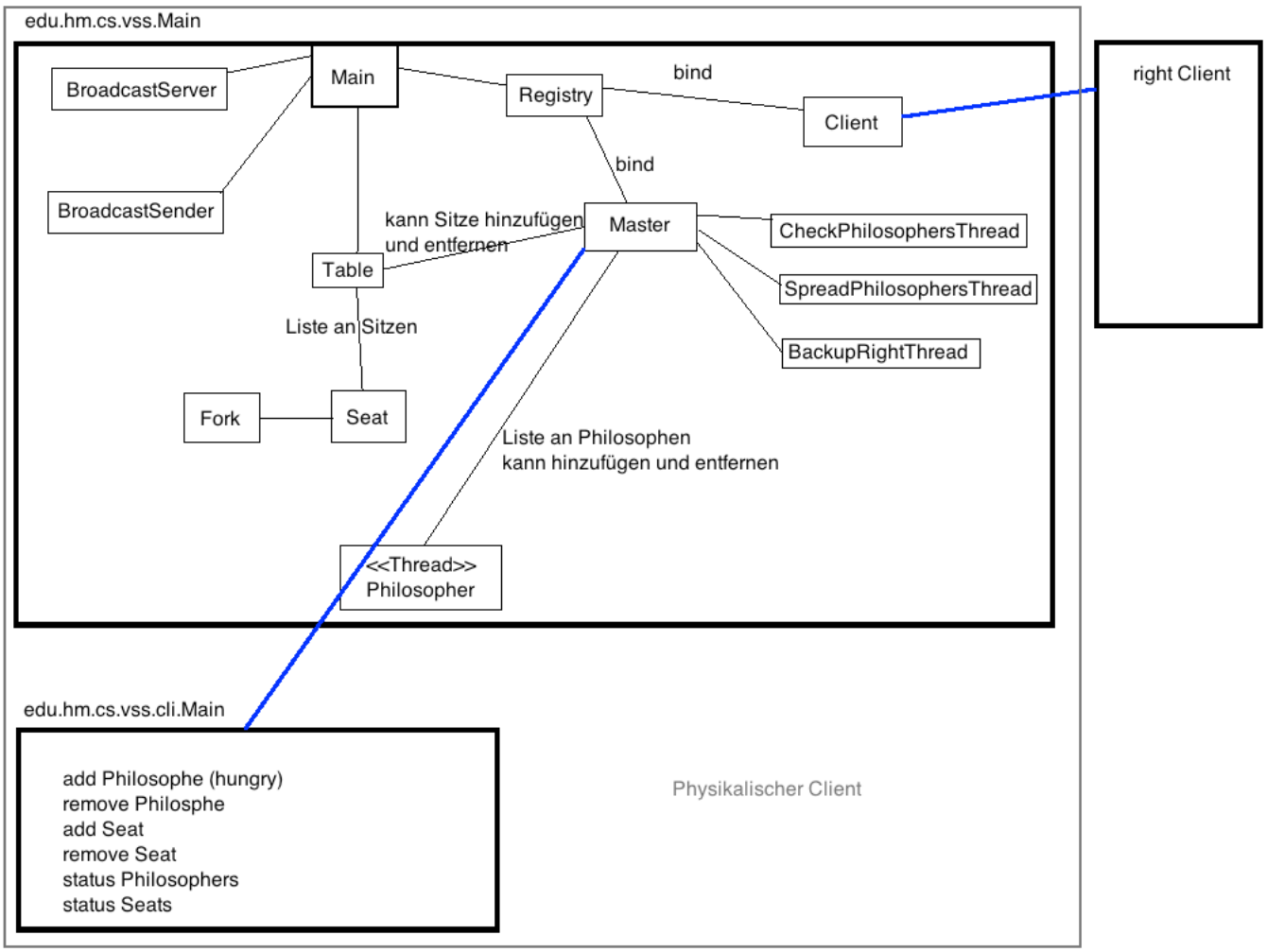
Verteilte Philosophen

von Elvira Hauer, Michael Freiwald

Wir entschieden uns für eine komplett Dezentrale Lösung.

Unsere Clients mit den Tischteilstücken sind in einem Ring angeordnet.
Für den regulären Ablauf des Programms benötigen ein Client nur seinen direkten linken und rechten Nachbarn. (Falls ein Client ausfällt kennt er seinen übernächsten linken sowie rechten Nachbarn.) Jeder Client muss greift aber immer nur auf seinen rechten Partner zu.

Zur Anmeldung in unserem System sendet ein Client, sobald er mit dem Aufbau einer Infrastruktur (erzeugen von Master zur Tisch verwalten sowie erzeugen der Plätze und Gabeln) fertig ist, drei Broadcast-Signale an einen festgelegten Port.
Wenn er in einem bestimmten Zeitraum eine Antwort erhält beginnt das eingliedern in das System.



Broadcast-Server

Stellt einen UDP-Server zur Verfügung, welcher auf Anfragen von Clients wartet, die sich mit ins Netzwerk einbinden wollen.
Sobald ein Packet empfangen wurde, wird geprüft ob der Request von einem anderen Client gekommen ist, um auszuschließen, dass der Server nicht auf die Anfrage seines eigenen BroadcastSenders antwortet.
Anschließend schickt der Server einfach ein Packet zurück, womit bestätigt wird, dass der anfragende Client eine Verbindung aufbauen darf.

Broadcast-Sender
Der BroadcastSender verschickt eine Broadcast-Nachricht mit der UUID des Clients und wartet anschließend eine bestimmte Zeit auf Antworten und sammelt alle ankommenden Nachrichten.
Nach dem die Zeit abgelaufen ist, gibt er eine Liste aller Clients zurück, die geantwortet haben, damit mit diesen Probiert werden kann, eine Verbindung einzugehen.

Registry
Stellt bindet den Client und Master in die lokale Registry

Client
Der Client ist unsere Verbindung nach Außen zu allen anderen Clients.

Dieser Versucht eine Verbindung mit anderen Clients einzugehen, damit er in ein vorhandenes Netzwerk integriert wird.

Immer wenn von einer Komponente auf den lokalen Client die Methode getRight() aufgerufen wird, überprüft dieser, ob der rechte Partner noch aktiv ist und gibt diesen zurück. Sollte dieser nicht mehr aktiv sein, wird die Ausfallbehandlung abgearbeitet. Neuer rechter Client wird der übernächster Rechte, anschließend dürfen alle alle angebundenen Clients (Links1, Links2, Rechts1, Rechts2) wieder ihre Nachbarn suchen. Nach dem dies abgeschlossen ist, wird das Backup wieder eingespielt.

Master
Der Master ist für das Erzeugen der Philosophen, sowie zusätzliche Sitzplätze zuständig. Außerdem Sorgt er mit einem extra Thread dafür, dass sein Rechter-PartnerClient gebackupt wird um einen möglichen Ausfall auszugleichen. Er ist auch dafür zuständig, falls dieser Client ausfällt die Philosophen in ihrem letzten Zustand zu sichern.

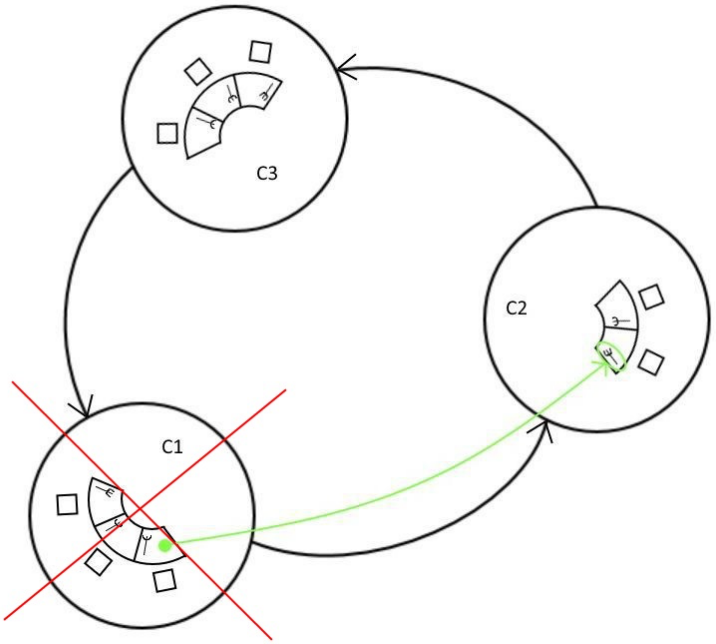
Erklärung Backup BackupRightThread
Für den Fall das ein Client ausfällt legt ein Client immer ein Backup von seinem Rechten Partner an um die Philosophen und die Tische im Fall eines Ausfalls wieder herstellen zu können. Das Backup beinhaltet die Anzahl der Essvorgänge der Philosophen.

Falls ein Client ausgefallen ist wird dies von seinem linken Nachbarn bemerkt, sobald eine Komponente auf den rechten Client zugreifen möchte. Ist dies der Fall schließt der

Client den Ring wieder indem er eine direkte Verbindung mit seinem Übernächsten Nachbarn eingeht und stellt mit Hilfe des Backups, dass er in regelmäßigen Abständen anlegt, die verlorenen Philosophen wieder her.

Ausfallproblem erläutern und Lösung:

Hierbei trat ein Problem auf, welches mit beachtet werden musste.
Wird von C1 per Remote auf die Gabel von C2 zugegriffen und C1 fällt aus, würde die Gabel nie wieder freigegeben werden. Hierfür speichern wir uns beim nehmen der Gabel, ob diese lokal oder remote genommen wurde. Wurde sie remote genommen und wir erfahren, das unser linker Nachbar ausgefallen ist, so erstellen wir einfach eine neue Gabel auf dem Platz.



Verteilung der Philosophen im System - SpreadPhilosopherThread

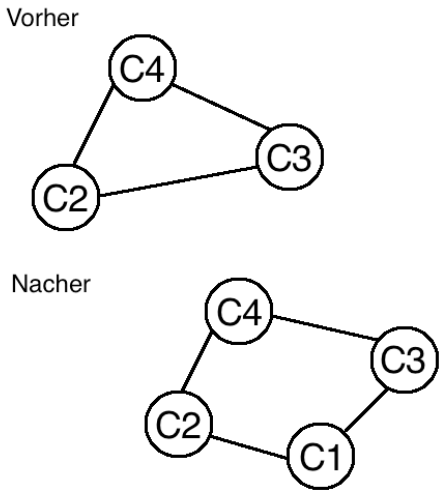
Jeder Client prüft die seine aktuelle Anzahl an Philosophen mit der seines rechten Partners. Außerdem vergleicht er seine Anzahl an CPUs mit der des rechten Partners. Anhand dieser Kennzahlen kann er ausrechen, wie viel Philosophen bei ihm und wie viel bei seinem Partner laufen sollten.
Laufen auf dem Client selbst zu viele Philosophen, exportiert er die bestimmte Anzahl und Import diese bei seinem rechten Partner.
Laufen auf dem Client selbst zu wenige Philosophen, so senden er dem rechten Partner einen export-Befehl, welcher die übergebenen Anzahl an Philosophen stop und sie dem linken Partner schickt. Dieser kann diese bei sich anschließend importieren.

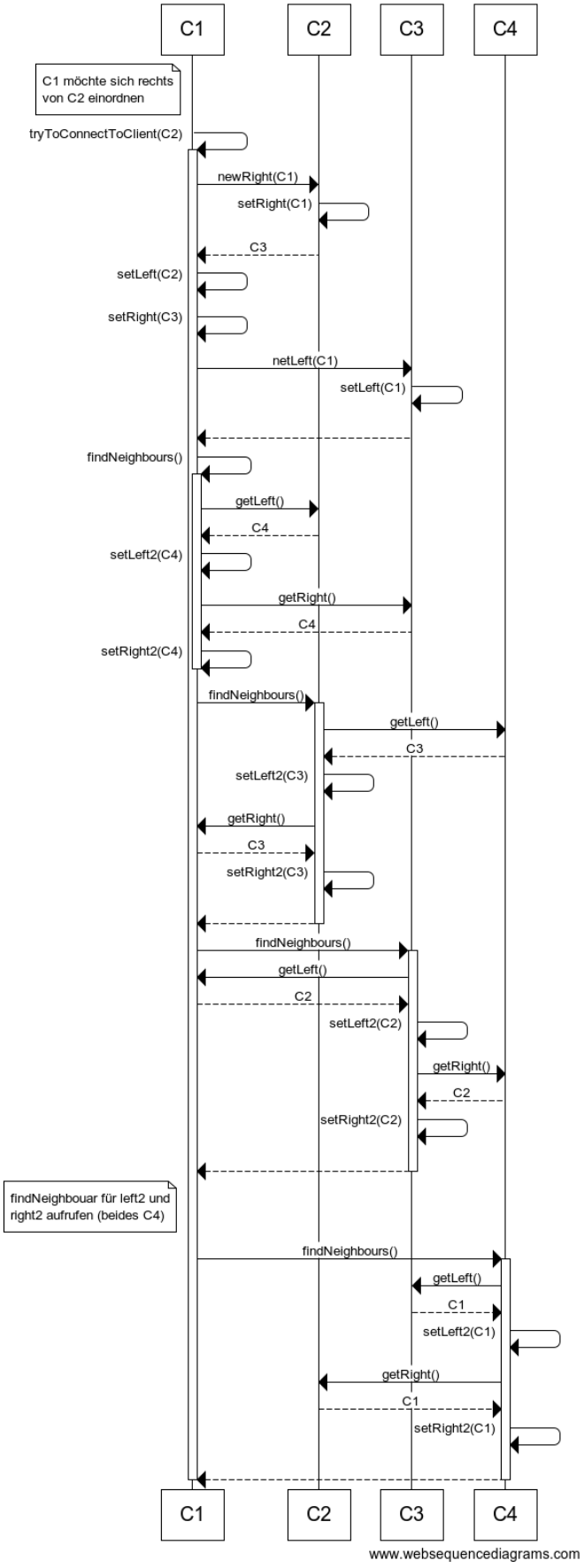
Verhindern, dass Philosophen ausarten - CheckPhilosophersThread

Der Thread frägt in einem festgelegten Intervall den globalen Durchschnitt an Essvorgänge pro Philosoph ab und durchläuft anschließend alle lokalen Philosophen. Sobald ein Philosoph mehr gegessen hat als der globale Durchschnitt + DELTA (festgelegter Wert) wird er aufgefordert zu warten.
Alle Philosophen die warten müssen, werden erst fortgesetzt, wenn ihr Durchschnitt kleiner gleich dem globalen Durchschnitt entsprechen.

Verbindung eines Clients in ein vorhandenes Netzwerk
Hat ein Client eine Antwort bekommen um sich in ein vorhandenes Netzwerk einzugliedern, so frägt er bei dem Client der geantwortet an, ob er sein neuer rechter Partner werden darf.

In der Abbildung ist zu sehen, wie das Netzwerk zuvor war, und wie es nach der Eingliederung auszusehen hat.




C1
C2
C3
C4
C1 möchte sich rechts von C2 einordnen
tryToConnectToClient(C2)
newRight(C1)
setRight(C1)
C3
setLeft(C2)
setRight(C3)
netLeft(C1)
setLeft(C1)
findNeighbours()
getLeft()
C4
setLeft2(C4)
getRight()
C4
setRight2(C4)
findNeighbours()
getLeft()
C3
setLeft2(C3)
getRight()
C3
setRight2(C3)
findNeighbours()
getLeft()
C2
setLeft2(C2)
getRight()
C2
setRight2(C2)
findNeighbouar für left2 und right2 aufrufen (beides C4)
findNeighbours()
getLeft()
C1
setLeft2(C1)
getRight()
C1
setRight2(C1)
C1
C2
C3
C4
www.websequencediagrams.com