Visualisierungsframework für Konzepte der Verteilten Systeme

Dominik Psenner
Christoph Caks

Visualisierungsframework für Konzepte der Verteilten Systeme

von Dominik Psenner und Christoph Caks

Zusammenfassung

Lehrveranstaltungen über Verteilte Systeme beschäftigten sich vornehmlich mit verteilten Algorithmen. Diese werden meist als Pseudo-Code dargestellt und Studenten haben oft Probleme sich darunter etwas vorszustellen. Es soll eine Möglichkeit geschaffen werden diese Algorithmen anschaulich, optisch ansprechend und interaktiv aufzubereiten.

Ziel dieser Arbeit ist es ein Framework zu konzeptionieren, welches diese Darstellung ermöglicht. Es muss eine API entworfen werden, die es ermöglicht, Verteilte Algorithmen zu implementieren. Das Framework muss diese Verteilten Algorithmen auf einem Rechner simulieren können, optisch ansprechend darstellen und dem Benutzer eine Möglichkeit bieten, das simulierte System zu beeinflussen. Außerdem sollen bereits einige verteilte Algorithmen implementiert werden.

Die Arbeit verschafft einen Überblick über das Design und die Implemntation dieses Frameworks. Das Framework ist jetzt voll funktionsfähig und wird für Lehrveranstaltung der verteilten Systeme verwendet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Beispiele

Danksagung

danke.

The component suffered from three failings:

- It was slow
- It ran hot
- It didn't actually work

Of these three, the last was probably the most important.

Kapitel 1. Einleitung

1. Vision

In diesem Abschnitt wird das Ziel dieser Bakkalaureatsarbeit beschrieben.

Zunächst werden die Anwendungsmöglichkeiten im Bereich der Visualisierung aufgezeigt. Anschließend werden die Anwendungsbereiche von VIDIS abgesteckt.

Der Name VIDIS wurde abgeleitet aus der englischen Betitelung des Projektnamens unter der üblichen Konvention von Open Source Projekten. VIDIS steht für „Visualisation Framework of Distributed Systems“. Diese Bakkalaureatsarbeit steht für ein einfach zu verwendendes, anschauliches und didaktisch hochwertiges Framework zur Simulation und Visualisierung für Verteilte Systeme. Entsprechend gliedern sich die Hauptanwendungsbereiche in „Visualisierung“ und „Entwicklung“.

1.1. VIDIS für Visualisierungen

VIDIS wurde als Visualisierungsframework geplant und entworfen. Es soll die Möglichkeit bieten Prinzipien von Verteilten Systemen schnell, verständlich und anschaulich zu präsentieren.

1.2. VIDIS für Entwickler

VIDIS kann dazu verwendet werden um verschiedene Implementierungen und Arten von Algorithmen auf diversen Netzwerkstrukturen zu testen und zu entwickeln. Es eignet sich zur Analyse von bestehenden Algorithmen indem ihre Funktionsweise veranschaulicht wird.

2. Arbeitsteilung

VIDIS wurde von Dominik Psenner und Christoph Caks in Zusammenarbeit mit Prof. Max Berger entworfen und implementiert. In diesem Abschnitt wird die Arbeitsteilung beschrieben.

2.1. Dominik Psenner

- Simulator
- TODO

2.2. Christoph Caks

- Visualisierung
- TODO

Kapitel 2. Related Work

1. Technologien und Bibliotheken

1.1. Java

1.1.1. Ant

1.1.2. Java WebStart

1.2. OpenGL

OpenGL® steht für „Open Graphics Library“ und ist eine öffentlich spezifizierte[1] Software-Schnittstelle für Grafik-Hardware. Die Spezifiezierungskommission besteht aus Firmen wie SGI, IBM, HP, Sun, Intel und Microsoft.

Abbildung 2.1. Das OpenGL® Logo (Quelle: [4] [33])

OpenGL® 2.1 ist derzeit noch aktuell, die Kommission hat den 3.0 Standard am 11. August 2008 verabschiedet. Derzeit arbeiten die Grafikkartenhersteller noch an der Anpassung der Treiber an die neue Spezifikation [10] [33].

Außerdem sind noch 2 weitere Versionen in Planung[9] [33]:

- OpenGL® 3.x „Longs Peak Reloaded“
- OpenGL® 3.x „Mount Evans“

1.2.1. OpenGL Java Bindings

Es existieren einige Projekte, die es ermöglichen den OpenGL® Standard auch in anderen Sprachen wie zb Java zu verwenden. Einige sind einfache API-Wrapper, die über JNI einach alle Funktionen der API zugänglich machen, andere haben ein komplett eigenes Objektmodell und verstecken die OpenGL® API vor dem Programmierer.

[1] Die Spezifikation ist unter [4] [33] zu finden.

1.2.1.1. Java3D™

Die Java3D™ API ist ein Open Source Projekt. Die API setzt auf ein Objektmodell mit Szenengraph.

1.2.1.2. Java OpenGL (JOGL)

JOGL[6] [33] steht für Java OpenGL und implementiert den JSR-231[7] [33] Standard. Dieser gibt genau vor wie die OpenGL® API Aufrufe in Java auszusehen haben.

1.2.1.3. Lightweight Java Gaming Library (LWJGL)

Diese Library implementiert nicht alle Funktionen der OpenGL® API. Die Entwickler haben bewusst auf veraltete und imperformate Operatinen verzichtet. Dadurch hat die Bibliothek eine geringe Größe, und eignet sich unter anderem besonders für embedded Devices.

1.3. XML

1.3.1. Sax

1.3.2. DocBook

2. Entwicklungswerkzeuge

2.1. Eclipse

2.2. Netbeans

2.3. Subversion

2.4. Testing and Performance Tools Platform

2.5. Metrics

3. Verwendete verteilte Algorithmen

3.1. Vector Clock Algorithmus

3.2. Bully Election Algorithmus

3.3. Byzantinische Generäle

3.4. Echo Algorithmus

3.5. Flooding Algorithmus

Kapitel 3. Design

1. Aufbau

Komponentendiagramm

2. Modul

Ein Modul soll eine Implementierung eines verteilten Algorithmus sein. Einhergehend mit der Definition geht die Beobachtung, dass alle Verteilten Systemen mit Hilfe von drei Komponenten dargestellt werden können. So gibt es in jedem Verteilten System Peers, welche über Verbindungen untereinander miteinander kommunizieren. Somit bietet sich folgender naiver Ansatz für ein Grundgerüst an um ein Verteiltes System zu simulieren.

Man unterscheidet drei Komponenten:

- Node
- Link
- Packet

Diese drei Komponenten können in dieser Form direkt in die Klassenstruktur übernommen werden.

Ein Node (oder auch Knoten) sei eine ausführbare Einheit, auf welchem ein gewisser Algorithmus implementiert wurde und zur Simulation ausgeführt wird. Gewissermaßen ist dies gleichzusetzen mit einem Peer in einem Verteilten System.

Ein Packet (oder auch Paket) sei ein Datenträger, welcher zum Datenaustausch zwischen Nodes verwendet wird.

Ein Link (oder auch Verbindung) sei eine ausführbare Einheit, auf welcher ein oder mehrere Packet von einem Node zu einem anderen transportiert werden können.

2.1. Voraussetzungen

Im folgenden werden die Voraussetzungen für die Komponenten beschrieben.

2.1.1. Voraussetzungen für eine Node

Ein Node soll auf die Verbindungen mit anderen Nodes zugreifen können. Seine Funktionalität, das heißt, der Code der bei jedem Simulationsschritt ausgeführt wird, soll frei programmierbar sein. Weiters soll ein Node auf den Erhalt von Paketen reagieren können.

2.1.2. Voraussetzungen für ein Packet

Ein Packet soll beliebige Daten abspeichern und transportieren können. Außerdem soll ein Packet den Absender und den Weg den es genommen hat kennen, um die Arbeit damit zu erleichtern.

2.1.3. Voraussetzungen für einen Link

Ein Link soll zwei Nodes bidirektional verbinden. Dabei soll es möglich sein die Länge des Links anzugeben (Delay). Es soll möglich sein die Paketübertragung zu stören um auch instabile Netzwerke beziehungsweise Netzwerkstörungen simulieren zu können.

2.2. Interfaces

Im folgenden werden die Interfaces der Komponenten Node, Packet und Link definiert.

2.2.1. Node

Ein Node ist die Entität eines Peers. Sein Verhalten ist deterministisch vorhersehbar. Dieses Verhalten soll in dieser Klasse kodiert und bei der Simulation ausgeführt werden.

Ein Node definiert die für den implementierenden Anwender relevanten Methoden:

- Diese Funktion wird dazu verwendet um eine Verbindung zu einem anderen Knoten aufzubauen. Sollte bereits eine Verbindung bestehen hat ein Aufruf dieser Funktion keinerlei Effekt.

```
    public void connect(
IUserNode
n
,

Class<? extends IUserLink> lclazz
,
                      long delay);
```

- Diese Funktion wird dazu verwendet um eine bestehende Verbindung zu einem anderen Knoten zu trennen. Sollte keine Verbindung zum anderen Knoten bestehen hat ein Aufruf dieser Funktion keinerlei Effekt.

```
    public void disconnect(
IUserNode
n
);
```

- Diese Funktion wird dazu verwendet um Pakete zu empfangen. Jedes Paket, das diese Node empfängt, wird über diese Funktion an den implementierenden Anwender weitergeleitet damit dieser angemessen darauf reagieren kann. Sie bildet, zusammen mit der Funktion "execute", die Hauptfunktionalität eines Moduls.

```
    public void receive(
IUserPacket
packet
);
```

- Diese Funktion wird beim Start einer Simulation ausgeführt. In dieser Funktion können Initialisierungen für Variablen und dergleichen implementiert werden.

```
public void init();
```

- Diese Funktion wird bei jedem Simulationsschritt einmal ausgeführt. Sie bildet, zusammen mit der Funktion "receive", die Hauptfunktionalität eines Moduls.

```
public void execute();
```

2.2.2. Link

Ein Link soll zwei Nodes bidirektional verbinden. Dabei soll es möglich sein die Länge des Links anzugeben (Delay). Es soll möglich sein die Paketübertragung zu stören um auch instabile Netzwerke beziehungsweise Netzwerkstörungen simulieren zu können.

Ein Link definitiert folgende, für den implementierenden Anwender, wichtigen Methoden:

- Diese Funktion wird mit jedem Simulationsschritt ausgeführt.

```
public void execute();
```

- Mit Hilfe dieser Funktion kann man ein Paket aus der Liste der zu übertragenden Pakete entfernt werden. So kann man Paketverluste ("Packet loss") simulieren.

```
public final void dropPacketOnLink(IUserPacket packet);
```

- Diese Funktion entfernt alle Pakete, die derzeit übertragen werden, von diesem Link. So kann man Paketverluste ("Packet loss") simulieren.

```
public final void dropPacketsOnLink();
```

- Diese Funktion bewirkt, dass diese Verbindung zwischen zwei Knoten gekappt wird.

```
public void disconnect();
```

- Mit Hilfe dieser Funktion erhält man die Verzögerung (Delay) dieses Links. Die Verzögerung ist die Länge des Links. Ein Paket braucht zur Übertragung von einem Node zum anderen genau so viele Simulationsschritte.

```
public long getDelay();
```

- Diese Funktion kann dazu verwendet werden, um den Nachbarknoten mit dem man über diesen Link verbunden ist, herauszufinden.

```
public IUserNode getOtherNode(IUserNode me);
```

2.2.3. Packet

Ein Packet wird durch folgende wichtigen Methoden definiert:

```
public IUserLink getLinkToSource();
```

Diese Funktion gibt den Link zum Absender des Pakets zurück.

```
    public IUserNode getSource();
```

Diese Funktion gibt den Node des Absenders zurück.

3. Simulator

3.1. Funktionsweise

Der Simulator ist die ausführende Einheit eines Moduls. Dieser hat folgende Aufgaben:

- Modul laden
- Objekte instantiieren
- Schrittweise simulieren

Nach jedem Simulationsschritt werden alle Veränderungen an den Objektinstanzen des geladenen Moduls auf der Benutzeroberfläche dargestellt.

3.2. Player

Der Player ist die kontrollierende Einheit für den Simulator. Er kümmert sich darum, einen Simulationsschritt anzustossen. Somit ist er das Verbindungsstück zwischen Simulation und Visualisierung.

Der Player hat mehrere Zustände.

- Ist der Simulator nicht gestoppt, kodiert dieser Zustand ob ein weiterer Simulationsschritt ausgeführt werden soll.

```
public
boolean
paused
;
```

- In diesem Zustand ist der Simulator gestoppt, das Modul neu geladen, die Objekte richtig instantiiert und der Simulator bereit zu starten.

```
public
boolean
stopped
;
```

- Kodiert den Zustand des korrekten Beendens des Simulators.

```
public
boolean
```

```
killed
;
```

Folgendes Zustandübergangsdiagramm veranschaulicht die Zustandsmaschine Player.

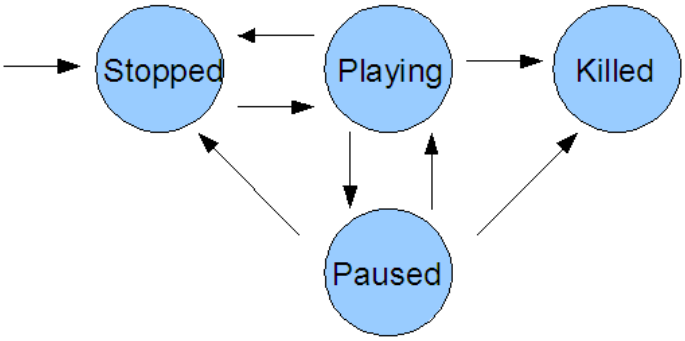

Abbildung 3.1. Zustandsübergangsdiagramm des Players

Er implementiert folgende Funktionalität:

- Diese Funktion startet den Simulator falls er noch nicht gestartet wurde. Wurde der Simulator bereits gestartet, hat diese Funktion keinen Effekt.

```
  public void play();
```

- Diese Funktion pausiert den Simulator. Wurde der Simulator bereits pausiert, startet diese Funktion den Simulator.

```
  public void pause();
```

- Diese Funktion startet bzw stoppt den Simulator. Wurde der Simulator gestoppt, ruft diese Funktion play() auf. Wurde der Simulator hingegen nicht gestoppt, ruft diese Funktion pause() auf.

```
  public void playPause();
```

- Diese Funktion stoppt den Simulator. Ist der Simulator bereits gestoppt hat ein Aufruf dieser Funktion keine Auswirkung.

```
  public void stop();
```

- Diese Funktion veranlasst den Simulator dazu, alle allokierten Ressourcen freizugeben und erlaubt dem Player-Thread zu sterben.

```
  public void kill();
```

3.3. Datenstruktur (intern)

Die Datenstruktur des Simulators ist ein Tupel (M, t), wobei M die Menge der ausführbaren Komponenten ist und t der aktuelle Simulationsschritt ist.

Diese Datenstruktur ist in der Klasse SimulatorData implementiert. Die Klasse enthält die Felder:

```
private
long
now
;
```

```
private
List<AComponent>
components
;
```

Die Klasse enthält im Wesentlichen diese Funktionen:

```
    public long getTime();
```

```
    public void registerComponent(
AComponent
component
);
```

```
    public void unregisterComponent(
AComponent
component
);
```

```
    public void executeComponents();
```

4. Darstellung

4.1. Graphische Benutzeroberfläche

Die Oberfläche muss einfach zu bedienen sein und sich optimal in die 3D Umgebung eingliedern. Existierende APIs wie Java™ Swing oder Eclipse RCP wurden ausgeschlossen, da ... #FIXME

Es wurde eine einfache 3D API entworfen, die genau die benötigten Bedürfnisse erfüllt. Sie besteht prinzipiell aus einem Container und einigen Widgets, um Benutzerineraktion zu ermöglichen.

Eine kurze Übersicht der GUI API:

- BasicGuiContainer

- ScrollContainer
- Button
- CheckBox
- Label
- NodeField

Abbildung 3.2. Klassendiagramm GUI API

4.2. Visualisierung

Das simulierte, verteilte System muss anschaulich dargestellt werden. Prinzipell müssen die folgenden Elemente gezeichnet werden:

- Node
- Link
- Paket

Die einfachste Darstellung wäre eine zweidimensionale Ebene auf der die Nodes als Punkte gezeichnet werden. Diese werden mit Linien, um die Links darzustellen, verbunden. Außerdem braucht man noch Punkte, um die Pakete darzustellen. Diese Punkte müssen sich natürlich von den Nodepunkten unterscheiden und können sich auf den Links (also den Linien) bewegen.

Es sollte aber eine weitaus ansprechendere Darstellung geschaffen werden.

Der Überschaubarkeit halber wird diese zweidimensionale Ebene beibehalten, aber nur als Grundfläche in einem dreidimensionalen Raum. Die Nodes werden zu Kugeln, die Links zu transparenten, nach oben gebogenen Rohren, die sich wölben wenn sie ein Paket trasportieren. Die Pakete werden auch zu Kugeln, die im Inneren der Links zu sehen sind. Dadurch wird die ganze Simulation plastischer vermittelt.

5. Dead Ends

Was haben wir uns doch alles schönes gedacht und schlussendlich kam alles anders.

Kapitel 4. Implementierung

1. Simulator

Im folgenden Abschnitt definieren und beschreiben wir den Simulator.

1.1. Definitionen

Folgende Definitionen werden im Simulator verwendet.

1.1.1. Komponente

Wir bezeichnen eine Komponente als eine ausführbare, vom Benutzer implementierte Klasse.

Es wird unter folgenden Komponententypen unterschieden:

1.1.1.1. Node

Ein Node, oder auch "Knoten", ist in der Terminologie gleichzusetzen mit einem Peer aus einem Verteilten Systemen.

In einem Netzwerk aus Computern wäre ein Knoten einer dieser Rechner.

Ein Knoten kann sich über Links mit anderen Knoten verbinden und so mit diesen über Pakete kommunizieren.

Mehr Informationen zur Funktionalität eines Knoten findet man im Abschnitt "Implementation > Modul > Node".

1.1.1.2. Link

Ein Link, oder auch "Verbindung", ist in der Terminologie gleichzusetzen mit einer aktiven Verbindung zwischen zwei Peers aus einem Verteilten System.

Mehr Informationen zur Funktionalität eines Knoten findet man im Abschnitt "Implementation > Modul > Link".

1.1.1.3. Packet

Ein Packet, oder auch "Paket", ist in der Terminologie gleichzusetzen mit einem Datenpaket das über eine aktive Verbindung zwischen zwei Peers geschickt wird.

Mehr Informationen zur Funktionalität eines Knoten findet man im Abschnitt "Implementation > Modul > Paket".

1.1.2. Modul

Ein Modul ist eine Menge aus Komponenten

1.1.3. Steps

Ein "Step", oder auch "Schritt" ist die kleinste Zeiteinheit im Simulator.

1.2. Arbeitsweise

Der Simulator arbeitet abschnittsweise. Ein Abschnitt stellt einen vollständigen Simulationsschritt dar.

1.2.1. Grundlegende Arbeitsweise

Zunächst werden falls nötig alle Variablen initialisiert.

Anschließend werden simulatorinterne Operationen ausgeführt. Im Falle eines Links wäre dies der Transport eines Paketes.

Am Schluss werden nochmals alle Variablen auf Änderungen überprüft. Gegebenenfalls wird die Visualisierungskomponente über die Änderungen informiert.

1.2.2. Detaillierte Beschreibung der Ablaufreihenfolge

- Node:
  - Wenn Simulationsschritt #1: module.init();
  - processPacketQueue();
  - propagateVariableChanges();
  - Wenn nicht schlafend: module.execute();
- Link:
  - processQueue();
  - propagateVariableChanges();
  - Wenn nicht schlafend: module.execute();
- Paket:
  - propagateVariableChanges();
  - execute(); (leere final Methode)

2. Visualisierung

2.1. OpenGL® Bindings

2.1.1. Java3D

2.1.2. JOGL

2.1.3. LWJGL

2.1.4. Fazit

und deshalb hama JOGL gnommen

2.2. Problemstellungen

2.2.1. Picking

Unter Picking versteht man den Prozess, herauszufinden auf welches 3D Objekt ein Benutzer geklickt hat. Man unterscheidet zwischen verschiedenen Varianten wie ein solcher Agorithmus arbeitet:

2.2.1.1. Ray Picking

Beim Ray Picking werden die 2D Koordinaten des Klicks zu einer Geraden im 3D Raum umgerechnet (vgl. Abbildung 4.1 [17] ray). Dieser „Strahl“ schneidet alle Objekte, die sich unter der Maus befinden.

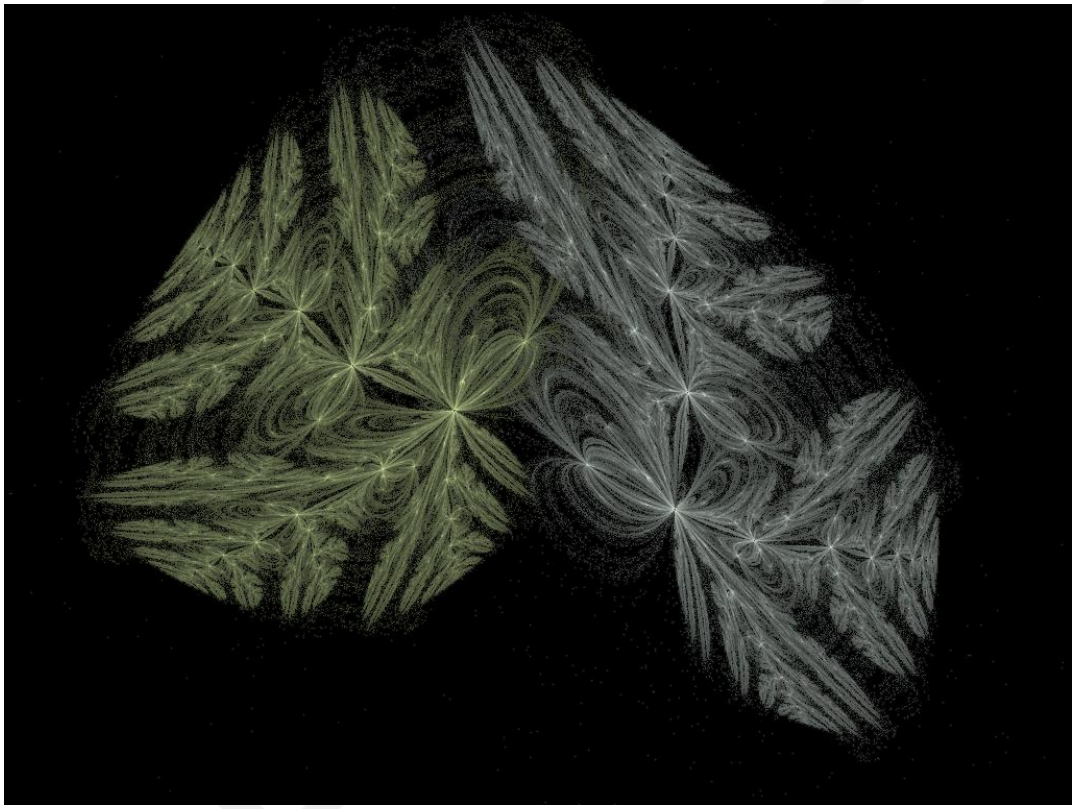

Abbildung 4.1. Ray Picking

2.2.1.2. Farb Picking mit Back Buffer

Die Idee hinter dieser Technik ist es, sich die Renderingpipeline der Grafikhardware zunutze zu machen. Jedes klickbare Objekt wird mit einer eindeutigen Id, in Form einer Farbe, versehen. Erfolgt nun ein Klick, wird in einen kleinen Viewport, der die Mausposition als Zentrum hat, die Szene gerendert. Der Farbwert unter der Maus identifiziert angeklickte Objekt eindeutig.

2.2.1.3. Picking mit Id Stack

Auch dieser Ansatz macht sich die Renderingpipeline zunutze. Ähnlich wie bei der zuvor vorgestllten Methode werden für alle klickbaren Objekte IDs vergeben. Allerdings werden diese nicht durch die Farbe codiert, sondern mit dafür vogesehenen Befehlen zusätzlich gesetzt.

Für nähere Informationen und Implementierungsbeispiele aller drei Methoden wird auf das Redbook verwiesen [1] [33].

2.2.2. Gebogene Links

Die Links werden als gewölbte Rohre dargestellt (vgl. Abbildung 4.2 [18]). Da die Positionen der Nodes variabel sind kann kein fixes 3D Modell für den Link verwendet werden. Bei der Erstpositionierung und jeder weiteren Neupositionierung einer Node, müssen immer alle damit verbundenen Links neu berechnet werden.

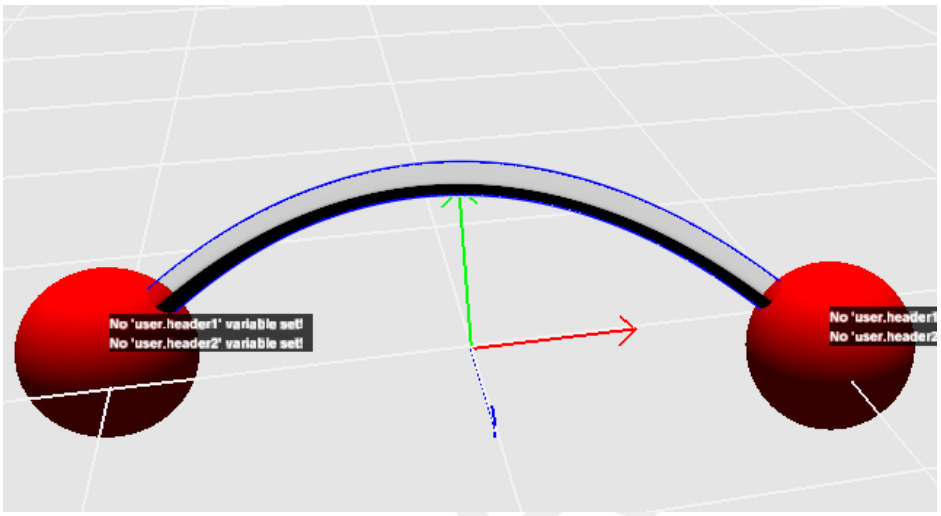

Abbildung 4.2. Ein Link aus der finalen Programmversion

Dieses "gebogene Rohr" folgt einer quadratischen Bézierkurve und wurde durch 4 Bézier Flächen dargestellt. OpenGL® bietet Evaluatoren an, um Bézier Berechnungen durchzuführen. Es werden nur die Kontrollpunkte und die Anzahl der gewünschten interpolierten Punkte angegeben und OpenGL® Evaluatoren übernehmen den Rest.

Im Zuge dieser Bakkarbeit wurde ... implementiert.

2.2.3. Linkwölbung für Pakete

Ein Link wölbt sich um alle sich darin befindenden Pakete (vgl. Abbildung 4.3 [19]).

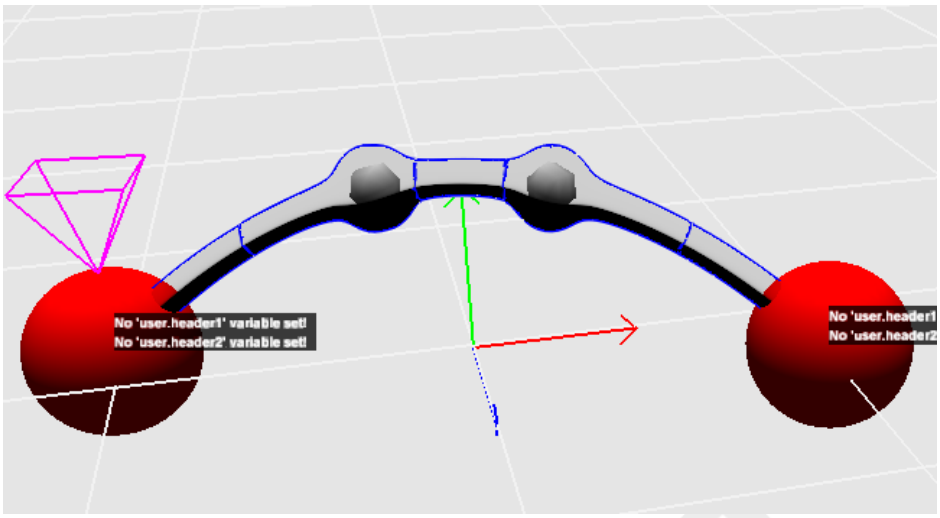

Abbildung 4.3. Ein Link mit 2 Paketen

Um dies zu realisieren müssen die Vertexdaten des gebogenen Links (vgl. Abschnitt 2.2.2 [18]) verändert werden. Da der Link durch die OpenGL® Evaluatoren erzeugt wird, sind der Anwendung nur die Kontrollpunkte, aber nicht die kompletten Vertexdaten bekannt.

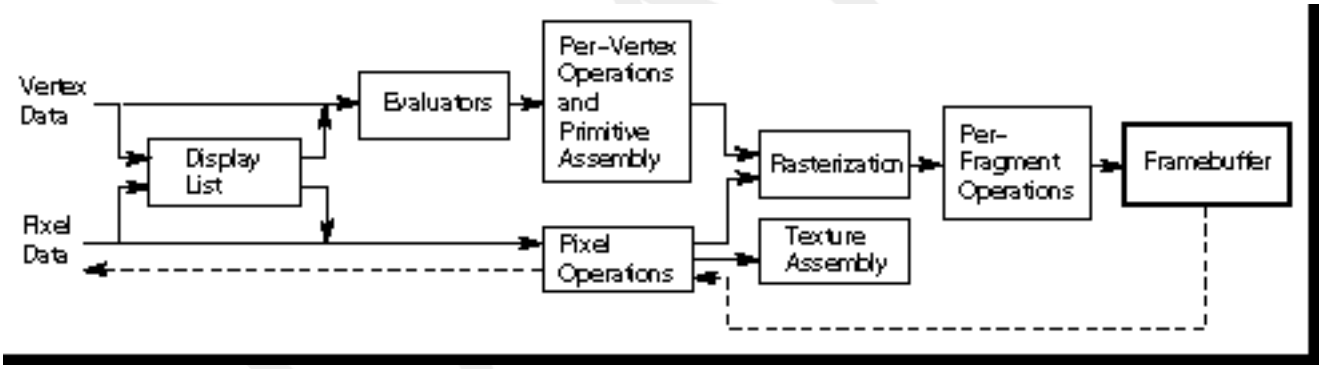

Abbildung 4.4. Die Reihenfolge der OpenGL® Operationen (Quelle: [1] [33])

Wie in Abbildung 4.4 [19] zu sehen ist, werden nach den Evaluatoren die Vertexoperationen durchgeführt. Vertexshader[1] kommen genau an dieser Stelle zum Einsatz. Deshalb wurde ein Vertexshader, mit der Aufgabe den Link um die Pakete zu vergrößern, implementiert. Der Shader nimmt als Parameter die Position von bis zu 9 Paketen und ist immer nur dann aktiv, wenn gerade ein Link gerendert wird. Die Kontrollpunkte werden von den Evaluatoren evaluiert und die berechneten Vertexdaten an den Shader übergeben. Dieser berechnet jetzt aus dem aktuellen Vertex, und alle ihm bekannten Paketpositionen, ob und wieviel sich der Vertex verschiebt.

Diese Lösung hat den Nachteil dass Systeme, die keinen Shadersupport bieten, auf diese Animation verzichten müssen.

[1] Mehr Information zu Vertexshader sind im Kapitel ??? oder in [2] [33] zu finden.

3. Module

In diesem Kapitel wird die Funktionalität der Module beschrieben.

3.1. Beschreibung

Ein Modul ist eine Simulationseinheit für den Simulator. Darin werden mehrere, mitunder verschiedene, Implementierungen von Nodes, Packets und Links zusammengefasst und ausgeführt.

3.2. Funktionsweise

Die Grundlage eines Modules bildet eine Datei, welche das Modul beschreibt. Im Folgenden wird diese Datei auch "MSIM-Datei" oder "MSIM-File" genannt.

Eine solche Datei beschreibt das Modul indem sie jede einzelne Komponente des Moduls als Instanz aufführt und darin klargestellt wird wie diese Instanzen sich zueinander verhalten.

Der Simulator ist dafür zuständig eine solche Datei zu analysieren und entsprechend der Inhalte die Instanzen der Objekte zu erstellen.

Anschließend ist der Simulator bereit das Modul auszuführen.

3.3. MSIM-Datei

Im folgenden wird die Struktur einer MSIM-Datei erläutert.

3.3.1. Beschreibung

Eine MSIM-Datei wird eine Datei mit der Endung .msim genannt. Diese kann dazu verwendet werden um eine Struktur eines Modules zu beschreiben. Man gibt dabei Nodes, Links und die Verbindungen dazwischen an.

```
<module>
 <id>Module ID</id>
 <classpath>vidis.modules.someModule</classpath>
 <nodeDensity>0.5</nodeDensity>
 <objects>
  <node>
   <id>node01</id>
   <class>SomeNode</class>
  </node>
  <node>
   <id>nodeWithVariables</id>
   <class>SomeNode</class>
   <variables>
    <someVariable>value</someVariable>
   </variables>
  </node>
  <link>
   <id>link01</id>
   <class>SomeLink</class>
   <delay>5</delay>
  </link>
 </objects>
 <connections>
  <connection>
   <nodeA>node01</nodeA>
   <nodeB>nodeWithVariables</nodeB>
   <link>link01</link>
 </connections>
</module>
```

Beispiel 4.1. Beispiel für eine MSIM-Datei

3.3.2. Format

Im Folgenden werden die XML Knoten einer MSIM-Datei beschrieben. [2]

3.3.2.1. Module

Dies ist das Wurzelelement. Es beinhaltet 5 Kindelemente. Es folgt eine Auflistung dieser Elemente.

[2] Alle Schlüssel der XML-Elemente sind nicht case-sensitiv, d.h. es macht z.B. keinen Unterschied wenn der Wurzelschlüssel "module" oder "Module" oder "MoDuLe" benannt wird.

3.3.2.1.1. ID

Dieses Element beinhaltet einen String, welches dieses Modul beschreibt. [3]

```
<id>someId</id>
```

Beispiel 4.2.

3.3.2.1.2. Classpath

Dieses Element beinhaltet einen String zum classpath der Modul Klassen.

```
<classpath>some.specific.classpath.where.all.module.classes.reside</classpath>
```

Beispiel 4.3. Ein ausführliches Beispiel für ein Classpath Element

```
my.vidis.modules.mymodule
```

Beispiel 4.4. Ein Beispiel für einen Classpath

3.3.2.1.3. Objects

Dieses Element beinhaltet alle zu ladenden Link- und Node-Objektinstanzen.

3.3.2.1.3.1. Node

Dieses Element beschreibt eine Node-Objektinstanz.

```
<node>
 <id>uniqueNodeId</id>
 <class>implementedClassName</class>
</node>
```

Beispiel 4.5. Beispiel für ein Node Element mit Variablen

[3] z.B. Demo Module 1

```
<node>
 <id>uniqueNodeId</id>
 <class>implementedClassName</class>
 <variables>
  <var1>var1Value</var1>
  <var2>var2Value</var2>
 </variables>
</node>
```

Beispiel 4.6. Beispiel für ein Node Element mit Variablen

3.3.2.1.3.1.1. ID

Ein String, welcher diese Instanz eindeutig (UNIQUE) beschreibt.

3.3.2.1.3.1.2. CLASS

Ein String, welche die Klasse dieser Instanz beschreibt. Zusammen mit dem Classpath-Element ist dies der Classpath der zu ladenden Klasse.

3.3.2.1.3.1.3. Variables

Dieses optionale Element beinhaltet Variablendefinitionen, welche beim Laden dieses Moduls gesetzt werden sollen.

3.3.2.1.3.1.3.1. Variable

Dieses Element repräsentiert eine Variable. Dabei ist der Schlüssel der Name der Variable und der Wert der Variable ist ein String.

```
<variableName>variableValue</variableName>
```

Beispiel 4.7. Beispiel für eine Variable

```
<variables>
 <name>some name</name>
 <var2>I am variable 2</var2>
</variables>
```

Beispiel 4.8. Ein generelles Beispiel

```
<node>
 <id>node01</id>
 <class>SomeNodeClass</class>
 <variables>
  <name>Node 01</name>
 </variables>
</node>
```

Beispiel 4.9. Ein Beispiel für Variablen auf einer Node

3.3.2.1.3.2. Link

Dieses Element beschreibt eine Link-Objektinstanz.

```
<link>
 <id>uniqueLinkId</id>
 <class>implementingClass</class>
 <delay>5</delay>
</link>
```

Beispiel 4.10. Beispiel für ein Link-Element

```
<link>
 <id>uniqueLinkId</id>
 <class>implementingClass</class>
 <delay>5</delay>
 <variables>
  <var1>var1value</var1>
  <var2>var2value</var2>
 </variables>
</link>
```

Beispiel 4.11. Beispiel für ein Link-Element mit Variablen

3.3.2.1.3.2.1. ID

Ein String, welcher diese Instanz eindeutig (UNIQUE) beschreibt.

3.3.2.1.3.2.2. CLASS

Ein String, welche die Klasse dieser Instanz beschreibt. Zusammen mit dem Classpath-Element ist dies der Classpath der zu ladenden Klasse.

3.3.2.1.3.2.3. DELAY

Eine Zahl im Wertebereich eines Java Long Integers, welcher den Delay (die Länge des Links) beschreibt.

3.3.2.1.3.2.4. Variables

Dieses optionale Element beinhaltet Variablendefinitionen, welche beim Laden dieses Moduls gesetzt werden sollen.

3.3.2.1.3.2.4.1. Variable

Dieses Element repräsentiert eine Variable. Dabei ist der Schlüssel der Name der Variable und der Wert der Variable ist ein String.

```
<variableName>variableValue</variableName>
```

Beispiel 4.12. Beispiel für eine Variable

```
<variables>
 <name>some name</name>
 <var2>I am variable 2</var2>
</variables>
```

Beispiel 4.13. Ein generelles Beispiel

```
<node>
 <id>node01</id>
 <class>SomeNodeClass</class>
 <variables>
  <name>Node 01</name>
 </variables>
</node>
```

Beispiel 4.14. Ein Beispiel für Variablen auf einer Node

3.3.2.1.4. Connections

Dieses Element beinhaltet alle Verbindungen, welche zwischen Nodes mit Hilfe von Links hergestellt werden sollen.

```
<connections>
 <connection>
  <nodeA>A</nodeA>
  <nodeB>B</nodeB>
  <link>L</link>
 </connection>
</connections>
```

Beispiel 4.15. Beispiel für eine einzelne Verbindung zwischen Node A und Node B über Link L.

```
<connections>
 <connection>
  <nodeA>node0</nodeA>
  <nodeB>node1</nodeB>
  <link>link0-1</link>
 </connection>
 <connection>
  <nodeA>node1</nodeA>
  <nodeB>node2</nodeB>
  <link>link1-2</link>
 </connection>
 <connection>
  <nodeA>node3</nodeA>
  <nodeB>node4</nodeB>
  <link>link3-4</link>
 </connection>
 <connection>
  <nodeA>node4</nodeA>
  <nodeB>node0</nodeB>
  <link>link4-0</link>
 </connection>
</connections>
```

Beispiel 4.16. Beispiel mit mehreren Verbindung zwischen diversen Nodes in einer Ring Topologie.

3.3.2.1.4.1. Connection

Dieses Element beschreibt eine Verbindung zwischen zwei Nodes.

3.3.2.1.4.1.1. NodeA

Die id der ersten Node. [4]

3.3.2.1.4.1.2. NodeB

Die id der zweiten Node. [5]

3.3.2.1.4.1.3. Link

Die id des zu verwenden Links. [6]

[4] z.B. node01
[5] z.B. node02
[6] z.B. link_node01_node02

3.3.2.1.5. NodeDensity

Dieses Element beinhaltet einen Zahlwert im Wertebereich [0..1] mit Nachkommastellen. Er beschreibt wie nahe die Knoten zu einander stehen sollen. 0 ist dabei der Minimalabstand, 1 der Maximalabstand. Dieser Parameter beeinflußt direkt die vorhandenen Layouts.

```
<nodeDensity>0.5</nodeDensity>
```

Beispiel 4.17. Beispiel für ein NodeDensity Element.

4. Grafische Benutzeroberfläche

Um dem Benutzer die Möglichkeit zu geben, mit der Simulation zu interagieren, wurde eine grafische Benutzeroberfläche entworfen und umgesetzt. Diese wird in den folgenden Kapiteln kurz vorgestellt und erklärt.

4.1. GUI Elemente

Im folgenden wird auf die sichtbaren Benutzeroberflächen-Elemente näher eingegangen.

4.1.1. Menu

Am unteren linken Rand des Bildschirms befindet sich das Menu. Mit diesem kann der Benutzer Einfluss auf die Simulation ausüben.

Das Menu ist durch ein Composite Pattern realisiert. Jede Node fasst ein Steuerelement, eine Liste der Kinderelemente und das Vaterelement. Zudem verfügt sie noch über die Information ob die Kinderelemente dargestellt, oder ausgeblendet werden sollen. Wird eine Node aufgeklappt, rutschen alle Geschwisterelemente nach oben und dazwischen werden ihre Kinder gerendert.

Auf der Hauptebene dieser Struktur befinden sich 3 dauerhaft sichtbare Elemente: Im folgenden werden diese Element (von unten nach oben) erklärt:

4.1.1.1. Die Player Kontrollen

Die Player Kontrollen befinden sich ganz unten am Bildschirmrand. Sie sind immer sichtbar und werden nicht durch das Menu verschoben. Sie bestehen aus den 3 Buttons Play, Pause und Reset (vgl. Abbildung 4.5 [28]).

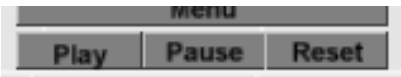

Abbildung 4.5. Bildschirmfoto von den Playerkontrollen

4.1.1.2. Das Systemmenu

Im Systemmenu können Module geladen und Einstellungen vorgenommen werden (vgl. Abbildung 4.6 [29]).

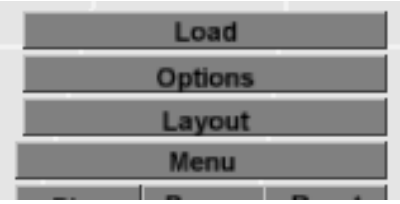

Abbildung 4.6. Bildschirmfoto vom Systemmenu

- Optionen

  Hier kann unter anderem in den Wireframe Modus gewechselt werden.
- Layout

  In diesem Untermenu kann man zwischen den vordefinierten Layouts wählen.
- Laden

  Dieses Untermenu stellt alle verfügbaren Module dar.

4.1.1.3. Das Aktionsmenu

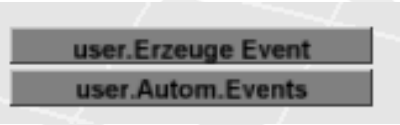

Abbildung 4.7. Bildschirmfoto vom Aktionsmenu

Für die aktuell selektierte Node werden hier Aktionen zur Verfügung gestellt. Dies äußert sich durch einen Button pro möglicher Aktion. Voraussetzung dafür ist das der Modul-Entwickler in der Node Implementation Methoden vom Typ void mit @Display annotiert hat (vgl. Beispiel 4.18 [29]).

```
@Display( name="Erzeuge Event" )
public void erzeugeEvent() {
 createSomeEvent();
}

@Display( name="Autom.Events" )
public void toggleAutoEvents {
 autoEvents = !autoEvents;
}
```

Beispiel 4.18. Auszug aus einer Modul-Node-Implementation

Es wird zwischen 2 Arten von Methoden unterschieden:

- Methoden ohne Parameter

Methoden ohne Parameter werden durch einen einfachen Button dargestellt. Klickt man darauf wird direkt die vom Modul-Entwickler implementierte Methode ausgeführt.

- Methoden mit Parameter

  Diese Methode kann ohne vorherige Eingabe der Parameter nicht ausgeführt werden. Darum werden durch einen Klick auf diesen Button Felder für alle benötigten Parametern, sowie eine "ausführen" Button dargestellt.

4.1.2. Informations Panel

Das Informationspanel befindet sich auf der rechten Seite. Es zeigt den aktuellen Status von den ausgewählten Objekten an. Für jede Methode (mit Rückgabewert), die vom Modul-Entwickler mit @Display annotiert wurde, ist hier eine Zeile mit Namen und Wert zu sehen.

4.2. Maus Interaktion

4.3. Tastatur Interaktion

Kapitel 5. Conclusions

Dominik stellt sich folgendes vor: [VORSCHLAG (BLENDTEXT)> VIDIS ist ein einfaches und tolles Werkzeug für didaktische Lehrzwecke und erfüllt somit unsere Erwartungen. -: für größere Netzwerke geht die Grafik in die Knie -: der Simulator könnte performanter sein weil 1] interne node/packet/link klassenstruktur ist eigentlich überflüssig 2] man könnte alles über Adjazenzlisten beschreiben und so sehr viel performance gewinnen 3] sei artig und hör auf simon, wenn simon sagt -: eclipse 64bit ist scheiße -: performance optimizer ist teilweise scheiße (generiert display listen wo es gscheider wär endlich a ruah zu gebm) -: (will niemand wissen) -: java und openGL ist ein 2schneidiges schwert, das sehr schnell deine computer-gedärme in stücke reißt In Summe steht hier da ein funktionierendes Programm für die Vorlesung "Verteilte Systeme", genau wie gewollt weil ja das Maxerl sehr geholfen hat. >VORSCHLAG]

Literaturverzeichnis

[1] *OpenGL® Programming Guide*. The Official Guide to Learning OpenGL®, Version 2. Fifth Edition. 0-321-33573-2. Dave Shreiner, Mason Woo, Jackie Neider und Tom Davis.

[2] *OpenGL® Shading Language*. Second Edition. 0321-33489-2. 978-321-33489-3. Randy J. Rost.

[3] *Distributed Systems*. Principles and Paradigms. Second Edition. 0-13-613553-6. Andrew S. Tanenbaum und Maarten Van Steen.

[4] http://www.opengl.org .

[5] https://java3d.dev.java.net/ .

[6] https://jogl.dev.java.net/ .

[7] http://jcp.org/en/jsr/detail?id=231 .

[8] http://lwjgl.org/ .

[9] *„Birds of Feather“ Presentation SIGGRAPH 2007*. http://www.khronos.org/developers/library/2007_08_siggraph/siggraph%202007%20bof.pdf .

[10] <linke></linke> .

Glossar

S

Shader	Programm, dass in der Grafikhardware ausgeführt wird.