DIPLOMARBEIT

Objektorientierte Entwicklung eines GUI-basierten Tools für die Simulation ereignisbasierter verteilter Systeme

Durchgeführt an der

Fachhochschule Aachen

mit Erstprüfer und Betreuer Prof. Dr.-Ing. Martin Oßmann

und Zweitprüfer Prof. Dr. rer. nat. Heinrich Fassbender

durch

Cand. Inform. (FH) Paul C. Bütow

Matthiashofstr. 15

D-52064 Aachen

Aachen, den 21. Juni 2008

Danksagungen

Ohne die Hilfe bestimmter Personen wäre die Anfertigung dieser Diplomarbeit viel schwieriger gewesen. Daher möchte ich mich bei den Folgenden bedanken:

- Martin Oßmann für die Betreuung der Diplomarbeit und der Bereitstellung des für mich sehr interessanten Themas
- Andre Herbst, für das Testen des Simulators; durch seine Hilfe wurden viele Mängel und Bugs aufgedeckt
- Meinem Bruder Florian Bütow, für Tipps und Tricks rund um Java, für die Bereitstellung eines Buches sowie für das Testen des Simulators
- Meinen Eltern Jörn und Leslie Bütow, die mir das Studium ermöglichten und stets für alle Dinge ein offenes Ohr hatten sowie für das Sponsoring eines weiteren Buches

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1.

Einleitung

1.1. Was ist ein Verteiltes System?

1.2. Motivation

[Bri03] Zitat.

Kapitel 2.

Grundbegriffe

Für das Verständnis wie die Simulation von verteilten Systemen funktioniert, werden hier einige Grundbegriffe erläutert. Eine Vertiefung findet erst in den nachfolgenden Kapiteln statt.

2.1. Client/Server Modell

Der Simulator basiert auf dem Client/Server Prinzip. Bei jeder sinnvollen Simulation gibt es mindestens einen teilnehmenden Client und einen Server, die miteinander über Nachrichten (s.u.) kommunizieren (Abbildung 2.1). Bei komplexen Simulationen können auch mehrere Clients und/oder Server mitwirken. In der Regel empfangen Server nur Nachrichten, die von Clients verschickt wurden und virce versa.

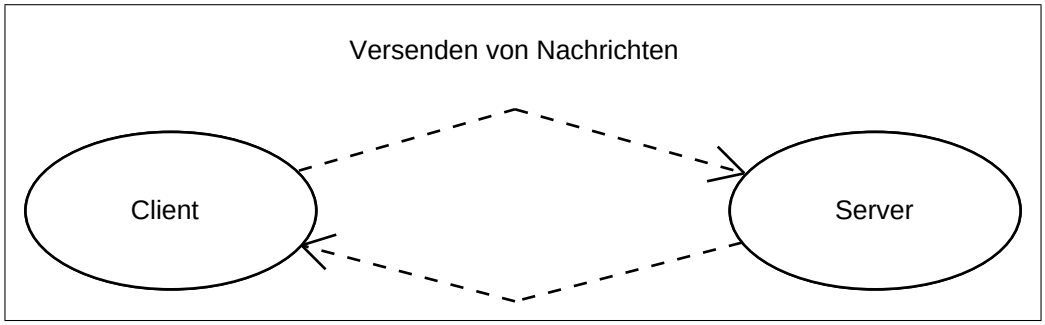

Abbildung 2.1.: Client/Server Modell

2.2. Prozesse und deren Rollen

Ein verteiltes System wird anhand von Prozessen simuliert. Jeder Prozess nimmt hierbei eine oder mehrere Rollen ein. Beispielsweise kann ein Prozess die Rolle eines Clients einnehmen und ein weiterer Prozess die Rolle eines Servers. Ein Prozess kann auch Client und Server gleichzeitig sein. Es ist auch möglich, dass ein Prozess die Rollen mehrerer Server und Clients aufeinmal einnimmt. Ob das sinnvoll ist hängt vom Szenario ab. Um einen Prozess zu kennzeichnen besitzt jeder Prozess eine **eindeutige** Prozess-Identifikationsnummer (PID).

2.3. Nachrichten

Damit das Client/Server Modell angewandt werden kann, müssen Nachrichten verschickt werden können. Eine Nachricht kann von einem Client- oder Serverprozess verschickt werden und kann beliebig viele Empfänger haben. Der Inhalt einer Nachricht hängt vom verwendeten Protokoll (s.u.) ab. Um eine Nachricht zu kennzeichnen besitzt jede Nachricht eine **eindeutige** Nachrichten-Identifikationsnummer (NID).

2.4. Lokale und globale Uhren

In einer Simulation gibt es **genau eine** globale Uhr. Sie stellt die aktuelle und **immer korrekte** Zeit dar. Eine globale Uhr geht nie falsch.

Zudem besitzt jeder beteiligter Prozess eine eigene lokale Uhr. Sie stellt die aktuelle, jedoch nicht zwangsmäßig global-korrekte, Zeit des jeweiligen Prozesses dar. Wenn die Prozesszeit nicht korrekt ist (nicht der globalen Zeit gleicht), dann wurde die Prozessuhr entweder im Laufe einer Simulation neugestellt oder sie besitzt eine Uhrabweichung. Eine Uhrabweichung gibt an, um wieviel eine Uhr falsch geht. Wenn eine lokale Uhr nicht neugesetzt wird und auch keine Uhrabweichung hat, dann gibt sie stets die korrekte globale Zeit wieder.

2.5. Ereignisse

Eine Simulation besteht aus der Hintereinanderausführung von endlich vielen Ereignissen. Beispielsweise kann es ein Ereignis geben, welches einen Prozess eine Nachricht verschicken läßt oder den Prozess selbst abstürzen läßt. Jedes Ereignis tritt zu einem bestimmten Zeitpunkt ein. Wenn es zeitgleiche Ereignisse gibt, so werden sie ebenso hintereinander ausgeführt, behalten aber in der Simulation die selben Ausführzeiten.

2.6. Protokolle

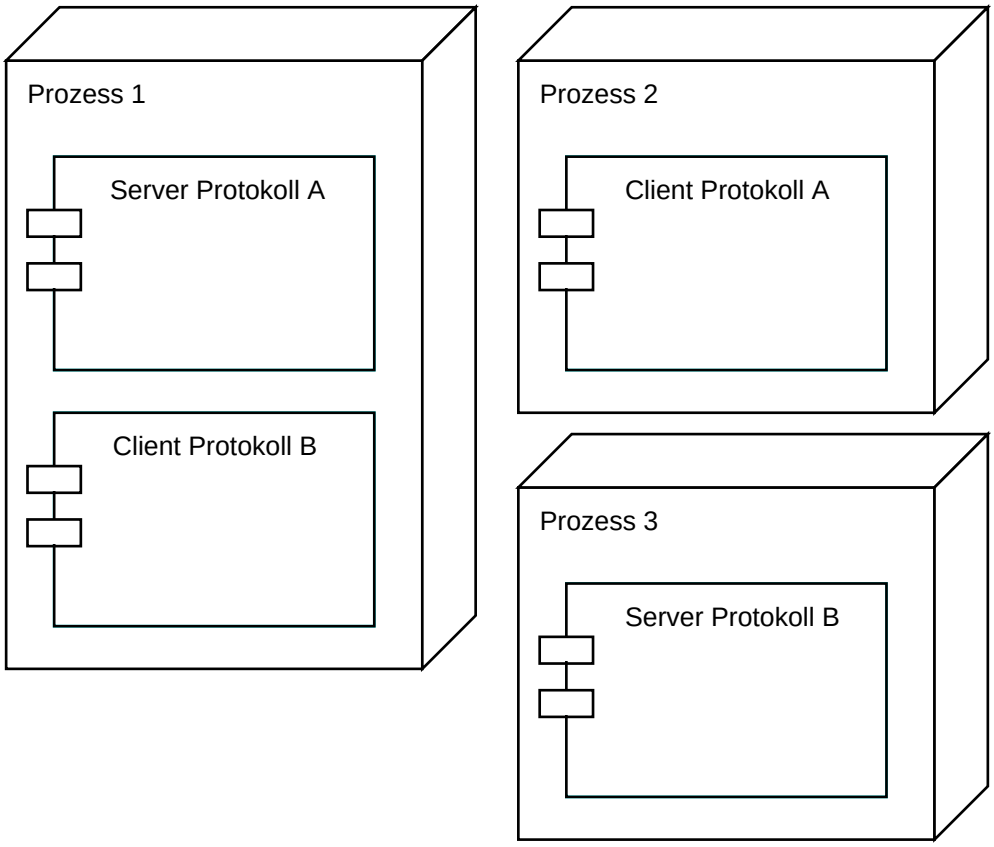

Abbildung 2.2.: Client/Server Protokolle

Eine Simulation besteht aus der Anwendung von Protokollen. Es wurde bereits erwähnt, dass ein Prozess die Rollen von Servern und/oder Clients annehmen kann. Bei jeder Server- und Clientrolle muss zusätzlich das dazugehörige Protokoll spezifiziert werden. Ein Protokoll definiert, wie ein Client und ein Server Nachrichten verschicken und wie sie bei Ankunft von

Nachrichten reagieren. Ein Prozess verarbeitet eine empfangene Nachricht nur, wenn er das jeweilige Protokoll versteht.

In Abbildung 2.2 sind 3 Prozesse dargestellt. Prozess 1 unterstützt serverseitig das Protokoll “A” und clientseitig das Protokoll “B”. Prozess 2 unterstützt clientseitig das Protokoll “A” und Prozess 3 serverseitig das Protokoll “B”. D.h., Prozess 1 kann mit Prozess 2 via Protokoll “A” und mit Prozess 3 via Protokoll “B” kommunizieren. Die Prozesse 2 und 3 sind zueinander inkompatibel.

Kapitel 3.

Der Simulator

3.1. Graphical User Interface (GUI)

3.1.1. Toolbar

3.1.2. Visualisierung

Zeitachse

Prozesse

Nachrichten

3.1.3. “Sidebar”

Ereignisse

Variablen

3.1.4. Loggfenster

3.1.5. Expertenmodus

Lamport- und Vektorzeit

Loggfilter

Lokale und globale Ereignisse

3.2. Ereignisse

3.2.1. Prozessabsturz

3.2.2. Prozesswiederbelebung

3.2.3. Aktivierung und Deaktivierung von Protokollen

Kapitel 4.

Die Implementierung

4.1. Gliederung der Pakete

4.2. Editoren

4.3. Ereignisse

4.3.1. Interne Ereignisse

4.4. Protokolle

4.4.1. Protokoll-API

4.5. Serialisierung von Simulationen

4.5.1. Rückwärtskompatibel

4.6. Programmierrichtlinien

4.7. Entwicklungsumgebung

Kapitel 5.

Ausblick

Anhang A.

Schemata

Anhang B.

Akronyms

GUI Graphical User Interface

NID Nachrichten-Identifikationsnummer

PID Prozess-Identifikationsnummer

VS Verteiltes System

Anhang C.

Literaturverzeichnis

[Bri03] Brian.H.Wilcox. Robotic vehicle system engineering. Internet, April 2003. http://team.caltech.edu/members/RoboticVehicleSystemEngineering3.pdf.