DIPLOMARBEIT

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

Durchgeführt an der

Fachhochschule Aachen
Fachbereich Elektrotechnik und Informationstechnik

Eupener Str. 70
D-52066 Aachen

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

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

durch

Paul C. Bütow

Matr.Nr.: 266617
Matthiashofstr. 15
D-52064 Aachen

Aachen, den 24. Juli 2008

Danksagungen

Ohne die Hilfe folgender Personen wäre die Anfertigung dieser Diplomarbeit in diesem Maße nicht möglich gewesen. Daher möchte ich mich bedanken bei:

- 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
- Mein Bruder Florian Bütow, für Tipps und Tricks rund um Java, für die Bereitstellung eines Buches sowie für das Testen des Simulators
- Meine 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
- Die Open Source Gemeinde; diese Diplomarbeit wurde ausschließlich mithilfe von Open Source Software erstellt

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1.

Einleitung

1.1. Motivation

In der Literatur findet man viele verschiedene Definitionen eins verteiltes Systems. Vieler dieser Definitionen unterschieden sich untereinander, so dass es schwerfällt eine Definition zu finden, die als Alleinige als die Richtige gilt. Andrew Tanenbaum und Marten van Steen haben für die Beschreibung eins verteilten Systems die Folgende lockere Charakterisierung formuliert:

[Tan03] *“Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen”*

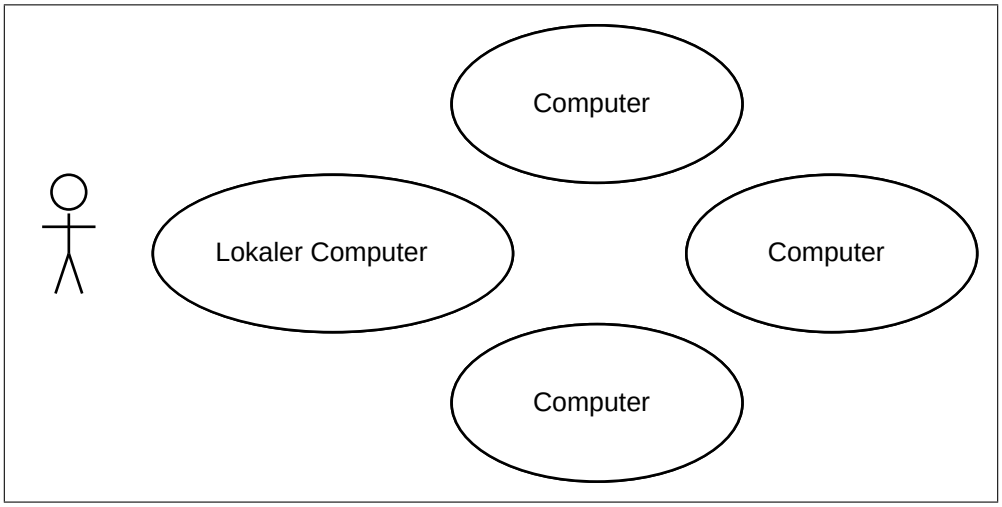

Abbildung 1.1.: Ein verteiltes System bestehend aus 4 Computern

Demnach erscheint dem Benutzer ein aus mehreren Computern bestehendes verteiltes System wie ein System, welches lediglich aus einem einzigen Computer besteht. Der Benutzer muss sich nur mit dem lokalen vor ihm befindenden Computer auseinandersetzen (Abbildung 1.1). Die Software des lokalen Computer stellt die reibungslose Kommunikation mit anderen Computern des verteilten Systems sicher und der Benutzer muss sich darum nicht selbst kümmern.

Hier betrachten wir ein verteiltes System von einer anderen Perspektive. Wir nehmen nicht die Sichtweise eines Endbenutzers ein, sondern wollen die grundlegenen Funktionsweisen, wie unabhängige Computer in einem verteilten System miteinander agieren, verstehen. Es sollen alle relevanten Ereignisse eines verteilten Systems sichtbar und verständlich repräsentiert werden.

Um dieses Ziel zu erreichen wurde ein Simulator entwickelt, der dies ermöglicht. Der Simulator wurde insbesondere für Lehr- und Lernzwecke entwickelt. Er inspiriert auch die Entwicklung eigener verteilter Systeme.

1.2. Grundlagen

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

Client/Server Modell

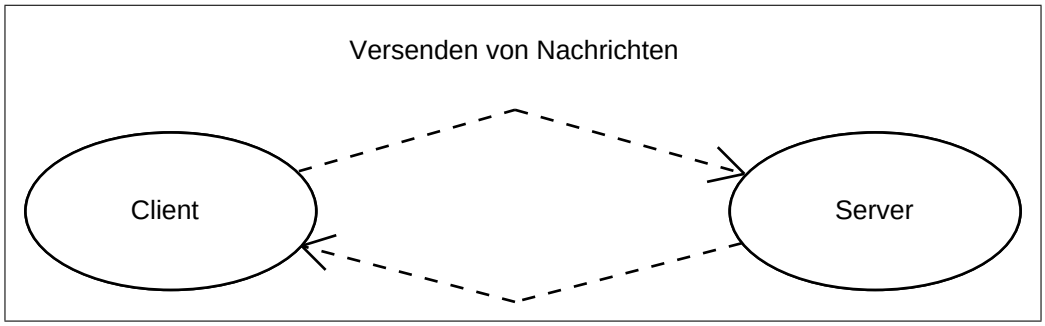

Abbildung 1.2.: Client/Server Modell

Der Simulator basiert auf dem Client/Server Prinzip. Jeder Simulation besteht in der Regel aus einen teilnehmenden Client und einen Server, die miteinander über Nachrichten kommunizieren (Abbildung 1.2). 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.

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).

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 ab. Was unter einem Protokoll zu verstehen ist, wird später behandelt. Um eine Nachricht zu kennzeichnen besitzt jede Nachricht eine **eindeutige** Nachrichten-Identifikationsnummer (NID).

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 global-korrekt ist (nicht der globalen Zeit gleicht), dann wurde die Prozessuhr entweder im Laufe einer Simulation neugesetzt, oder sie geht wegen einer Uhrabweichung falsch. Die Uhrabweichung gibt an, um welchen Faktor die Uhr falsch geht. Wenn eine lokale Uhr nicht

neugesetzt wird und auch keine Uhrabweichung hat, dann gleicht ihre Zeit die der globalen Uhr.

Neben diesen “normalen” Uhren sind auch die **Vektor-Zeitstempel** sowie die **logische Uhr von Lamport** von Interesse. Jeder Prozess besitzt zusätzlich einen Vektor-Zeitstempel für die Vektorzeit, sowie einen Lamportzeitstempel für die Lamportzeit. Für die Vektor- und Lamportzeiten gibt es hier, im Gegensatz zu der normalen Zeit, keine globalen Äquivalente.

Konkrete Beispiele zu den Lamport- und Vektorzeiten werden später anhand einer Simulation behandelt.

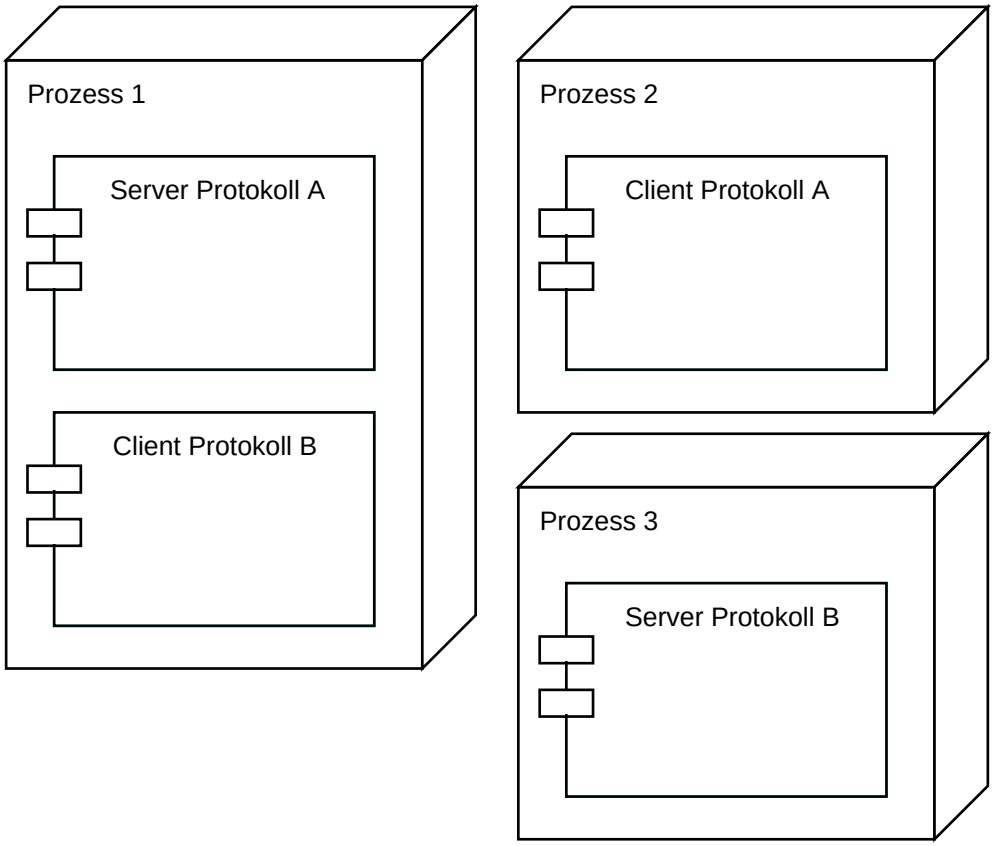

Abbildung 1.3.: Client/Server Protokolle

Ereignisse

Eine Simulation besteht aus der Hintereinanderausführung von endlich vielen Ereignissen. Beispielsweise kann es ein Ereignis geben, welches einen Prozess eine Nachricht verschicken- oder den Prozess selbst abstürzen läßt. Jedes Ereignis tritt zu einem bestimmten Zeitpunkt

ein. Wenn es zeitgleiche Ereignisse gibt, so werden sie in Wirklichkeit ebenso hintereinander ausgeführt, erscheinen aber in der Simulation als ob sie parallel ausgeführt würden. Dieser Umstand ist auf die Implementierung des Simulators zurückzuführen, worauf später noch genauer eingegangen wird. Dem Benutzer des Simulators stört dies jedoch nicht, da Ereignisse aus seiner Sicht parallel ausgeführt werden.

Protokolle

Eine Simulation besteht auch 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 verschickt und wie bei Ankunft einer Nachricht reagiert wird. Ein Protokoll legt auch fest, welche Daten in einer Nachricht enthalten sind. Ein Prozess verarbeitet eine empfangene Nachricht nur, wenn er das jeweilige Protokoll versteht.

In Abbildung 1.3 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 und können voneinander erhaltene Nachrichten nicht verarbeiten.

In der Regel können Clients nicht mit Clients und Server nicht mit Server kommunizieren. Je nach verwendetem Protokoll kann dies jedoch variieren. Alle vom Simulator verfügbaren Protokolle werden später genauer behandelt.

Kapitel 2.

Der Simulator

2.1. Die grafische Benutzerschnittstelle

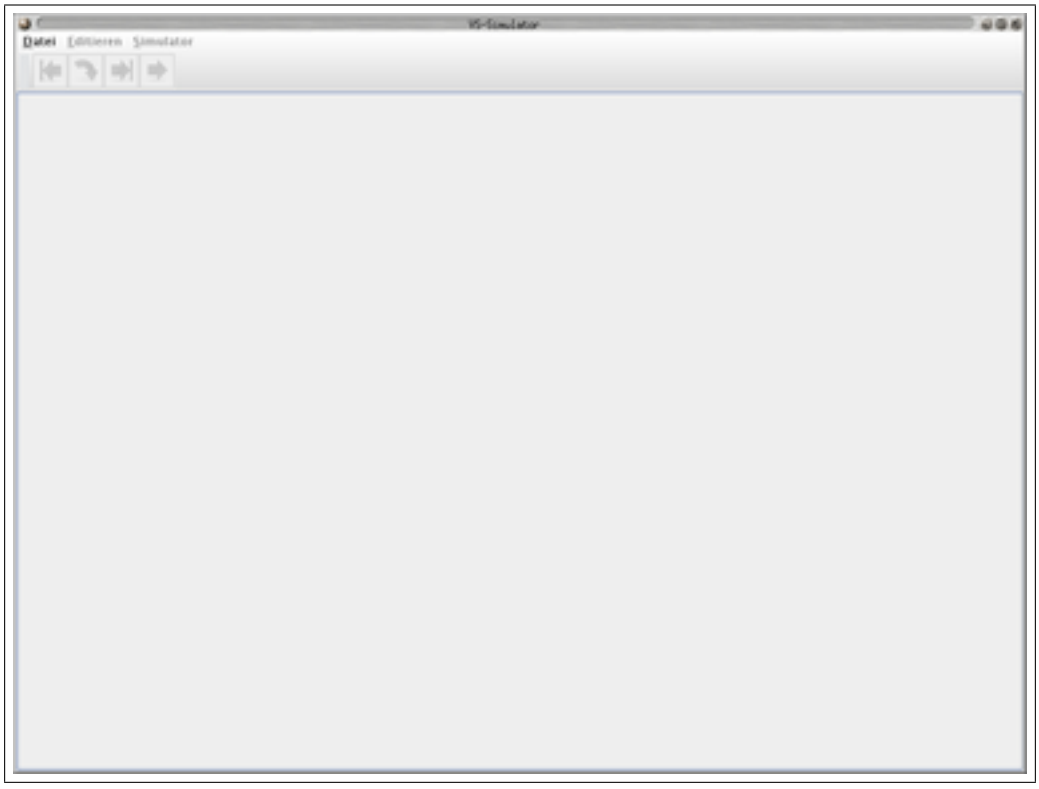

Abbildung 2.1.: Der Simulator nach dem ersten Starten

Der Simulator präsentiert sich nach dem ersten Starten wie in Abbildung 2.1. Für die Erstellung einer neuen Simulation wird im Menü “Datei” (Abbildung 2.2) der Punkt “Neue Simulation” ausgewählt, wo anschließend das Einstellungsfenster für die neue Simulation erscheint.

Auf die einzelnen Optionen wird später genauer eingegangen und es werden nun nur die Standardeinstellungen übernommen. Die GUI mit einer frischen Simulation sieht dann wie in Abbildung 2.3 aus.

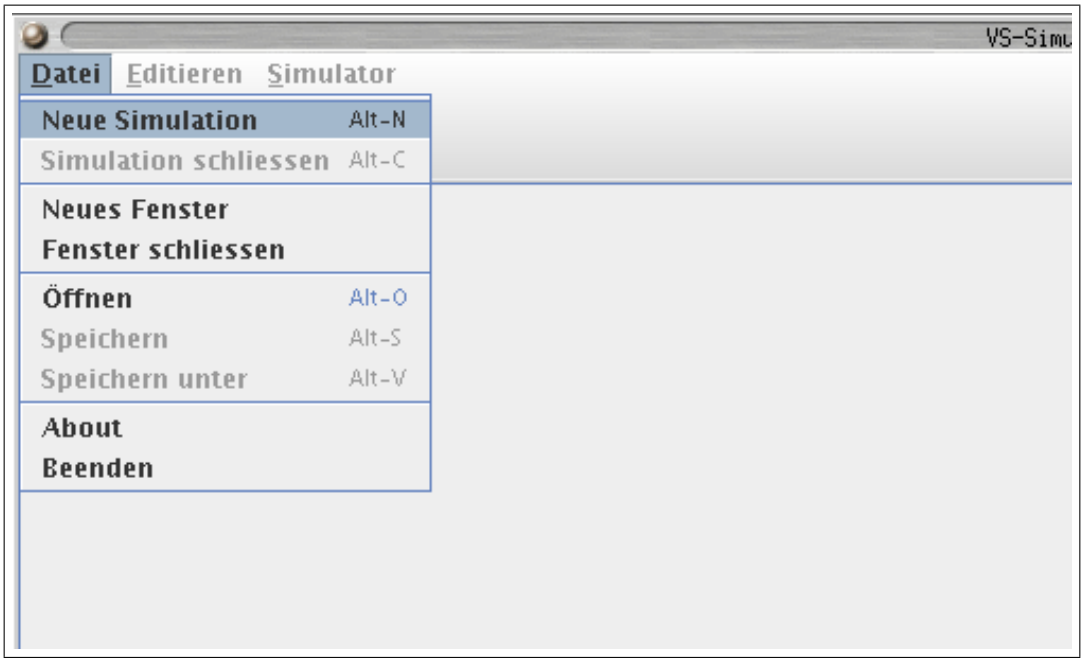

Abbildung 2.2.: Datei-Menü

Die Menüzeile

Im Datei-Menü (Abbildung 2.2) lassen sich neue Simulationen erstellen oder die aktuell geöffnete Simulation schliessen. Neue Simulationen öffnen sich standardmäßig in einem neuen Tab. Es können allerdings auch neue Simulationsfenster, die wiederrum eigene Tabs besitzen, geöffnet oder geschlossen werden. In jedem Tab befindet sich eine von den Anderen vollständig unabhängige Simulation. Es können somit beliebig viele Simulationen parallel ausgeführt werden. Die Menüeinträge “Öffnen”, “Speichern” und “Speichern unter” dienen für das Laden und Speichern von Simulationen.

Über das Editieren-Menü gelangt man zu den Simulationseinstellungen, worauf später genauer eingegangen wird. Es werden in diesem Menü auch alle beteiligten Prozesse zum Editieren aufgelistet. Wählt man dort einen Prozess aus, dann öffnet sich der dazugehörige Prozesseditor. Auf diesen wird ebenso später genauer eingegangen. Das Simulator-Menü bietet die selben Optionen wie die Toolbar, welche im nächsten Teilkapitel beschrieben wird.

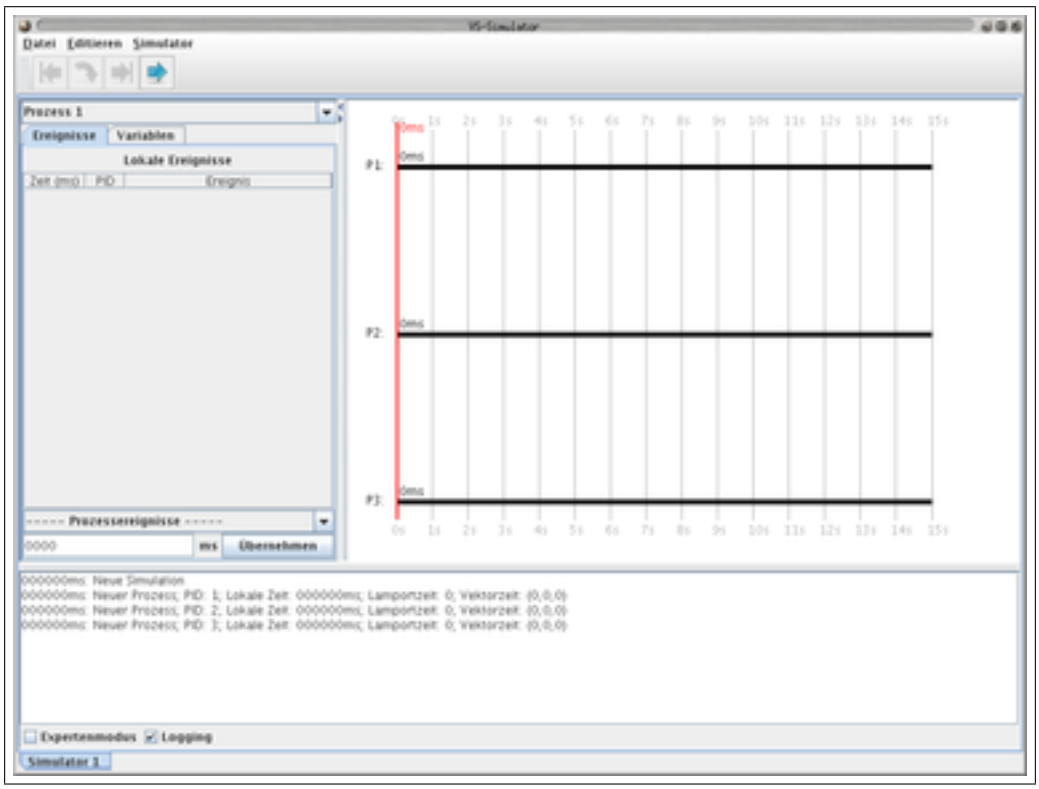

Abbildung 2.3.: Eine neue Simulation

Einige Menüunterpunkte sind erst erreichbar, wenn im aktuellen Fenster bereits eine Simulation erstellt oder geladen wurde.

Die Toolbar

Oben links im Simulator befindet sich die Toolbar (Abbildung 2.4). Die Toolbar enthält die Funktionen, die vom Benutzer am häufigsten verwendet werden.

Abbildung 2.4.: Die Menüzeile inklusive Toolbar

Die Toolbar bietet vier verschiedene Funktionalitäten an:

- Starten der Simulation; kann nur betätigt werden, wenn die Simulation derzeit nicht läuft.

- Pausieren der Simulation, kann nur betätigt werden, wenn die Simulation derzeit läuft.
- Wiederholen der Simulation, kann nicht betätigt werden, wenn die Simulation noch nicht gestartet wurde.
- Zurücksetzen der Simulation, kann nur betätigt werden, wenn die Simulation pausiert wurde oder wenn die Simulation abgelaufen ist.

Die Toolbar lässt sich auch nach Belieben repositionieren (z.B. links, rechts oder unten des Simulatorfensters). Hierfür muss per “Drag-n-Drop” die “raue Fläche” zur Zielposition gezogen werden.

Die Visualisierung

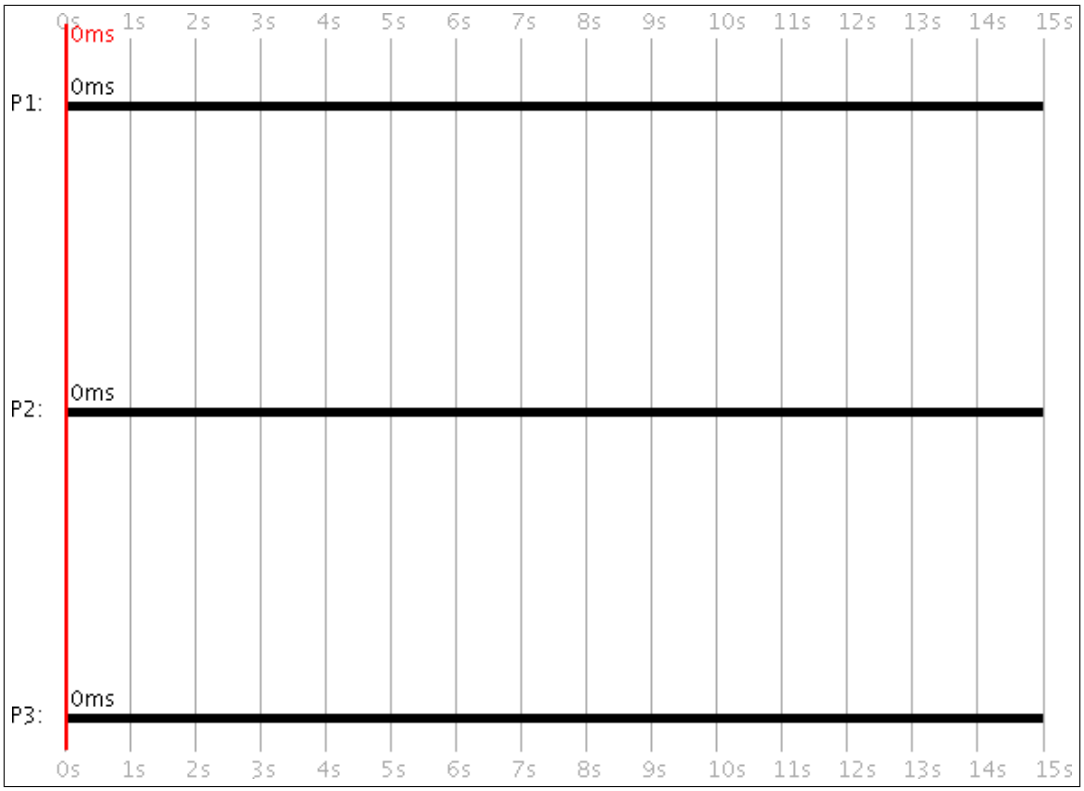

Abbildung 2.5.: Visualisierung einer noch nicht gestarteten Simulation

Mittig rechts in Abbildung 2.3 befindet sich die grafische Repräsentation der Simulation. Die X-Achse gibt die Zeit in Millisekunden an. Unsere Demo-Simulation endet nach genau 15 Sekunden. In Abbildung 2.5 sind 3 Prozesse (mit den PIDs 1, 2 und 3) dargestellt, die jeweils einen eigenen horizontalen schwarzen Balken besitzen. Auf diesen Prozessbalken kann der

Benutzer die jeweilige lokale Prozesszeit ablesen. Die vertikale rote Linie stellt die globale Simulationszeit dar.

Die Prozessbalken dienen auch für Start- und Zielpunkte von Nachrichten. Wenn beispielsweise Prozess 1 eine Nachricht zum Prozess 2 verschickt, so wird eine Linie vom einen Prozessbalken zum Anderen gezeichnet. Nachrichten, die ein Prozess an sich selbst schickt, werden nicht visualisiert. Sie werden aber im Loggfenster (mehr dazu später) protokolliert.

Eine andere Möglichkeit einen Prozesseditor aufzurufen ist ein Linksklick auf den zum Prozess gehörigen Prozessbalken. Dies muss also nicht zwingend über das Simulator-Menü geschehen. Ein Rechtsklick hingegen öffnet ein Popup-Fenster mit weiteren Auswahlmöglichkeiten (Abbildung 2.6). Ein Prozess kann über das Popup-Menü nur dann abstürzen oder wiederbelebt werden, wenn die Simulation aktuell läuft.

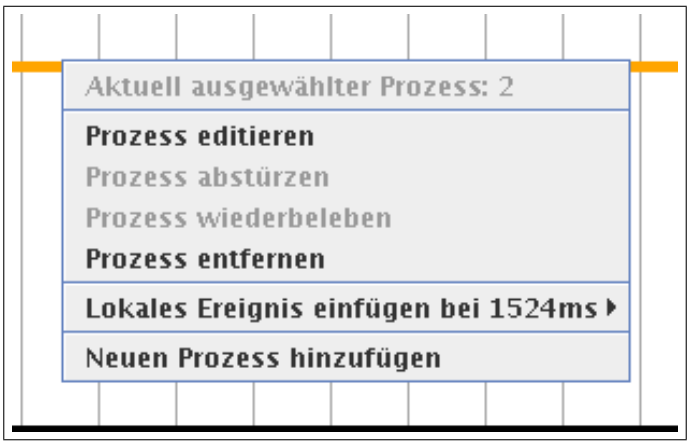

Abbildung 2.6.: Rechtsklick auf einen Prozessbalken

Generell kann die Anzahl der Prozesse nach belieben variieren. Die Dauer der Simulation beträgt mindestens 5 -und maximal 120 Sekunden. Die Simulation endet erst, wenn die globale Zeit 15 Sekunden erreicht hat, und nicht, wenn eine lokale Prozesszeit die 15 Sekunden erreicht.

Farbliche Differenzierung

Farben helfen dabei die Vorgänge einer Simulation zu deuten. Standardmäßig werden die Prozesse (Prozessbalken) und Nachrichten mit den Farben wie in Tabelle 2.1 aufgelistet dargestellt. Dies sind lediglich die Standarfarben, welche man über die Einstellungen umkonfigurieren kann.

Prozessfarbe	Bedeutung
Schwarz	Simulation läuft derzeit nicht (z.B. noch nicht gestartet, abgelaufen oder pausiert)
Orange	Die Maus befindet sich über den Prozessbalken
Rot	Der Prozess ist abgestürzt
Nachrichtfarbe	**Bedeutung**
Grün	Die Nachricht ist noch unterwegs und hat das Ziel noch nicht erreicht
Blau	Die Nachricht hat das Ziel erfolgreich erreicht
Rot	Die Nachricht ging verloren (entweder weil der Zielprozess abgestürzt ist oder weil sie unterwegs verloren ging)

Tabelle 2.1.: Farbliche Differenzierung von Prozessen und Nachrichten

Die Sidebar

Mithilfe der Sidebar lassen sich Ereignisse von Prozessen verwalten. Ganz oben in Abbildung 2.7 ist der zu verwaltende Prozess selektiert (hier mit der PID 1). In dieser Prozessauswahl gibt es auch die Möglichkeit “Alle Prozesse” auszuwählen, womit die Ereignisse aller Prozesse gleichzeitig verwaltet werden können. Unter “Lokale Ereignisse” versteht man diejenigen Ereignisse, die auftreten, wenn eine bestimmte lokale Zeit des dazugehörigen Prozesses eingetreten ist. Die darunterliegende Ereignistabelle listet alle programmierten Ereignisse (hier noch keine vorhanden) mitsamt Eintrittszeiten sowie den PIDs auf.

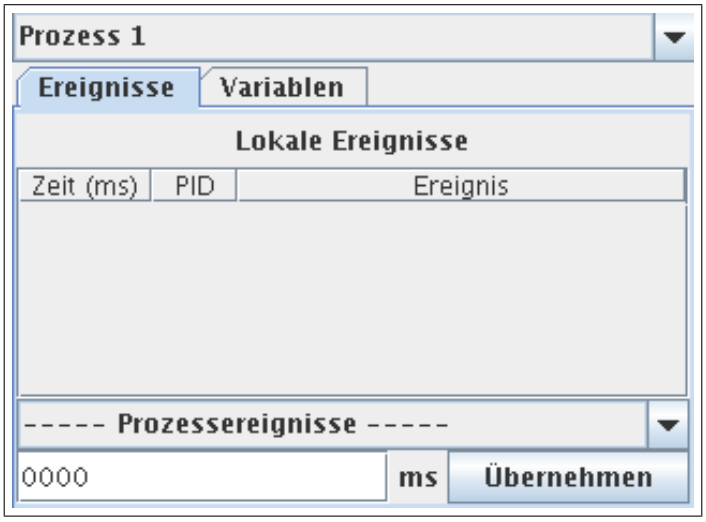

Abbildung 2.7.: Die Sidebar mit leerem Ereigniseditor

Für die Erstellung eines neuen Ereignisses kann der Benutzer entweder mit einem Rechtsklick

auf einen Prozessbalken (Abbildung 2.6) klicken, oder unterhalb der Ereignistabelle ein Ereignis auswählen (Abbildung 2.8), im darunterliegendem Textfeld die Zeit eintragen und auf “Übernehmen” klicken. Beispielsweise wurden auf Abbildung 2.9 drei Ereignisse hinzugefügt: Absturz nach 123ms, Wiederbelebung nach 321ms und erneuerter Absturz nach 3000ms des Prozesses mit der ID 1.

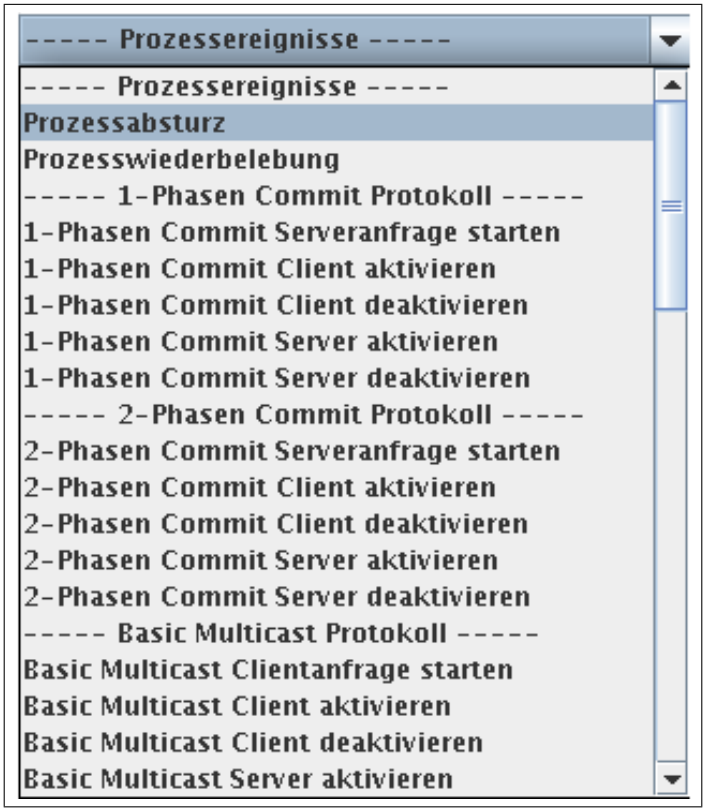

Abbildung 2.8.: Die Ereignisauswahl via Sidebar

Mit einem Rechtsklick auf den Ereigniseditor lassen sich alle selektierten Ereignisse entweder kopieren oder löschen. Die Einträge der Spalten für die Zeit und der PID lassen sich nachträglich editieren. Somit besteht eine komfortable Möglichkeit bereits programmierte Ereignisse auf eine andere Zeit zu verschieben oder einem anderen Prozess zuzuweisen.

In der Sidebar gibt es neben dem Ereignis-Tab einen weiteren Tab “Variablen”. Hinter diesem Tab verbirgt sich der Prozesseditor des aktuell ausgewählten Prozesses. Dort können alle Variablen des Prozesses editiert werden. Dies wird später genauer behandelt.

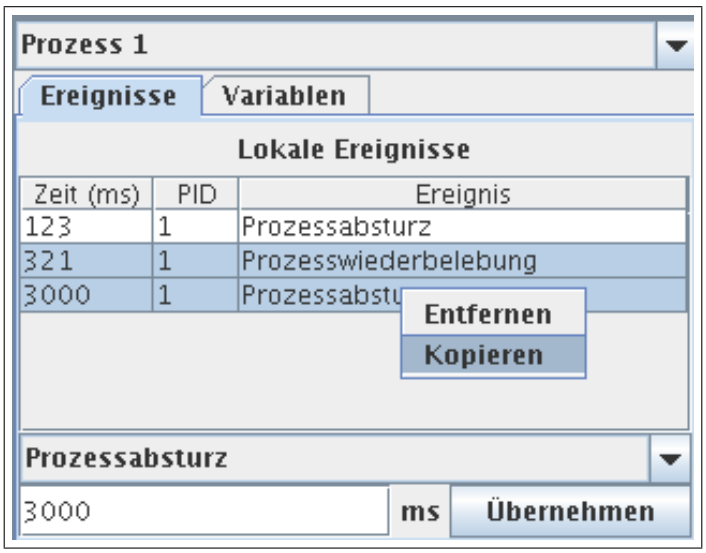

Abbildung 2.9.: Der Ereigniseditor mit 3 programmierten Ereignissen

Das Loggfenster

Das Loggfenster (Abbildung 2.3, unten) protokolliert in chronologischer Reihenfolge alle eingetroffenen Ereignisse. Auf Abbildung 2.10 sieht man das Loggfenster nach Erstellung unserer Simulation, an welcher 3 Prozesse beteiligt sind. Am Anfang eines Loggeintrages wird stets die globale Zeit in Millisekunden protokolliert. Bei jedem Prozess wird ebenso seine lokale Zeit sowie die Lamport- und die Vektor-Zeitstempel aufgeführt. Letztere werden später genauer behandelt.

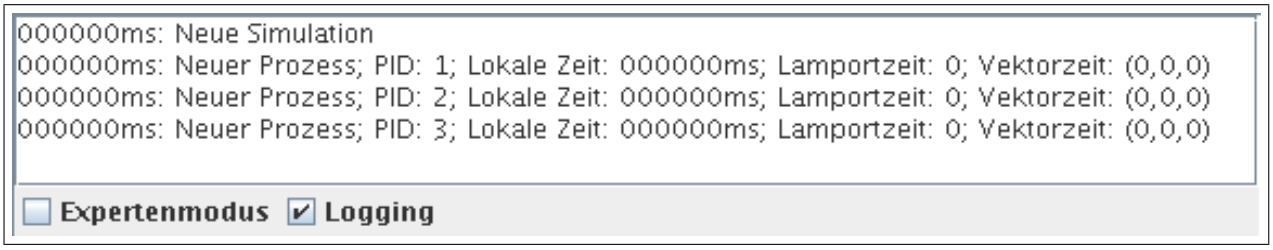

Abbildung 2.10.: Das Loggfenster

Mit dem Deaktivieren der Checkbox “Logging” läßt sich das direkte Loggen von Nachrichten temporär deaktivieren. Ohne aktivierter Checkbox erscheinen keine neuen Nachrichten mehr im Loggfenster. Nach Reaktivieren der Checkbox werden alle ausgelassenen Nachrichten nachträglich in das Fenster geschrieben. Eine Deaktivierung des Loggings kann zu verbessertem Leistungsverhalten des Simulators führen (z.B. kein Rucklen; ist vom verwendeten Computer, auf dem der Simulator läuft, abhängig). Dieser Umstand ist der sehr langsamen

Java-Implementierung der JTextArea-Klasse zu verdanken.

Über die Checkbox “Expertenmodus” wird der Expertenmodus aktiviert beziehungsweise deaktiviert.

2.2. Der Expertenmodus

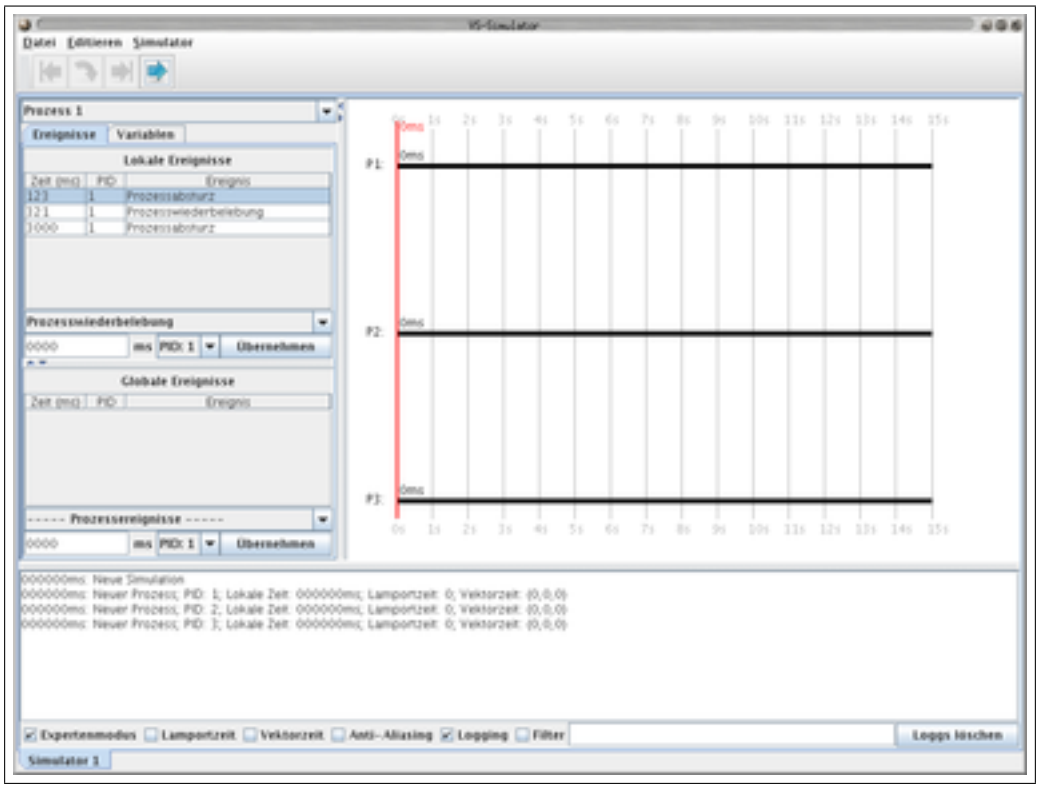

Abbildung 2.11.: Der Simulator im Expertenmodus

Der Simulator kann in zwei verschiedenen Modi betrieben werden. Es gibt einen einfachen- und einen Expertenmodus. Der Simulator started standardmäßig im einfachen Modus, so dass sich der Benutzer nicht mit der vollen Funktionalität des Simulators auf einmal auseinandersetzen muß. Der einfache Modus ist übersichtlicher, bietet jedoch weniger Funktionen an. Der Expertenmodus eigent sich für mehr erfahrene Anwender und bietet dementsprechend auch mehr Flexibilität. Der Expertenmodus kann über die gleichnamige Checkbox unterhalb des Loggfensters oder über die Simulationseinstellungen aktiviert oder deaktiviert werden. Auf Abbildung 2.11 ist der Simulator im Expertenmodus zu sehen. Wenn man den

Simulator im Expertenmodus mit Abbildung 2.3 vergleicht, dann fallen einige Unterschiede auf, die nun aufs Weitere behandelt werden.

Abbildung 2.12.: Die Sidebar im Expertenmodus

Der erste Unterschied ist in der Sidebar erkennbar (Abbildung 2.12). Dort sind nun, zusätzlich den lokalen Ereignissen, auch globale Ereignisse editierbar. Wie bereits erwähnt, sind unter lokale Ereignisse diejenigen Ereignisse zu verstehen, die auftreten, wenn eine bestimmte lokale Zeit des dazugehörigen Prozesses eingetreten ist. Globale Ereignisse hingegen sind diejenigen Ereignisse, die auftreten, wenn eine bestimmte globale Zeit eingetreten ist. Ein globales Ereignis nimmt die globale Zeit- und ein lokales Ereignis die lokale Prozesszeit als Eintrittskriterium. Globale Ereignisse machen somit nur einen Unterschied, wenn sich die lokalen Prozesszeiten von der globalen Zeit unterscheiden.

Eine weitere neue Funktionalität ist die Möglichkeit einem neuzuerstellenen Ereignis direkt die PID zuzuweisen. Im einfachen Modus wurde, wenn man ein neues Ereignis erstellte, standardmäßig immer die PID des aktuell ausgewählten Prozesses (in der obersten Combo-Box) verwendet. In dieser Combo-Box sollte man gegebenenfalls “Alle Prozesse” selektieren, damit im Ereigniseditor stets die Ereignisse aller Prozesse aufgelistet werden.

Weitere Unterschiede machen sich unterhalb des Loggfensters bemerkbar. Dort gibt es un-

ter Anderem zwei neue Checkboxen “Lamportzeit” und “Vektorzeit”. Aktiviert man eine dieser beiden Checkboxen, dann wird die Lamport- beziehungsweise Vektorzeit in die Visualisierung dargestellt. Übersichtshalber kann der Benutzer nur jeweils eine dieser beiden Checkboxen aktivieren. Wenn die Lamportzeit-Checkbox bereits aktiviert ist und der Benutzer versucht die Vektorzeit-Checkbox zusätzlich zu aktivieren, so wird die Lamportzeit-Checkbox automatisch deaktiviert und virce versa.

Die Anti-Aliasing-Checkbox ermöglicht dem Benutzer Anti-Aliasing zu aktivieren und deaktivieren. Mit aktiviertem Anti-Aliasing werden alle Grafiken der Visualisierung gerundet dargestellt. Aus Performancegründen ist Anti-Aliasing standardmäßig deaktiviert.

Je komplexer eine Simulation wird, desto unübersichtlicher werden die Einträge im Loggfenster. Hier fällt es zunehmend schwerer die Übersicht aller Ereignisse zu behalten. Um dem entgegenzuwirken gibt es im Expertenmodus einen Loggfilter, welcher es ermöglicht nur die wesentlichen Daten aus den Loggs zu filtern. Der Loggfilter wird anhand der dazugehörigen Checkbox “Filter” aktiviert beziehungsweise deaktiviert. In der dahinterliegenden Eingabezeile kann ein regulärer Ausdruck in Java-Syntax angegeben werden. Beispielsweise werden mit “`PID: (1|2)`” nur Loggzeilen angezeigt, die entweder “`PID: 1`” oder “`PID: 2`” beinhalten. Alle anderen Zeilen, beispielsweise mit “`PID: 3`”, werden dabei nicht angezeigt. Mit aktiviertem Loggfilter werden nur die Loggzeilen angezeigt, auf die der reguläre Ausdruck passt. Der Loggfilter kann auch nachträglich aktiviert werden. Bereits protokollierte Ereignisse werden jedes Mal erneuert gefiltert. Der Loggfilter kann auch während einer laufenden Simulation verwendet werden. Wenn der Loggfilter deaktiviert wird, dann werden wieder alle Nachrichten (auch nachträglich) im Loggfenster angezeigt.

2.3. Ereignisse

Es wird zwischen zwei verschiedenen Haupttypen von Ereignissen unterschieden: Programmierbare Ereignisse und nicht-programmierbare Ereignisse. Programmierbare Ereignisse lassen sich im Ereigniseditor editieren und deren Eintrittszeiten hängen von den lokalen Prozessuhren oder der globalen Uhr ab. Nicht-programmierbare Ereignisse lassen sich hingegen nicht im Ereigniseditor angeben und treten nicht aufgrund einer Uhrzeit auf sondern beispielsweise wenn eine Nachricht eintrifft.

Prozessabsturz- und Wiederbelebung (programmierbar)

Die beiden grundliegensten Ereignisse sind “Prozessabsturz” sowie “Prozesswiederbelebung”. Wenn ein Prozess abgestürzt ist, so wird sein Prozessbalken in rot dargestellt. Ein abgestürzter Prozess kann keine weiteren Ereignisse mehr verarbeiten und, wenn er eine Nachricht empfangen sollte, geht diese verloren. Die einzige Ausnahme bildet ein Wiederbelebungsereignis. Ein abgestürzter Prozess kann nichts, ausser wiederbelebt werden. Während eines Prozessabsturzes läuft die lokale Prozessuhr, abgesehen der Lamport- und Vektor-Uhren, wie gewohnt weiter. D.h. es könnte sein, dass ein Prozess einige seiner lokalen Ereignisse gar nicht ausführt, da er zu den Ereigniseintrittszeiten abgestürzt ist. Selbiges trifft natürlich auch auf globale Ereignisse zu. Wenn im echten Leben ein Computer abstürzt oder abgeschaltet wird, dann läuft dort die Hardwareuhr, unabhängig vom Betriebssystem, auch weiter.

Aktivierung und Deaktivierung von Protokollen (programmierbar)

Wir wissen bereits, dass ein Prozess mehrere Protokolle Client- und auch Serverseitig unterstützen kann. Welches Protokoll von einem Prozess unterstützt wird, kann der Benutzer anhand von Protokollaktivierungs- und Protokolldeaktivierungsereignissen konfigurieren. Somit besteht die Möglichkeit, dass ein gegebener Prozess ein bestimmtes Protokoll erst zu einem bestimmten Zeitpunkt unterstützt und gegebenenfalls ein anderes Protokoll ablöst. Jedes Protokoll kann entwender Server- oder Clientseitig aktiviert beziehungsweise deaktiviert werden. Welche Protokolle es gibt wird später behandelt.

Weitere Protokollereignisse (programmierbar)

Der Benutzer hat die Auswahl zwischen fünf weiteren Protokollereignissen:

- Aktivierung des Clients des gegebenen Protokolls
- Aktivierung des Servers des gegebenen Protokolls
- Deaktivierung des Clients des gegebenen Protokolls
- Deaktivierung des Servers des gegebenen Protokolls
- Starten einer Client/Server-Anfrage des gegebenen Protokolls

Ob sich das Ereignis für das Starten einer Anfrage auf den Client oder Server bezieht, hängt vom verwendeten Protokoll ab. Man unterscheidet von Protokollen die Clientseitig- oder Serverseitig eine initiale Anfrage starten. Beispielsweise startet bei dem “Ping-Pong Protokoll” der Client- und bei dem “Commit-Protokollen” der Server immer die erste Anfrage. Es gibt kein Protokoll, wo Client und Server jeweils eine initiale Anfragen starten können.

Bei allen dieser fünf Ereignissen kann der betroffene Prozess noch beliebig andere Dinge, abhängig vom Protokoll, tun. Beispielsweise kann er den Inhalt der Nachricht generieren oder lokale Variablen initialisieren oder eine der lokalen Uhzeiten ändern oder Wecker für “Callback Ereignisse” setzen (mehr dazu später).

Nachrichtenempfang sowie Antwortnachrichten (nicht-programmierbar)

Nachdem ein Prozess eine Anfragenachricht versendet hat, und ein weiterer Prozess diese Nachricht erhält, so überprüft der Empfängerprozess zunächst ob er das dazugehörige Protokoll versteht. Wenn es sich um eine Clientnachricht handelt, so muß der Empfänger ein Server sein und virce versa. Passt alles, so führt der Empfängerprozess die vom Protokoll definierten Aktionen aus. In der Regel berechnet der Prozess einen Wert und schickt eine Antwortnachricht zurück. Es können aber auch beliebig andere Aktionen ausgeführt werden.

Callback-Ereignisse (nicht-programmierbar)

Ein Callback-Ereignis kann von einem Protokoll ausgelöst werden. Das Protokoll setzt einen Wecker zur welcher lokalen Uhrzeit eine weitere Aktion ausgeführt werden soll. Zum Beispiel lassen sich hiermit Timeouts realisieren, wenn ein Protokoll eine Antwort erwartet, diese aber nicht eintrifft. Nach dem Timeout kann dann eine Anfrage erneuert verschickt werden! Es können beliebig viele Callback-Ereignisse definiert werden. Wenn sie noch nicht ausgeführt wurden und aufgrund eines anderen Ereignisses nicht mehr benötigt werden, können sie vom Protokoll auch wieder entfernt werden. Wenn ein Callback-Ereignis ausgeführt wird, kann es sich selbst wieder für eine weitere Ausführung erneuert planen. So lassen sich periodisch wiedereintreffende Ereignisse realisieren. Beispielsweise verwendet das “Reliable Multicast Protokoll” Callback-Ereignisse, indem solange Anfragen verschickt werden, bis alle benötigten Antworten vorliegen.

2.4. Protokolle

Im Folgenden werden alle bisher verfügbaren Protokolle behandelt. Wie bereits beschrieben wird bei Protokollen zwischen Server- und Clientseite unterschieden. Server können auf Clientnachrichten, und Client auf Servernachrichten antworten. Jeder Prozess kann beliebig viele Protokolle sowohl Clientseitig als auch Serverseitig untestützen. Theoretisch ist es auch möglich, dass ein Prozess für ein bestimmtes Protokoll gleichzeitig Server und Client ist. Der Benutzer kann auch weitere eigene Protokolle in der Programmiersprache Java mittels einer speziellen API (Application Programming Interface) erstellen. Wie eigene Protokolle erstellt werden können wird später behandelt.

2.4.1. Beispiel (Dummy) Protokoll

Das Dummy-Protokoll dient lediglich als leeres Template für die Erstellung eigener Protokolle. Bei der Verwendung des Dummy-Protokolls werden bei Ereignissen lediglich Loggnachrichten ausgegeben, jedoch keine weiteren Aktionen ausgeführt.

2.4.2. Das Ping-Pong Protokoll

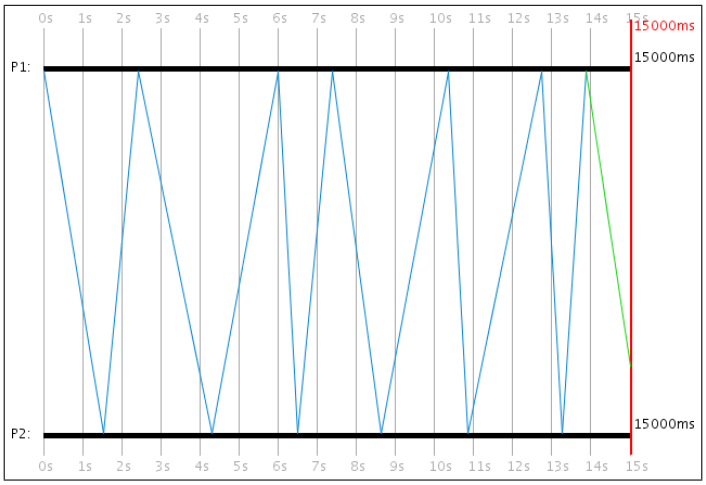

Abbildung 2.13.: Das Ping-Pong Protokoll

Bei dem Ping-Pong Protokoll (Abbildung 2.13) werden zwischen Client und Server ständig Nachrichten hin- und hergeschickt. Der Ping-Pong Client startet die erste Anfrage, worauf der Server dem Client antwortet. Auf diese Antwort wird vom Client wiederum geantwortet und

so weiter. Jeder Nachricht wird ein Zähler mitgeschickt, der bei jeder Station um eins inkrementiert wird und jeweils im Loggfenster protokolliert wird. Es werden erst keine Nachrichten mehr versendet, wenn entweder eine Nachricht verloren geht, oder wenn die Simulationszeit das Ende erreicht hat.

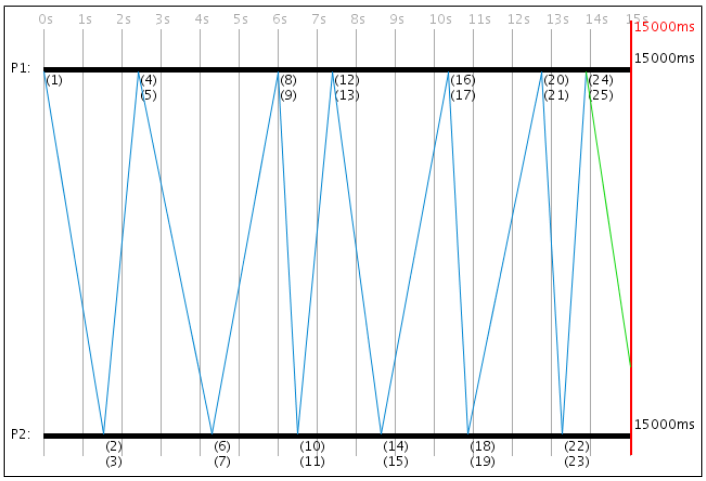

Abbildung 2.14.: Das Ping-Pong Protokoll mit Lamport-Zeitstempel

Das Ping-Pong Beispiel eignet sich auch für ein erstes Beispiel der Lamport- und Vektorzeitangaben. In Abbildung 2.14 wurde im Simulator die Lamportzeit-Checkbox aktiviert und in Abbildung 2.15 die der Vektorzeit. Selbstverständlich lassen sich die Lamport- und Vektorzeiten auch bei jedem anderen Protokoll darstellen.

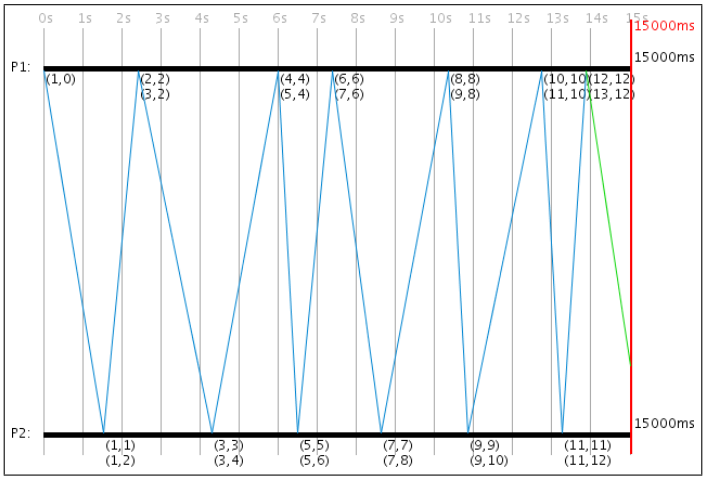

Abbildung 2.15.: Das Ping-Pong Protokoll mit Vektor-Zeitstempel

2.4.3. Das Broadcast-Sturm Protokoll

2.4.4. Das Protokoll zur internen Synchronisierung in einem synchronen System

2.4.5. Christians Methode zur externen Synchronisierung

lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds

flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj

lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj lsdkfjds lfjds flsjfsljsd flsdjf sldkfjsdlfkj

2.4.6. Berkeley Algorithmus zur internen Synchronisation

2.4.7. Das Ein-Phasen Commit Protokoll

2.4.8. Das Zwei-Phasen Commit Protokoll

2.4.9. Der ungenügende (Basic) Multicast

2.4.10. Der zuverlässige (Reliable) Multicast

2.5. Einstellungen

2.5.1. Simulationseinstellungen

2.5.2. Prozesseinstellungen

2.5.3. Protokolleinstellungen

Kapitel 3.

Die Implementierung

3.1. Gliederung der Pakete

3.2. Editoren

3.3. Ereignisse

3.3.1. Interne Ereignisse

3.4. Protokolle

3.4.1. Protokoll-API

3.5. Serialisierung von Simulationen

3.5.1. Rückwärtskompatibel

3.6. Programmierrichtlinien

3.7. Entwicklungsumgebung

Kapitel 4.

Ausblick

Anhang A.

Schemata

Anhang B.

Akronyms

API Application Programming Interface

GUI Graphical User Interface

NID Nachrichten-Identifikationsnummer

PID Prozess-Identifikationsnummer

VS Verteiltes System

Anhang C.

Literaturverzeichnis

[Tan03] Andrew Tanenbaum. Verteilte systeme - grundlagen und paradigmen. Buch, 2003. 2. Autor Marten van Steen; ISBN: 3-8273-7057-4.