

Anwendung zur Darstellung einer verteilten, virtuellen Welt

Patrick Pauchard
Engelhardstrasse 30A
3280 Murten
E-Mail: patrick.pauchard@unifr.ch

Diplomarbeit
Universität Freiburg, Schweiz
Wirtschafts - und Sozialwissenschaftliche Fakultät
Betreuer: Patrik Fuhrer
Referent: Prof. Dr. Jacques Pasquier-Rocha

2. April 2004

Zusammenfassung

Die Arbeit stellt einen Editor vor, welcher zur Erstellung und Darstellung einer verteilten, virtuellen Welt dient. Der beschriebene Editor ermöglicht es, eine gewünschte Strukturierung der virtuellen Welt zu definieren, diese im Format XML abzuspeichern und sie entsprechend der Definition auf den gewünschten Rechnern zu installieren.

Inhaltsverzeichnis

1	Einleitung	4
1.1	Über MaDViWorld	4
1.2	Problemstellung	6
1.3	Zielsetzung und Lösungsansätze	7
1.4	Dokumentaufbau	8
1.5	Konventionen	8
2	XML	9
2.1	Definition	9
2.2	Document Type Description (DTD)	9
2.3	XML Schema	10
2.4	Parser	11
2.5	XML und MaDViWorld	11
2.5.1	DTD einer virtuellen Welt	12
2.5.2	XML Schema einer virtuellen Welt	13
2.5.3	Ein XML-Dokument einer virtuellen Welt	14
2.5.4	MaDViWorld und DOM	14
3	Aufbau und Funktion des Editors	17
3.1	Worldpanel	18
3.2	XML-Panel	18
3.3	Helppanel	19
3.4	Menüs und Toolbar	20
3.4.1	Statuslabel	23
4	Programmerklärung	24
4.1	Paket viworld	24

4.1.1	Struktur der virtuellen Welt als Composite-Pattern	24
4.1.2	Implementierung der virtuellen Welt	25
4.2	Paket <code>gui</code>	27
4.2.1	Strukturen zur Darstellung von speziellen virtuellen Welten	27
4.2.2	Algorithmen zur optimierten Darstellung von virtuellen Welten	29
4.2.3	XML-Highlighting	34
4.2.4	Installation einer Welt	34
5	Zusammenfassung	36
5.1	Ergebnis	36
5.2	Verbesserungsvorschläge	36
A	Installation und Anwendung	41
A.1	MaDViWorld-Framework installieren	41
A.2	Konfiguration	41
A.3	Ant installieren	41
A.4	Starten der Anwendungen	42
B	UML	43
B.1	Klassendiagramm <code>viworld</code>	43
B.2	Klassendiagramm <code>gui</code>	44
C	CD-ROM	45
D	Eidesstaatliche Erklärung	46

Kapitel 1

Einleitung

Dieses Kapitel gibt eine kurze Einführung über das Framework MaDViWorld und beschreibt anschliessend Probleme, welche die aktuelle Version dieses Frameworkes mit sich bringt. Anhand diesen Problemen werden anschliessend Lösungsansätze und das Ziel dieser Arbeit bestimmt. Zudem erklärt dieses Kapitel, wie diese Dokumentation aufgebaut ist.

1.1 Über MaDViWorld

Diese Arbeit befasst sich mit virtuellen Welten. Eine virtuelle Welt ist laut Definition ein System, in welchem mehrere Benutzer und aktive Objekte in einem Raum agieren können, wobei sie einen direkten Einfluss aufeinander haben. In einem Chat beispielsweise können sich mehrere Benutzer miteinander unterhalten.

Das Web hingegen gilt nicht als eine virtuelle Welt. Dokumente können nur kopiert werden und auf dem eigenen, lokalen Rechner angesehen werden. Dies geschieht unabhängig zu den anderen Benutzern und er kann keinen Einfluss auf diese Dokumente nehmen. Die Dokumente können nicht wie in einer virtuellen Welt direkt bearbeitet werden.

Oft werden die gängigen virtuellen Welten nicht etwa wie das Web von einem zentralen Server aus verwaltet. Fällt dieser Server aus, funktioniert die ganze virtuelle Welt nicht mehr. Sie ist von einem einzigen Server abhängig. Zudem ist eine virtuelle Welt nicht unendlich ausbaubar, da die ganze Speicherkapazität von einem Server abhängt.

Die Software Entwicklungsgruppe der Universität Freiburg [1] modellierte das Framework **Massive Distributed Virtual World** (MaDViWorld), welches diese Schwachpunkte der gängigen virtuellen Welten vermeiden sollte. Eine solche verteilte, virtuelle Welt wird nicht von einem zentralen Server aus verwaltet, sondern wie es der Name schon sagt verteilt von ver-

schiedenen Rechner, wobei auf jedem dieser Rechner ein Serverprogramm gestartet werden muss. Eine virtuelle Welt, welche mit dem MaDViWorld kreiert wurde, besteht aus verschiedenen Räumen. Diese Räume können untereinander mit Türen verbunden sein und sie können verschiedene Objekte beinhalten. Sogenannte Avatars können sich in den Räumen bewegen und von den Objekten Gebrauch machen. Es wurden bereits etliche solche Objekte implementiert. So kann der Benutzer verschiedene Spiele wie etwa ein Schach oder ein Chat in einem Raum plazieren und von diesen Gebrauch machen.

Abbildung 1.1 stellt graphisch ein physikalisches Modell einer solchen virtuellen Welt dar, welche aus 2 Räume und 3 Avatars besteht. Die Abbildung verdeutlicht ebenfalls wie die verschiedenen Rechner über ein Netz verbunden sind.

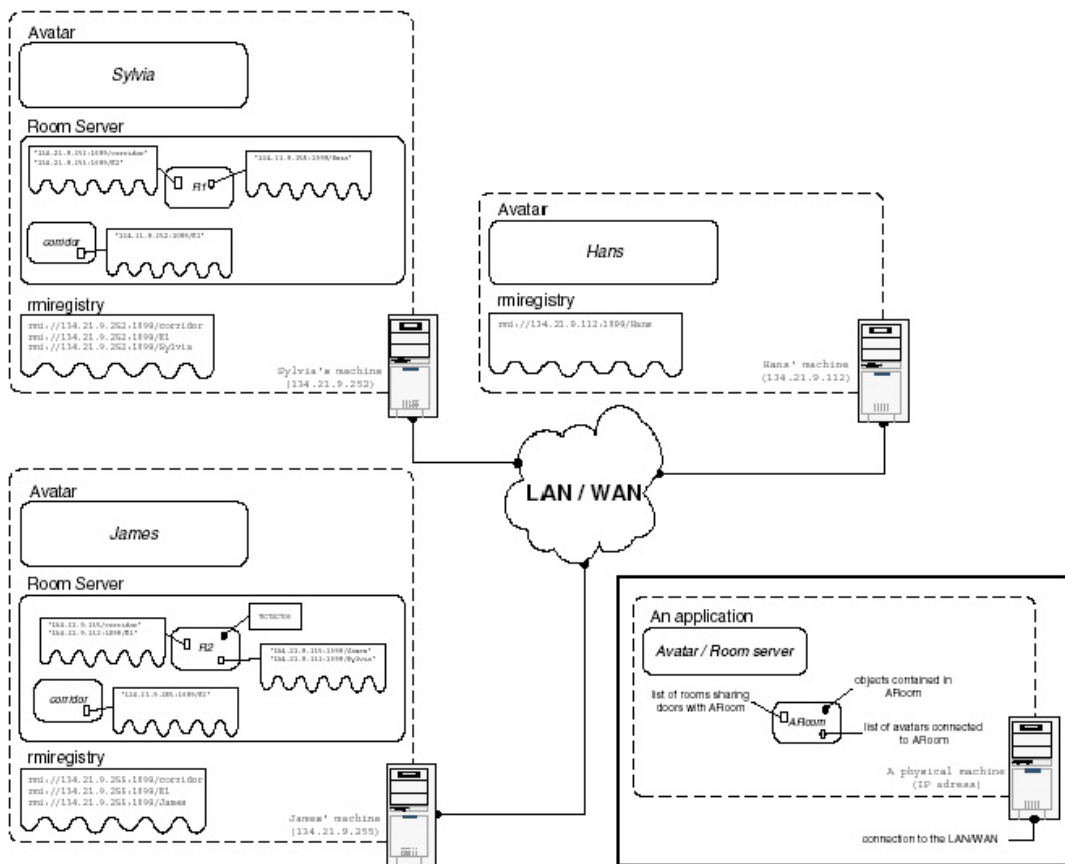


Abbildung 1.1: Modell einer einfachen Welt [4]

1.2 Problemstellung

Ein Problem des aktuellen Frameworks betrifft das Installieren der Räume. Um Räume zu installieren, wird dabei eine Anwendung gestartet, welches das Fenster in der Abbildung 1.2 aufruft. Das Fenster listet alle Räume mit dessen Objekten und Türen auf den erreichbaren, installierten Servern auf. Diese Art die Räume zu installieren, wird sehr aufwendig, sobald

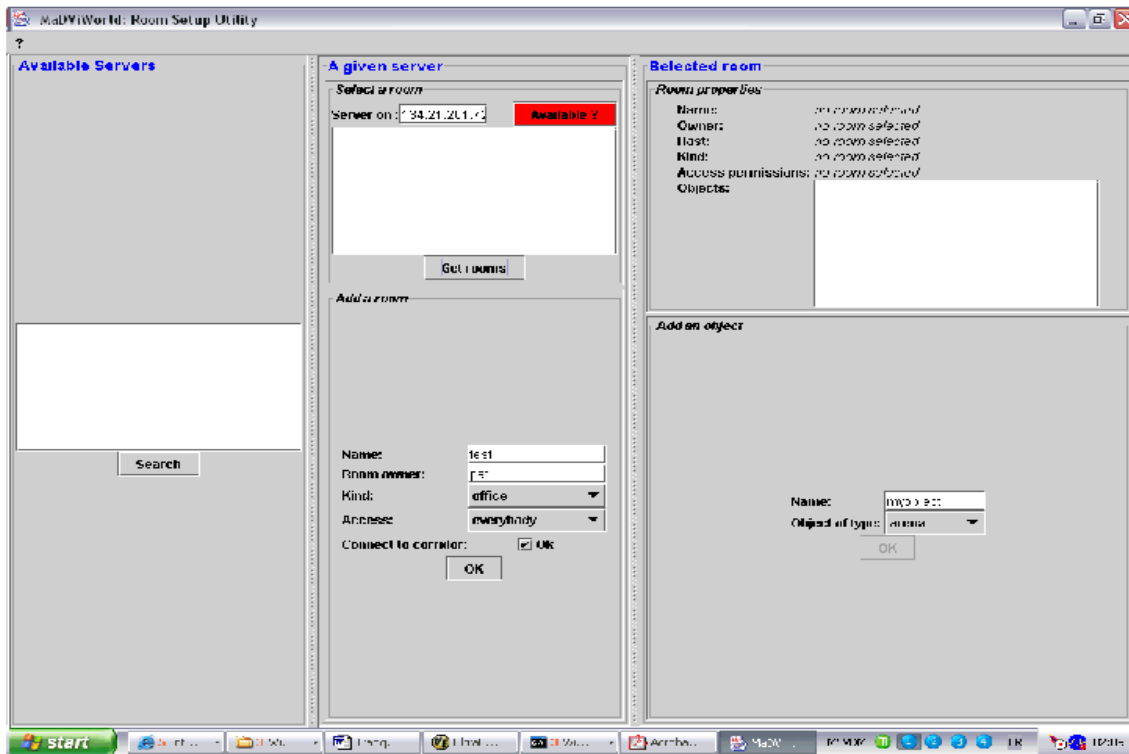


Abbildung 1.2: Erstellen von Räumen

der Benutzer eine virtuelle Welt mit grösseren Strukturen wie etwa einem Stern oder einer vollständiger Vernetzung erstellen möchte. Dabei muss er jeden einzelnen Raum und jedes einzelne Objekt nacheinander installieren. Auch die Definitionen der Türen ist dabei recht mühsam.

Ein weiterer Nachteil, welche die gängige Methode mit sich bringt, eine virtuelle Welt zu kreieren, ist der, dass der Benutzer leicht den Überblick über die konzeptuelle Welt verlieren kann. Eine Topologie ist nur schwer zu erahnen.

Schliesslich besteht keine Möglichkeit eine Struktur einer virtuellen Welt abzuspeichern, damit der Benutzer sie bei einer nächsten Sitzung wieder genau gleich installieren kann. Der Benutzer muss jeden Raum und jedes Objekt jedes Mal neu definieren.

1.3 Zielsetzung und Lösungsansätze

Als Lösungsansatz für die oben beschriebenen Probleme des aktuellen MaDViWorld-Frameworks soll in dieser Arbeit ein graphischer Editor mit der Programmiersprache JAVA entwickelt und beschrieben werden, welcher das automatische generieren von komplexen Welten und deren graphische Darstellung ermöglichen sollte. Zudem sollte es mit diesem Editor möglich sein, die Struktur einer virtuellen Welt abzuspeichern. Der Editor sollte die Funktionsweisen eines gängigen Editors wie etwa ein Menu oder einer Toolbar aufweisen, welche die Handhabung möglichst komfortabel und praktisch gestalten sollen. Auf Programmebene sollten die grundlegenden Regeln einer sauberen objektorientierten Programmierung eingehalten werden. Dazu zählt sicherlich eine saubere Architektur und ein gut dokumentierter Code, damit der Editor in Zukunft auf zusätzliche Bedürfnisse ausgebaut werden kann.

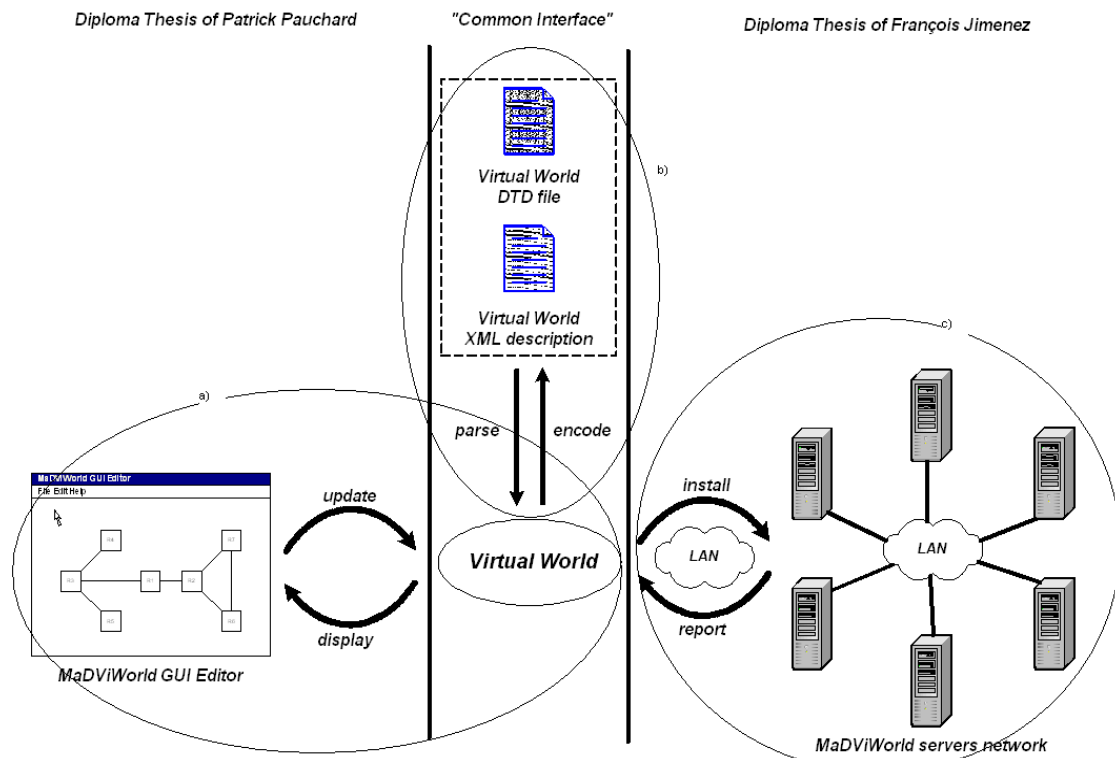


Abbildung 1.3: Struktureller Aufbau der Arbeit

Damit diese Ziele verwirklicht werden können, sollte der Editor aus drei Teilen bestehen:

- a) Der erste Teil sollte die Welt graphisch darstellen. Dabei sollte der Benutzer die Möglichkeit haben, neue Räume und Türen zu kreieren und dessen Merkmale wie zum Beispiel der Name des Raumes zu editieren.

- b) Der zweite Teil des Editors sollte zur Darstellung eines XML-Dokuments dienen, welches die Struktur einer virtuellen Welt beschreibt und es auch erlaubt, diese so abzuspeichern.
- c) Der dritte Teil der Arbeit umfasst die Installation der Welt auf die verschiedenen Rechner. Dieser Teil des Editors wird nicht direkt in dieser Arbeit behandelt, sondern wurde von François Jimenez in seiner Diplomarbeit [2] implementiert.

Abbildung 1.3 zeigt auf, wie die oben genannten Teile miteinander zusammenhängen.

1.4 Dokumentaufbau

Die Dokumentation befasst sich zunächst mit der Markupsprache XML, welches es ermöglicht, eine virtuelle Welt zu beschreiben und sie abzuspeichern. Dem Leser wird eine Definition dieser Sprache gegeben, anschliessend wird gezeigt inwiefern XML in der Arbeit verwendet wurde. Im anschliessenden Kapitel wird dem Leser der Editor in einer Benutzeranleitung beschrieben, bevor näher in dessen Programm eingegangen wird. Abgeschlossen wird die Arbeit mit einer Zusammenfassung und auf den Editor bezogenen Verbesserungsvorschläge.

1.5 Konventionen

Abbildungen werden jeweils mit 2 Zahlen durchnummeriert (z.B. Abbildung 1.2). Dabei gibt die erste Zahl an, in welchem Kapitel sich die Abbildung befindet und die zweite Zahl, die wievielte Abbildung sie innerhalb des Kapitels ist.

Verweise auf Bücher und Webseiten in der Dokumentation werden mit einer Zahl in einer eckigen Klammer gekennzeichnet. Diese verweisen auf den entsprechenden Eintrag im Literaturverzeichnis, in welchem diese Literatur näher bestimmt wird.

Auszüge aus Programme und Dokumente werden in einer anderen Schriftart geschrieben.

Kapitel 2

XML

Dieses Kapitel gibt einen kurzen Einblick in die Markup-Sprache XML. Näheres über diese Sprache kann in [12] und [13] nachgeschlagen werden. Der zweiten Teil dieses Kapitels zeigt auf, wie diese Art Strukturen zu beschreiben, einen wichtigen Bestandteil in dieser Arbeit einnimmt.

2.1 Definition

XML ist ein Standard für die Modellierung hierarchischer Daten. Es erlaubt Daten und ihre Strukturen zu beschreiben. Ein Ziel dieser Sprache ist es im Gegensatz zum HTML, Inhalt und Layout zu trennen. Ein XML-Dokument besteht aus XML-Elementen, welches durch einen Anfangs-Tag und einem End-Tag sowie dazwischenliegenden Daten aufgebaut ist. Nachfolgend ist ein Beispiel eines solchen XML-Elements aufgeführt, wobei das Element als Wert den Namen eines Raumes hat:

```
<roomname>R1</roomname>
```

Ein XML-Dokument gilt dann als wohlgeformt, wenn jedes Anfangs-Tag mit einem End-Tag abgeschlossen wird und wenn sich hierarchisch ineinander verschachtelte Tags nicht überlappen, das heisst ein Tag, der innerhalb eines anderen geöffnet wird, muss auch innerhalb wieder geschlossen werden.

2.2 Document Type Description (DTD)

Eine DTD definiert die erlaubten Tags eines XML-Dokuments und bestimmt damit dessen Struktur. Sie legt fest, wie die einzelnen XML-Elemente aufgebaut sind, z.B:

```
<!ELEMENT world (worldname,room*)>
```

Dieser Ausdruck besagt, dass das Element `<world>` aus den Elementen `<worldname>` und `<room>` besteht. Wie aus diesem Beispiel ersichtlich ist, können den Elementen jeweils auch verschiedene Zeichen nachgestellt werden:

- *: Das Element kommt 0 oder mehrmals vor
- ?: Das Element kommt optional vor
- +: Das Element kommt 1 oder mehrmals vor

Wird kein Zeichen hinter das Element gesetzt, heisst das, dass dieses Element genau einmal vorkommen muss. Das Element `<world>` besitzt folgendermassen genau ein Element `<worldname>` und keines oder mehrere Elemente `<room>`. Eine virtuelle Welt hat also laut dieser Definition einen Namen und kann mehrere Räume beinhalten.

Wenn ein XML-Dokument wohlgeformt ist und zudem eine DTD besitzt, ist das Dokument valid.

2.3 XML Schema

Die im vorhergehenden Abschnitt beschriebene DTD weist einige Mängel auf. So können den Elementen keine Datentypen zugewiesen werden und die DTD besteht nicht aus der XML Syntax. Diese Mängel wurden durch das XML Schema behoben.

Die Schemas werden in einem `xsd`-Dokument abgespeichert. In Code 2.5.2 wurde die DTD aus dem Code 2.5.1 in ein XML-Schema umgewandelt.

Aus diesem Beispiel eines XML-Schemas ist ersichtlich, dass für jedes Element ein Tag definiert wird, in welchem der Name und der Typ des Elements festgelegt wird. Das folgende Beispiel zeigt eine solche Definition eines Elementes:

```
<xsd:element name=worldname type=xsd:string/>
```

Dieses Element hat den Namen `worldname` und besteht aus einer Zeichenkette (`string`).

Es wird grundsätzlich zwischen einfachen und komplexen Typen unterschieden. Zu den einfachen Typen gehören neben der oben erwähnten Zeichenkette auch Zahlen, Datum, usw. Die komplexen Typen enthalten normalerweise eine Menge von Element-Deklarationen. So besteht der neu definierte `worldTyp` aus den noch zu definierenden Elementen `worldname` und `room`:

```
<xsd:complexType name= worldTyp
<xsd:sequence>
<xsd:element name=worldname type=xsd:string/>
<xsd:element name=room type=roomType/>
</xsd:sequence>
```

```
</xsd:complexType>
```

Wie aus Code 2.5.2 ersichtlich ist, werden Häufigkeitsbeschränkungen im XML-Schema anders gehandhabt als bei der DTD. Den Attributen `minOccurs` und `maxOccurs` werden je nach erlaubter Anwendung Werte zugewiesen. Die folgende Liste vergleicht die Ausdrucksweisen der beiden Definitionsarten von XML-Dokumenten:

- *: `minOccurs=0`, `maxOccurs=unbounded`
- ?: `minOccurs=0`
- + `maxOccurs=unbounded`

Ein Tutorial über das XML Schema ist in [15] zu finden.

2.4 Parser

Software, welche es ermöglichen, XML-Dokumente zu durchlaufen und aus ihnen eine hierarchische Struktur zu kreieren, werden Parser genannt. Es wird zwischen nicht-validierenden und validierenden Parser unterschieden. Ein nicht-validierender Parser überprüft, ob ein XML-Dokument wohlgeformt ist. Ein validierender Parser verifiziert zusätzlich, ob das XML-Dokument valid ist.

XML-Parser können auf zwei grundlegende Arten Zugriff auf ein XML-Dokument herstellen:

- SAX basiert auf spezielle Ereignisse, welche beim Parsen auftreten.
- Durch DOM erzeugt der Parser ein Baum aus dem XML-Dokument, welche das Navigieren der Knoten erleichtert. Einzelene Knoten können entfernt, verändert oder hinzugefügt werden.

Das ganze Object Model steht beim DOM die ganze Zeit im Speicher zur Verfügung. Darum ist SAX schneller und einfacher als DOM, ist aber nur dann geeignet, wenn das XML-Dokument nur einmal durchlaufen werden soll.

2.5 XML und MaDViWorld

XML dient in dieser Arbeit dazu, eine virtuelle Welt zu beschreiben und abzuspeichern. Die Struktur eines XML-Dokuments ist leicht ausbaubar, das heisst, dass bei einer Definitionsänderung der virtuellen Welt die DTD leicht angepasst werden kann. Dieser Abschnitt geht auf die aktuelle Version der DTD einer virtuellen Welt ein und zeigt ein Beispiel einer solchen Welt. Zudem wird beschrieben, wie die Umwandlung eines `world`-Objektes in ein XML-Dokument verläuft.

2.5.1 DTD einer virtuellen Welt

Code 2.5.1 DTD einer virtuellen Welt

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT world (worldname,room*)>
<!ELEMENT room (roomname,doorlist?,objectlist?, access, owner,
kind, host, coord?)>
<!ELEMENT doorlist (door*)>
<!ELEMENT objectlist (object*)>
<!ELEMENT door (doorHost, doorName)>
<!ELEMENT object (objectname, description, type)>
<!ELEMENT coord (dim+)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT kind (#PCDATA)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT doorHost (#PCDATA)>
<!ELEMENT doorName (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT roomname (#PCDATA)>
<!ELEMENT worldname (#PCDATA)>
<!ELEMENT objectname (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT information (#PCDATA)>
<!ELEMENT access (#PCDATA)>
<!ELEMENT dim (#PCDATA)>
```

Die DTD einer virtuellen Welt ist in Code 2.5.1 aufgelistet. Sie wurde anhand der Struktur einer virtuellen Welt im Framework definiert. Eine virtuelle Welt kann demzufolge aus mehreren Räumen bestehen, welche selber verschiedene Objekte und Türen untereinander beinhalten können. Diese einzelnen Bestandteile einer virtuellen Welt oder eben Elemente sind durch einige Merkmale näher definiert. Nachfolgend werden die wichtigsten Merkmale erklärt.

Element	Beschreibung
----------------	---------------------

access	Dient der Sicherheit und bestimmt welche Personen und Personengruppen Zutritt zu den Räumen haben
host	IP-Adresse des Rechners, auf welcher der Raum installiert werden soll
type	Objekttyp, welcher nur existierende Werte annehmen kann (z.B. chat, clock, ...)

Ausserdem können die Räume ein `<coord>`-Element besitzen, welches, wie später näher erläutert wird, zur Positionierung im graphischen Teil des Editors dient. Ein `<coord>`-Element besitzt ein oder mehrere `<dim>`-Elemente, je nachdem wieviele Dimensionen zur Darstellung der Räume erforderlich sind. Für die aktuelle Version des Editors sind 2 Dimensionen vorgesehen.

2.5.2 XML Schema einer virtuellen Welt

Code 2.5.2 wie das XML Schema einer virtuellen Welt aussehen könnte. Dieses Schema wurde von der DTD des vorhergehenden Abschnitts abgeleitet.

Code 2.5.2 XML Schema einer virtuellen Welt

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="world" type="worldTyp"/>
<xsd:element name="Kommentar" type="xsd:string"/>
<xsd:complexType name="worldTyp">
  <xsd:sequence>
    <xsd:element name="worldname" type="xsd:string"/>
    <xsd:element name="room" minOccurs="0"
      maxOccurs="unbounded" type="roomType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="roomType">
  <xsd:sequence>
    <xsd:element name="roomname" type="xsd:string"/>
    <xsd:element name="doorlist" minOccurs="0">
<xsd:sequence>
  <xsd:element name="door" minOccurs="0">
<xsd:sequence>
  <xsd:element name="doorHost" type="xsd:string"/>
  <xsd:element name="doorName" type="xsd:string"/>
</xsd:sequence>
</xsd:element>
</xsd:sequence>
</xsd:element>
    <xsd:element name="objectlist" minOccurs="0">
      <xsd:sequence>
        <xsd:element name="object" minOccurs="0">
          <xsd:sequence>
            <xsd:element name="objectname" type="xsd:string"/>
            <xsd:element name="description" type="xsd:string"/>
          </xsd:sequence>
        </xsd:element>
      </xsd:sequence>
    </xsd:element>
    <xsd:element name="type" type="xsd:string"/>
  </xsd:sequence>
</xsd:element>
  <xsd:element name="access" type="xsd:string"/>
  <xsd:element name="owner" type="xsd:string"/>
  <xsd:element name="kind" type="xsd:string"/>
  <xsd:element name="host" type="xsd:string"/>
  <xsd:element name="coord" minOccurs="0">
<xsd:sequence>
  <xsd:element name="dim" maxOccurs="unbounded" type="xsd:int"/>
</xsd:sequence>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

2.5.3 Ein XML-Dokument einer virtuellen Welt

Code 2.5.3 zeigt ein Beispiel einer möglichen virtuellen Welt. Sie besteht aus zwei Räumen mit den Namen R1 und R2. Beide Räume haben einigen Merkmalen, welchen diese Räume beschreiben. Davon gibt ein Merkmal an, auf welchem Host der jeweilige Raum laufen soll. Beide Räume sind durch eine Türe miteinander verbunden. Raum R1 besitzt zudem ein Objekt `clock`. Die 2. Zeile des XML-Dokumentes gibt an, dass es der entsprechenden DTD zugrunde liegt.

Code 2.5.3 Beispiel eines XML-Dokuments einer virtuellen Welt

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE world SYSTEM "http://diuf.unifr.ch/softeng/projects/
madviworld/DTD/world.dtd">
<world>
  <worldname>myworld</worldname>
  <room>
    <roomname>R1</roomname>
    <doorlist>
      <door>
        <doorHost>134.21.9.64</doorHost>
        <doorName>R2</doorName>
      </door>
    </doorlist>
    <objectlist>
      <object>
        <objectname>swatch</objectname>
        <description>This is a clock.</description>
        <type>clock</type>
      </object>
    </objectlist>
    <access>everybody</access>
    <owner>pat</owner>
    <kind>office</kind>
    <host>134.21.9.117</host>
    <coord>
      <dim>60</dim>
      <dim>124</dim>
    </coord>
  </room>
  <room>
    <roomname>R2</roomname>
    <doorlist>
      <door>
        <doorHost>134.21.9.117</doorHost>
        <doorName>R1</doorName>
      </door>
    </doorlist>
    <access>everybody</access>
    <owner>yve</owner>
    <kind>office</kind>
    <host>134.21.9.64</host>
    <coord>
      <dim>20</dim>
      <dim>200</dim>
    </coord>
  </room>
</world>
```

2.5.4 MaDViWorld und DOM

Um die Umwandlung eines `world`-Objektes in ein XML-Dokument oder auch der umgekehrte Weg zu gewährleisten, braucht der Editor einen Parser. Da die Struktur der XML-Dokumente einer virtuel-

len Welt einfach sind und ein schneller Zugang zu den einzelnen Knoten erwünscht ist, ist ein Parser mit einem DOM-Zutritt zu den XML-Dokumenten wünschenswert.

Diese Umwandlung wird durch die Klasse `XMLTranslatorImpl` vorgenommen. Sie ist im Paket `MadViWorldMadViWorldw.source.ch.unifr.diuf.madviworld.util.virtualworld.GVirtualWorldImpl` zu finden. In Code 2.5.4 ist das Interface zu dieser Klasse aufgeführt.

Code 2.5.4 Interface `XMLTranslator`

```
package ch.unifr.diuf.madviworld.util.worldXMLtranslator;
import java.io.File;

public interface XMLTranslator {
    public File toXML(VirtualWorldImpl viworld);
    public File toXML(GVirtualWorldImpl viworld);
    public VirtualWorldImpl fromXML(File f);
    public GVirtualWorldImpl fromXML(File f);
}
```

Die Methode `toXML()` wandelt ein `world`-Objekt in ein XML-Dokument um. Dabei wird die virtuelle Welt in seine Räume und Objekte aufgeteilt und der DTD entsprechend eine Baumstruktur aufgebaut. Der oberste Knoten des Baumes ist immer ein `Document`-Element. Mit der Methode `appendChild()` wird Knoten für Knoten an den wachsenden Baum gehängt. Der Ausdruck `world.appendChild(roomel)` hängt zum Beispiel ein Raumelement dem Weltelement an. Beinhaltet die Baumstruktur einmal alle Elemente der virtuellen Welt, so wird der Baum in ein XML-Dokument umgewandelt und dieses wird als Resultat zurückgegeben.

Die Methode `fromXML()` durchläuft genau den umgekehrten Weg. Vom XML-Dokument wird ein Baum erstellt, durch welcher ein `world`-Objekt kreiert wird. Das Durchlaufen des Baumes erfolgt durch die Methode `getFirstChild()`, welche den Baum von oben nach unten durchläuft und je nach Element das entsprechende Objekt kreiert.

Abbildung 2.1 stellt das Beispiel des XML-Dokumentes aus Code 2.5.3 in einem DOM-Baum dar. Der oberste Knoten `Document` ist in jedem DOM-Baum enthalten. Die Blätter der Baumstruktur (jeweils die rechtsgelegenen Knoten in kursiver Schrift) definieren den Inhalt des Elementes des Vaterknotens. Die übrigen Knoten stellen die Elemente eines XML-Dokumentes dar.

Kapitel 3

Aufbau und Funktion des Editors

Dieses Kapitel beschreibt den Editor, welcher im Rahmen dieser Arbeit entwickelt wurde, indem dessen Aufbau und Funktionsweise erklärt wird. Abbildung 3.1 zeigt den ganzen Editor. Im weiteren werden die verschiedenen Teile dieses Editors näher erläutert.

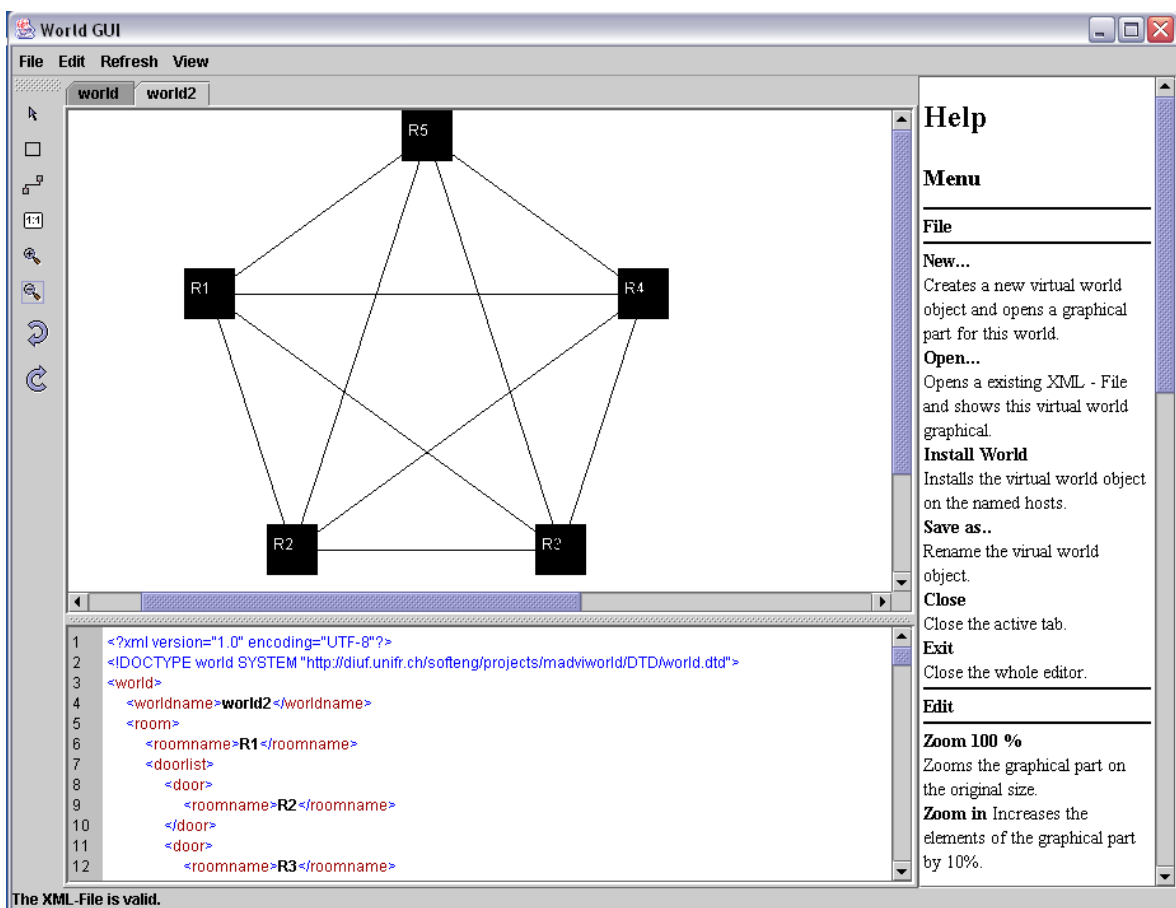


Abbildung 3.1: MaDViWorld Editor

3.1 Worldpanel

Im Worldpanel wird die kreierte, virtuelle Welt graphisch dargestellt, wobei die Räume als Knoten mit einem schwarzen Quadrat und die Türen mit einer Linie zwischen zwei Räume dargestellt werden.

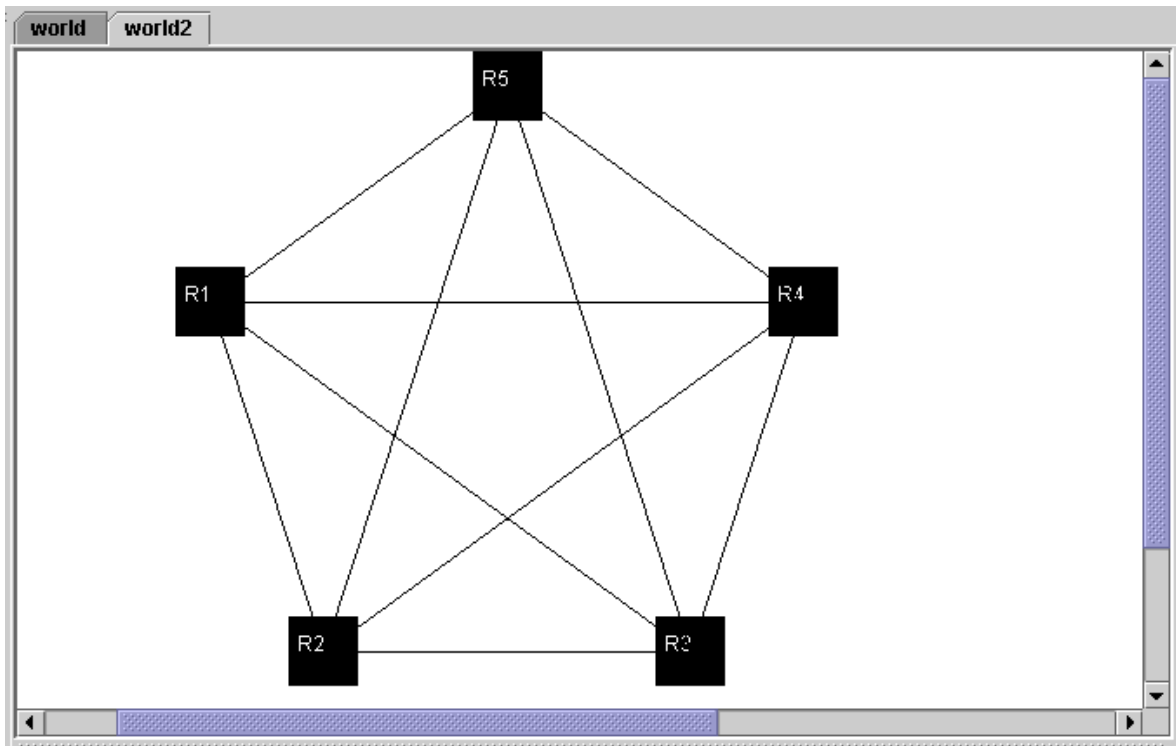


Abbildung 3.2: Worldpanel

Die im Worldpanel in Abbildung 3.2 dargestellte, virtuelle Welt besitzt demzufolge fünf Räume. Jeder dieser Räume ist jeweils mit einer Tür mit den anderen Räumen verbunden. Die Räume sind völlig miteinander vernetzt.

Das Worldpanel kann aus mehreren Tabs bestehen, welche es erlauben von einer Welt zur anderen zu gelangen. Sie sind jeweils mit den Namen der Welt gekennzeichnet. Ist dem Namen zusätzlich ein Stern nachgestellt, bedeutet dies, dass die aktuelle graphische Welt noch nicht in einem XML-Dokument abgespeichert worden ist.

3.2 XML-Panel

Der untere Teil des Editors umfasst das XML-Panel. Es ist in Abbildung 3.3 abgebildet. Im XML-Panel wird jeweils die geladene, virtuelle Welt mit der Markup-Sprache XML nach der letzten Speicherung beschrieben (siehe Kapitel 2.5.3). Kommentare und Dokumentdefinitionen des XML-Dokuments werden blau gekennzeichnet, die Elementnamen in braun und die Werte der Elemente in schwarz. Das

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE world SYSTEM "world.dtd">
3 <world>
4   <worldname>world2</worldname>
5   <room>
6     <roomname>R1</roomname>
7     <doorlist>
8       <door>
9         <roomname>R2</roomname>
10        </door>
11      <door>
12        <roomname>R3</roomname>
13      </door>
14      <door>
15        <roomname>R4</roomname>
16      </door>
17      <door>
18        <roomname>R5</roomname>
```

Abbildung 3.3: XML-Panel

XML-Dokument kann mit diesem Panel verändert werden.

3.3 Helppanel

Das Helppanel erklärt alle Elemente des Menus und der Toolbar auf ihre Funktionsweise, dabei wird ein einfaches HTML-Dokument ohne Verweise geladen. Es ist in Abbildung 3.4 abgebildet.



Abbildung 3.4: Helppanel

3.4 Menus und Toolbar

Das Menu und die Toolbar ermöglichen es dem Benutzer Funktionen wie das Zeichnen von Räumen und Türen auf dem Editor vorzunehmen. Nachfolgend sind die in der Menubar enthaltene Befehle mit dessen Shortcut und Beschreibungen aufgelistet:

File Menu Befehle

New...

Ctrl-N

Funktion

Ruft Fenster auf, welches eine neue Welt definiert und speichert sie in einem Dokument mit dem Namen `Name_der_Welt.xml`. Im Dialogfenster kann bestimmt werden, ob die neue Welt eine Ketten-, Stern- oder Ringstruktur aufweisen soll. Es kann auch einfach eine leere Welt kreiert werden. Auch der Standardwert des Rechners, auf welchem die Räume installiert werden sollen, kann beim Erstellen einer neuen Welt angegeben werden.



Open...

Ctrl-O

Öffnet ein XML-Dokument einer bestehenden, abgespeicherten Welt und stellt sie im Editor graphisch dar. Das zu öffnende XML-Dokument muss zum DTD-Dokument (siehe Abschnitt 2.5.1) valid sein.

Install World Ctrl-W	Installiert die Räume und Türen der aktuell geladenen Welt auf den angegebenen Rechnern. Ist die Installation abgeschlossen wird im Worldpanel ein neues Tab mit den erfolgreich installierten Komponenten geöffnet. Diese installierte Welt wird mit dem Namen <code>worldname_inst</code> bezeichnet.
Save world as... Ctrl-S	Speichert die virtuelle Welt im anzugebenen XML-Dokument ab. Der Name der Welt wird ebenfalls geändert.
Close Ctrl-C	Schliesst die aktuell geladene Welt.
Close All	Schliesst alle im Editor enthaltene Welten.
Exit Ctrl-E	Schliesst den Editor.
Edit Menu Befehle	Funktion
Zoom In	Vergrössert den graphischen Teil des Editors um 10%.
Zoom Out	Verkleinert den graphischen Teil des Editors um 10%.
Fit In	Definiert die Grösse der graphischen Welt genau so gross, so dass sie gerade ins Worldpanel passt.
Add Rooms... Ctrl-A	Fügt eine neue Struktur von Räume und Türen zur geladenen Welt bei.
Replace Rooms Ctrl-P	Plaziert die Räume der geladenen Welt mit dem in Layouts angegebenen Algorithmus.
Layouts... ▶	Beinhaltet eine Liste mit Algorithmen zur Platzierung der Räume.

Refresh Menu Befehle

Funktion

Refresh Graphics

Erneuert den graphischen Teil des Editors gemäss dem geladenen XML-Dokument.

Refresh XML

Erneuert das XML-Feld gemäss der aktuell geladenen, graphischen Welt.

View Menu Befehle

Funktion

Show Help

Öffnet das Helppanel.

Close Help


Schliesst das Helppanel.

Look and Feel... ▶

Beinhaltet Liste mit verschiedenen „Look-and-Feels“.

Ist ein Menu Befehl in grauer Farbe geschrieben und nicht anklickbar, so kann diese Funktion in dieser Situation nicht gewählt werden. So können viele dieser Menubefehle zum Beispiel nur gewählt werden, wenn auch mindestens eine Welt geladen ist.

Zusätzlich steht dem Benutzer eine Toolbar zur Verfügung. Die verschiedenen Icons werden im Weiteren näher erklärt:

 Ermöglicht das Selektieren von einzelnen oder mehreren Räume und Türen. Ist der Cursor auf einem Raum und wird die linke Maustaste betätigt, so wird der einzelne Raum selektiert. Dies wird mit einem roten Rahmen um den Raum gekennzeichnet. Durch das Rechtsklicken der Maus können selektierte Räume und Türen editiert oder entfernt werden.

Zeichnet ein neuer Raum an der Stelle, an welcher die Maus gedrückt wurde.



Verbindet 2 Räume mit einer Türe. Die Türen werden durch eine Gerade repräsentiert.



Setzt den Zoom des graphischen Teils des Editors auf normale Grösse.



Vergrössert den graphischen Teil des Editors um 10%.



Verkleinert den graphischen Teil des Editors um 10%.



Erneuert das XML-Feld gemäss der aktuell geladener, graphischer Welt.



Erneuert den graphischen Teil des Editors gemäss dem geladenen XML-Dokument.

3.4.1 Statuslabel

Am unteren Ende des Editors ist ein Statuslabel zu finden. Es ist in Abbildung 3.5 ersichtlich. Das Label gibt an, ob das XML-Dokument valid ist. Ist das XML-Dokument fehlerhaft, wird die Stelle, an welcher das Problem aufgetaucht ist, im Statuslabel angezeigt.

The XML -File is valid.

Abbildung 3.5: Statuslabel

Kapitel 4

Programmerklärung

Dieses Kapitel befasst sich mit dem Programmaufbau des Editors. Das Programm ist in drei Pakete aufgebaut. In diesem Kapitel werden auf zwei dieser drei Pakete näher eingegangen. Das dritte Paket beinhaltet die von François Jimenez erstellte Klasse, welche die vom Editor kreierten Welt auf Rechnern installiert [2].

4.1 Paket `viworld`

Der Pfad dieses Paketes lautet `MaDViWorld.MaDViWorldw.source.ch.unifr.diuf.madviworld.util.viworld` und umfasst die Klassen, welche die Struktur der virtuellen Welt beschreiben. Das UML-Klassendiagramm des ganzen Paketes ist im Anhang in Abbildung B.1 ersichtlich. Im Weiteren wird näher auf einzelne Teile des Paketes eingegangen.

4.1.1 Struktur der virtuellen Welt als Composite-Pattern

Wie in dieser Arbeit schon erläutert wurde, kann eine virtuelle Welt aus mehreren Räume bestehen. Ihr liegt demzufolge eine hierarchische Struktur zu Grunde. Sind Objekte wie in diesem anhand einer Baumstruktur aufgebaut, kann auf das Composite-Pattern [9] zurückgegriffen werden. Es hat zum Vorteil, dass der Client Zugriff auf die hierarchische Struktur haben kann, ohne dessen genauer Aufbau zu kennen.

Abbildung 4.1 zeigt das Composite-Pattern einer virtuellen Welt anhand eines UML-Diagrammes [10]. Die abstrakte Komponente ist die Klasse `GraphicalVirtualWorldElement`. Von ihr werden die Klassen `GRoomImpl` und `GVirtualWorldImpl` abgeleitet, wobei die zweitgenannte Klasse das Kompositobjekt einnimmt. Die abstrakte Klasse enthält die Methoden `draw()` und `move()`. Werden diese Methoden auf dem Kompositobjekt aufgerufen, werden sie auf ihre „Kinderkomponente“ übertragen, in diesem Fall auf `GRoomImpl`. Der Zugriff auf die hierarchische Struktur

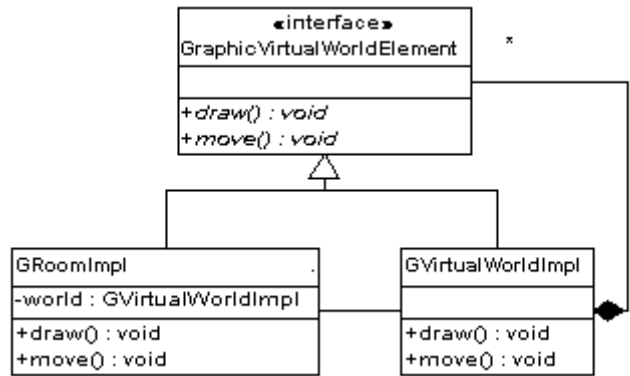


Abbildung 4.1: Virtuelle Welt als Composite-Pattern

erfolgt durch das `worldpanel`. Im Gegensatz zum ursprünglichen Composite Pattern kennen die Kinderkomponente in dieser Struktur das Kompositobjekt. Dies erleichtert die Bestimmung der maximalen Koordinaten, welche gebraucht werden, um den graphischen Teil des Editors zu definieren. So werden diese Koordinaten bei Erneuerung eines `GRoomImpl` im `GVirtualWorldImpl` bei Bedarf neu gesetzt. Sonst müssten jedesmal alle Räume durchlaufen werden, um den äussersten Punkt des graphischen Teils zu bestimmen.

4.1.2 Implementierung der virtuellen Welt

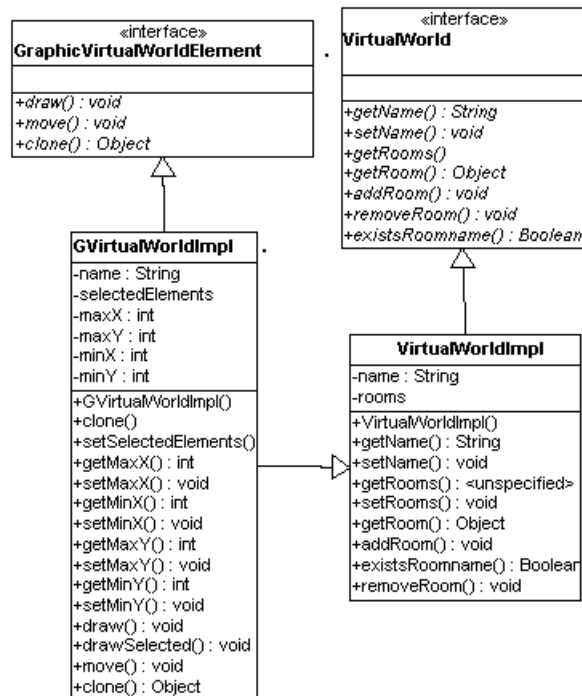


Abbildung 4.2: Welt-Objekt

Die Klassen, welche eine Welt beschreiben, sind in Abbildung 4.2 aufgelistet. Das Ziel bei der Implementierung der Welt war es, die graphischen Elemente von den Daten und der Struktur der Welt zu trennen, damit allfällige Änderungen bei einer der Teile des Welt-Objektes effizienter erfolgen können. Der graphische Teil beinhaltet alle Angaben, welche zur Darstellung der Welt im Editor erforderlich sind. Dazu gehören die äussersten Koordinaten der Welt, um das `worldpanel` entsprechend zu definieren oder auch einen Vector mit den zu selektierenden Räumen. Der strukturelle Teil beinhaltet unter anderem ein Vector und Methoden, welche das Verwalten der Räume der virtuellen Welt ermöglichen.

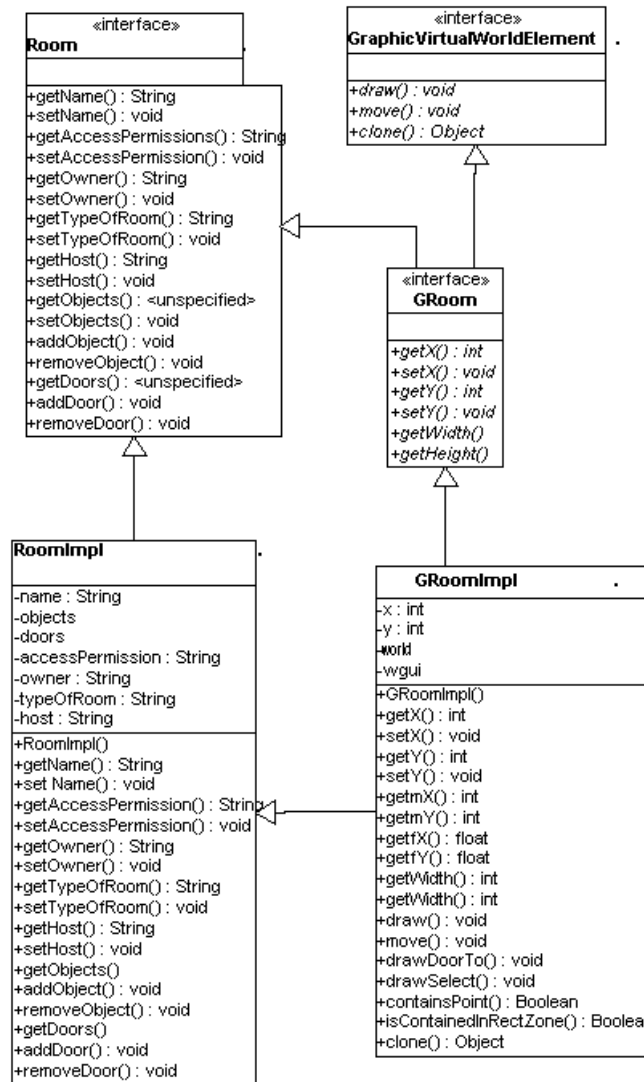


Abbildung 4.3: Raum-Objekt

Bei der Implementierung der Räume wurde gleich vorgegangen wie bei der Implementierung der Welt. Auch hier wurde der graphische Teil vom strukturellen Teil getrennt. Die resultierenden Klassen sind in Abbildung 4.3 ersichtlich. Der strukturelle Teil eines Raum-Objektes beinhaltet alle Angaben eines Raumes, welche zur Definition des Aufbaus dienen. Dazu zählen zum Beispiel der Typ oder der

Name des Raumes. Der graphische Teil des Raum-Objektes hingegen umfasst alle Elemente zur Darstellung des Raumes im Editor. So werden im graphischen Teil die Koordinaten oder Darstellungsart des Raumes festgelegt.

4.2 Paket gui

Das Paket gui ist unter `MaDViWorld.MaDViWorldw.source.ch.unifr.diuf.madviworld.setup.XMLtemplate.gui` zu finden. Dieses Paket enthält sämtliche Klassen, welche für die Darstellung des Editors notwendig sind. Die Abbildung B.2 im Anhang zeigt das ganze UML-Klassendiagramm dieses Paketes. Die Klasse `MaDViWorldCreatorGUI` ist die Hauptklasse dieser Anwendung. Diese Klasse benutzt auch die Klassen der anderen Pakete. Dies ist in Abbildung 4.4 ersichtlich.

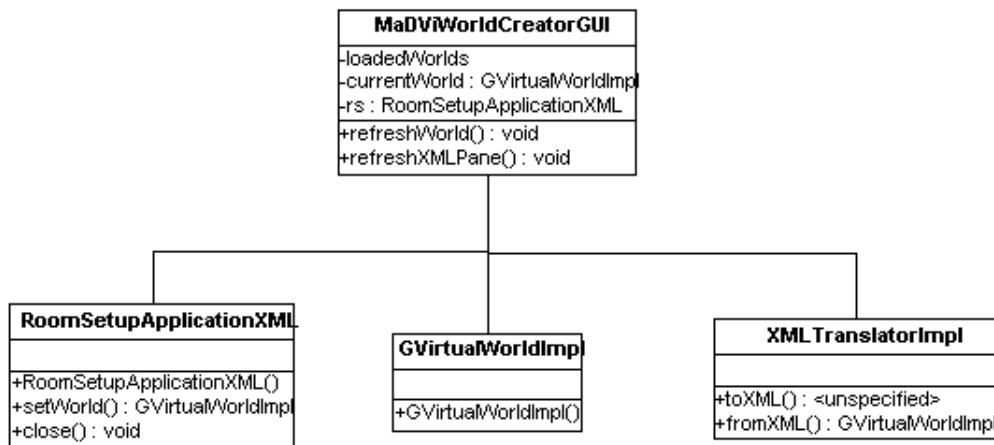

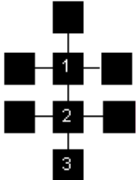
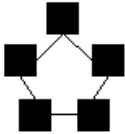
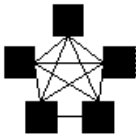
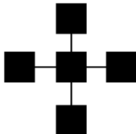


Abbildung 4.4: Verbindungen zwischen den verschiedenen Paketen

Im Weiteren wird näher in Teile des Sourcecodes und in andere Merkmale dieses Paketes eingegangen.

4.2.1 Strukturen zur Darstellung von speziellen virtuellen Welten

Manchmal ist es sinnvoll eine virtuelle Welt anhand von gegebenen, symmetrischen Strukturen aufzubauen. So kann beim Erstellen von neuen Welten oder beim Hinzufügen von Räumen auf vordefinierte Strukturen zurückgegriffen werden. Die nachfolgende Tabelle beschreibt einige solcher Strukturen und gibt Beispiele, wo sie eingesetzt werden können:

Bezeichnung	Grafik	Merkmale	Beispiel
Kette (chain)		Ein Ende kann als Startpunkt und das andere Ende als Endpunkt definiert werden. Die Türen zum nächsten Raum müssen dabei durch eine Aktion freigemacht werden.	Geschäftsprozess
Komplexe Kette		Um zur nächsten Raum zu gelangen, müssen im Gegensatz zu der einfachen Kette mehrere Aktionen erfüllt werden.	Universitätskurs, Computerspiel
Ring		Die Räume sind kreisförmig angeordnet, wobei nur die nebeneinander liegenden Räume mit Türen verbunden sind.	
Komplexer Ring		Im Gegensatz zum einfachen Ring ist jeder Raum mit allen anderen durch eine Türe verbunden.	Internet
Stern (star)		Alle Räume sind mit einem zentralen Raum durch eine Türe verbunden	

Wie aus der Tabelle zu entnehmen ist, wird als Beispiel für eine Kette ein Geschäftsprozess angegeben. In einem solchen Fall stellen die Räume die zusammen wirkende Prozesse zur Erstellung eines Produktes oder einer Dienstleistung dar. Dabei muss jeweils erst die Türe zum nächsten Raum freigegeben werden (Prozess fertig gestellt), um in den nächsten Raum der Kette zu gelangen. Gelangt man in den Zielraum, so ist der Geschäftsprozess beendet.

Beim Universitätskurs oder beim Computerspiel beinhaltet jeder Raum eine Lektion mit dazugehörigem Theorieteil und Tests bzw. Levels.

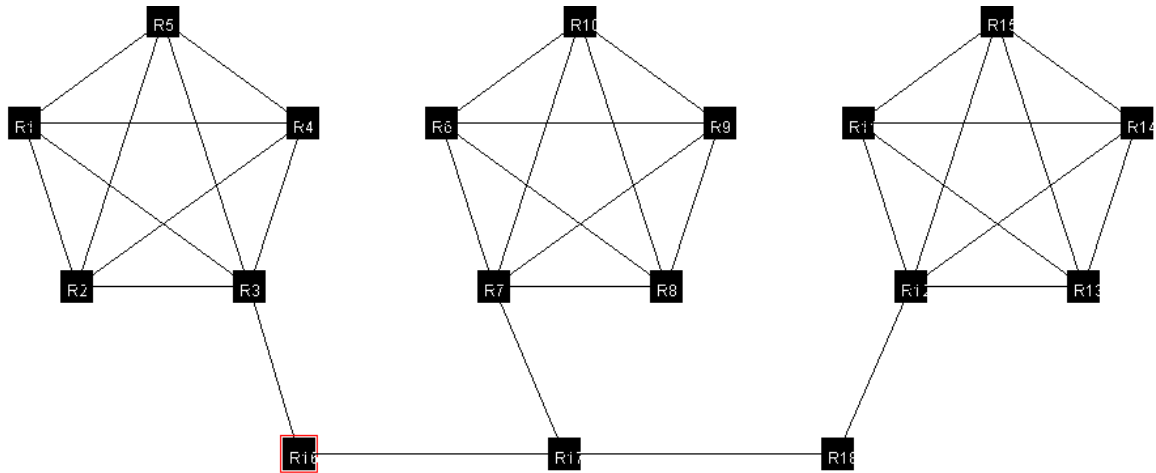


Abbildung 4.5: Komplexe Struktur einer virtuellen Welt

Der Editor ermöglicht es, mehrere oben vorgestellte Strukturen in einer Welt zu integrieren. So können komplexe Strukturen erschaffen werden. Eine solche komplexe Struktur ist in Abbildung 4.5 abgebildet. Sie beinhaltet eine Kette sowie 3 Ringstrukturen. Diese virtuelle Welt könnte zum Beispiel ein Gebäude mit drei Stockwerken simulieren, wobei die Räume eines Stockwerkes vollständig miteinander verbunden sind.

4.2.2 Algorithmen zur optimierten Darstellung von virtuellen Welten

Eine virtuelle Welt kann als einen Graphen angesehen werden [5], wobei die Räume Knoten und die Türen Kanten darstellen. Eine möglichst optimierte Darstellung von Graphen hängt von verschiedenen Kriterien ab und je nachdem bei welchen Bereichen dies auftritt, werden diese Kriterien verschiedenartig gewichtet. Andere Bereiche sind z.B. Datenstrukturen, Petrinetze oder Schaltkreise. Mögliche Kriterien für die optimierte Darstellung eines Graphen sind unter anderem:

- Anzahl der Kreuzungen von Kanten
- Benötigte Fläche

- Länge der Kanten
- Symmetrische Darstellung (gleichmässige Winkel und Kantenlänge)

Bei dieser Arbeit ist es vorteilhaft, dass Strukturen wie sie im Abschnitt 4.2.1 näher erklärt wurden, symmetrisch dargestellt werden und es möglichst keine Überschneidungen von Türen mit Räumen gibt. Einfachhalber werden die Kanten (Türen) mit Geraden dargestellt. Weiter unten werden zwei einfache Algorithmen beschrieben, welche die oben genannten Kriterien teilweise erfüllen und in dieser Arbeit angewendet werden.

Wenn wie in diesem Fall mehrere Algorithmen für den gleichen Zweck eines Programms eingesetzt werden, kann auf das Strategy Pattern [9] zurückgegriffen werden. Es besagt, dass die verschiedenen Algorithmen in verschiedenen Klassen gekapselt werden sollen. Für diese Arbeit bedeutet das, dass die zwei Algorithmen zur Platzierung der Räume jeweils in einer separaten Klasse verwaltet werden soll. Mit dem Strategy Pattern kann ohne grossen Aufwand ein neuer Algorithmus zur Darstellung der Räume eingeführt werden.

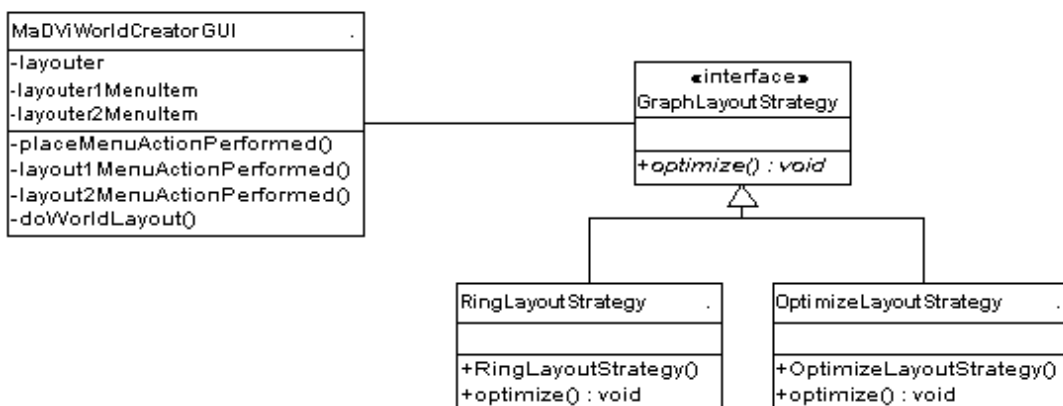


Abbildung 4.6: Layout als Strategy Pattern

Ringförmige Anordnung der Räume

Die erste Darstellungsart ordnet die Räume kreisförmig an. Diese Darstellungsart hat zum Vorteil, dass es bei einer relativen geringen Fläche keine Überschneidungen von Räume durch Türen gibt. Diese Darstellungsart liefert daher bei vielen Räumen mit vielen Türen eine gute Lösung. Abbildung 4.7 zeigt auf wie die Darstellung der virtuellen Welt von einer Ausgangsposition zu einer ringförmigen Anordnung umplaziert wurde.

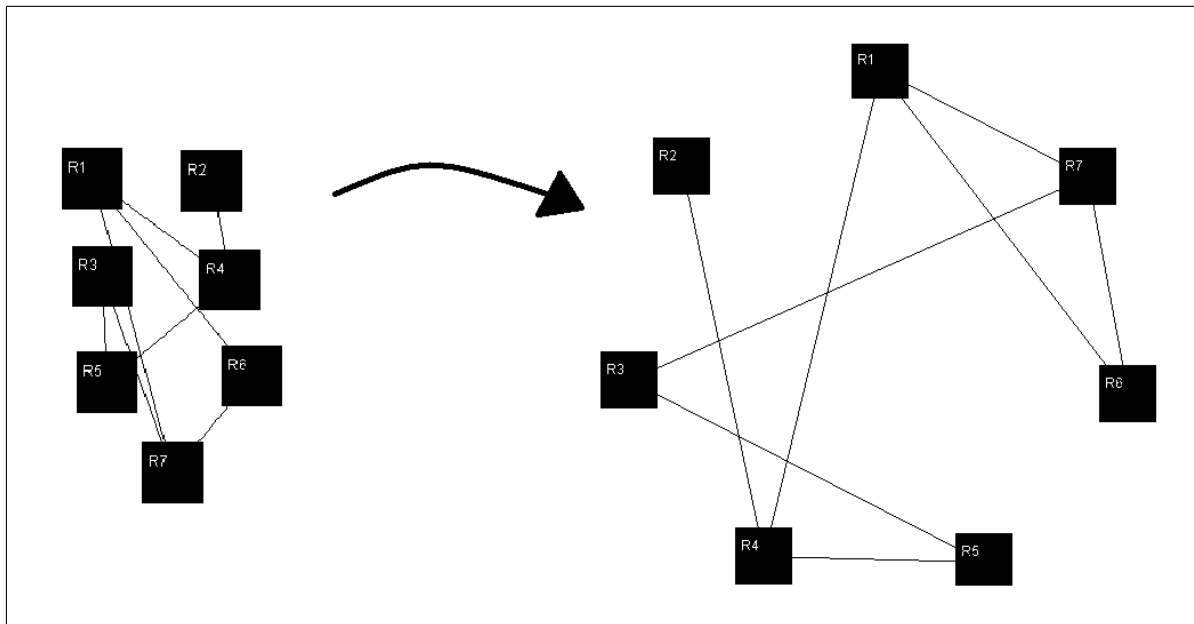


Abbildung 4.7: Umplazierung einer virtuellen Welt zu einer ringförmigen Anordnung

Code 4.2.1 Algorithmus zur ringförmigen Anordnung der Räume

```

public void optimize() {
    Vector rooms = world.getRooms();
    int nr = rooms.size();
    double decAngle, radAngle;
    double l = 225;
    int alpha = 360 / nr;
    double xValue, yValue;
    int x, y;
    for (int i = 0; i < nr; i++) {
        decAngle = i * alpha;
        radAngle = degreeToAngle(decAngle);
        xValue = -(Math.sin(radAngle)) * l;
        yValue = -(Math.cos(radAngle)) * l;
        x = (int)xValue;
        y = (int)yValue;
        x = x + CENTER_X;
        y = y + CENTER_Y;
        GRoomImpl room = (GRoomImpl)rooms.elementAt(i);
        room.setX(x);
        room.setY(y);
    }
}

```

Durch die Methode `optimize()` der Klasse `RingLayoutStrategy` erfolgt die Umplazierung der Welt. Dieser Algorithmus ist in Code 4.2.1 ersichtlich. Um den Punkt `(CENTER_X, CENTER_Y)` werden alle Räume der Reihe nach platziert. Der erste Raum wird genau oberhalb von diesem Punkt gesetzt. Darauf werden die Koordinaten der anderen Räume in Abhängigkeit ihres Winkel bestimmt. Abbildung 4.8 zeigt wie dies in einer Welt mit 5 Räumen aussieht.

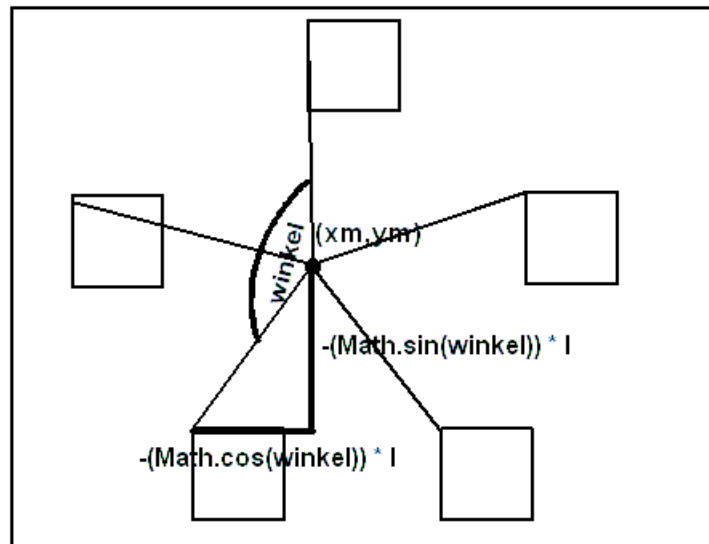


Abbildung 4.8: Berechnung der Raumkoordinaten bei einer ringförmigen Anordnung

Optimierte Anordnung

Eine komplexere und bessere Anordnung der Räume liefert der zweite Algorithmus dieser Arbeit. Dieser Algorithmus wird in der Tabelle 4.1 anhand einem einfachen Beispiel Schritt für Schritt erläutert, wobei auch die Veränderungen der Variablen (Arrays) angegeben werden.

	roomlist	placedrooms	nextrooms	newdoors
	((1,2),(2,2),(4,2),(3,3),(5,3))	()	()	()
	((2,2),(4,2),(3,3),(5,3))	(1)	(1)	(4,5)

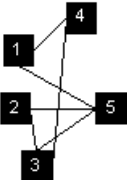
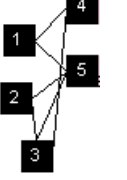
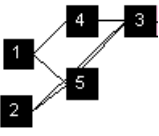
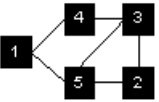
	roomlist	placedrooms	nextrooms	newdoors
	((2,2),(3,3),(5,3))	(1,4)	(4)	(5,3)
	((2,2),(3,3))	(1,4,5)	(5)	(3,2)
	((2,2))	(1,4,5,3)	(3)	(2)
	()	(1,4,5,3,2)	(2)	()

Tabelle 4.1: Umformung in eine optimierte Darstellung

Die Variable `roomlist` enthält alle noch neu zu platzierende Räume. Die erste Zahl jedes Eintrages in diesem Array liefert die Position des Raumes, an welchem er im Objekt `virtualworld` abgespeichert ist. Die zweite Zahl gibt an, wieviele Türen vom jeweiligen Raum ausgehen. Die Einträge im Array `roomlist` sind nach den Anzahl Türen geordnet.

Der erste Raum, welcher platziert wird, ist der ersten Eintrag der `roomlist`. In der Tabelle 4.1 ist dies der Raum 1. Von ihm aus werden alle Räume platziert, welche eine Türe zu diesem Raum besitzen (Raum 4 und 5). Dieser Vorgang wird danach solange wiederholt bis alle Räume gesetzt wurden. Der oben vorgestellte Algorithmus gehört zur Familie der „greedy“ Algorithmen [11]. Sie folgen dem Prinzip der lokalen Optimierung, das heisst, es wird jeweils der nächste Schritt gewählt, ohne die ganze Struktur zu betrachten. „Greedy“ Algorithmen haben meist nur linearen bis quadratischen Aufwand, können aber zu global ungünstigen Lösungen führen.

Dieser Algorithmus ordnet die Räume bei symmetrischen Strukturen relativ gut an. Auch bei unstrukturierten Anordnungen liefert diese Darstellungsart eine guten Lösungsansatz. Bei komplexen Anordnungen aber kann dieser Algorithmus dazu führen, dass sich Räume überschneiden.

4.2.3 XML-Highlighting

Wie schon in Abschnitt 3.2 beschrieben wurde, werden die verschiedene Teile des XML-Dokuments im Editor mit jeweils verschiedenen Farben dargestellt. Die Methoden, welche diese Kennzeichnung vornehmen, sind in der Klasse `XMLPane` zu finden. Dabei wird das XML-Dokument durchlaufen und je nachdem, ob es sich um ein Elementnamen, einem Attribut, einem Wert oder einem Kommentar handelt, mit der entsprechenden Farben formatiert. Das Parsen läuft durch reguläre Ausdrücke ab. Sie wurden von [16] übernommen.

4.2.4 Installation einer Welt

Nachdem mit dem Editor eine Welt kreiert wurde, kann sie auf die gewünschten Hosts installiert werden. Als Veranschaulichung einer Installation soll das UML-Sequenzdiagramm in Abbildung 4.9 dienen. Mit der Methode `setWorld()` aus der Hauptklasse `MaDViWorldXMLCreator` wird die Installation gestartet, nachdem die Klasse `XMLRoomSetup` aus der Arbeit von François Jimenez [2] instanziiert wurde.

Bei der Installation einer Welt können Fehler auftreten. So kann zum Beispiel auf dem gewünschten Host keine Server-Applikation laufen oder ein gewünschtes Objekt ist nicht verfügbar. Ist dies der Fall, so wird der entsprechende Raum oder das entsprechende Objekt aus dem Welt-Objekt entfernt. Als Resultat wird die Welt zurückgegeben, welche die Elemente enthält, welche auch wirklich installiert werden konnten.

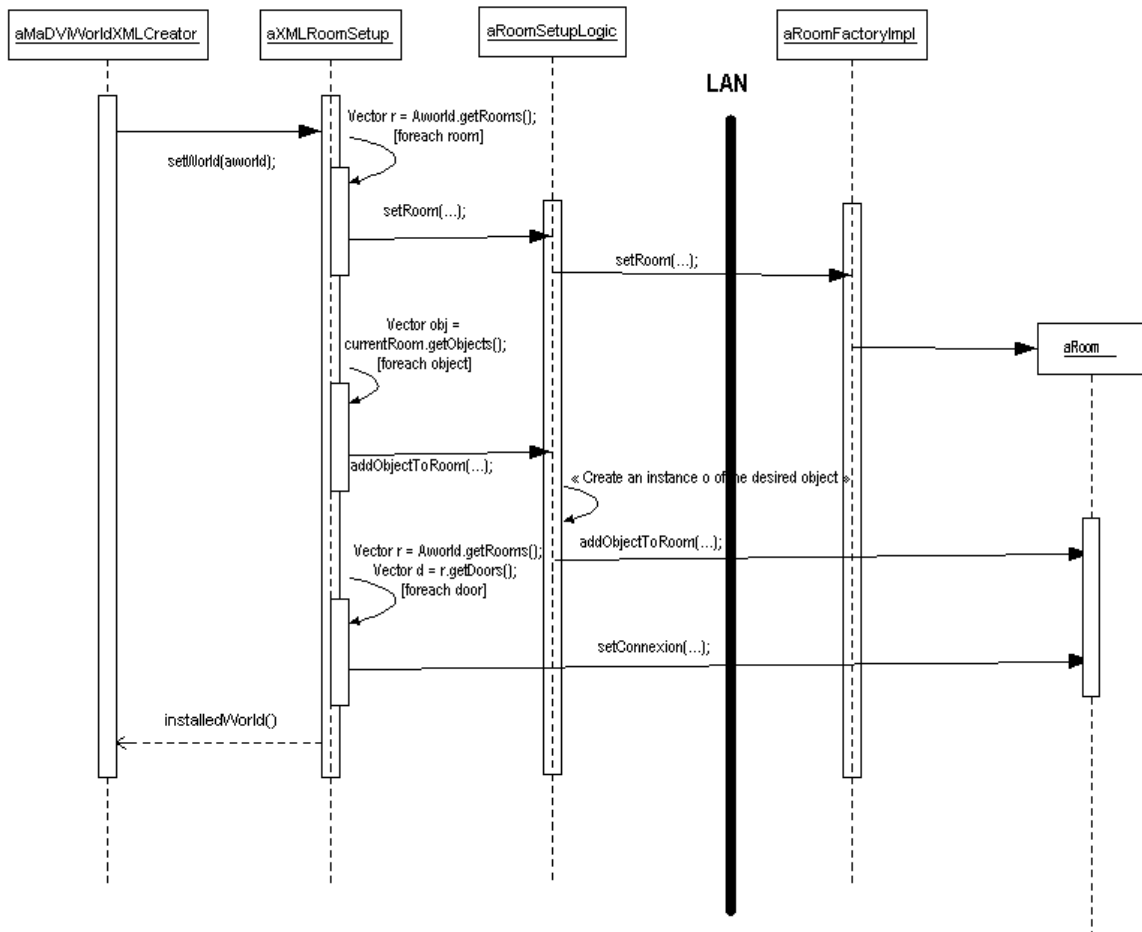


Abbildung 4.9: UML-Sequenzdiagramm: Installation einer Welt

Kapitel 5

Zusammenfassung

Dieses abschliessende Kapitel überprüft, ob das Resultat dieser Arbeit mit den gesteckten Zielen übereinstimmt und es werden einige Verbesserungsvorschläge angebracht, welche wegen zu teils grossem Zeitaufwand nicht realisiert werden konnten.

5.1 Ergebnis

Das Ziel war es, ein Editor zu entwickeln, welcher es erlaubt, virtuelle Welten zu kreieren. Zudem sollte es möglich sein, diese in einem XML-Dokument abzuspeichern und sie auf die angegebenen Rechner zu installieren. Wenn das Resultat dieses Editors betrachtet wird, kann der Benutzer feststellen, dass all die oben genannten Ziele erreicht wurden.

Zudem wurde bei der Implementierung des Editors auf eine saubere Architektur geachtet, damit er leicht ausgebaut werden kann. So können durch das Strategy Pattern ohne grossen Aufwand neue Algorithmen zur Umplazierung der Räume integriert werden. Auch die Trennung der Daten von den graphischen Elementen einer Welt erleichtert eine spätere Programmerweiterung. So kann eine neue Darstellungsart der Räume schnell umgesetzt werden. Zudem wurde darauf geachtet, dass der Programmcode nach den Regeln von Javadoc [7] kommentiert wurde. Javadoc ist ein Werkzeug, welches es erlaubt, HTML Seiten zu schaffen, welche die Klassen, Methoden und Felder eines Programmes beschreiben.

5.2 Verbesserungsvorschläge

- Zum Editieren der Räume wird ein Fenster aufgerufen, welche alle Merkmale eines Raumes beinhaltet. Dieses Fenster ist statisch, was nicht anderes heisst, als dass wenn zu den bestehenden Merkmalen ein neues hinzugefügt werden sollte, muss dieses entsprechend in diesem

Fenster geändert werden. Es wäre aber ideal, wenn die Merkmale im Fenster direkt von der DTD aufgelistet werden, welche die innere Struktur eines Raumes festlegt.

- Die graphische Darstellung der Räume könnte noch besser sein, als die Darstellungen, welche die Algorithmen hervorbringen, welche im Abschnitt 4.2.2 erläutert werden. Ein guter Lösungsansatz eines Editors zur Darstellung eines Graphen ist unter [17] zu finden. Vor allem das zirkuläre Layout (siehe Abbildung 5.1) entspricht in etwa den Bedürfnisse, welche die Graphen einer virtuellen Welt entsprechen.

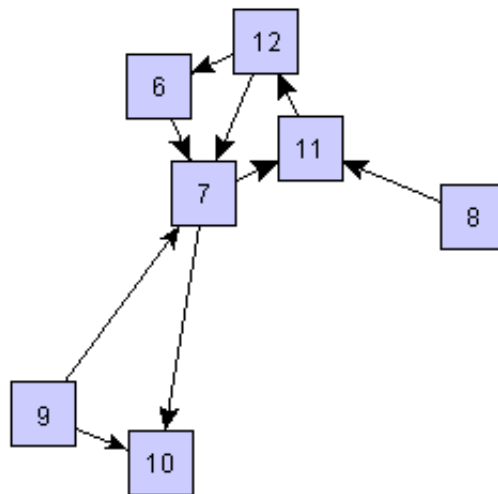


Abbildung 5.1: Beispiel eines zirkulären Layouts

- Wünschenswert wären Funktionen wie `undo` und `redo`, welche die Handhabung des Editors weiter erleichtern könnten.
- Die Türen werden in der aktuellen Version des Editors im graphischen Teil durch eine Gerade dargestellt. Dies kann schnell einmal zu Überschneidungen mit dazwischenliegenden Räumen führen. Abhilfe zu diesem Problem könnten eckige oder runde Türen bieten.
- Bei der Installierung einer virtuellen Welt wird ein Welt-Objekt zurückgegeben, das beschreibt, welche Komponente der ursprünglich gedachten Welt auch wirklich installiert wurden. Könnten nicht alle Komponenten installiert werden, so wird diese Welt ebenfalls im Editor angezeigt. Wünschenswert wäre aber eine bessere Fehlerbehandlung. So kann zum Beispiel vorgestellt werden, dass die nicht installierten Komponenten in der ursprünglichen, virtuellen Welt mit einer anderen Farbe gekennzeichnet würden.

- Die in einem XML-Dokument abgespeicherte Welt hängt von der DTD in Code 2.5.1 ab. Wie in dieser Arbeit beschrieben wurde, weist die DTD einige Mängel auf, welche durch das XML Schema behoben werden. Darum wäre es sinnvoll, wenn die DTD durch das XML Schema aus Code 2.5.2 ersetzt werden würde.

Literaturverzeichnis

- [1] Webseite der Software Engineering Group
<http://diuf.unifr.ch/softeng/projects/madviworld/> (letzter Besuch 08. März 2004)
- [2] Webseite der Diplomarbeit von François Jimenez
<http://diuf.unifr.ch/fuhrer/studproj/jimenez/chat.html> (letzter Besuch 08. März 2004)
- [3] Aktuellste Version des MaDViWorld-Frameworks
<http://diuf.unifr.ch/softeng/projects/madviworld/latestbuild.html> (letzter Besuch 08. März 2004)
- [4] P. Fuhrer and J. Pasquier-Rocha, Massively Distributed Virtual Worlds: a Framework Approach, Department of Informatics Internal Working Paper no 02-16, University of Fribourg, Switzerland, December 2002.
- [5] P. Fuhrer and J. Pasquier-Rocha, Massively Distributed Virtual Worlds: a Formal Approach, Department of Informatics Internal Working Paper no 03-14, University of Fribourg, Switzerland, August 2003.
- [6] D. Flanagan, Java in a nutshell, O'Reilly, Sebastapol, 1999.
- [7] Javadoc Tool, Sun Microsystems
<http://java.sun.com/j2se/javadoc/> (letzter Besuch 29. März 2004)
- [8] The apache ant project
<http://ant.apache.org/> (letzter Besuch 30. März 2004)
- [9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns, Addison Wesley, USA, 1995.
- [10] M. Fowler, UML: A brief guide to the stanard object modeling language, Addison Wesley, USA, 2003.
- [11] T. Cormen, C. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, USA, 2001.
- [12] E. Ray, Einführung in XML, O'Reilly, USA, 2004.

- [13] E. R. Harold, Processing XML with Java, Addison Wesley, Boston, 2003.
- [14] W. Dehnhardt, Scriptsprachen für dynamische Webauftritte, Hanser, München, 2001.
- [15] W3Schools: XML Schema Tutorial
<http://www.w3schools.com/schema/> (letzter Besuch 31. März 2004)
- [16] R. D. Cameron, REX: XML Shallow Parsing with Regular Expression, School of Computing Science, Simon Fraser University, Kanada
<http://www.cs.sfu.ca/cameron/REX.html> (letzter Besuch 08. März 2004)
- [17] yEd, JAVA Graph Editor, yWorks GmbH, Deutschland
http://www.yworks.com/en/products_yed_about.htm (letzter Besuch 08. März 2004)

Anhang A

Installation und Anwendung

A.1 MaDViWorld-Framework installieren

Installiere das MaDViWorld-Framework von der beiliegenden CD-ROM oder von der Webseite [3] in das gewünschte Verzeichnis. Das MaDViWorld-Framework besteht aus 3 Unterverzeichnisse:

- `MaDViworlds`: Beinhaltet die Server-Anwendungen
- `MaDViworldc`: Beinhaltet die Client-Anwendungen
- `MaDViworldw`: Beinhaltet die Wizard-Anwendungen inklusive Editor

A.2 Konfiguration

Jedes der drei Unterverzeichnisse besitzt eine Datei `build.properties`. In jeder dieser Datei müssen folgende Variablen angepasst werden:

- `jinihome`: Legt den Pfad fest, in welchem Jini zu finden ist.
- `host`: Definiert die IP-Adresse, welche den Host bestimmen, auf welchem die Anwendungen gestartet werden sollen.
- `madviworldhome`: Legt den Pfad fest, auf welchem die Unterverzeichnisse abgelegt wurden.

A.3 Ant installieren

Damit der Editor gestartet werden kann, muss das Apache Build-Werkzeug Ant installiert werden. Ant ist vergleichbar mit Make und steuert die Erzeugung ausführbarer und auch anderer Dateien.

Unter [8] kann Ant heruntergeladen werden. Nachdem das Ant Build System entpackt wurde, muss die Umgebungsvariable `ANT_HOME` gesetzt werden:

```
set ANT_HOME=c:\java\ant
```

A.4 Starten der Anwendungen

Damit anhand dem Editor eine virtuelle Welt installiert werden kann, muss zuerst die Server-Anwendung gestartet werden. Dabei gibt der Benutzer folgenden Befehl in die Konsole ein:

```
C:\MaDViWorld\MaDViWorlds>ant run
```

Dieser Befehl startet die Server-Anwendung und ruft das Fenster in Abbildung A.1 auf. Dieses Fens-

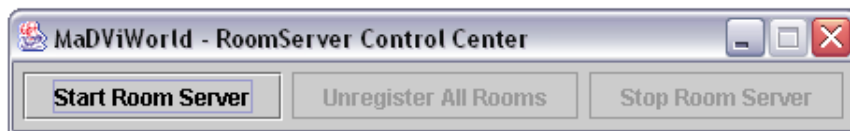


Abbildung A.1: Server-Fenster

ter erlaubt es, den Server zu starten oder zu stoppen sowie die registrierten Räume zu entfernen.

Ist der Server einmal gestartet, kann der Editor aufgerufen werden. Dies geschieht ähnlich zur Server-Anwendung:

```
C:\MaDViWorld\MaDViWorldw>ant runxml
```

Dieser Befehl startet den in dieser Arbeit vorgestellten Editor ohne einer geladenen Welt.

Anhang B

UML

B.1 Klassendiagramm viworld

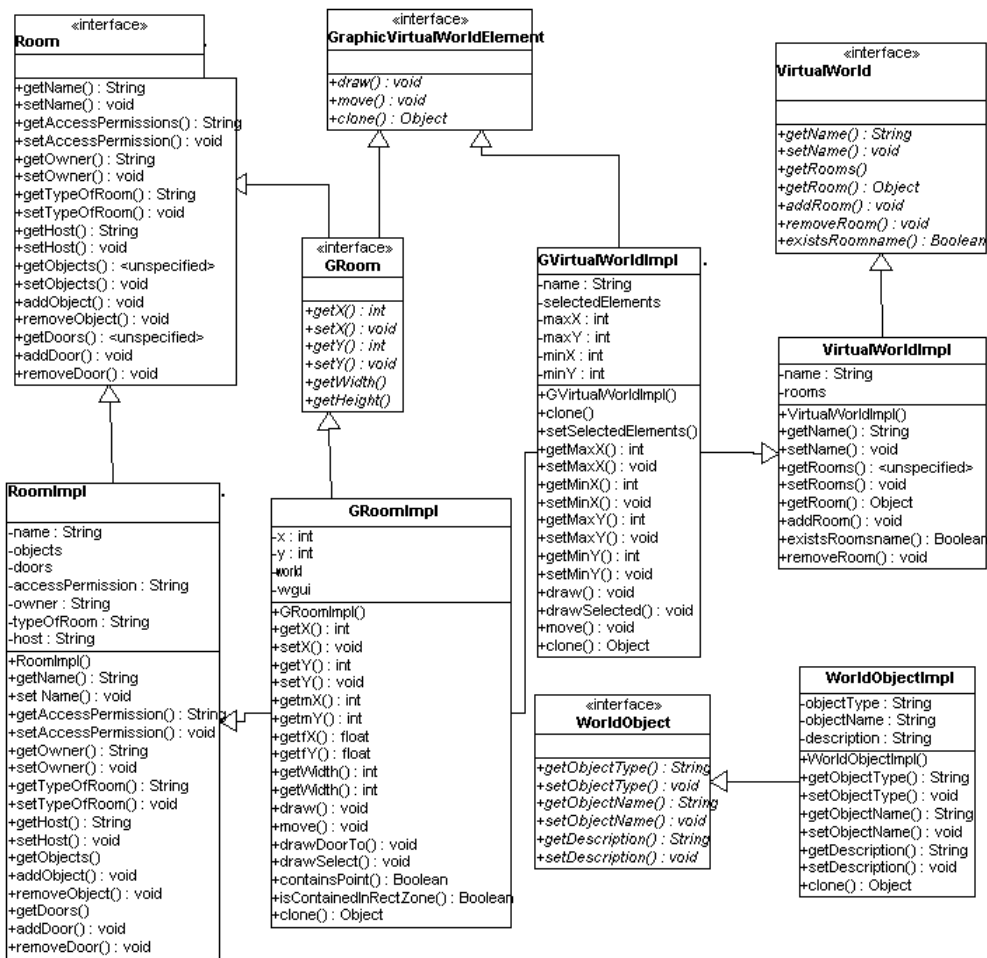


Abbildung B.1: Paket viworld

B.2 Klassendiagramm gui

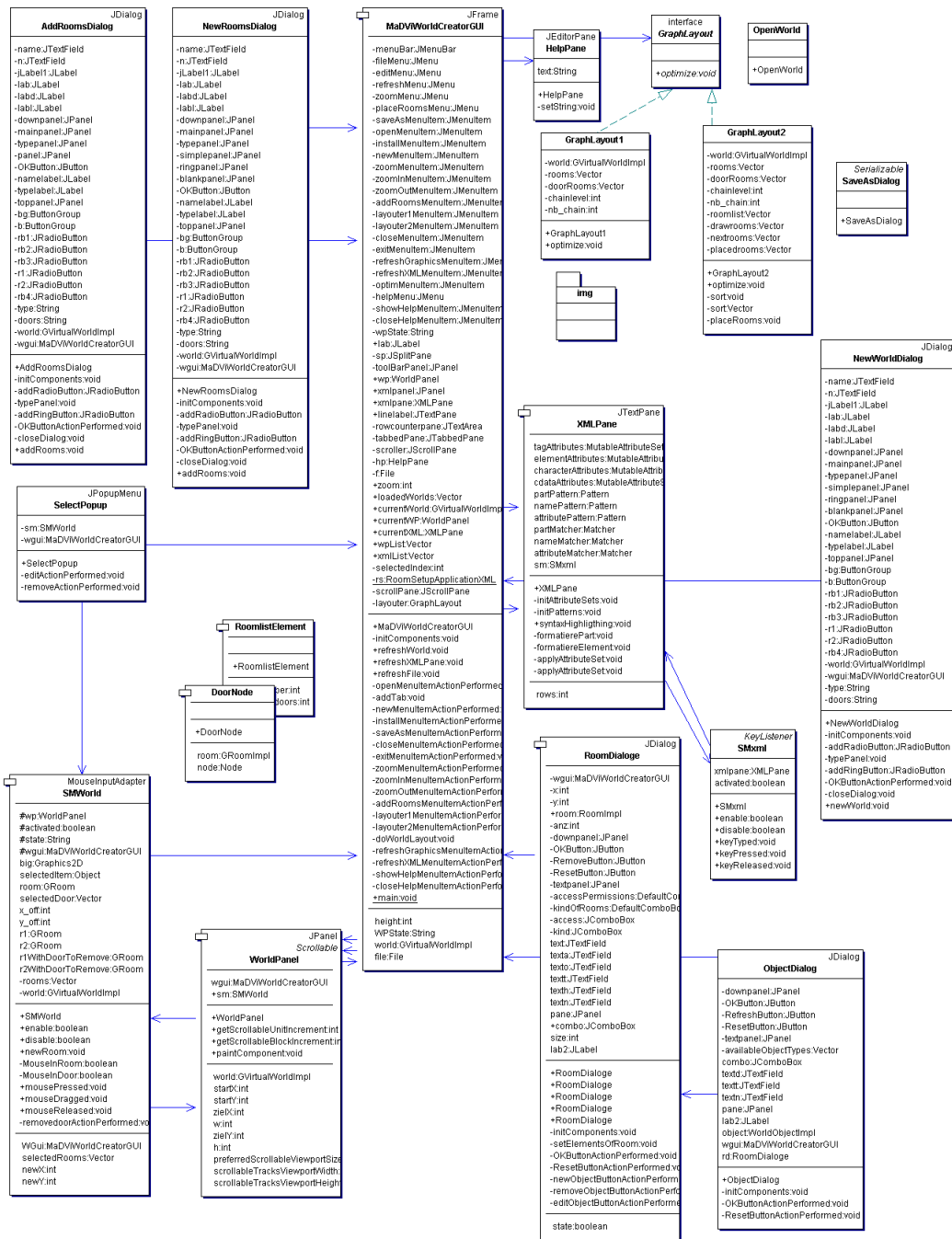


Abbildung B.2: Paket gui

Anhang C

CD-ROM

Die beigelegte CD-ROM beinhaltet:

- MaDViWorld-Framework mit Editor
- Dokumentation im Format PDF
- Dokumentation in LaTeX mit den Abbildungen

Anhang D

Eidesstaatliche Erklärung

Ich versichere, dass ich die vorliegende Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht veröffentlicht und noch keiner Prüfungsbehörde vorgelegt worden.

Murten, den

Unterschrift