine {From,{vote,Initiator}} guckt er nach ob die hälfte des Timeout-Zeit vergangen ist. Ist das der Fall, antwortet er mit {voteYes, Name}. Sonst nichts.
- Kommt es zum Timeout wird die Methode vote() aufgerufen. Der ggT-Prozess übergeht in den Vote-Modus.

Precondition

Die Methode start() wurde erfolgreich ausgeführt.

Postcondition

Mi wurde berechnet.

Return

Void

vote()

Definition

- Startet über multicast ein Voting({From,{multicast,vote,meinname}}).
- Wartet auf die Antworten({voteYes,Name}).
- Wird die benötigte Quote erreicht, benachrichtigt den Koordinator({From,briefterm,{Clientname,CMi,CZeit}}).

Precondition

Der Timer ist abgelaufen.

Postcondition

Der Koordinator wurde beim erfolgreichen Voting benachrichtigt.

Return

Void

kill()

Definition

- Jeder Zeit kann ein laufender ggT-Prozess ein kill-Nachricht kriegt. Ist das geschehen, wird diese Methode aufgerufen.
- Der ggT-Prozess registriert sich von Namensdienst({From,{unbind,meindienst}}) und Erlang-Node(unregister(Name)) ab.

Precondition

kill erhalten

Postcondition

Programm wird beendet.

Return

void

starter.erl

start()

Definition

- Erfragt bei Koordinator die Steuerndewerte({From,getsteeringval}) und wartet auf die Antwort vom Koordinator({steeringval,ArbeitsZeit,TermZeit,Quota,GGTProzessnummer}). Liest restliche Werte aus der ggt.cfg aus.

- Startet die vorgegebene Anzahl an ggT-Prozessen.

Precondition

Koordinator wurde gestartet.

Postcondition

Koordinator-Prozess wurde korrekt gestartet.

Return

Void

koordinatorSteuerung.erl

startCC()

Definition

Liest die Benutzer Eingaben von der Konsole und leitet diese an Koordinator weiter.

Precondition

Koordinator-Prozess wurde gestartet.

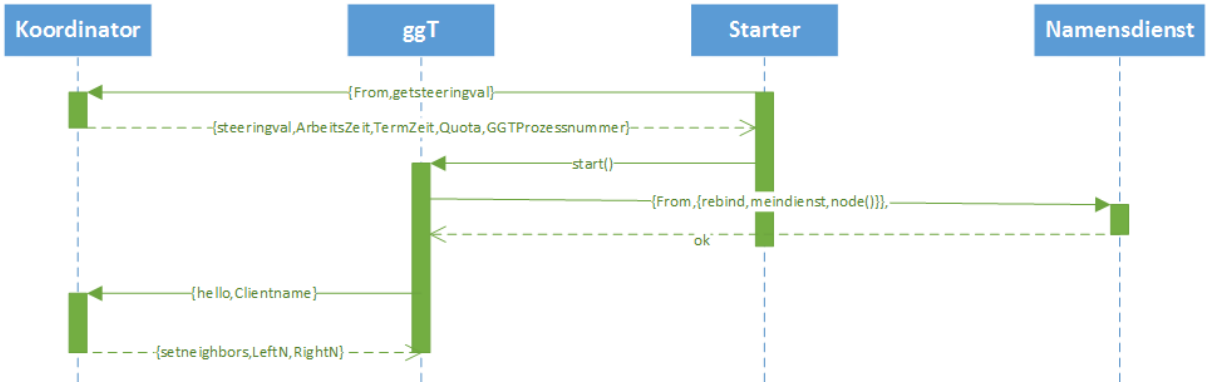
Postcondition

Return

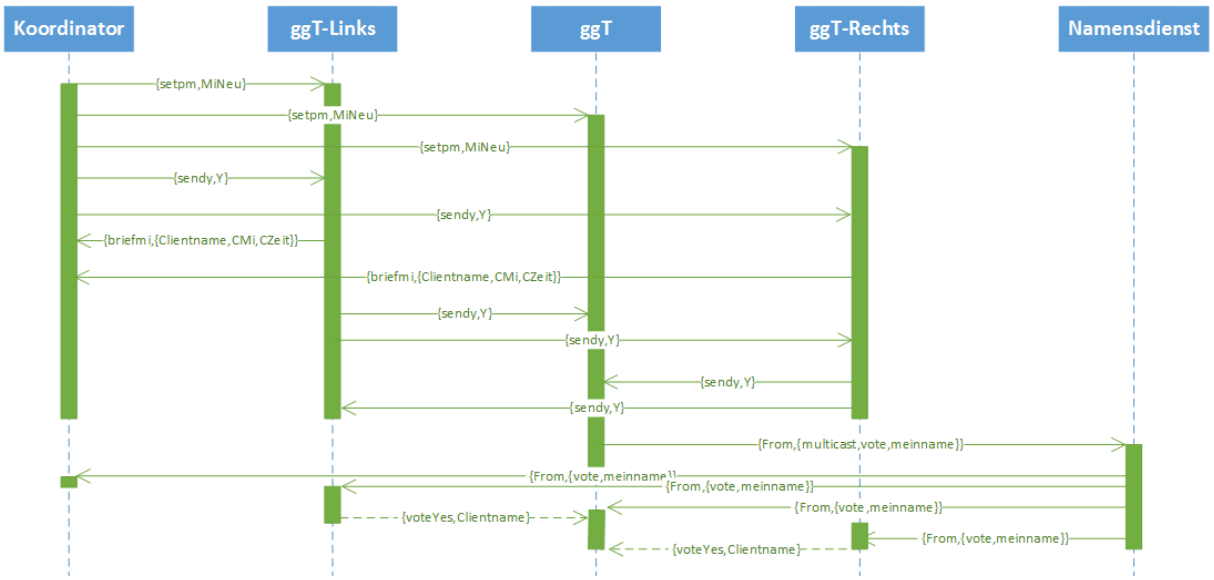
Void

Sequenzdiagramme

Initialphase



Arbeitsphase



Beendigungsphase