Marlin Zapp 13.07.2024

Verteilte Systeme – Übung 2

Umsetzung

Ich habe die Aktoren und den Observer mittels `extends Node` implementiert. Die Simulation hat diese Knoten instanziiert und damit gestartet. In der Simulation wurde die Instanz des `Simulator` genutzt, um die Simulation sauber zu terminieren, nachdem vom Observer erkannt wurde, dass keine Firework-Nachrichten mehr im Umlauf sind. Die Anzahl der Knoten kann per Kommandozeilen-Parameter übergeben werden und die Wahrscheinlichkeit zum Aufwecken eines Aktors nach einer Firework-Nachricht wird immer auf p=p/2 reduziert.

Erfahrungen im Umgang mit dem Simulator

Das Importieren der uber-jar hat mehr Zeit gebraucht als nötig, da ich so etwas bisher noch nicht gemacht hatte. In Eclipse darf man anscheinend nicht die Datei `module-info.java` zur Verwaltung der Java-Modules nutzen, wenn man jar-Dateien als Bibliothek nutzen möchte, was erst zu Problemen geführt hat.

Die Klasse `Node` war einfach zu benutzen und zu verstehen, genau wie die Standard-Funktionen `send()` und `receive()`. Die `Simulator`-Instanz habe ich zuerst nicht verwendet, wodurch die Simulation zwar einwandfrei funktioniert hat, aber nicht sauber beendet wurde.

Daraufhin habe ich die `Simulator`-Instanz genutzt und den Start der Knoten nicht mehr wie vorher im Konstruktor erledigt, sondern wie gewollt in die Methode `engage()` ausgelagert. Anschließend wurde das Programm wie gewollt terminiert.

Ergebnisse

Die Aufgabe 2, also das Versenden von Firework-Nachrichten hat problemlos funktioniert. Genutzt habe ich dafür die `send()`-Methode und eine `Message` mit dem Payload-Schlüssel `type` und dem Wert `firework`. Die Wahrscheinlichkeit für ein weiteres Firework wurde pro ankommender Firework-Nachricht halbiert.

In Aufgabe 3 hat das Implementieren des Observers auch kaum Probleme gemacht. Hier habe ich vom Observer im Sekundentakt Nachrichten an alle anderen Knoten geschickt mit dem `type status`. Daraufhin antworten die Knoten dem Observer damit, wie viele Nachrichten sie seit der letzten Status-Anfrage an welche Knoten versendet haben. Das ist genau der Algorithmus aus der Vorlesung und dieser hat gut funktioniert. Man muss nur gut aufpassen, an welchen Stellen Nachrichten als versendet und als Empfangen gewertet werden, damit die Terminierung die richtige Rückmeldung gibt. Die Nachricht darf nämlich erst nach der `sleep()`-Zeit des Aktors als Empfangen gelten, ansonsten können noch neue Fireworks nach dem Schlafen des Threads starten.

Bis zu 1000 Aktoren sind kein Problem, aber ab 10000 Knoten kommt es zu Heap-Problemen aufgrund der Vielzahl an Nachrichten. Auch wenn die Nachrichtenweiterleitung um einen Faktor 10 reduziert wird, läuft der Heap über, auch wenn es eine Weile dauert.

Zwar sind viele zufällige Faktoren in dem Programm, die die Laufzeit beeinflussen, aber trotzdem ist hier im Folgenden noch eine Tabelle angegeben, wie lange das Programm in einem Beispiel gelaufen ist, während die Nachrichten-Weiterleitungs-Wahrscheinlichkeit sich jedes Mal halbiert hat.

Anzahl Aktoren	Zeit in Millisekunden
1	`6075`
10	`14097`
100	`10174`
1000	`32439`
10000	Java Heap Space Exception