Diplomarbeit

Fachgebiet der Diplomarbeit:
Graphische Datenverarbeitung

Thema der Diplomarbeit:

Eine Animationsplattform für virtuelle Charaktere auf Basis von vordefinierten Animationen

durchgeführt am
Zentrum für Graphische Datenverarbeitung e.V. (ZGDV)
in Darmstadt

Diplomand: Mario Langlitz
Referentin: Prof. Dr. Monika Lutz
Korreferent: Prof. Dr. Stephan Euler
Betreuer am ZGDV: Dipl.-Inform. Ido Iurgel, M.A.

Sommersemester 2004
Fachhochschule Gießen/Friedberg
Bereich Friedberg
Fachbereiche IEM, MND, MNI
Studiengang Medieninformatik

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Gleichzeitig versichere ich, diese Arbeit in gleicher oder ähnlicher Form weder veröffentlicht noch einer anderen Prüfungsbehörde vorgelegt zu haben.

Gießen, den 14. April 2004

Mario Langlitz

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung einer Animationsplattform für virtuelle Charaktere mit emotionaler Gestik und Mimik und einer synthetischen Sprachausgabe. Die Animationsplattform soll es ermöglichen, virtuelle Charaktere als menschenähnliche Gesprächspartner darzustellen, die zur Vermittlung von Informationen im Projekt Art-E-Fact eingesetzt werden können. Art-E-Fact ist ein von der EU gefördertes Projekt, in dem eine Arbeitsumgebung für interaktives Storytelling in einer Mixed Reality Umgebung erstellt wird. Künstler sollen diese Arbeitsumgebung nutzen können, um interaktive Ausstellungen und Mixed Reality Objekte zu erschaffen. Der Benutzer hat die Möglichkeit, dargestellte Bilder virtuell zu bearbeiten oder diese z. B. mit einer Lupe zu untersuchen, was durch Gestenerkennung ermöglicht wird. Dem Benutzer werden im Dialog mit virtuellen Charakteren Informationen über die dargestellte Kunst oder Antworten auf Fragen anhand eines implementierten Geschichtsmodells gegeben. Das Geschichtsmodell wird durch Datenbanken gespeist, welche von den Künstlern angelegt werden, wodurch dem Benutzer das Wissen des Künstlers anhand der virtuellen Charaktere vermittelt wird.

"Physische" Interaktionen der Charaktere mit ihrer virtuellen Umwelt, wie beispielsweise das Anheben eines Stuhls oder das Drücken einer Türklinke, finden nicht statt, weshalb der Schwerpunkt bei der Realisierung der Animationsplattform auf der Darstellung von emotionalen und sprachbegleitenden Animationen liegt.

Vorgegeben ist eine Grobstruktur der Plattform in Form eines UML-Klassendiagramms sowie einige Animationshilfsklassen, so dass die Hauptaufgabe darin besteht, das Diagramm umzusetzen, zu verfeinern und, wenn nötig, zu korrigieren. Zur Aufgabe der Umsetzung des statischen Diagramms gehört die Festlegung und Beschreibung der dynamischen Aspekte sowie die Vervollständigung der Speicher-, Austausch-, Beschreibungs- und Konfigurationsformate.

Weiterhin soll die Plattform um geeignete Editoren erweitert werden, welche die Steuerung der virtuellen Charaktere im Rahmen ihrer Funktionalität ermöglichen. Es ist besonders auf den sinnvollen automatisierten Einsatz von vorgefertigten Animationen zu achten, wozu die Entwicklung von Kontrollmechanismen für die Auswahl und für den Einsatz von Animationen notwendig ist. Diese Kontrollmechanismen setzen maschinenlesbare Be-

schreibungen der Animationen voraus, die ebenfalls im Rahmen dieser Arbeit entwickelt werden sollen.

Wegen des engen Zusammenhangs mit dem Projekt Art-E-Fact ist es eine ausdrückliche Anforderung an die Diplomarbeit, eine stabile Implementierung zur Verfügung zu stellen. Zusätzlich sollen aber auch komplexere Lösungen beschrieben werden, deren Implementierung im Rahmen des Projektzeitplanes noch nicht sinnvoll ist.

Für die Realisierung der Animationsplattform ist die Programmiersprache Java der Firma Sun Microsystems zu verwenden. Weiterhin soll versucht werden, die Renderumgebung Avalon zur Darstellung in die Plattform zu integrieren. Avalon ist eine Eigenentwicklung des Fraunhofer-Instituts und des ZGDV und stellt eine Darstellungsplattform für Virtual-Reality- und Augmented-Reality-Anwendungen dar [10]. Avalon ermöglicht die Darstellung dreidimensionaler Objekte, die in der Virtual Reality Modelling Language (VRML) beschrieben sind [11].

Ergebnis der Diplomarbeit soll eine flexible Animationsplattform sein, die leicht konfigurierbar ist, stabil läuft, flüssige und variable Animationen einsetzt und über Werkzeuge für Autoren verfügt.

1.2 Gliederung der Arbeit

Im nachfolgenden Kapitel 2 werden zunächst verschiedene Kategorien von Animationen vorgestellt. Nachfolgend wird der Begriff des virtuellen Charakters erläutert und anhand von zwei Beispielen die Möglichkeiten der Realisierung von Animationsplattformen abhängig von ihrem Einsatzgebiet aufgezeigt.

Kapitel 3 beschreibt, welche Anforderungen die Plattform erfüllen soll, um innerhalb des Projektes Art-E-Fact eingesetzt werden zu können. In Kapitel 4 wird ein Lösungsansatz für die Animationsplattform vorgestellt, der sich aus der vorgegebenen UML-Struktur und den in Kapitel 3 beschriebenen Anforderungen ergeben hat. Es wird ebenfalls die Klassenstruktur und das Laufzeitverhalten des Ansatzes beschrieben.

In Kapitel 5 befasst sich mit der Realisierung der Animationsplattform. Es werden ein Editor zum Steuern der Plattform, der Aufbau eines virtuellen Charakters und die Verar-

beitung von Animationen sowie die Funktionsweise und Implementierung der Sprachsynthese beschrieben.

Kapitel 6 beschreibt theoretische Ansätze zur Entwicklung eines Beschreibungsformates für Animationen und der Verwendung einer auf XML basierenden Kommunikationssprache innerhalb der Klassenstruktur.

Kapitel 7 beinhaltet eine Zusammenfassung der Ergebnisse dieser Arbeit und einen Ausblick auf mögliche Weiterentwicklungen und Einsatzgebiete der Animationsplattform.

2 Stand der Forschung

2.1 Einleitung

In diesem Kapitel werden zunächst die Verfahren der Keyframe-Animation und der inversen Kinematik beschrieben sowie ein Format zur Beschreibung von Rotationen, so genannte Quaternione. Anschließend folgen Erläuterungen zum Begriff des virtuellen Charakters. Das Beispiel von zwei bestehenden Animationsplattformen verdeutlicht, wie Animationen erzeugt werden können, und welche Voraussetzungen, beispielsweise die Skelettstruktur eines virtuellen Charakters, zur Darstellung von menschenähnlichen Bewegungsabläufen nötig sind.

2.2 Animationstechniken

Animationen lassen sich in drei Kategorien einteilen. Die erste Kategorie bilden vordefinierte Animationen, die durch Motion-Capture-Verfahren aufgezeichnet oder in 3D-Programmen entworfen werden. Eine Möglichkeit Animationen dieser Kategorie zu Speichern sind Keyframe-Animationen, d. h. in bestimmten Zeitintervallen werden Rotationswinkel, Translationswerte oder Skalierungswerte für die Achsen x, y oder z gespeichert. Für Rotationen werden dabei Eulerwinkel verwendet. Die Berechnung eines genauen Wertes beim Abspielen einer Keyframe-Animation ist durch eine einfache lineare Interpolation zwischen zwei gespeicherten Werten möglich. Eine zweite Möglichkeit zur Speicherung von Keyframe-Animationen bieten Kurvenformate wie Bézier-Kurven oder B-Splines, die den Verlauf einer Animation anhand von wenigen Stützstellen in einer Kurve beschreiben (Abbildung 2.1). Werden Animationen anhand von Stützstellen gespeichert, ist die Berechnung eines Wertes zwischen zwei Stützstellen aufwendiger, weil anstelle einer linearen Interpolation zwischen zwei Punkten ein Punkt auf einer Kurve berechnet werden muss. Der Vorteil bei der Verwendung von Kurvenformaten ist, dass eine Interpolation zwischen dem Ende einer Animation und dem Anfang einer zweiten Animation bessere Ergebnisse ergibt. Die Manipulation einzelner Stützstellen bietet zudem die Möglichkeit, über einen bestimmten Zeitraum innerhalb der Animation eine Veränderung an der Ausprägung der Animation vorzunehmen.

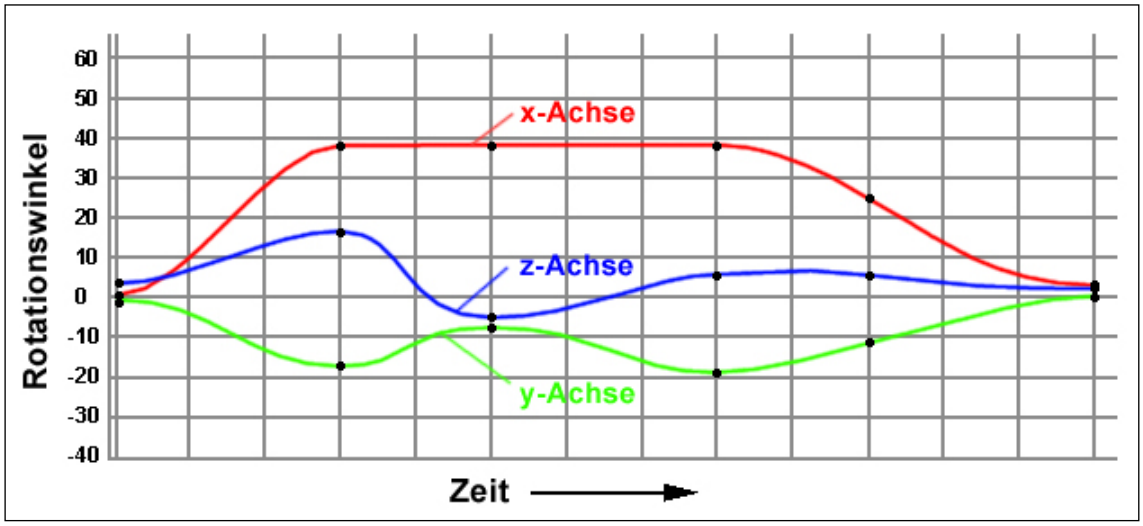


Abbildung 2.1: Winkeländerungen der x, y und z-Achse eines Knotens beschrieben anhand von sechs Stützstellen

Die zweite Kategorie von Animationen bilden so genannte prozedurale Systeme, die Animationen anhand von Algorithmen berechnen. Ein solches prozedurales Verfahren ist die inverse Kinematik (IK). Gleicher [3] definiert IK für die Animation von virtuellen Charakteren als einen Prozess, bei dem die Konfiguration der Parameter eines Charakters (bekannt als Pose) aufgrund von vorher spezifizierten Merkmalen einer Pose, beispielsweise einer Endeffektorposition, gefunden wird. Ein Endeffektor kann beispielsweise eine Fingerspitze sein, aufgrund deren Position die Winkel der Gelenke des Armes berechnet werden. Aufgrund der vielen Möglichkeiten der Stellung von Gelenken sind natürlich aussehende Gesten durch IK nur mit hohem Aufwand zu verwirklichen.

Die dritte Kategorie verbindet prozedurale Systeme und vorgefertigte Animationen, um natürlich aussehende Bewegungen umzusetzen, ohne rechenintensive komplexe Algorithmen zu verwenden. Gleicher [3] beschreibt verschiedenen Kombinationen von inverser Kinematik und Keyframe-Animationen, um beispielsweise die Einhaltung von minimalen und maximalen Rotationswerten einzelner Gelenke auch bei der Verwendung von Keyframe-Animationen zu verwirklichen.

Shoemake [6] beschrieb die Verwendung von Quaternionen zur Interpolation von Rotationen. Im Gegensatz zu Euler-Winkeln, die Rotationen im dreidimensionalen Raum anhand von drei vordefinierten Achsen beschreiben, verwenden Quaternione einen Vektor

zur Abbildung der Rotationsachse und einen Skalar für den Rotationswinkel zur Beschreibung einer Rotation. Quaternione ermöglichen die Kombination zweier Rotationen durch Multiplikation, was unter Verwendung von Matrizen und Euler-Winkeln nur sehr schwer möglich ist. Ein weiterer Vorteil von Quaternionen: Der Verlust von Freiheitsgraden durch Rotationen, das so genannte "gimbal lock", ist nicht möglich (Abbildung 2.2). Das "gimbal lock" kann bei mehrfachem Aufmultiplizieren von Rotations-Matrizen auftreten, wenn eine bestimmte Kombination von Werten in der Rotationsmatrix 0 ergibt. Eine Multiplikation eines Punktes im dreidimensionalen Raum mit dieser Matrix führt nicht mehr zu dem gewünschten Ergebnis. Das "gimbal lock" tritt beispielsweise bei der Rotation um 90° um die y-Achse auf, wonach die Achsen x und z in der gleichen Ebene liegen.

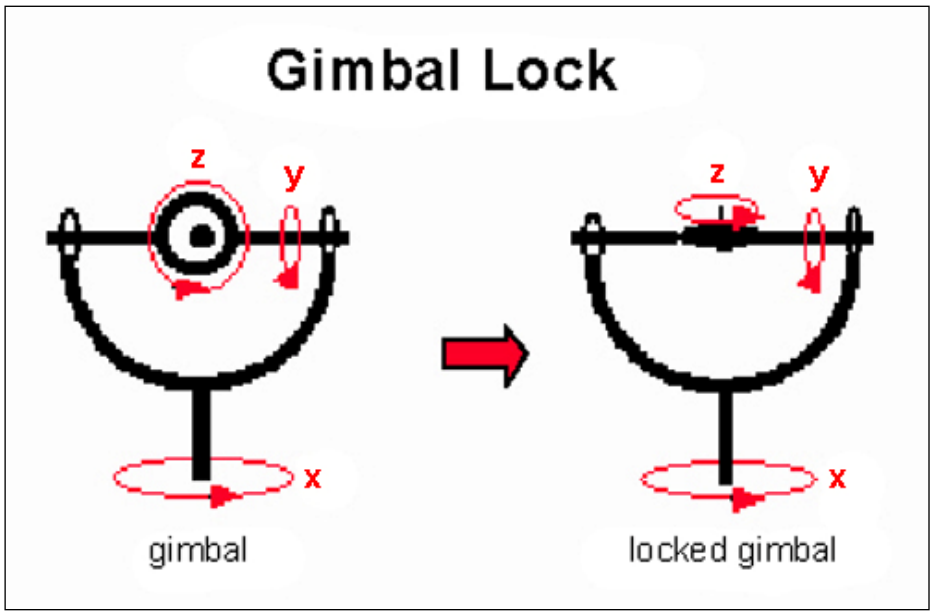


Abbildung 2.2: Verlust eines Freiheitsgrades durch Rotationen [7]

Die Verwendung von Quaterionen verhindert eine durch das Auftreten eines "gimbal lock" mögliche fehlerhafte Darstellung von Animationen. Zudem stellen Quaternione im Vergleich zu Matrizen eine schnellere Variante zur Verbindung von verschiedenen Rotationen dar, was für die Animationsplattform von Vorteil wäre. Quaternione wurden im Java3D-Package von Sun Microsystems® bereits in einer Klasse realisiert, welche Methoden zur Interpolation und Multiplikation von Quaternionen zur Verfügung stellt, was deren Verwendung in der Animationsplattform vereinfachen würde [15].

2.3 Virtuelle Charaktere

Virtuelle Charaktere stellen menschenähnliche Modelle dar, die, abhängig von der eingesetzten Technologie, Handlungen ohne das Einwirken eines Menschen ausführen können. Komplettsysteme, die eine Umsetzung von realistischem Aussehen mit einem physikbasierenden und menschlichen Handeln eines virtuellen Charakters in Echtzeit ermöglichen, sind derzeit noch nicht erhältlich. Die Gestaltung eines virtuellen Charakters ist zurzeit noch stark von seinem Einsatzgebiet abhängig. So werden beispielsweise zu Konstruktionszwecken virtuelle Charaktere eingesetzt, deren Aufbau und Abmessungen denen des Menschen nachempfunden sind, um ergonomische Gesichtspunkte beim Design von Maschinen, Fahrzeugen oder Arbeitsplätzen zu überprüfen. Solche virtuellen Charaktere müssen mit ihrer virtuellen Umwelt interagieren können, um Gegenstände anzufassen oder zu bewegen, was eine Berechnung der Bewegungen aufgrund physikalischer Gesetzmäßigkeiten voraussetzt. Für eine solche Simulation werden die natürlichen Einschränkungen der Bewegungsrichtungen einzelner Gliedmaßen, der Schwerpunkte zum Halten des Gleichgewichts und Eigenschaften wie Stärke oder Gewicht des virtuellen Charakters benötigt, deren Berechnung eine hohe Rechenleistung in Anspruch nimmt.

Badler et. al [1] haben im Projekt Jack ein System realisiert, mit dem "physikalische" Handlungen in einer virtuellen Welt möglich sind. Bei Jack handelt es sich sowohl um eine Animationsplattform, als auch um eine Entwurfsplattform für virtuelle Charaktere. Sie bietet die Möglichkeit virtuelle Charaktere koloriert durch ein Low-Resolution-Modell mit 110 Polygonen darzustellen, um die Berechnungszeiten gering zu halten, oder für darstellungsorientierte Anwendungen texturierte Modelle mit 1180 Polygonen zu verwenden.

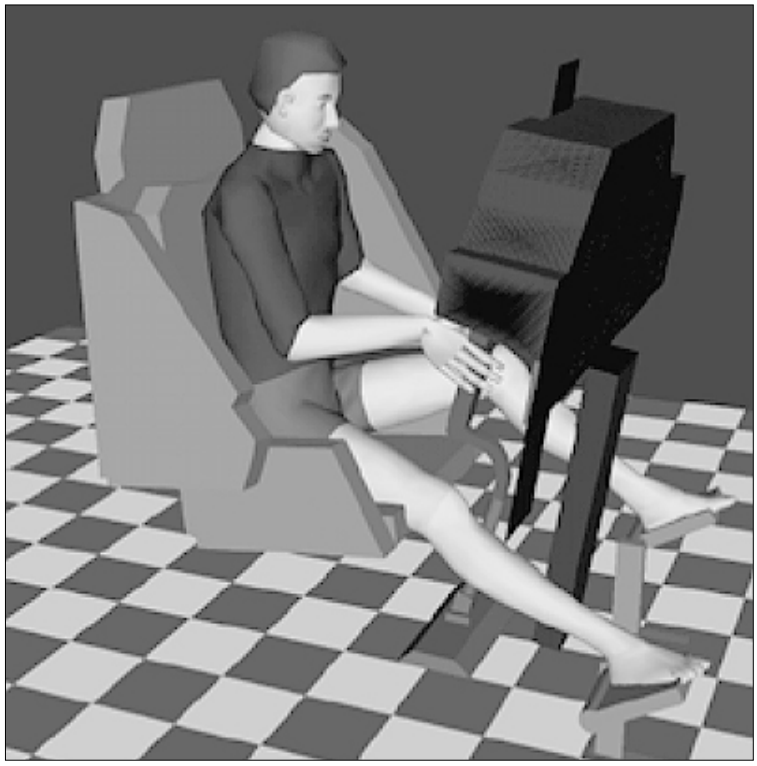


Abbildung 2.3: Jack als High-Resolution-Modell in einer Apache-Hubschrauber-Simulation [1]

Virtuelle Charaktere in der Jack-Software wurden aus 69 einzelnen Segmenten zusammengesetzt, wobei ein Skelett mit 68 Knoten verwendet wurde. Jede Hand wurde mit 16 Knoten umgesetzt, um das Greifen von Gegenständen realistisch darstellen zu können. Die Human Animation Spezifikation (H-Anim), die als Richtlinie zur Modellierung humanoider Modelle von einer Arbeitsgruppe des Web3D Konsortiums erstellt wurde, sieht ein Skelett mit 94 Knoten vor, wovon zur Darstellung einer Hand 19 Knoten vorgesehen sind. Das in der H-Anim-Spezifikation beschriebene Skelett soll die exakte Nachbildung menschlicher Bewegungsabläufe ermöglichen [8]. Durch eine vorgegebene Namensgebung für alle Knoten und den einheitlichen Aufbau der Skelette soll die Verwendung der H-Anim-Spezifikation die Verwendbarkeit von humanoiden Modellen in unterschiedlichen Einsatzgebieten ermöglichen.

Bei dem im Projekt Jack verwendeten Skelett wurden die Bewegungsrichtungen einzelner Knoten auf ein oder zwei Achsen beschränkt, um bei der Berechung von Rotationswerten weniger Rechenleistung zu benötigen. Diese Beschränkungen werden mit degree of freedom (DOF) bezeichnet und geben die Summe aller Bewegungsachsen im Skelett eines 3D-

Modells an. Bei Jack wurde der DOF auf 136 anstatt der möglichen 204 Bewegungsrichtungen (68 Knoten und drei Rotationsachsen) reduziert. Für jeden Knoten wurden Grenzwerte für Rotationen festgelegt, die nicht überschritten werden können, was ein unnatürliches Verdrehen von Gliedmaßen verhindert. Die Einschränkung der Rotationen sowie die Verwendung von DOF sind nötig, um den Einsatz von inverser Kinematik zu vereinfachen. Werden in einer Animationsplattform nur vordefinierte Animationen abgespielt, kann die Einschränkung von Rotationswinkeln eine fehlerhafte Berechnung von Rotationen und ein dadurch mögliches Verdrehen von Gliedmaßen verhindern, was jedoch einen zusätzlichen Rechenaufwand bedeutet.

Im Projekt Jack wird IK eingesetzt, um beispielsweise das Annähern der Hände an einen Gegenstand zu ermöglichen, was bei einer wechselnden Positionierung von Gegenständen nicht durch vorgegebene Animationen umgesetzt werden kann.

Abbildung 2.4: Low-Resolution-Modell von Jack [1]

Zum Annähern der Hand an einen Gegenstand wird im Projekt Jack ein Programm aufgerufen, welches anhand von IK die Winkel des Handgelenks, des Ellenbogens und der Schulter abhängig von der Position der Hand berechnet. Zum Greifen des Gegenstands wird ein Programm aufgerufen, welches Collision-Detection verwendet, um die Hand um

einen Gegenstand zu schließen. Die Verwendung einzelner Programme ermöglicht es, aufwändige Berechnungen nur bei Bedarf auszuführen und bei einfacheren Handlungen wegzulassen. Für einfachere Handlungen wurden bei Jack vorgefertigte Bewegungsabläufe in einer Datenbank gespeichert, die über Motion-Capture-Verfahren erzeugt wurden, um ein möglichst natürliches Aussehen zu erreichen.

Da im Projekt Art-E-Fact keine Interaktion zwischen virtuellen Charakteren und virtuellen Objekten geplant sind und die Animationsplattform hauptsächlich zur Darstellung von Gesten eingesetzt werden soll, kann auf den Einsatz von IK verzichtete werden. Vordefinierte Animationen, die in einer 3D-Modellierungssoftware erstellt oder durch Motion-Capture-Verfahren aufgezeichnet werden, stellen eine ausreichende Möglichkeit zur Realisierung von Gesten dar. Die Verwendung von vordefinierten Animationen hat zudem den Vorteil, dass ein Durchdringen von Körperteilen nicht anhand von Collision-Detection-Methoden verhindert werden muss. Die Beschränkungen der Bewegungsrichtungen werden ebenfalls nicht benötigt, da die Berechnungen der Animationswerte nur wenig Rechenleistung benötigen und die Werte sich nur innerhalb der durch die Animation vorgegebenen Bereiche bewegen.

Im Gegensatz zur prozeduralen Berechnung von Bewegungen wie im Projekt Jack, hat Perlin [2] versucht, unter Verwendung einfacher Algorithmen menschenähnliche Animationen am Beispiel einer virtuellen Tänzerin zu erzeugen. Das Skelett der Tänzerin ist in 19 Segmente unterteilt. Zur Animation der einzelnen Segmente werden minimale und maximale Rotationswerte für die drei Achsen eines Knotens festgelegt. In der in Abbildung 2.5 dargestellten Animation ist der rechte Arm im ersten Schritt angewinkelt, im dritten Schritt gestreckt und im letzten Schritt wieder angewinkelt, wobei sich der Ellenbogen zwischen den minimalen und maximalen Rotationswerten hin- und herbewegt. Während der Animation wird periodisch zwischen den festgelegten Extremwerten interpoliert. Die Interpolation wird anhand von zwei Funktionen berechnet, die jeweils mit normaler und doppelter Geschwindigkeit verwendet werden können, wodurch für jeden Knoten vier unterschiedliche Animationen zur Verfügung stehen.

Abbildung 2.5: Beispiel einer periodischen Animation [9]

Zum Erzeugen einer Animation werden für jeden Knoten im Skelett Rotationswerte in der folgenden Form festgelegt:

{15, 0, 5} {-15,0,-5} {s1, 0, s1} LChest

Die Werte in den ersten beiden Klammern beschreiben die minimalen und maximalen Rotationswinkel für die Achsen x, y und z. In der dritten Klammer wird für jede der drei Achsen eine der vier Interpolationsarten angegeben. Ein solches Set mit Rotationswerten für alle 19 Knoten des Skeletts beschreibt eine periodische Animation. Durch den Einsatz von minimalen zufälligen Winkeländerungen, so genannten Noise-Bewegungen, wird jede Periode individualisiert. Durch die Verwendung von verschiedenen Sets können unterschiedliche Animationen abgespielt werden.

Das Zusammenstellen eines Sets stellt jedoch einen erheblichen Aufwand dar, da die einzelnen Rotationswerte manuell eingegeben werden müssen, was das Erzeugen einer menschenähnlichen Bewegung erschwert. Zudem sind nur wenige Animationen in periodischer Form darstellbar, was den Einsatz dieses Verfahrens für die Animationsplattform ausschließt.

Die Übergänge von einer Animation zur nächsten, werden durch eine Veränderung von Gewichtungen ermöglicht (Abbildung 2.6).

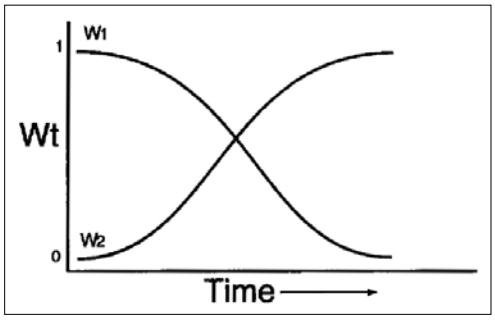


Abbildung 2.6: Verlauf der Gewichtung zweier Animationen [2]

Die Übergänge zwischen den Animationen werden am Ende einer Periode umgesetzt, indem die Rotationswerte für beide Animationen berechnet und im Verhältnis der Gewichtungen interpoliert werden.

Der Benutzer hat die Möglichkeit, einzelne Animationen anhand von Buttons ein- und auszuschalten, während sich die Tänzerin bewegt. Daraus können sich Kombinationen von Animationen ergeben, die nicht ausgeführt werden können, beispielsweise das Drehen einer Pirouette während die Figur läuft. Verhindert wird eine solche Kombination durch die Verwendung von Abhängigkeiten zwischen den einzelnen Gewichtungen der Animationen. Stetige Animationen wie das Laufen oder Tanzen werden durch das Ausführen von einfachen Animationen unterbrochen und nach dem Abspielen der Animationen wieder fortgesetzt. Durch den Aufruf einer Pirouette wird die Gewichtung des Laufens automatisch auf null reduziert und am Ende der Pirouette wieder auf eins angehoben.

Die im Projekt Jack verwirklichte Animationsplattform bietet die Möglichkeit menschliche Verhaltensweisen anhand von virtuellen Charakteren in einer virtuellen Umgebung zu simulieren. Für die im Projekt Art-E-Fact benötigte Darstellung von Gestik und Mimik würde der Einsatz der Jack-Software einen unnötigen Mehraufwand an Rechenleistung durch eine Implementierung nicht benötigter Kontroll- und Animationsverfahren bedeuten. Das Abspielen von detailreichen vordefinierten Keyframe-Animationen stellt im Vergleich zur prozeduralen Berechnung von Animationen beispielsweise durch inverse Kinematik, eine ausreichende Lösung für die im Rahmen dieser Diplomarbeit zu erstellende Animationsplattform dar. Das von Perlin verwendete Verfahren zur Realisierung von Ani-

mationen ist durch die Einschränkung der periodischen Verarbeitung der Animationen und wegen des hohen Aufwandes zum Erzeugen dieser Animationen kein ausreichendes Verfahren zur Darstellung menschlicher Gestik.

Aus diesen Gründen ist es sinnvoll, für das Projekt Art-E-Fact eine Animationsplattform zu entwickeln, die das Abspielen von vordefinierten Animationen ermöglicht, die beispielsweise in einer 3D-Modellierungssoftware entworfen werden.

3 Anforderungen an die Animationsplattform

Aus den Vorgaben für das Projekt Art-E-Fact und der Verwendung von vordefinierten Animationen ergeben sich die im Folgenden beschriebenen Anforderungen an die im Rahmen dieser Diplomarbeit zu realisierende Animationsplattform.

Die Hauptanforderung an die Animationsplattform besteht darin, virtuelle Charaktere mit annähernd menschlicher Gestik und Mimik darzustellen und mit synthetischer Sprachausgabe zu versehen. Dabei liegt der Schwerpunkt nicht auf der physikalisch korrekten Umsetzung der Gestik, was beispielsweise für Halten des Gleichgewichts eines virtuellen Charakters benötigt würde, sondern auf dem natürlichen Aussehen der Animationen. Für das Projekt Art-E-Fact wird keine Interaktion zwischen Charakteren und ihrer virtuellen Umwelt benötigt, weshalb auf eine Einbindung von inverser Kinematik oder Collision-Detection-Verfahren verzichtet werden kann.

Autoren müssen durch einfache Vorgaben in einem Editor Gesten, Emotionen und Text kombinieren können, wobei die endgültige Auswahl der Gesten von der Plattform vorgenommen werden soll. Hierzu muss die Plattform über eine Möglichkeit verfügen, Animationen anhand einer Beschreibung auswählen zu können, um beispielsweise verschiedene Kombinationen aus Emotionen und Gesten zu realisieren. Sollen zwei Gesten miteinander verknüpft werden, die beide den rechten Arm eines virtuellen Charakters bewegen, muss eine Beschreibung vorhanden sein, die es der Plattform ermöglicht, eine der beiden Animationen auszuwählen.

Werden Gesten nacheinander ausgeführt, ist es wichtig, nahtlose Übergänge zu schaffen. Da Animationen nicht immer die gleiche Ausgangs- und Endpose besitzen, ist ein Überblenden zwischen dem Ende einer Animation und dem Anfang einer zweiten Animation erforderlich. Dazu soll die Möglichkeit untersucht werden, Animationen zu verwenden, welche die Endposition einer Animation mit der Anfangsposition einer zweiten Animation verbinden. Anstelle von Animationen können auch expressive Posen eingesetzt werden.

Beim Abspielen einzelner Animationen sollte die Möglichkeit bestehen, bestimmte Bereiche der Animation zu wiederholen, was z. B. beim Winken oder Klatschen nötig ist, um die Animation zu verlängern. Ein einfaches Skalieren der Dauer der Animation wäre in diesem Fall nicht ausreichend. Für Gesten wie das Deuten in eine bestimmte Richtung, wäre es sinnvoll, die Animation an einer Stelle für eine bestimmte Zeit anhalten zu können.

Die Animationen sollten auf andere virtuelle Charaktere übertragbar sein, um eine flexible Nutzung der Plattform mit wechselnden Charakteren zu ermöglichen und um den Verwaltungsaufwand bei der Auswahl von Animationen gering zu halten.

Um die Animationsplattform auch auf weniger leistungsfähigen Rechnern laufen lassen zu können, sollte sie auch auf verteilten Systemen ausführbar sein. Hierbei ist die eventuelle Nutzung über das Internet zu berücksichtigen.

Im nächsten Kapitel wird der Lösungsansatz für die Umsetzung der Plattform beschrieben, der aus den Anforderungen, die in diesem Kapitel beschrieben wurden und unter Berücksichtigung bereits bestehender Animationstechniken entwickelt wurde.

4 Hauptmerkmale der Animationsplattform

4.1 Einleitung

In diesem Kapitel werden das Konzept der Plattform und der Aufbau der dazugehörigen Klassenstruktur erläutert. Eine der Vorgaben der Arbeit war, Avalon zur Darstellung zu verwenden. Daraus ergibt sich die Verwendung von VRML als Beschreibungsformat für virtuelle Charaktere. Zu Beginn der Diplomarbeit wurde die Darstellung der Animationsplattform in Java3D realisiert, da die benötigte Schnittstelle in Avalon zu Verwendung von Java erst zu einem späteren Zeitpunkt zur Verfügung stand.

4.2 Lösungsansatz

Für ein ähnliches Projekt des ZGDV wurden bereits Animationen in Maya erstellt und zusammen mit einem 3D-Modell in VRML-Dateien exportiert, d. h. das 3D-Modell wurde zusammen mit der Animation in eine Datei gespeichert. Die einzelnen Dateien wurden dann zur Laufzeit in einem VRML-Player nachgeladen. Die Manipulation einer solchen Animation wäre nur durch Veränderungen an der entsprechenden VRML-Datei zu erreichen. Das Überblenden zu einer zweiten Animation ist dabei nur mit sehr hohem Aufwand zu verwirklichen. Animationen können dabei auch nur durch einen erneuten Export der Modelle aus Maya auf andere Charaktere übertragen werden, was diese Möglichkeit der Animation für das Projekt Art-E-Fact ausschließt.

Der Ansatz für die Animationsplattform beruht auf der Idee, statische VRML-Modelle durch Zuweisen von Transformationswerten zu animieren. Der Vorteil dabei ist, dass zum Erzeugen von Transformationswerten unterschiedliche Quellen wie beispielsweise Keyframe-Animationen oder inverse Kinematik verwendet werden können. Die Möglichkeit der prozeduralen Berechnung von Animationen, wie zum Beispiel von Badler [1] verwendet, um zielgerichtete Aktionen auszuführen, wurde jedoch aufgrund der mangelhaften optischen Ergebnisse und der hohen benötigten Rechenleistung vorerst außer Acht gelassen.

Die Auswahl des Formates, in dem die Animationen gespeichert werden können, ist abhängig von den unterstützten Exportformaten der Modellierungssoftware. In Maya wird das .anim-Format angeboten, welches Animationen in Form von Bézierkurven beschreibt. Dadurch wäre es möglich, Animationen nicht nur in ihrer Dauer zu skalieren, sondern durch die Veränderung von Amplitudenwerten die Ausprägung der Animation zu verrin-

gern oder zu erhöhen. Die 3D-Modelle für das Projekt Art-E-Fact und die dazugehörigen Animationen werden jedoch in 3DStudioMax erstellt, wodurch sich die Auswahl des Animationsformates lediglich auf VRML beschränkt, weil das Schreiben eines Exporters für das .anim-Format für 3DStudioMax einen zu großen Aufwand darstellt. Da es sich bei in VRML-Dateien gespeicherten Animationen um Keyframe-Animationen handelt, sind keine Veränderungen an der Ausprägung der Animationen möglich.

Bei der Modellierung der 3D-Modelle in VRML wurde die H-Anim-Spezifikation verwendet, worin festgelegt ist, wie die Knotenhierarchie für humanoide VRML-Modelle aufgebaut sein muss und wie diese Knoten benannt werden müssen (Abbildung 4.1) [8]. Die anhand dieser 3D-Modelle erstellten Animationen beschreiben die Veränderung der Rotationswerte eines Knotens und verwenden ebenfalls die Namensgebung nach H-Anim-Spezifikation.

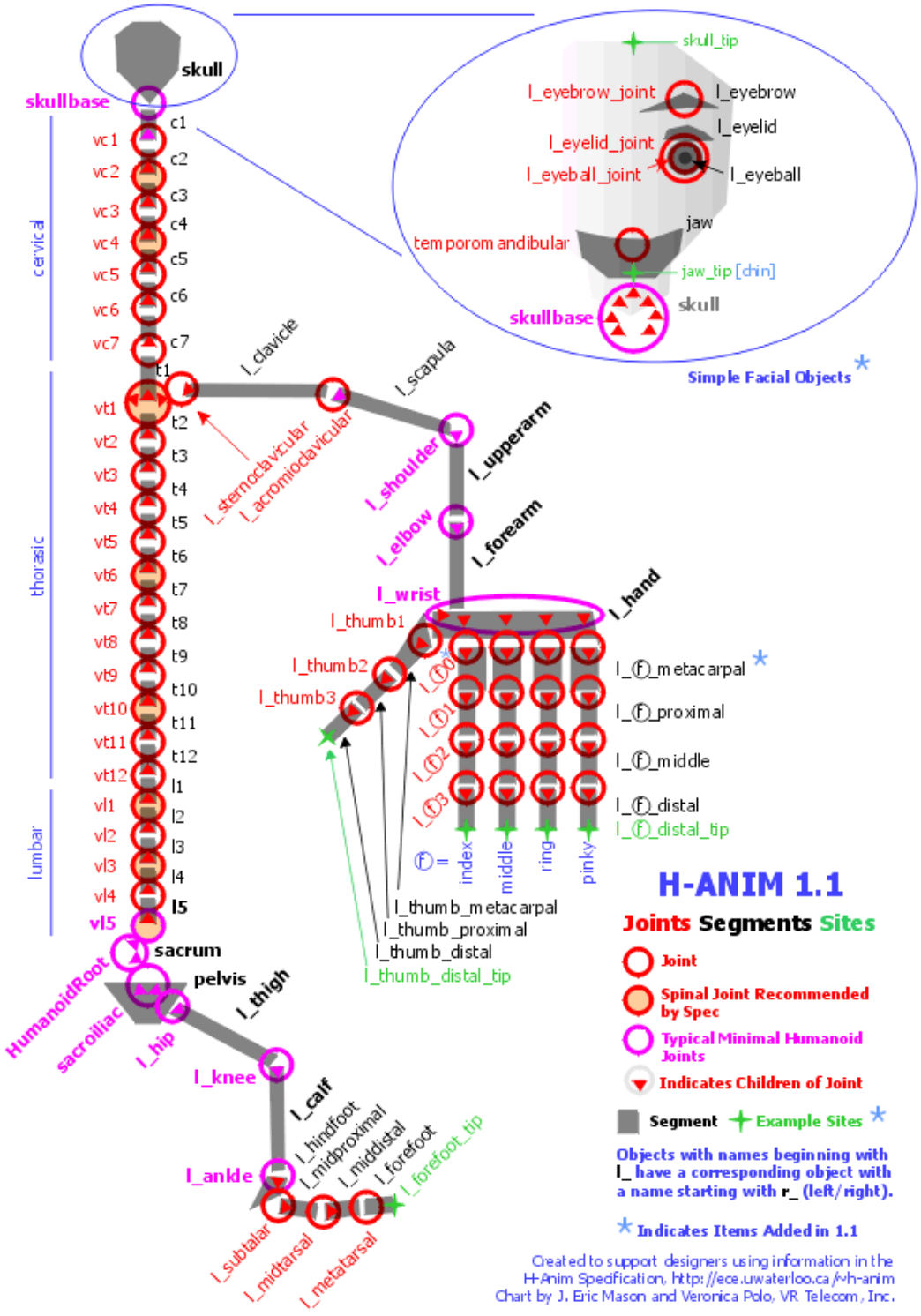


Abbildung 4.1: Aufbau eines humanoiden VRML-Modells nach H-Anim Spezifikation 2.0 [8]

Zur Unterscheidung von Modellen wird jedem Knoten der Name oder eine Beschreibung des Modells vorangestellt (Tabelle 4.1).

Knotenname	**Knoten im VRML-Modell Professor**
r_shoulder	professor_r_shoulder

Tabelle 4.1: Verwendung von Knotennamen in VRML-Modellen

Zur Darstellung von Emotionen werden für jeden virtuellen Charakter drei zusätzliche VRML-Dateien mit verschiedenen Körperhaltungen verwendet. Die Rotationswerte für eine bestimmte Emotion werden zur Laufzeit auf das 3D-Modell des virtuellen Charakters übertragen. Zur Vereinfachung der Implementierung werden lediglich die Emotionen Glück, Trauer und Wut verwendet. Diese Emotionen stellen drei von sechs Grundemotionen dar, deren Mimik unabhängig vom kulturellen Hintergrund eines Landes von Menschen erkannt wird [4] (Abbildung 4.2).

	Glück	Furcht	Überraschung	Zorn	Ekel, Abscheu	Trauer
Japan	87	71	87	63	82	74
Brasilien	97	77	82	82	86	82
Chile	90	78	88	76	85	90
Argentinien	94	68	93	72	79	85
USA	97	88	81	69	82	73

Abbildung 4.2: Richtige Zuordnung (in Prozent) von Gesichtsausdrücken und Emotionsbegriffen in fünf verschiedenen Kulturen. Die Beurteiler waren Studenten/innen [4]

4.3 Modulstruktur

Das vorgegebene UML-Klassendiagramm, wurde aus einem ähnlichen Projekt übernommen und auf das Projekt Art-E-Fact angepasst. Der Grundgedanke, auf dem die übernommene Klassenstruktur basiert, ist, dass die Intelligenz der Klassen mit zunehmender Hierarchietiefe abnimmt. Die virtuellen Charaktere werden dabei durch zwei Klassen dargestellt, die AgentivePerson und ManifestPerson.

Die AgentivePerson soll die Persönlichkeit eines Charakters erzeugen, indem gewisse Grundemotionen gespeichert und entsprechende Gesten ausgewählt werden, die zu dem Charakter passen. Zusätzlich sollen in der AgentivePerson Zufallsbewegungen erzeugt und Entscheidungen über Positionen wie z. B. der Blickrichtung getroffen werden.

Die Klasse ManifestPerson soll die Berechnung der Animationen übernehmen. Hierfür wird für jede Art von Animation ein Modulator erzeugt, welcher Animationsbeschreibungen bekommt und für einen bestimmten Zeitpunkt Transformationswerte berechnet.

Zum Verteilen der Animationsbeschreibungen an die Modulatoren wird in die ManifestPerson die Klasse Controller eingefügt, welche diese anhand vorgegebener Kriterien auswertet, evtl. verändert und an die entsprechenden Modulatoren übergibt. Die berechneten Werte werden von den Modulatoren an eine übergeordnete Klasse, die ModulatorLoop zurückgegeben. Diese Klasse fügt alle Werte, die zur Darstellung von Position, Gestik, Mimik und Emotion berechnet werden, zusammen und überträgt die Werte über eine Schnittstelle an die entsprechende Darstellungsebene. Eine weitere Aufgabe der Klasse ModulatorLoop ist es, die Berechnung von Animationswerten innerhalb der Modulatoren in einem bestimmten Intervall zu starten. Die Modulatoren und die Klasse ModulatorLoop sind dabei nur ausführende Komponenten und enthalten keinerlei Entscheidungsoptionen (Abbildung 4.3).

Die Klasse Player ist den virtuellen Charakteren übergeordnet und koordiniert das Abspielen einzelner Handlungen. Einzelne Handlungen oder ganze Szenen werden in einem Editor erzeugt, welcher in der Klasse RRLEditor umgesetzt ist. Zum Speichern von Handlungen bietet sich die auf XML basierende Rich Representation Language (RRL) an, die zu diesem Zweck im Rahmen des Projektes NECA entworfen wurde (Kapitel 5.2.2).

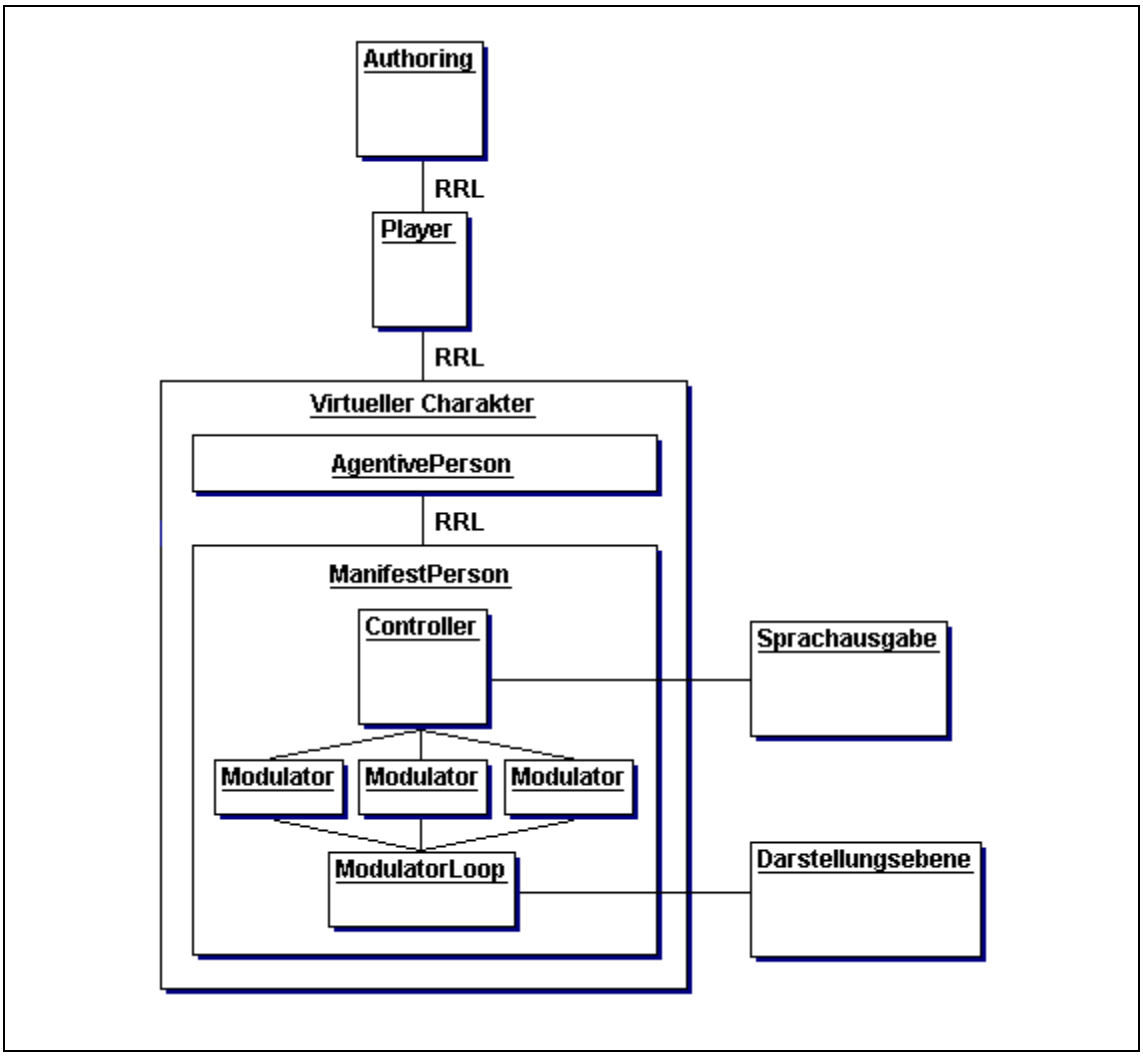


Abbildung 4.3: Hierarchie der Komponenten der Animationsplattform

Zum Starten der Animationsplattform wird in der Klasse LocalPlatformStart eine Instanz der Darstellungsebene erzeugt. Anschließend werden für jeden virtuellen Charakter eine Instanz der Klasse AgentivePerson und eine Instanz der Klasse ManifestPerson erzeugt. Für jeden virtuellen Charakter wird in der Darstellungsebene ein 3D-Modell geladen. Die Namen der virtuellen Charaktere sowie die Pfadangaben zu den VRML-Dateien der 3D-Modelle werden aus einer XML-Datei ausgelesen und in einem Objekt der Klasse Avatar-Participant gespeichert. Für jeden virtuellen Charakter wird ein Objekt der Klasse Avatar-Participant erzeugt und der Klasse ManifestPerson des virtuellen Charakters im Konstruktor übergeben. Nach den virtuellen Charakteren werden je eine Instanz der Klassen Player

und RRLEditor erzeugt. Danach ist der Startvorgang der Animationsplattform abgeschlossen.

Im nächsten Kapitel wird der Ablauf der Verarbeitung von RRL innerhalb der Animationsplattform beschrieben.

4.4 Ablauf der Verarbeitung

Der Ablauf der Verarbeitung beginnt mit dem Erzeugen von Handlungen für einen oder mehrere virtuelle Charaktere im RRL-Editor. Die Handlungen beinhalten Sprache in Form von Text, Gesten und Emotionen. Alle Handlungen werden im RRL-Editor in ein RRL-Dokument im XML-Format gespeichert. Das Abspielen der Handlungen in der Animationsplattform wird über den Send-Button des Editors gestartet. Durch Drücken des Send-Buttons wird das RRL-Dokument an die Klasse Player übertragen, welche die Reihenfolge, in der die Handlungen festgelegt wurden, überwacht, und die einzelnen Handlungen in Form von RRL an die virtuellen Charaktere überträgt. Die virtuellen Charaktere werden durch die Klassen AgentivePerson und ManifestPerson dargestellt. In der Klasse ManifestPerson wird das RRL mit den Handlungen für diesen virtuellen Charakter ausgewertet. Der Text den der virtuelle Charakter sprechen soll, wird an die Sprachsynthese übertragen, Gesten und Emotionen an den dafür vorgesehenen Modulator, der die Animationen berechnet. Die Animationswerte werden nach der Berechnung den 3D-Modellen zugewiesen. Wurden alle Handlungen für diesen virtuellen Charakter ausgeführt, meldet dieser der Klasse Player seinen aktuellen Status. Daraufhin verteilt der Player die nächsten Handlungen an den jeweiligen virtuellen Charakter, bis alle im RRL-Editor erzeugten Handlungen ausgeführt wurden.

Im nächsten Kapitel wird die Realisierung der in diesem Kapitel beschriebenen Komponenten beschrieben. Die Reihenfolge, in der die Komponenten beschrieben werden, ist an die Reihenfolge der Verarbeitung innerhalb der Animationsplattform angelehnt.

5 Realisierung der Animationsplattform

5.1 Einleitung

In diesem Kapitel wird die Funktionsweise und Umsetzung der einzelnen Komponenten der Laufzeitumgebung erläutert. Die Reihenfolge, in der die Komponenten beschrieben werden, ist dabei an die Verarbeitung innerhalb der Plattform angelehnt. Es wird zunächst das Erstellen von Szenen beschrieben (Kapitel 5.2). Danach wird die Kommunikationsstruktur innerhalb der Plattform erläutert (Kapitel 5.3), sowie das Verteilen der Handlungen auf die virtuellen Charaktere (Kapitel 5.4). Anschließend wird die Verarbeitung von Daten innerhalb der virtuellen Charaktere dargestellt (Kapitel 5.5). Am Ende dieses Kapitels werden die Implementierung der Darstellungsebene der Animationsplattform (Kapitel 5.6) und die Generierung synthetischer Sprache (Kapitel 5.7) sowie eine Klasse zur Konvertierung von XML-Bestandteilen (Kapitel 5.8) beschrieben.

5.2 RRL-Editor zum Erstellen von Szenen

5.2.1 Einleitung

In diesem Kapitel wird zunächst der Aufbau der Rich Representation Language beschrieben. Anschließend werden die Anforderungen an den RRL-Editor und verschiedene Lösungsansätze zur Implementierung des Editors vorgestellt. Danach erfolgen eine Beschreibung des Editors, sowie die Konvertierung einer Szenenbeschreibung in die Notation des RRL.

5.2.2 Rich Representation Language

Die Rich Representation Language (RRL) wurde im Rahmen des NECA-Projektes[1] entwickelt, um alle möglichen Handlungen virtueller Charaktere in textueller Form beschreiben und steuern zu können. Das Projekt NECA[2] hat das Ziel eine Plattform für das Internet zu erstellen, die eine Kommunikation zwischen animierten virtuellen Charakteren ermöglicht, wobei die virtuellen Charaktere persönliche Verhaltensweisen und bevorzugte

[1] http://www.ai.univie.ac.at/NECA/RRL/

[2] Net enviroment for embodied emotional conversational agents

Gewohnheiten besitzen sollen. RRL basiert auf der Extended Markup Language (XML). Zu Beginn des Art-E-Fact Projektes lag die RRL-Spezifikation in der Version 0.4 vor, in welcher die Möglichkeit besteht, Gespräche mit Gestik, Lippenbewegungen und synthetischer Sprache zwischen mehreren virtuellen Personen zu beschreiben.

Die Spezifikation sieht eine Erstellung des RRL in mehreren Schritten vor, wobei im ersten Schritt nur die Gesten und die kompletten Sätze, die ein virtueller Charakter sprechen soll, in den entsprechenden XML-Tags gespeichert werden. In weiteren Schritten werden unter Verwendung von synthetischer Sprachgenerierung einzelne Sprachbestandteile, so genannte Phoneme, in das RRL gespeichert. Phoneme sind im SAMPA-Code (Speech Assessment Methods Phonetic Alphabet Code) notiert und stellen ein Alphabet für die verschiedenen Laute dar, die zur Generierung von synthetischer Sprache benötigt werden. Zu jedem Phonem werden die Dauer des Phonems in Millisekunden, sowie Pitch-Angaben zur Regulierung der Tonhöhe gespeichert. Aus der Gesamtdauer der Phoneme lassen sich eine zeitlich genauere Zuordnung der Gesten, sowie eine textbasierte Darstellung der Lippenbewegungen, so genannte Viseme, erzeugen.

Im Projekt Art-E-Fact wird das RRL in einem einzigen Schritt generiert. Aus diesem Grund konnte auf die Verwendung des in der Spezifikation vorgesehenen processingState-Tags verzichtet werden, welches Auskunft über den Stand der Verarbeitung gibt. Zur Verwendung von Emotionen wurde RRL noch um entsprechende Tags erweitert.

Ein RRL-Dokument gliedert sich in die drei Bereiche "participants", "temporalOrdering" und "setOfActs".

```
<RRL>
    <participants>
    </participants>
    <temporalOrdering>
    </temporalOrdering>
    <setOfActs>
    </setOfActs>
</RRL>
```

Abbildung 5.1: Gliederung eines RRL-Dokuments

Der Bereich "participants" enthält die Beschreibung der virtuellen Personen, welche in einer Szene verwendet werden. Für jede Person wird der Name als Attribut im person-Tag, das Geschlecht (gender-Tag) und die in der Sprachsynthese verwendete Stimme (voice-Tag) gespeichert. Angaben über Charaktereigenschaften sind in der Spezifikation vorgesehen, wurden jedoch in der bisherigen Implementierung der Plattform nicht verwendet.

```
<participants>
    <person id="Karl">
        <gender type="[male/female]"/>
        <voice name="de2"></voice>
    </person>
    <person> ... </person>
</participants>
```

Abbildung 5.2: Aufbau des Knotens participants

Im Bereich "temporalOrdering" wird der zeitliche Ablauf der Handlungen der virtuellen Charaktere festgelegt. Zur Steuerung können Angaben über sequentielle und parallele Verarbeitung anhand von <seq>- und <par>-Tags gemacht werden, wodurch die Charaktere Handlungen nacheinander oder parallel ausführen können. Innerhalb der seq- und par-Knoten werden act-Knoten erstellt, in welchen das id-Attribut eines dialogueAct gespeichert wird.

```
<temporalOrdering>
    <seq>
        <act id="d_1">
        <par>
            <act id="d_2">
            <act id="d_3">
        </par>
        <act id="d_4">
    </seq>
</temporalOrdering>
```

Abbildung 5.3: Aufbau des Knotens temporalOrdering

Der Bereich "setOfActs" beinhaltet alle in einer Szene ausgeführten Handlungen aller Charaktere. Die Handlungen einer Person zu einem bestimmten Zeitpunkt werden in "dia-

logueActs" zusammengefasst. Diese werden im setOfActs-Tag als Unterknoten gespeichert.

```
<setOfActs>
    <dialogueAct> … </dialogueAct>
    <dialogueAct> … </dialogueAct>
    …
<setOfActs>
```

Abbildung 5.4: Aufbau des Knotens setOfActs

Für jeden "dialogueAct" ist ein Sprecher und Adressat festzulegen. Innerhalb der dialogueAct-Knoten werden alle Sätze aufgelistet, die während dieses dialogueActs gesprochen werden sollen. In welcher Form die Texte gespeichert werden, ist abhängig von der Verarbeitung innerhalb der Sprachsynthese. Im NECA-Projekt wurden zum Speichern sprachbezogener Daten so genannte sentence-, word-, syllable- und ph-Knoten verwendet, welche direkt von der Sprachsynthese interpretiert werden konnten. Für das Projekt Art-E-Fact wurde diese Struktur nicht verwirklicht, da die verwendeten Sprachsynthesen damit nicht kompatibel sind. Anweisungen für die Sprachsynthese werden hier als Textobjekte in den sentence-Knoten gespeichert. Jeder "dialogueAct" kann mehrere sentence-Knoten enthalten, welche ein für das Dokument eindeutiges id-Attribut bekommen.

Innerhalb des dialogueActs befindet sich der animationSpec-Knoten, in welchem alle Arten von Animationen gespeichert werden. Bei den Animationen kann es sich um Gesten und Viseme handeln, welche unter Verwendung der <seq> und <par> Tags nacheinander oder parallel angeordnet werden können. Da sowohl für Gesten als auch für Viseme der gesture-Knoten benutzt wird, enthalten die Tags für Viseme zusätzlich das Attribut modality="viseme", wodurch eine Unterscheidung zwischen Geste und Visem ermöglicht wird. Weitere Attribute für gesture-Tags sind begin, welches die Anfangszeit einer Geste in Millisekunden angibt, oder duration, das die Dauer eines Visems oder einer Geste in Millisekunden angibt. Das Attribut identifier gibt den Namen des Visems oder den Namen der ausgewählten Geste an. Jeder gesture-Tag erhält ebenfalls eine für das RRL-Dokument eindeutige ID.

Für das Art-E-Fact-Projekt wurde zum animationSpec-Knoten der Knoten emotion hinzugefügt, welcher ebenfalls die Attribute begin und identifier enthält und es ermöglicht,

einer virtuellen Person Emotionen in Form von Körperhaltung, Gesichtsausdruck und je nach verwendeter Sprachsynthese auch emotionale Sprache zuzuordnen.

```
<dialogueAct id="d_1">
  <speaker id=""/>
  <adressee id=""/>
  <sentence id="s_1"> Text </sentence>
  <sentence id="s_2"> Text </sentence>
  ...
  <animationSpec>
    <par>
      <seq>
        <emotion begin="100" identifier="sad">
      </seq>
      <seq>
        <gesture begin="234" dur="1400" id="g_1"
                                              identifier="winken"/>
        <par>
          <gesture begin="2000" dur="1400" id="g_2"
                                                 identifier="..."/>
          <gesture begin="3000" dur="1000" id="g_2"
                                                 identifier="..."/>
        </par>
      </seq>
      <seq>
        <gesture dur="150" identifier="v_ac" modality="viseme"/>
        <gesture dur="64" identifier="v_p" modality="viseme"/>
        <gesture dur="126" identifier="v_ac" modality="viseme"/>
        <gesture dur="97" identifier="v_p" modality="viseme"/>
        <gesture dur="135" identifier="v_aI" modality="viseme"/>
        ...
      </seq
    </par>
  </animationSpec>
</dialogueAct>
```

Abbildung 5.5: Struktur eines dialogueActs

5.2.3 Anforderungen an den RRL-Editor

Der RRL-Editor soll es einem Autor ermöglichen, Gespräche mit einem vorgegebenen Ablauf zwischen mehreren Charakteren zu erstellen. Hierbei müssen Texte einem Sprecher zugeordnet werden können. Weiterhin soll die Möglichkeit bestehen, Gesten und Emotionen an bestimmten Stellen im Text einzufügen und deren Dauer festzulegen. Bei der Verwendung von Gesten muss eine Überlappung zwischen dem Ende der ersten und dem Anfang der zweiten Geste vorgesehen werden, um nahtlose Übergänge verwirklichen zu können.

Erstellte Szenen müssen gespeichert, geladen und in ein RRL-Dokument konvertiert werden können. Der Editor sollte ebenfalls die Möglichkeit bieten, die erstellten RRL-Dokumente zu bearbeiten, ggfs. zu im- und exportieren und zum Testen an die Animations-Plattform zu senden.

5.2.4 Lösungsansätze

Im Folgenden werden zwei Lösungsansätze für die Implementierung des RRL-Editors vorgestellt.

Einer der Lösungsansätze ist graphisch und besteht darin, dass für jeden virtuellen Charakter, der in einer Szene verwendet werden soll, eine endlose Zeitleiste erstellt wird. Jede Zeitleiste enthält ein Feld zur Texteingabe und weitere drei Zeilen, in denen Gesten und Emotionen eingefügt werden können. Anfang und Ende einer Animation werden an den Buchstaben des eingegebenen Textes oder anderen Animationen ausgerichtet, wobei eine Animation des Charakters A auch am Text des Charakters B ausgerichtet werden kann. Die Ausrichtung soll wie in der Abbildung 5.6 zu sehen, anhand von Zeigern, dargestellt durch senkrechte Pfeile, realisiert werden. Dies soll ein zeitlich synchronisiertes Abspielen von Animationen ermöglichen, z. B. als Reaktion von Charakter A auf einen Text des Charakters B.

Animationen werden in Form von Schiebereglern eingefügt, wobei die Animation durch eine Drop-Down-Liste im Reglerelement ausgewählt werden kann. Zur Begrenzung der Spieldauer einer Animation wird ein zweites Reglerelement verwendet. Die Überlagerung zweier Animationen wird durch die Verwendung mehrerer Animationszeilen ermöglicht. Bei der Verwendung von Emotionen ist ein Eingabefeld für die Gewichtung der möglichen Emotionszustände vorzusehen.

Wann ein Charakter beginnt einen Text zu sprechen, wird ebenfalls durch die Verwendung von Zeigern festgelegt, die zwischen den Textfeldern der einzelnen Zeitleisten eingesetzt werden können.

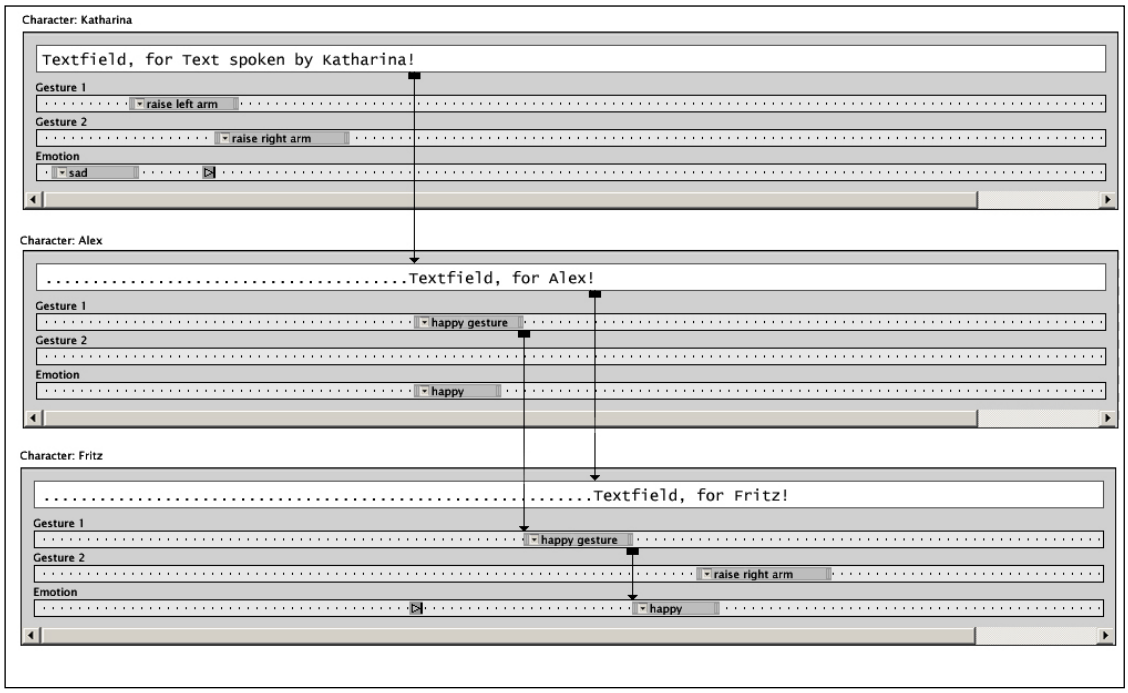


Abbildung 5.6: Entwurf einer GUI zum Erstellen einer Szene

Im zweiten Lösungsansatz wird ein Texteditor verwendet, der das Erstellen, Markieren und Bearbeiten von Texten ermöglicht. Zur Zuordnung eines Sprechers oder einer Animation zu einem Textabschnitt werden Tags eingesetzt, die entweder vor oder nach einem markierten Textbereich oder an der aktuellen Position des Cursors eingefügt werden. Die Auswahl der Sprecher, Gesten und Emotionen wird über ein Menü getroffen, das mit der rechten Maustaste geöffnet werden kann. Für die Verwendung von Emotionen muss eine Eingabemöglichkeit für Gewichtungswerte der einzelnen Emotionen vorgesehen werden.

Aufgrund des hohen programmiertechnischen Aufwands, den die Implementierung des graphischen Ansatzes mit sich bringt, fiel die Entscheidung zugunsten der textbasierten

Variante. Die Realisierung des graphischen Entwurfs wird jedoch von einer am Projekt Art-E-Fact beteiligten Institution vorgenommen.

5.2.5 Umsetzung

5.2.5.1 Graphical User Interface

Der RRL-Editor wurde in der Klasse RRLEditor im Package characterPlatform.authoring realisiert. Das Graphical User Interface (GUI) des RRL-Editors wurde mit Hilfe des GUI-Editors der Entwicklungsumgebung NetBeans erstellt. Der GUI-Editor ist ein WYSIWYG-System, mit dessen Hilfe man Benutzeroberflächen graphisch gestalten kann, wobei der Editor den Java-Quelltext generiert. Hierbei werden entsprechend der verwendeten Objekte bereits Methodenrümpfe, die z. B. zur Einbindung eines Buttons nötig sind, automatisch im Quelltext eingefügt. Die zur Benutzung von Buttons und Shortcuts nötigen Listener-Objekte werden ebenfalls im Quelltext erzeugt. Die Oberfläche wurde mit Elementen aus dem Java-Swing-Paket erstellt.

Dem Hauptfenster wurde eine Menüleiste mit den Einträgen Datei, Bearbeiten, Ansicht und Hilfe hinzugefügt. Im Dateimenü wurden die Einträge Öffnen, Schließen, Speichern, Speichern unter, Importieren, Exportieren und Beenden eingefügt. Die hierfür benötigten Dialogfelder zum Öffnen und Speichern sowie die dazu nötigen Dateioperationen wurden ebenfalls über den GUI-Editor der Entwicklungsumgebung erstellt. Im Menüfeld Bearbeiten wurden die Einträge Ausschneiden, Kopieren und Einfügen zur Textbearbeitung und im Menüfeld Ansicht der Eintrag ViewRRL erzeugt.

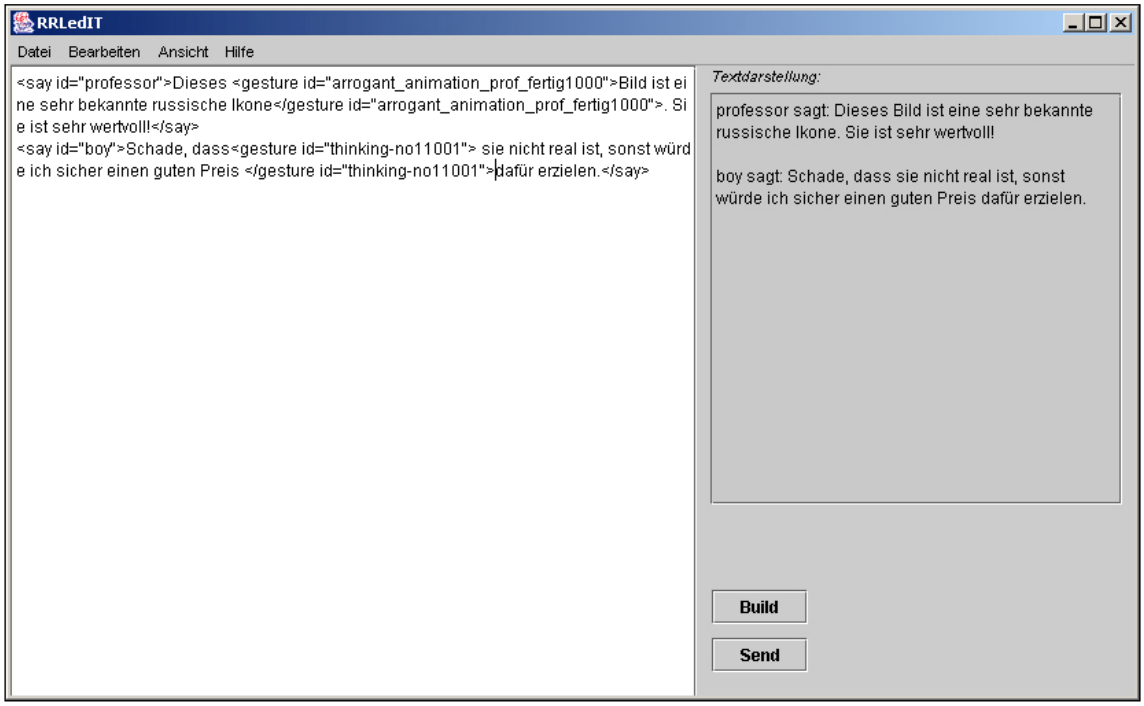


Abbildung 5.7: GUI des textbasierten RRL-Editors

Zur Realisierung eines editierbaren Textfeldes wurde eine Instanz der Klasse JEditorPane verwendet, welche Methoden zur Verfügung stellt, um einfache Textoperationen wie z. B. Markieren, Kopieren und Einfügen durchzuführen. Dem Textfeld wurde ein MouseListener hinzugefügt, welcher zur Verwendung eines PopUp-Menüs benötigt wird. Die Einträge in diesem PopUp-Menü sind Avatar, Geste, Emotion und Adressat, sowie zur Textbearbeitung Kopieren, Ausschneiden und Einfügen.

5.2.5.2 Erstellen einer Szene

Bei der Erstellung einer Szene benötigt der RRL-Editor Informationen über die zur Verfügung stehenden virtuellen Charaktere. Diese Informationen erhält er durch ein Array von AvatarParticipants. Die Einträge in den Menüfeldern Avatar und Adressat werden durch die Namen der virtuellen Charaktere erzeugt, die im Array der AvatarParticipants gespeichert sind. Die zur Verfügung stehenden Emotionen werden ebenfalls aus diesem Array ausgelesen. Weiterhin bekommt er Informationen über die zur Verfügung stehenden Animationen durch eine Referenz auf ein Objekt der Klasse AnimationService.

Nach der Initialisierung des Editors kann der Benutzer nun durch Eingabe von Text und Zuweisen von Sprecher, Gesten und Emotionen über das Menü der rechten Maustaste Gespräche zwischen mehreren virtuellen Personen erstellen.

Durch Stringoperationen werden markierte Textbereiche mit der Notation des Editors erweitert und Informationen über zugewiesene Sprecher, Gesten und Emotionen zugewiesen. Das folgende Beispiel zeigt zwei Sätze, die von zwei verschiedenen Charakteren gesprochen werden und Emotionen und Gesten verwenden.

```
<say id="charakter1"> <emotion neutral="0,0" happy="1,0" angry="0,0"
sad="0,0"/>Das <gesture id="deuten1001"> ist nur ein Test!</gesture
id="deuten1001"></say>
<say id="charakter2"> <emotion neutral="0,0" happy="1,0" angry="0,0"
sad="0,0"/>Aber au<gesture id="ja1001">ch der ist wichtig.</gesture
id="ja1001"></say>
```

Tabelle 5.1: Erzeugte Notation für zwei Sprecher mit Text, Emotionen und Gesten

Im Beispiel ist gut zu erkennen, dass bei der Verwendung von mehreren Gesten und Emotionen der Text nur noch schwer lesbar ist. Aus diesem Grund wurde dem Editor noch ein Darstellungsfenster hinzugefügt, in welchem die Namen der virtuellen Charaktere mit nachstehendem Text ohne Notation angezeigt werden.

Das fertige Skript kann nun in RRL konvertiert werden, wozu im Editor eine Instanz der Klasse RRLBuilder erzeugt wird (Kapitel 5.2.6). Über den Build-Button des Editors wird das erzeugte Skript als String-Objekt an die Methode buildRRL() der Klasse RRLBuilder übergeben. Diese Methode gibt ein Objekt vom Typ Document zurück, welches das fertige RRL im XML-Format darstellt.

Durch die Auswahl des Unterpunktes ViewRRL im Menüpunkt Ansicht wird das erstellte RRL-Dokument unter Verwendung der Methode documentToString() der Klasse XMLConverter in ein String-Objekt umgewandelt und in einem neuen Fenster dargestellt (Kapitel 5.8). Hierzu wurde ebenfalls eine Instanz der Klasse JEditorPane verwendet, um Änderungen am RRL-Dokument vornehmen zu können. Dem ViewRRL-Fenster wurde noch ein Button hinzugefügt, wodurch Änderungen am RRL übernommen werden können.

Durch Verwendung der Optionen Importieren und Exportieren im Datei-Menü des Editors können fertige RRL-Dokumente in Dateien gespeichert oder aus Dateien in den Editor gelesen werden.

Zum Abspielen des RRL in der Animationsplattform wurde dem Editorfenster ein Send-Button hinzugefügt, welcher das RRL an die Klasse Player überträgt (Kapitel 5.4). Der Player verteilt das RRL an die virtuellen Charaktere.

5.2.5.3 Die Notation des RRL-Editors

Die Notation des RRL-Editors sieht vor, bei der Zuordnung von Charakteren, Gesten, Emotionen und Adressaten auf XML basierende Tags zu verwenden. Es existieren zwei unterschiedliche Arten von Tags. Eine Art der Tags schließt Textbereiche ein, wobei die Attribute des Tags Informationen zu diesem Textbereich beinhalten. Diese Tags haben die folgende Form:

```
<tag-Name [Attribut="Wert"]> Text </tag-Name>
```

Die zweite Form der verwendeten Tags beinhaltet keinen Textbereich, sondern wird lediglich zum Speichern von Werten benutzt. Sie haben die Form:

```
<tag-Name [Attribut="Wert"]/>
```

Auf dieser Basis wurde die im Folgenden beschriebene Notation erstellt.

Jeder Text, der von einem Charakter gesprochen werden soll, muss von einem say-Tag umschlossen sein. Die Zuordnung des Charakters geschieht über das ID-Attribut des say-Tags (Abbildung 5.8).

```
<say id="Charaktername"> Text </say>
```

Abbildung 5.8: say-Tag in der Notation des RRL-Editors

Zur Zuweisung von Gesten werden zwei unterschiedliche Formen von Gesture-Tags verwendet. Gesture-Tags können nur innerhalb eines Textbereiches eingesetzt werden, der von einem say-Tag umgeben wird. Die erste Form der gesture-Tags ermöglicht die Skalierung einer Geste auf die Dauer des eingeschlossenen Textes. Hierbei kann die Dauer je

nach verwendeter Sprachsynthese variieren. Bei Verwendung der zweiten Form wird die Geste in der Originaldauer abgespielt (Abbildung 5.9).

```
Form 1:
  <gesture id="Gestenname1000"> Text </gesture id="Gestenname1000">

Form 2:
  <gesture id="Gestenname1000"/>
```

Abbildung 5.9: Formen der gesture-Tags in der Notation des RRL-Editors

Eine Überlagerung von Gesten ist durch eine Überschneidung von Gesture-Tags möglich (Abbildung 5.10). In der in Abbildung 5.10 dargestellten vierten Form werden beide Gesten in der Originaldauer abgespielt. Ist die Dauer der ersten Geste kürzer als die Dauer des Textes zwischen den beiden Tags führt diese Form nicht zu einer Überlagerung mit der zweiten Geste. Um eine eindeutige Zuordnung von Anfangs- zu End-Tag zu gewährleisten, wurde dem ID-Attribut eine für diese Szene eindeutige vierstellige Zahl hinzugefügt. Ohne diese Zuordnung wäre eine Überlagerung von Anfangs- und End-Tags in der Form 1 und 2, bei der Verwendung von identischen Gesten, nicht zu unterscheiden.

```
Form 1:
  <gesture A> Sein oder<gesture B> nicht</gesture A> sein</gesture B>
Form 2:
  <gesture A> Sein oder<gesture B> nicht</gesture B> sein</gesture A>
Form 3:
  <gesture A> Sein oder<gesture B/> nicht</gesture A> sein
Form 4:
  <gesture A/> Sein oder nicht<gesture B/> sein das ist hier die Frage
```

Abbildung 5.10: Möglichkeiten der Überlagerung von gesture-Tags

Emotionen werden durch so genannte emotion-Tags beschrieben und dürfen wie gesture-Tags nur innerhalb eines von say-Tags eingeschlossenen Textbereiches eingefügt werden. Den einzelnen Emotionen können Gewichtungswerte im Bereich zwischen 0.0 und 1.0 zugeordnet werden (Abbildung 5.11). Abhängig von der verwendeten Sprachsynthese-Software können Emotionen nicht nur auf die Mimik und Körperhaltung eines virtuellen Charakters, sondern auch auf die Sprachausgabe angewendet werden. Dabei sind Emotio-

nen jedoch nur auf vollständige Sätze anwendbar. Werden in einem Satz mehrere emotion-Tags verwendet, so hat nur der erste Auswirkungen auf die Sprachausgabe und auch nur auf den Satz, in dem er steht. Jeder weitere emotion-Tag beeinflusst jedoch die Körperhaltung und den Gesichtsausdruck des Charakters und hat so lange Gültigkeit, bis ein weiterer emotion-Tag verwendet wird. Standardmäßig wird innerhalb der Plattform die Gewichtung der neutralen Emotion auf den Wert 1.0 gesetzt.

```
<emotion neutral="0.0" happy="0.0" angry="0.0" sad="0.0"/>
```

Abbildung 5.11: Emotion-Tag in der Notation des Editors

Der adressee-Tag gibt an, welchem Charakter sich der Sprecher zuwenden soll. Hierbei wird nur der erste Tag innerhalb eines say-Tags ausgewertet (Abbildung 5.12). Soll sich der Sprecher innerhalb eines Monologes zwei Charakteren nacheinander zuwenden, so ist dies durch zwei say-Tags mit unterschiedlichem adressee-Tag zu verwirklichen.

```
<adressee id="Charaktername"/>
```

Abbildung 5.12: Adressee-Tag der Notation des RRL-Editors

5.2.6 RRLBuilder

Zu Beginn der Diplomarbeit wurde die Konvertierung der Editornotation in das XML-Format der Rich Representation Language (RRL) im Editor selbst realisiert. Da im Bereich Authoring des Projektes ebenfalls Skripte in der Notation des Editors erstellt werden, wurde die Konvertierung in die eigene Klasse RRLBuilder ausgelagert und in der Methode buildRRL() umgesetzt. In diesem Kapitel wird zunächst das Erzeugen von XML-Dokumenten beschrieben, das zur Konvertierung der Notation benötigt wird. Im Anschluss daran wird der eigentliche Konvertiervorgang beschrieben.

In der gesamten Animationsplattform wird zum Erstellen von XML-Dokumenten das Xerces-Package aus dem Apache-XML-Projekt verwendet [17]. Dieses Package implemen-

tiert die Java-XML API (JAXP 1.2) und stellt DOM[3]- und SAX[4]-Parser zum Auslesen sowie Klassen zum Erstellen von XML-Dokumenten zur Verfügung.

In JAXP werden XML-Dokumente über das Interface Document definiert. Jedes XML-Dokument enthält einen Wurzelknoten, von dem aus eine Baumstruktur von Knoten erzeugt werden kann. Alle erzeugten Knoten müssen an das Wurzelelement des Dokuments oder an untergeordnete Knoten angehängt werden, wobei nur ein Wurzelelement pro Dokument verwendet werden darf. Knoten werden über das Interface Node definiert und bieten somit Methoden zum Hinzufügen und Entfernen von Knoten und Durchlaufen der Hierarchie an.

Im DOM werden insgesamt zwölf verschiedene Arten von Knoten definiert, wovon in der vorliegenden Arbeit nur die Knoten vom Typ Element und Text verwendet werden. Da es sich bei Document, Element und Text um Interfaces handelt, können keine direkten Instanzen gebildet werden. Zur Erstellung eines Objektes vom Typ Document wird folgender Ausdruck verwendet:

```
Document doc = new DocumentImpl();
```

Die Klasse DocumentImpl ist hierbei die Implementierung eines Documents aus dem Xerces Package.

Zur Erzeugung von Element- und Text-Objekten werden die Methoden createElement() und createTextNode() des Document-Objektes verwendet, wodurch Instanzen des jeweiligen Objektes erzeugt und von den Methoden zurückgegeben werden. Über die Methode setAttribute() können einem Element Attribute in Form von String-Parametern übergeben werden. Hierbei handelt es sich um Key-/Value-Paare, die in der Form key="value" in das entsprechende Element gespeichert werden.

<xml header> <rrl> <participants> <person id="Karl">	Document doc = new DocumentImpl(); Element root = doc.createElement("rrl"); Element participants = doc.createElement("participants");

[3] Document Object Model
[4] Simple API for XML

</person>
</participants>
</rrl>
	Element person = doc.createElement("person");
	person.setAttribute("id","Karl");
	participants.appendChild(person);
	root.appendChild(participants);
	doc.appendChild(root);

Tabelle 5.2: Die XML-Struktur auf der linken Seite wird durch den Quelltext auf der rechten Seite erzeugt

Bei der Konvertierung der Notation des Editors in das RRL-Format wird im ersten Schritt ein XML-document-Objekt erzeugt, womit der Wurzelknoten rrl sowie die Knoten participants, temporalOrdering und setOfActs erstellt werden.

Der Knoten participants, der alle Informationen über die virtuellen Charaktere beinhaltet, speichert unter anderem die Angabe der zu verwendenden Sprachbibliothek eines virtuellen Charakters. Dazu wird dem RRLBuilder im Konstruktor vom RRL-Editor das Array der AvatarParticipant-Instanzen übergeben. Die zum Erzeugen von Phonemen benötigte Sprachsynthese wird bereits beim Start der Plattform erzeugt und dem RRLBuilder als Referenz vom Typ JSAPIXtendedInterface im Konstruktor übergeben.

Die vom RRL-Editor als String an die Methode buildRRL() übergebene Szene setzt sich aus einzelnen say-Tags zusammen (Abbildung 5.13). Jedes say-Tag wird innerhalb der Methode buildRRL() in eine dialogueAct Knotenhierarchie konvertiert.

```
<say id="Professor">Das <gesture id="deuten1001"> ist nur ein
Test!</gesture id="deuten1001"></say>
<say id="Student">Aber au<gesture id="ja1002">ch der ist wich-
tig.</gesture id="ja1002"></say>
```

Abbildung 5.13: Beispiel einer erstellten Szene

Unter Verwendung der String-Methode split() wird die Szene in ein Stringarray mit dem Namen currentNotation aufgeteilt, wobei zum Unterteilen der Suchparameter "<say " verwendet wird. Jeder String im Array beginnt nun mit "id=", da der Suchparameter durch die Methode split gelöscht wurde (Abbildung 5.14).

```
id="Professor">Das <gesture id="deuten1001"> ist nur ein
Test!</gesture id="deuten1001"></say>
```

```
id="Student">Aber au<gesture id="ja1002">ch der ist wichtig.</gesture
id="ja1002"></say>
```

Abbildung 5.14: Inhalt des Stringarrays nach Ausführen der Methode split()

Das id-Attribut enthält den Namen des virtuellen Charakters, der die Handlungen in diesem Array-Eintrag ausführen soll. Das Stringarray wird an die Methode createPersonTag() der Klasse RRLBuilder übergeben. Hier werden die person-Tags innerhalb des Knotens participants erstellt. Dazu werden die Namen aller in der Szene vorkommenden Charaktere aus dem Stringarray ausgelesen und in den jeweiligen person-Tag gespeichert. Dem person-Tag wird der Knoten voice hinzugefügt, der die verwendete Sprachbibliothek eines virtuellen Charakters enthält. Die Sprachbibliothek wird anhand des Charakternamens aus dem AvatarParticipants-Array ausgelesen und als Attribut in den voice-Knoten gespeichert. Es muss vermieden werden, dass für einen Charakternamen mehrere Einträge im Knoten participants erzeugt werden. Deshalb wird der Charaktername nach dem Erstellen eines person-Knotens in eine HashMap gespeichert. Einträge mit gleichem Schlüsselwert werden innerhalb der HashMap überschrieben, was doppelte Einträge verhindert. Bevor ein neuer person-Knoten erzeugt wird, wird die HashMap nach dem aktuellen Namen durchsucht. Bei einer Übereinstimmung wird kein person-Knoten erstellt. Die Methode createPerson-Tag() gibt ein XML-element-Objekt zurück, welches in den participant Knoten eingefügt wird.

Alle Einträge im Stringarray currentNotation werden anschließend in einen dialogueAct konvertiert. Hierzu wird zuerst die Grundstruktur eines dialogueActs wie in Abbildung 5.5 dargestellt erzeugt, wobei dem dialogueAct-Knoten eine eindeutige ID zugeordnet wird. Für diesen dialogueAct wird gleichzeitig ein act-Knoten im temporalOrdering-Knoten angelegt, in welchem die ID des dialogueActs gespeichert wird.

Aus dem Eintrag des Stringarray currentNotation wird der Charaktername ausgelesen und als Attribut im speaker-Knoten des dialogueActs gespeichert. Beginnt ein String nicht mit "id", wird kein dialogueAct erzeugt, da offensichtlich bei der Eingabe ein Fehler verursacht wurde. Nachdem der speaker-Knoten erzeugt wurde, wird der restliche say-Tag aus dem String entfernt.

Als nächstes wird der String nach einem adressee-Tag durchsucht. Wird ein Eintrag gefunden, wird der Parameter im adressee-Knoten des dialogueAct gespeichert. Standardmäßig wird der Eintrag "camera" verwendet, der den Anwender als Adressaten angibt.

Zum Erzeugen der sentence-Knoten des dialogueAct wird eine Kopie des aktuellen String aus currentNotation verwendet.

Diese Kopie wird in der Methode getShortSentences() in einzelne Sätze unterteilt, wobei jeder Satz in ein Array gespeichert wird. Zum Unterteilen werden hierbei alle Satzendezeichen verwendet.

In einer for-Schleife wird jeder Satz nach Emotion-Tags durchsucht. Wurden einem Satz Emotionswerte zugewiesen, werden diese über die Methode setEmotion() und setMood() des Interface JSAPIXtendedInterface an die Sprachsynthese übergeben.

Aus dem aktuellen Satz werden durch die Methode removeNotation() der Klasse RRLBuilder alle Tags aus dem String gelöscht, da diese zu Fehlern in der Sprachausgabe führen würden.

Über das Interface JSAPIXtendedInterface wird in der Sprachsynthese die Methode phonemes() aufgerufen. Übergabeparameter ist der aktuelle Satz ohne Notation. Abhängig davon, ob die Sprachsynthese zum Erzeugen von synthetischer Sprache Phoneme verwendet oder Text direkt verarbeiten kann, wird ein String mit Phonemen oder Text zurückgegeben und als Textknoten in den sentence-Knoten gespeichert.

Nachdem ein sentence-Knoten erstellt worden ist, wird der aktuelle Satz an ein neues Stringobjekt angehängt. Sind alle Sätze eines dialogueAct-Knotens erstellt worden, befindet sich in diesem neuen String der komplette Text ohne Notation.

Zum Erstellen der Viseme wird der String ohne Notation an die Methode visemes() des JSAPIXtendedInterface übergeben. Diese gibt die einzelnen Viseme und deren Dauer in Millisekunden in einem String zurück. Visem und Dauer sind dabei durch eine Leertaste voneinander getrennt. Zwischen den einzelnen Visemen wird ein "#" als Separator verwendet. Anhand des Separators werden die Viseme mit der split() Methode in ein Array geschrieben. Für jeden Eintrag im Array wird ein gesture-Knoten erzeugt und an den animationSpec-Knoten angefügt.

Zum Erstellen der Gesten und Emotionen wird die Dauer jedes Wortes benötigt, um eine genaue Zuordnung der Animationen zu gewährleisten. Hierzu wird der gleiche String der zum Erzeugen der Viseme verwendet wurde, an die Methode getWordLength() des JSAPIXtendedInterface übergeben. Rückgabewert ist hier ein Array aus int-Werten. Jeder Eintrag im Array ist die Dauer eines Wortes in Millisekunden.

In einer for-Schleife wird der aktuelle String aus dem Array currentNotation durchlaufen (Abbildung 5.15). Handelt es sich bei einem Charakter um eine eckige Klammer, also um den Beginn eines Tags, wird ausgelesen, um welchen Tag es sich handelt, welche Parameter er enthält und ob es sich um ein einfaches Tag, ein Anfangs- oder Ende-Tag handelt. Innerhalb der for-Schleife laufen zwei Zähler mit, einer für die Wortanzahl und ein zweiter für die Position innerhalb eines Wortes. Anhand dieser Zähler lässt sich die Anfangszeit eines Tags berechnen. Hierbei werden die Zeiten der Wörter aufsummiert. Für jeden Buchstaben des aktuellen Wortes wird ein Mittelwert aufaddiert, der sich aus der Dauer aller Wörter und der Anzahl der Buchstaben im String berechnet.

Handelt es sich bei dem gefundenen Tag um eine Emotion, wird direkt ein Knoten erzeugt und zum emotionSpec-Knoten hinzugefügt. Im String wird der emotion-Tag gelöscht.

Bei einer Geste wird über den Gestennamen die Dauer der Geste mit der Methode getDuration() der Klasse AnimationService abgefragt. Handelt es sich bei dem Tag um einen Anfangs-Tag, werden die Anfangszeit und die Originaldauer in ein int-Array gespeichert. Dieses wird unter dem Gestennamen in eine HashMap gespeichert. Handelt es sich bei dem gefundenen Tag um einen End-Tag wird in der HashMap der Gestenname gesucht und anhand der im Array gespeicherten Anfangszeit die Dauer der Geste berechnet. Die gespeicherte Originaldauer wird hierbei überschrieben. Da beim Erstellen der gesture-Knoten darauf geachtet werden muss, ob zwei Gesten zeitlich überlappen, werden zuerst alle Tags im String ausgewertet und die Werte zwischengespeichert. Anhand der Anfangszeit und Dauer einer Geste wird verglichen, welchen der nachfolgenden Gesten die aktuelle Dauer überlagert und mit der aktuellen Geste innerhalb eines <par>-Knotens gespeichert werden muss.

Nach der Verarbeitung eines Eintrages aus dem Array currentNotation wird ein dialogueAct-Knoten zum Knoten setOfActs hinzugefügt. Die ID des dialogueAct-Knotens wird in der temporalOrdering gespeichert.

Wurden alle Einträge im Array currentNotation verarbeitet, werden die Knoten setOfActs und temporalOrdering dem root-Knoten hinzugefügt. Dieser wird zum document hinzugefügt und von der Methode buildRRL() zurückgegeben.

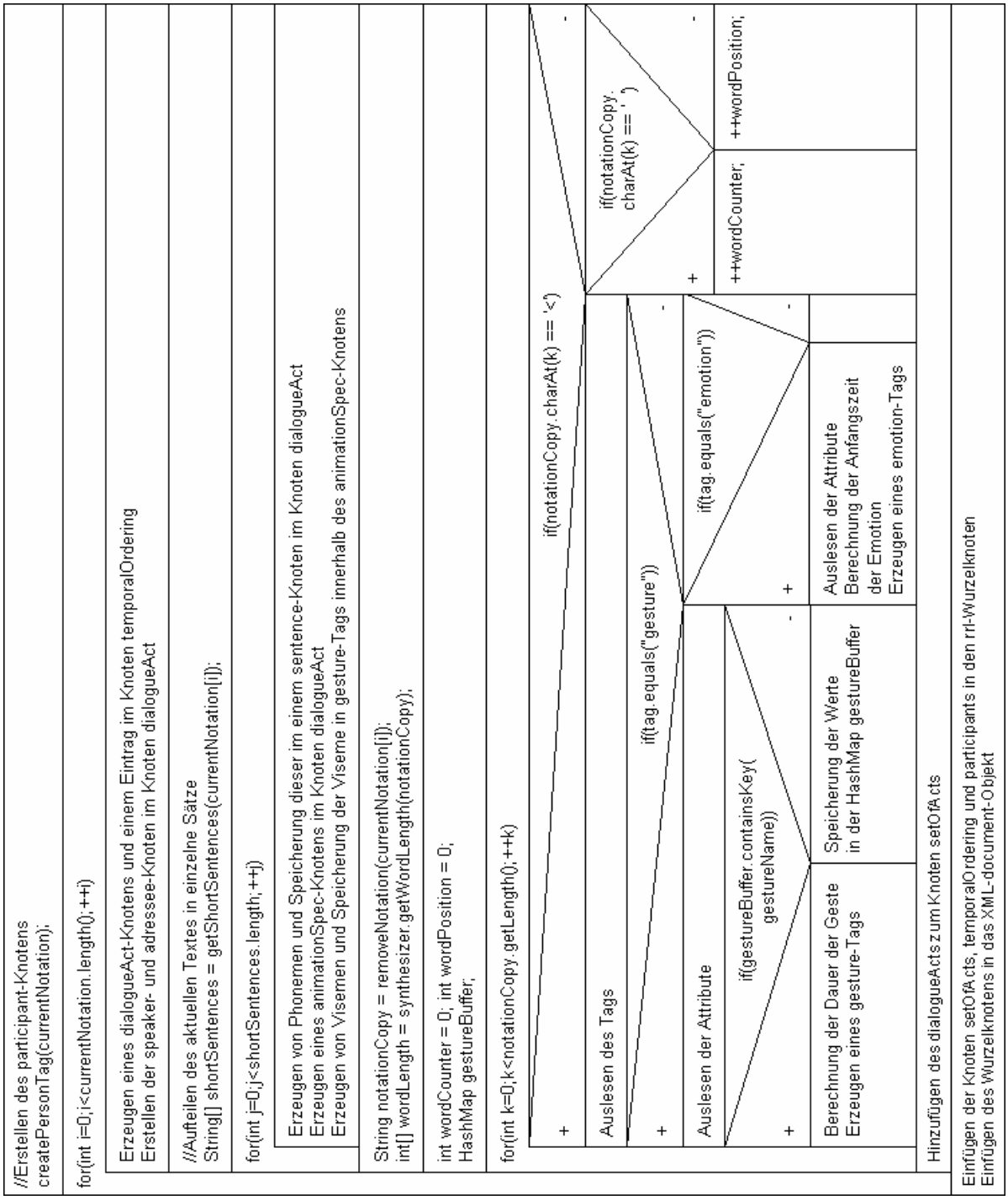


Abbildung 5.15: Konvertierung der Notation des Editors in ein RRL-Dokument

5.2.7 Zusammenfassung

Im RRL-Editor können virtuellen Charakteren zu sprechende Texte sowie Gesten und Emotionen zugewiesen werden, wodurch sich Gespräche zwischen mehreren virtuellen Charakteren umsetzen lassen. Die Szenenbeschreibungen des RRL-Editors werden in die Rich Representation Language konvertiert und können zum Abspielen an die Animationsplattform versendet werden.

Im nächsten Kapitel wird die Kommunikationsstruktur beschrieben, die verwendet wird, um RRL und Statusmeldungen an die einzelnen Komponenten der Animationsplattform zu versenden.

5.3 Kommunikation innerhalb der Plattform

Die Kommunikation zwischen den Klassen RRLEditor, Player, AgentivePerson, ManifestPerson, Controller und den Modulatoren der Animationsplattform beschränkt sich auf das Versenden von RRL-Dokumenten und den Austausch von verschiedenen Statusmeldungen zur Kontrolle der Verarbeitung. Die Übertragung dieser Daten darf die sendende Klasse nur für die Zeit der Übertragung blockieren. Die weitere Verarbeitung der Daten durch den Empfänger muss somit unabhängig vom Methodenaufruf der Senderklasse ausgeführt werden.

Die im Folgenden dargestellte Lösung, ist eine vorläufige Implementierung und soll durch den in dieser Arbeit zusätzlich entwickelten Lösungsansatz ersetzt werden (Kapitel 6.2). Der in diesem Kapitel dargelegte Ansatz wurde zunächst gewählt, da zum Zeitpunkt der Implementierung noch nicht feststand, in welcher Art die Events realisiert werden sollten, die beispielsweise die Art der Verarbeitung eines RRL-Dokuments angeben. Die Überlegung war, dass Events auch unabhängig von RRL-Dokumenten eingesetzt werden können, um beispielsweise die aktuelle Verarbeitung zu stoppen oder anzuhalten. Aus diesem Grund wurde das Versenden von RRL und Events über die Interfaces CommunicationInterface und IDEventListener realisiert.

Das CommunicationInterface wurde zur Übertragung von RRL entworfen. Der Versand erfolgt nur in eine Richtung innerhalb der Klassenhierarchie der Animationsplattform, wobei einer Klasse die ihr in der Klassenhierarchie untergeordnete Klasse als CommunicationInterface bekannt ist. Daher ist die Kommunikation nicht zielgerichtet, d. h. es kann kein

Empfänger und Sender angegeben werden. Da RRL nur in eine Richtung innerhalb der Hierarchie versendet werden muss, ist diese Lösung ausreichend. Zum Versenden von RRL wird die receive-Methode des Empfängers aufgerufen. Dieser speichert das RRL in einer Klassenvariablen. Wie die empfangenen Daten verarbeitet werden sollen, wird anhand eines Events festgelegt, der nach dem RRL an den Empfänger versendet wird.

```
package characterPlatform.core;
import  org.w3c.dom.Element;

public interface CommunicationInterface {
    public void receive(String rrl);
    public void receive(Element rrl);
    public void receive(Document rrl);
}
```

Quellcode 5.1: CommunicationInterface zur Übertragung von RRL

Zum Versand der Events an den Empfänger wird das Interface IDEventListener von den Klassen der Animationsplattform implementiert (Quellcode 5.2). Über die Methoden setEventParent() und setEventChild() werden die Listener innerhalb der Hierarchie festgelegt. An diese werden über die Methode idEvent() Event-Objekte verschickt (Quellcode 5.3).

```
package shared;
import java.util.*;

public interface IDEventListener {
    public void idEvent(IDEvent event);
    public void setEventParent(IDEventListener eventParent);
    public void setEventChild(IDEventListener eventChild);
}
```

Quellcode 5.2: Interface zum Versand von Statusmeldungen

Die verschiedenen IDs der Events geben die Art der Verarbeitung an. Zu diesem Zweck wurden folgende IDs festgelegt:

```
0 – Verarbeitung erfolgreich
1 – Fehler bei der Verarbeitung
2 – Verarbeitung beginnen
3 – Verarbeitung anhalten
4 – Verarbeitung abbrechen
5 – Verarbeitung beginnen (neuer dialogueAct)
6 – Ende des dialogueAct
20 – SlerpModulator – Verarbeitung erfolgreich
30 – EmotionModulator – Verarbeitung erfolgreich
40 – NoiseModulator – Verarbeitung erfolgreich
50 – VisemeModulator – Verarbeitung erfolgreich
60 – PositionModulator – Verarbeitung erfolgreich
```

Abbildung 5.16: Bedeutung der EventIDs

```
package shared;
import java.util.*;

public class IDEvent {
    public int id;
    public Object source;
    public String message;

    public IDEvent(Object source, int id, String message) {
        this.id = id;
        this.message = message;
        this.source = source;
    }
}
```

Quellcode 5.3: Die Klasse IDEvent

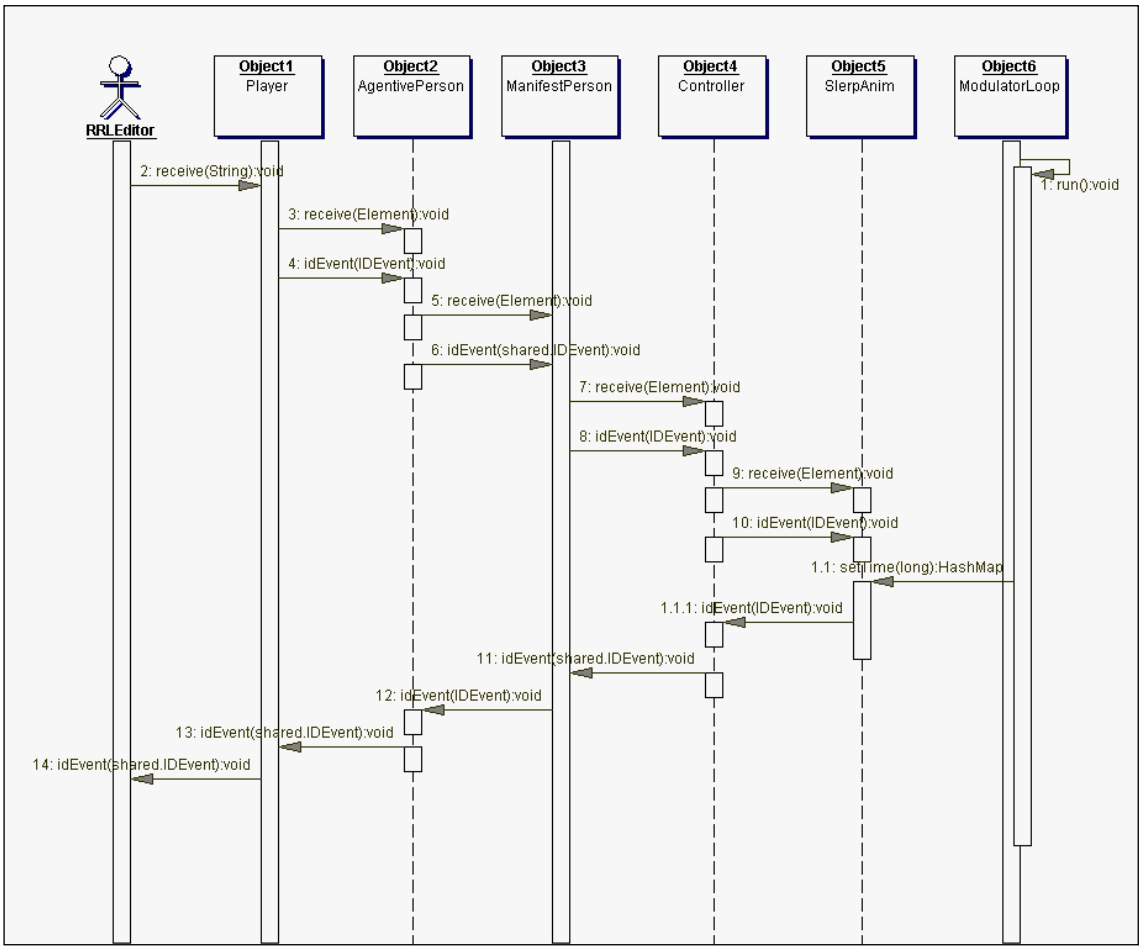


Tabelle 5.3: Ablauf der Verarbeitung innerhalb der Klassenhierarchie am Beispiel eines virtuellen Charakters

5.4 Verteilen von RRL – Klasse Player

Die Klasse Player übernimmt die Aufgabe der Steuerung von Szenen, die in Form von RRL durch den Editor generiert werden. Dazu muss das RRL-Dokument ausgewertet, und die einzelnen Bestandteile in der richtigen Reihenfolge an die virtuellen Charaktere verteilt werden. Es muss auch möglich sein, Handlungen an mehrere Charaktere annähernd gleichzeitig zu versenden, so dass parallele Handlungen ausgeführt werden können.

Die Kommunikation des Players mit den virtuellen Charakteren wird über das CommunicationInterface ermöglicht. Dem Konstruktor der Klasse Player wird dazu die HashMap

`characters` mit den Namen der virtuellen Charaktere übergeben, denen jeweils das entsprechende CommunicationInterface zugeordnet ist (Abbildung 5.17). Somit wird über diese HashMap das Versenden von RRL-Bestandteilen an die virtuellen Charaktere ermöglicht. Jedes CommunicationInterface in der HashMap stellt hierbei eine Referenz auf die Klasse AgentivePerson des jeweiligen virtuellen Charakters dar.

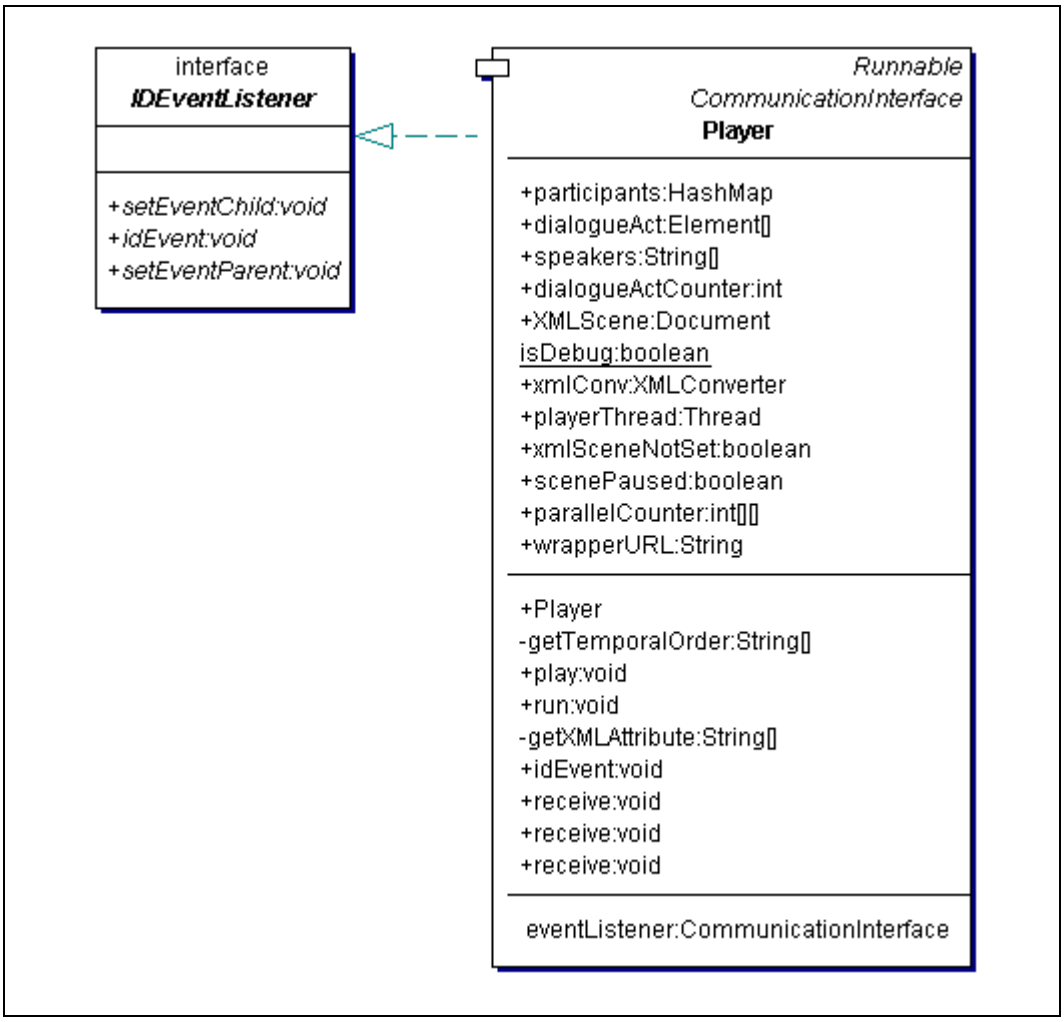


Abbildung 5.17: Klassendiagramm der Klasse Player im Package exhibitionPlatform

An die receive-Methode des CommunicationInterface der jeweiligen AgentivePerson wird ein String übergeben, welcher einen dialogueAct-Knoten beinhaltet. Die Reihenfolge in der die dialogueActs an die virtuellen Charaktere versendet werden müssen, wird aus dem Knoten temporalOrdering des RRL-Dokuments ausgelesen. Die im Knoten temporal-Ordering enthaltenen act-Knoten enthalten ein ID-Attribut, in welchem die ID des zuge-

hörigen dialogueActs gespeichert ist. Zum Abspielen von parallelen Aktionen werden die act-Knoten der parallel auszuführenden dialogueActs im Knoten temporalOrdering von par-Tags umschlossen (Tabelle 5.4).

Alle Einträge aus dem Knoten temporalOrdering werden in der String-Matrix `sceneOrdering` gespeichert, wobei jede Zeile der Matrix die IDs aller dialogueActs enthält, die gleichzeitig ausgeführt werden müssen (Tabelle 5.5).

```
<temporalOrdering>
  <act id="d_1">
  <act id="d_2">
  <par>
    <act id="d_3">
    <act id="d_4">
  </par>
  <act id="d_5">
</temporalOrdering>
```

Tabelle 5.4: Speicherung der Handlungs-Reihenfolge innerhalb der RRL

0	d_1			
1	d_2			
2	d_3	d_4		
3	d_5			

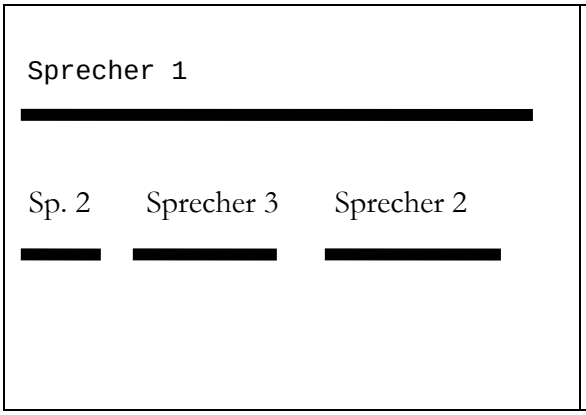
Tabelle 5.5: Speicherung der temporalOrdering in einer String-Matrix

Im nächsten Schritt werden die Namen der Sprecher aus den dialogueAct-Knoten ausgelesen. Das zugehörige CommunicationInterface wird anhand des Namens eines Sprechers aus der HashMap `characters` ausgelesen und zusammen mit der ID des dialogueActs in der HashMap `sceneCharacters` gespeichert. Die dialogueActs werden in String-Objekte konvertiert und ebenfalls anhand der jeweiligen ID in die HashMap `sceneActs` eingetragen.

Die Auswertung des RRL-Dokuments ist damit abgeschlossen. Die dialogueActs können jetzt in der Reihenfolge der IDs, die in der Matrix `sceneOrdering` gespeichert sind, versendet werden. Standardmäßig enthält jede Zeile der Matrix nur einen Eintrag, da normalerweise nur ein virtueller Charakter zu einer bestimmten Zeit eine Handlung ausführt. Der Player wartet nach dem Versenden einer oder mehrerer Handlungen bis von der entsprechenden virtuellen Person eine Rückmeldung in Form eines Events zurückkommt. Wurden parallele Handlungen versendet, wartet der Player bis alle virtuellen Charaktere ihre Handlungen ausgeführt haben. Hierfür wird eine boolesche Matrix in der gleichen Größe wie die sceneOrdering-Matrix erzeugt. Alle Werte der booleschen Matrix werden

auf true gesetzt, für jeden Eintrag in der `sceneOrdering` wird das entsprechende Feld in der booleschen Matrix auf false gesetzt. Meldet ein Event die erfolgreiche Verarbeitung eines dialogueActs, wird das aktuelle Feld der booleschen Matrix auf true gesetzt. Erst wenn alle Einträge in einer Zeile der Matrix auf true gesetzt sind, wird mit dem Versenden der nächsten Zeile der `sceneOrdering` begonnen.

Die gewählte Implementierung anhand einer Stringmatrix schließt jedoch das parallele Ausführen von sequentiellen Handlungen aus, da hierfür eine Verwendung von seq-Tags innerhalb der parallelen Handlungen nötig wäre. Ein Beispiel für diese Verwendung wäre, wenn ein virtueller Charakter einen Monolog halten würde und parallel dazu zwei weitere Charaktere einen Dialog führen (Tabelle 5.6). Eine solche Verschachtelung von par- und seq-Tags könnte durch die Verwendung einer Baumstruktur ermöglicht werden.




```
<par>
    <seq>
        Sprecher 1
    </seq>
    <seq>
        Sprecher 2
        Sprecher 3
        Sprecher 2
    </seq>
</par>
```

Tabelle 5.6: Die Gesprächsabfolge links lässt sich nur durch Verschachtelung von par- und seq-Tags wie im rechten Bild dargestellt umsetzen

Die Klasse Player verteilt die im RRL gespeicherten Handlungen, so genannte dialogueActs, anhand der im temporalOrdering-Tag des RRL gespeicherten Reihenfolge, an die virtuellen Charaktere. Dabei können durch die Verwendung von par-Tags, parallele Handlungen mehrerer Charaktere ausgeführt werden. Als nächstes wird der Aufbau eines virtuellen Charakters beschrieben.

5.5 Virtueller Charakter

5.5.1 Einleitung

Die Realisierung der virtuellen Charaktere wurde in zwei Bereiche unterteilt. Ein Bereich soll den Charakter durch Zuweisen von Grundemotionen und einer Vorauswahl von Animationen individualisieren. In diesem Bereich sollen auch Zufallsbewegungen erzeugt werden, um den Charakter in Bewegung zu halten, wenn keine vorgegebenen Handlungen ausgeführt werden. Für diesen Bereich wurde die Klasse AgentivePerson vorgesehen.

Im zweiten Bereich werden alle Handlungen des virtuellen Charakters ausgeführt. Dazu gehören das Berechnen von Animationswerten für Gestik, Mimik und Emotionen, sowie die Ausgabe von synthetischer Sprache. Dieser Bereich wurde durch die Klassenstruktur ManifestPerson realisiert.

Zur Beschreibung, welcher Charakter durch die im Package characterPlatform.core abgebildete Klassenstruktur dargestellt werden soll, wird für jeden virtuellen Charakter eine XML-Beschreibung in der Datei AvatarWorld.xml hinzugefügt (Abbildung 5.18). Die Klasse XMLWorldReadService aus dem Package shared liest diese Datei aus und erzeugt für jeden virtuellen Charakter ein Objekt der Klasse AvatarParticipant, das an die Konstruktoren der Klassen AgentivePerson und ManifestPerson übergeben wird.

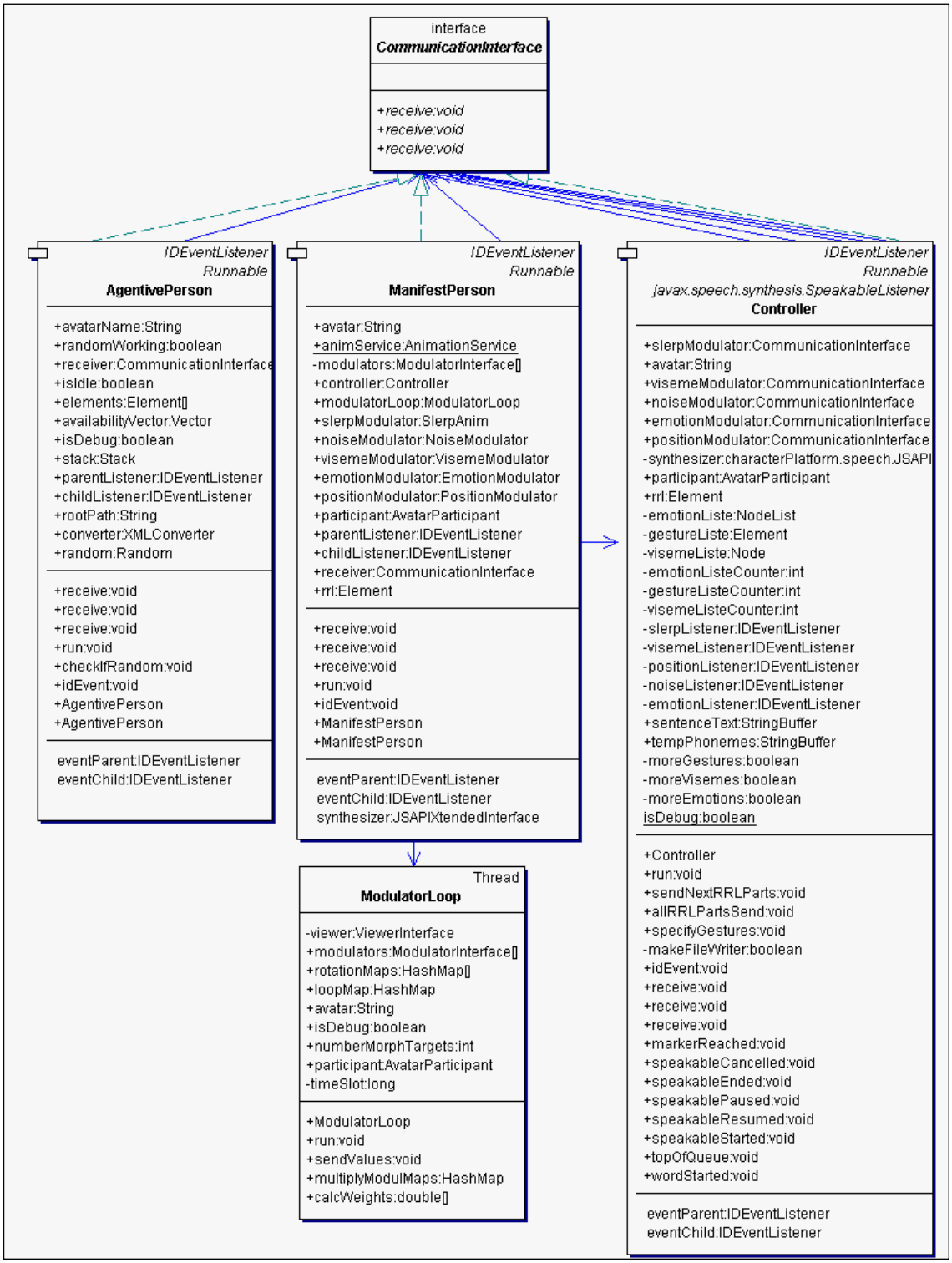


Abbildung 5.18: Klassendiagramm des Package characterPlatform.core

5.5.2 Klasse AgentivePerson

In der derzeitigen Implementierung ist aus zeitlichen Gründen ausschließlich das Erzeugen von Zufallsbewegungen umgesetzt worden. Die hierzu ausgewählten Animationen sind Gesten, wie z. B. das Verlagern des Gewichtes auf ein anderes Bein, und sollen den virtuellen Charakter in Bewegung halten, um einen lebendigen Eindruck zu erzeugen.

Innerhalb des RRL-Editors wird ein RRL-Dokument erzeugt, welches für jede Zufallsbewegung einen dialogueAct enthält (Abbildung 5.19). Das RRL-Dokument wird als Textdatei im Hauptverzeichnis der Plattform gespeichert. Im Konstruktor der AgentivePerson wird die Textdatei mit der Methode loadDOMFile() der Klasse XMLConverter in ein XML-document-Objekt konvertiert, anschließend werden alle dialogueActs in ein Element-Array gespeichert. Einzelne Gesten können jetzt über den Aufruf von receive() und einem idEvent() an die Klasse ManifestPerson übertragen werden.

```
<root>
  <dialogueAct id="d_1">
    <animationSpec>
      <par>
        <seq>
          <gesture begin="0" dur="2152" id="g_1"
                                        identifier="yes-prof-final"/>
        </seq>
      </par>
    </animationSpec>
  </dialogueAct>
  <dialogueAct id="d_2">
    <animationSpec>
      <par>
        <seq>
          <gesture begin="0" dur="3477" id="g_1"
                                         identifier="no-prof-final"/>
        </seq>
      </par>
    </animationSpec>
  </dialogueAct>
</root>
```

Abbildung 5.19: Textdokument mit Zufallsgesten in Form von einzelnen dialogueActs

Um sicher zu gehen, dass nicht immer die gleiche Geste abgespielt wird, werden alle dialogueActs aus dem Array in einen Vector gespeichert. Wurde eine Geste ausgewählt wird diese aus dem Vector entfernt. Die Auswahl eines Vectoreintrags wird über die Methode nextInt() der Klasse Random realisiert.

Anhand der eingehenden Events wird festgelegt, ob zurzeit Zufallsgesten erzeugt werden dürfen. Über receive() eingegangene RRL-Dokumente werden in einem Stack zwischengespeichert, für den Fall, dass gerade eine Zufallsgeste ausgeführt wird. Der aktuelle Status wird dabei über die boolesche Variable randomWorking definiert und in der Methode idEvent() überprüft. Wird keine Zufallsgeste ausgeführt, wird das erste Objekt aus dem Stack an die ManifestPerson übertragen und aus dem Stack entfernt (Abbildung 5.20).

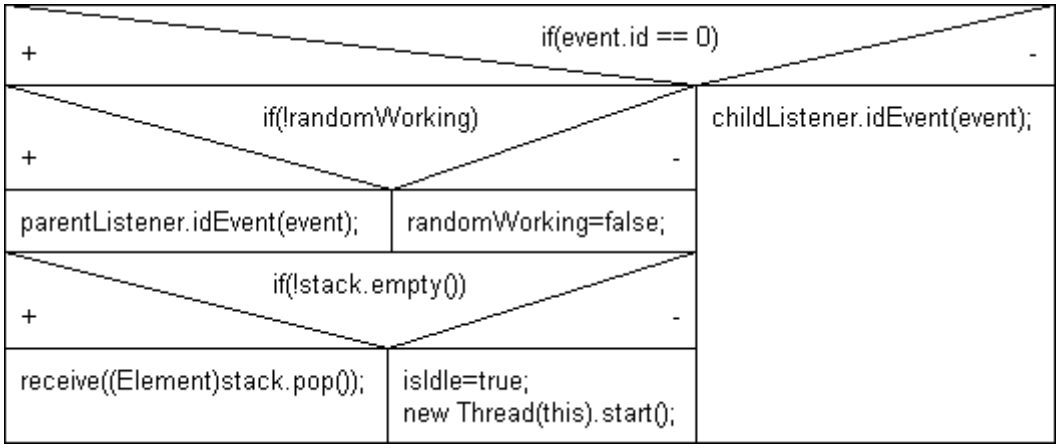


Abbildung 5.20: Struktogramm der Methode idEvent()

Befindet sich kein Eintrag im Stack, wird ein neuer Thread erzeugt, der die Methode checkIfRandom() aufruft. In dieser Methode werden die Zufallsgesten ausgewählt und an die Klasse ManifestPerson übertragen (Abbildung 5.21). Solange kein RRL von der Klasse Player eintrifft, lösen Events mit der ID 0 von der Klasse ManifestPerson neue Zufallsbewegungen aus.

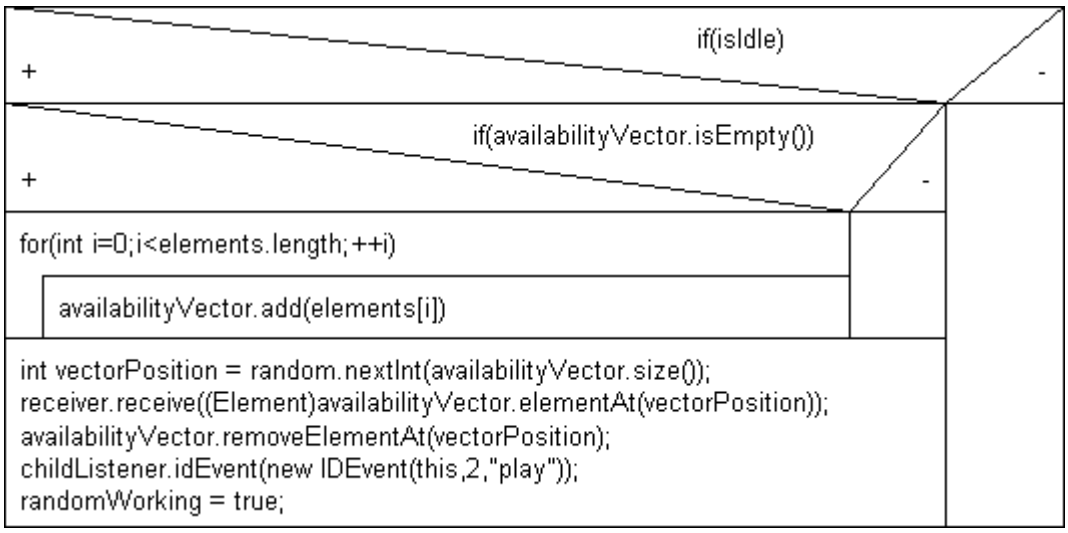


Abbildung 5.21: Funktionsweise der Methode checkIfRandom()

5.5.3 Berechnung von Animationswerten – Klasse ManifestPerson

Innerhalb der Klasse ManifestPerson müssen die im übertragenen dialogueAct gespeicherten Animationen und Sprachelemente ausgelesen und an die Modulatoren bzw. die Sprachsynthese verteilt werden. Für diese Aufgaben wird eine Instanz der Klasse Controller verwendet (Abbildung 5.18).

Zur Berechnung von Animationswerten werden in der Klasse ManifestPerson Instanzen der Klassen SlerpModulator, EmotionModulator, VisemeModulator, NoiseModulator und PositionModulator erzeugt. Die verschiedenen Modulatoren implementieren das Interface ModulatorInterface und stellen damit die Methode setTime(long systemTime) zur Verfügung. In der Methode setTime() der Modulatoren werden Animationswerte für die im Parameter systemTime übergebene aktuellen Systemzeit berechnet. Der Rückgabewert der Methode setTime() ist eine HashMap, welche die berechneten Werte enthält. Zum Aufrufen der setTime-Methoden der Modulatoren wird eine Instanz der Klasse ModulatorLoop erzeugt. Die Klasse ModulatorLoop erhält alle Modulatoren als Referenz im Konstruktor. Die Klasse ist hierbei als Thread umgesetzt worden und ruft in einer Endlosschleife die setTime()-Methode aller Modulatoren auf. Übergabeparameter ist die aktuelle Systemzeit.

Die Rückgabeparameter aus den Modulatoren werden in der ModulatorLoop zusammengefasst und an die Darstellungsebene[5] übertragen.

5.5.3.1 Controller

Eine Instanz der Klasse Controller übernimmt die Aufgaben das RRL auszuwerten, die Bestandteile für Gesten, Emotionen und Viseme an die entsprechenden Modulatoren zu verteilen und den zu sprechenden Text an die Sprachsynthese zu übertragen.

Beim Erzeugen der Modulatoren in der Klasse ManifestPerson wird der Klasse Controller je eine Referenz als CommunicationInterface übergeben. Die Sprachsynthese wird über ein JSAPIXtendedInterface im Konstruktor referenziert (Kapitel 5.7). Zum Ausführen der Methoden speak() und speakPlainText() der Sprachsynthese implementiert die Klasse Controller das Interface SpeakableListener, das eine Benachrichtigung über den aktuellen Verarbeitungsstatus der Sprachsynthese ermöglicht. Ein weiterer Parameter im Konstruktor ist eine Referenz auf das Objekt participant in der Klasse ManifestPerson, das die Angaben über die zu verwendende Stimme des virtuellen Charakters enthält.

Über den Aufruf der receive-Methode wird in der Klasse Controller ein neuer Thread erzeugt und gestartet. In der dabei ausgeführten run()-Methode werden zuerst die Sprachbestandteile aus den sentence-Tags des übertragenen dialogueActs ausgelesen und je nach verwendeter Sprachsynthese in einem String gespeichert, oder in eine Datei geschrieben (Kapitel 5.7).

Die in der animationSpec des RRL gespeicherten Animationen werden durch gesture- und emotion-Tags beschrieben. Bei Visemen und Emotionen werden dabei keine par-Tags verwendet, da diese Animationen nicht parallel ausgeführt werden. Alle Emotionen und Viseme eines dialogueActs werden nur von einem seq-Tag umschlossen. Dieser seq-Tag des XML wird für die weitere Verarbeitung gespeichert. Dazu wird die animationSpec nach emotion- und gesture-Tags durchsucht und der übergeordnete Knoten auf einer Variablen gespeichert. Gesten und Viseme werden dabei durch das modality-Attribut der Viseme unterschieden. Für Gesten wird hierbei noch überprüft, ob es sich bei dem übergeordneten Knoten um einen par-Tag handelt. Doppelte Verschachtelungen der par-Tags kommen dabei nicht vor, da nur zwei Gesten parallel abgespielt werden können.

[5] Die Darstellungsebene wird als ViewerInterface im Konstruktor der Klasse ManifestPerson referenziert.

Die Hierarchie des Knotens für Gesten wird noch einmal durchsucht, wobei jedes Element an die Methode specifyGesture(Element node) übergeben wird. Innerhalb dieser Methode sollen die Animationen anhand der Animation Description Language (animDL) ausgewertet und angepasst werden (Kapitel 6.3). Eine Implementierung im Rahmen dieser Diplomarbeit ist jedoch nicht vorgesehen.

Das Versenden der einzelnen Animationselemente an die Modulatoren wird in der Methode sendNextRRLParts(int eventID) vorgenommen (Abbildung 5.22). Der Übergabeparameter eventID gibt dabei an, welche Art von Animation übertragen werden soll. Am Ende der Methode run() wird diese Methode für jede Art von Animation einmal aufgerufen. In der Sprachsynthese wird über die Methode setVoice() die Stimme des virtuellen Charakters gesetzt und anschließend der zu sprechende Text entweder als String an die Methode speakPlainText() oder als URL einer Textdatei an die Methode speak() übergeben (Quellcode 5.4). Danach ist die Verarbeitung innerhalb des Controller-Threads beendet.

```
if(synthesizer.getName().equals("TEM")) {
  //phoFileOK = true, wenn Phoneme in Datei geschrieben wurden
  if(phoFileOK) {
    //Setzen der Stimme über JSAPIXtendedInterface
    synthesizer.setVoice( this.participant.getVoiceDataBase());
    //Phoneme in der angegebenen Datei werden gesprochen
    synthesizer.say(avatar+".pho", avatar);
  }
}
else if(synthesizer.getName().equals("ATT")) {
  synthesizer.setVoice(this.participant.getVoiceATT());
  synthesizer.speakPlainText(sentenceText.toString(),this);
}
```

Quellcode 5.4: Ausführen der Sprachsynthese

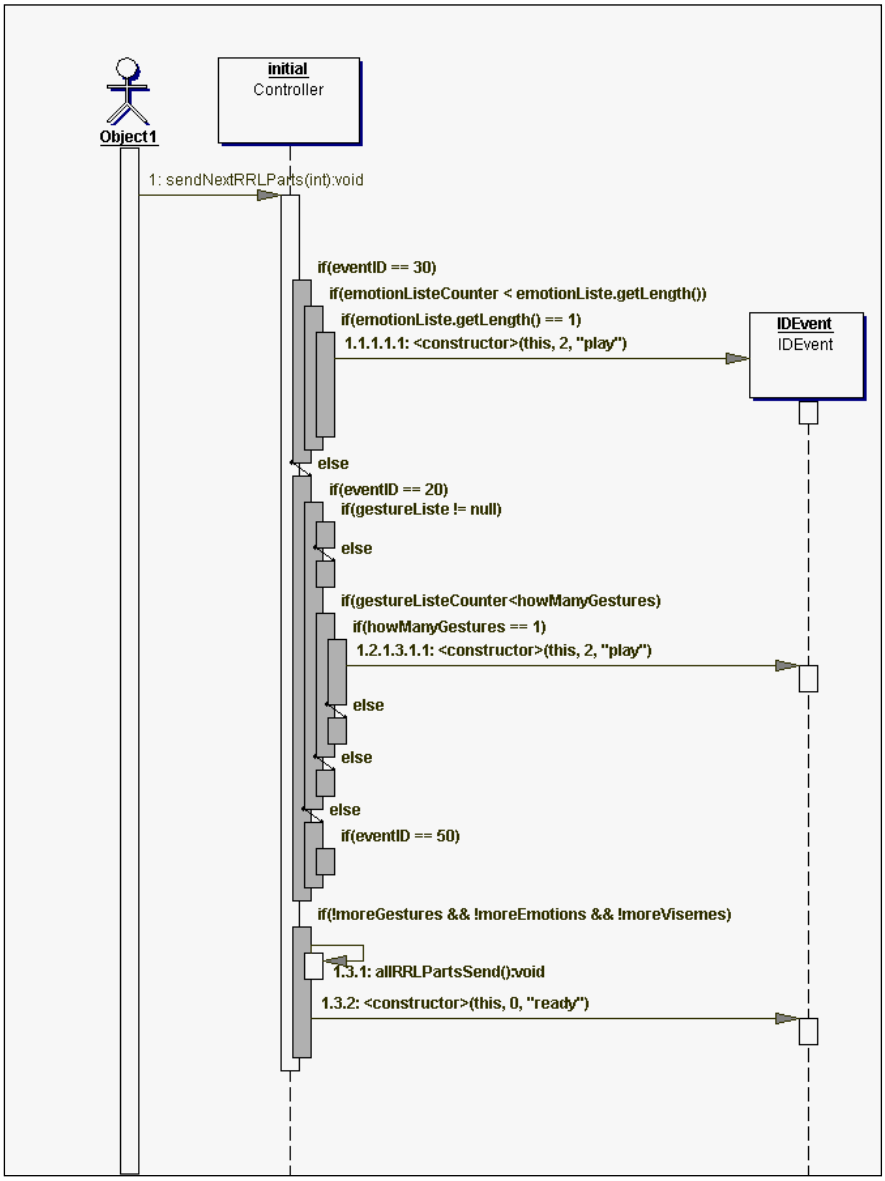


Abbildung 5.22: Ablauf innerhalb der Methode sendNextRRLParts()

Zur Verkürzung der Verarbeitungszeit wäre es möglich, die Aufgaben der Klasse Controller auf die Klasse ManifestPerson zu übertragen. Dadurch würden bei allen virtuellen Charakteren ein Schritt beim Versenden von Events und RRL sowie das Erzeugen eines neuen Threads wegfallen.

5.5.3.2 Die Modulatoren

Innerhalb der Plattform wurden fünf Modulatoren realisiert, der VisemeModulator zur Berechnung der Gewichtungen für den Morphknoten (Kapitel 5.6.2.1), der EmotionModulator zur Berechnung der Körperhaltung und Mimik, der SlerpModulator zur Berechnung von Gesten, der NoiseModulator zum Erzeugen von Augenbewegungen, und der PositionModulator, um virtuelle Charaktere platzieren zu können. Alle Modulatoren implementieren das ModulatorInterface (Abbildung 5.23).

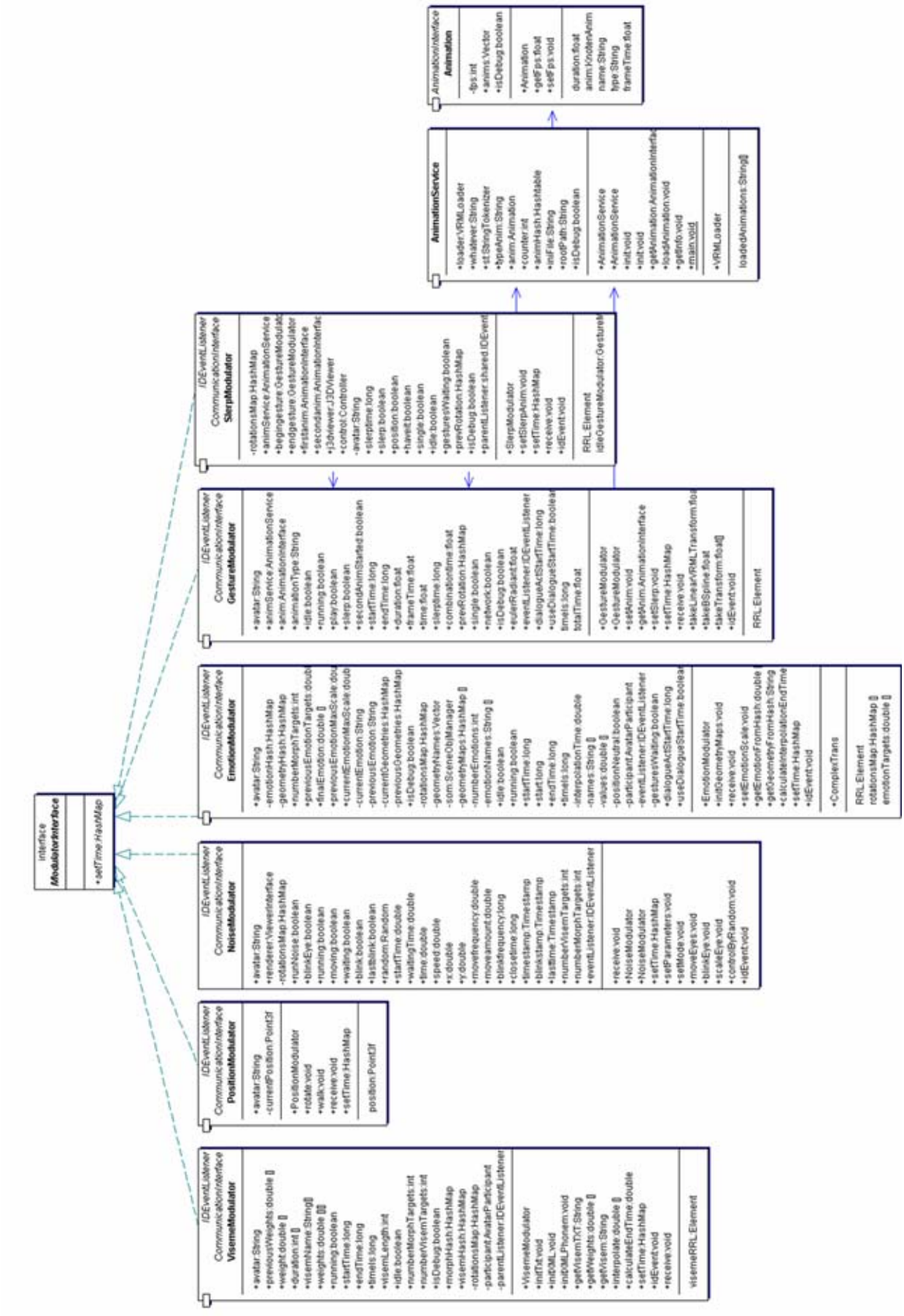


Abbildung 5.23: Das Package characterPlatform.modular

In der Klasse ModulatorLoop wird eine Endlosschleife ausgeführt, welche in regelmäßigen Intervallen über das ModulatorInterface, die Methoden setTime() aller Modulatoren aufruft und die aktuelle Systemzeit in Millisekunden als Parameter übergibt. Die Endlosschleife wird hierbei in der run()-Methode eines Threads ausgeführt (Abbildung 5.24). Rückgabeparameter der Methode ist eine HashMap, mit Rotationswerten in Form von Quaternionen, oder Gewichtungswerten in Form eines double-Arrays. Der einzige Modulator der Translationswerte erzeugt, ist hierbei der PositionModulator, der die Position des HumanoidRoot-Knotens eines 3D-Modells verändert (Abbildung 4.1). Die Werte werden in der HashMap anhand des Knotennamens gespeichert, dessen Transformationswerte verändert werden sollen.

Die Methode calculateWeights() fügt die Gewichte zur Animation des Morphknotens zusammen. Jedes Gewicht hat dabei einen Wertebereich von 0 bis 1 und steht für einen Gesichtsausdruck, der in Form einer Geometrie in den Morphknoten geladen wird. Dabei kann es sich um ein Lächeln, offene oder geschlossene Augen oder eine angehobene Augenbraue handeln. Setzen zwei Modulatoren Werte für das gleiche Gewicht, wird ein Mittelwert gebildet.

Die Methode multiplyModulMaps() fasst alle Quaternione zusammen, die von den Modulatoren für einen Knotennamen erzeugt wurden. Der dazu verwendete Algorithmus wurde von einer Mitarbeiterin der Abteilung Z5 des ZGDV entwickelt und implementiert.

Am Ende des Schleifendurchlaufs der ModulatorLoop existierten nur noch eine HashMap, die keine doppelten Einträge für Knotennamen enthält, und ein double-Array mit Gewichtungswerten. Diese werden über die Methode setValue() an die Darstellungsebene übertragen, wo sie dem 3D-Modell zugewiesen werden.

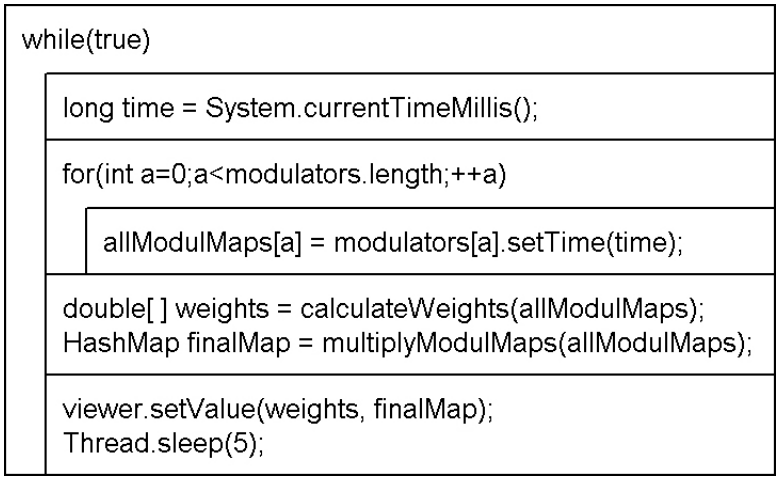


Abbildung 5.24: Methode run() der Klasse ModulatorLoop

Die Berechnung von Animationswerten innerhalb der setTime()-Methoden der Modulatoren, sowie das Laden der dafür benötigten Animationsformate wie VRML-Posen, VRML-Animationen und das Erzeugen von Augenbewegungen wurden von Mitarbeitern der Abteilung Z5 des ZGDV implementiert. Im Rahmen der Diplomarbeit wurden die Auswertung des RRL in den Modulatoren und die zeitliche Steuerung innerhalb der setTime()-Methode, sowie die Klasse SlerpModulator zum Erstellen von Gesten realisiert.

5.5.3.3 SlerpModulator

Die Klasse SlerpModulator erzeugt zwei Instanzen der Klasse GestureModulator, um zwei Gesten parallel berechnen zu können. Die Klasse GestureModulator wurde zum Erzeugen von Rotationswerten entworfen. Innerhalb der Methode receive() werden der Name und die Zeiten für Dauer und Anfang der Geste ausgelesen und auf Variablen gespeichert. Handelt es sich bei dem übertragenen RRL um einen par-Tag, wird die im Folgenden beschriebene Verarbeitung für beide Instanzen der GestureModulatoren ausgeführt. Die gespeicherte Anfangszeit gibt an, wie viele Millisekunden nach Beginn des dialogueActs eine Geste ausgeführt werden soll. Der Beginn eines dialogueActs wird der Klasse SlerpModulator durch die EventID 5 mitgeteilt, wodurch gleichzeitig die Verarbeitung gestartet wird (Quellcode 5.5).

```
public void idEvent(IDEvent event){
        if(event.id == 2) {
            gesturesWaiting = false;
```

```
            idle = false;
        }
        if(event.id == 5) {
            gesturesWaiting = true;
            idle = false;
        }
        if(event.id == 6) {
            gesturesWaiting = false;
            begingesture.useDialogueStartTime = false;
            endgesture.useDialogueStartTime = false;
        }
    }
```

Quellcode 5.5: Die Methode idEvent der Klasse SlerpModulator

In der Methode setTime() der GestureModulatoren wird abhängig von der booleschen Variablen gesturesWaiting die aktuelle Systemzeit auf der Variablen dialogueActStartTime gespeichert. Diese wird zur Berechnung der Startzeit (Variable startTime) aller Animationen eines dialogueActs verwendet (Abbildung 5.25).

Zum Laden der Animationen wird die Klasse AnimationService verwendet. Eine Instanz dieser Klasse wird in der Klasse ManifestPerson erzeugt und an den Konstruktor der Klasse SlerpModulator übergeben. Die Pfade zu den Animationsdateien werden in der Datei animationService.ini gespeichert und während der Initialisierung der Klasse ausgelesen. Für jede Animation wird ein Objekt der Klasse VRMLAnimation erzeugt. In der Klasse VRMLAnimation werden die Keyframe-Animationen für jeden animierten Knoten des 3D-Modells als Objekte der Klasse VRMLNodeAnimation zusammen mit dem entsprechenden Knotennamen in einer HashMap gespeichert. Die Umsetzung der Klasse AnimationService war nicht Bestandteil der Diplomarbeit.

Da es sich bei den Animationen um Keyframe-Animationen handelt, wird der aktuelle Frame und nicht die Zeit in Millisekunden berechnet. Zur Umrechnung wird die Dauer eines Frames berechnet (Formel 5.1).

$$frameTime = \frac{Animationsdauer}{Anzahl\ Keyframes}$$

Formel 5.1: Berechnung der Dauer eines Frames

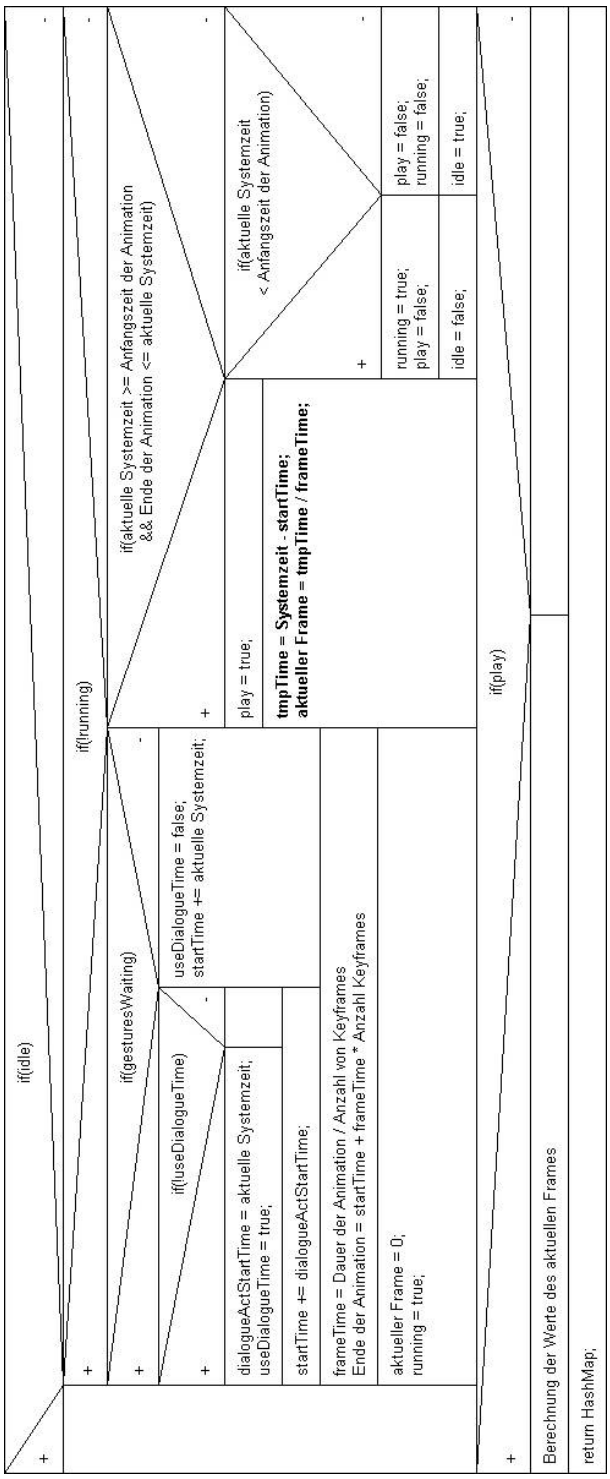


Abbildung 5.25: Die Methode setTime() der Klasse GestureModulator

Anhand der Dauer eines Frames wird die aktuelle Position innerhalb der Animation berechnet (Formel 5.2). Diese Umrechnung anhand der Dauer eines Frames ist nötig, da die Dauer einer Animation im RRLEditor verändert werden kann.

$$aktuellerFrame = \frac{Systemzeit - startTime}{frameTime}$$

Formel 5.2: Berechnung des aktuellen Frames

Die gespeicherten VRML-Animationen sind Keyframe-Animationen und beschreiben Rotationen anhand von vier float-Werten, die durch Leerzeichen voneinander getrennt sind. Die ersten drei Werte beschreiben die Rotationsachse anhand eines Vektors und haben einen Wertebereich von -1 bis 1. Der vierte Wert beschreibt den Winkel der Rotation im Bogenmaß [12]. Dieses Format der Rotationsbeschreibung ist in Java3D im Package javax.vecmath in der Klasse AxisAngle4f bereits umgesetzt. Der Unterschied zu Quaternionen, die ebenfalls einen Vektor zur Beschreibung der Rotationsachse und einen Skalar zur Abbildung des Winkels verwenden ist, das Quaternione der Einschränkung $x^2+y^2+z^2+w^2=1$ unterliegen, wodurch Rotationswerte dem Parameter w nicht direkt zugewiesen werden können.

Zur Berechnung des Rotationswertes des aktuellen Frames werden die beiden AxisAngle4f-Objekte, zwischen denen der aktuelle Frame liegt, interpoliert. Da die Klasse AxisAngle4f keine Methode zur Interpolation zur Verfügung stellt, werden zwei Quaternione erzeugt. Quaternione werden in Java3D im Package javax.vecmath in der Klasse Quat4f abgebildet und stellen die Methode set(AxisAngle4f) zur Verfügung, womit sich die Rotationswerte des AxisAngle4f-Objektes dem Quaternion zuweisen lassen. Die erzeugten Quat4f-Objekte werden interpoliert, wozu die Methode interpolate(Quat4f q1, Quat4f q2, float alpha) der Klasse Quat4f verwendet wird. Der Parameter alpha stellt einen Wert zwischen null und eins dar, und gibt an, in welchem Verhältnis die Quaternione interpoliert werden. Eine Interpolation mit dem Wert 0.5 würde als Ergebnis ein Quaternion erzeugen, dass den Mittelwert zwischen beiden Quaternionen darstellt.

Für jeden animierten Knoten der in der verwendeten Animation gespeichert ist, wird ein Quaternion erzeugt und anhand des Knotennamens in einer HashMap gespeichert. Diese wird als Rückgabeparameter der Methode setTime() an die Klasse ModulatorLoop übergeben wird.

Werden zwei Animationen parallel ausgeführt, erzeugt jede Instanz der Klasse GestureModulator ein Quaternion. Die Klasse SlerpModulator interpoliert diese beiden Quaternione in der Zeit delta t (Abbildung 5.26). In dieser Zeit wird der Parameter alpha der Methode interpolate() schrittweise von null auf eins gesetzt, um einen Übergang zwischen der ersten und der zweiten Animation zu erzeugen.

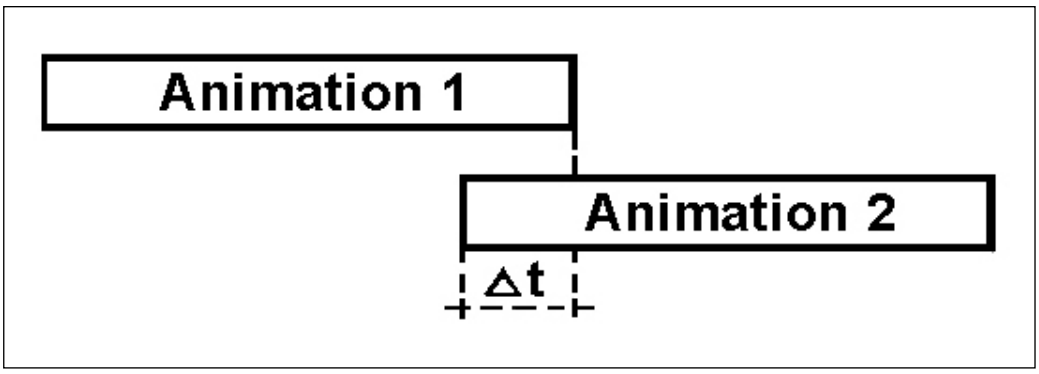


Abbildung 5.26: Überlappung zweier Animationen

5.6 Darstellung der Animationsplattform

5.6.1 Einleitung

In diesem Kapitel wird die Umsetzung der Darstellungsebene in Java3D und die Anbindung der Renderumgebung Avalon beschrieben (Abbildung 5.27). Dabei wird für beide Darstellungsplattformen das Einbinden der VRML-Modelle, die Umsetzung der Mimik durch Morphknoten und das Zuweisen von Animationswerten an virtuelle Charaktere beschrieben.

5.6.2 Java3D

Die Darstellung der Szene mit Java3D wurde in der Klasse J3DViewer umgesetzt. Darin wird ein Szenengraph unter Verwendung eines SimpleUniverse-Objektes erzeugt. Zum Laden der VRML-Modelle der Charaktere wird die Methode loadModel() beim Starten der Plattform aufgerufen. Die Methode load() der Klasse VrmlLoader aus dem Package com.sun.j3d.loaders lädt dabei eine VRML-Datei und liefert ein J3D-Scene-Interface zurück. Das Scene-Interface bietet Methoden um auf die geladene Knotenhierarchie zu-

zugreifen. Die Methode getSceneGroup() gibt den Rootknoten des VRML-Modells zurück, der in den Szenengraphen des J3DViewer eingebunden wird.

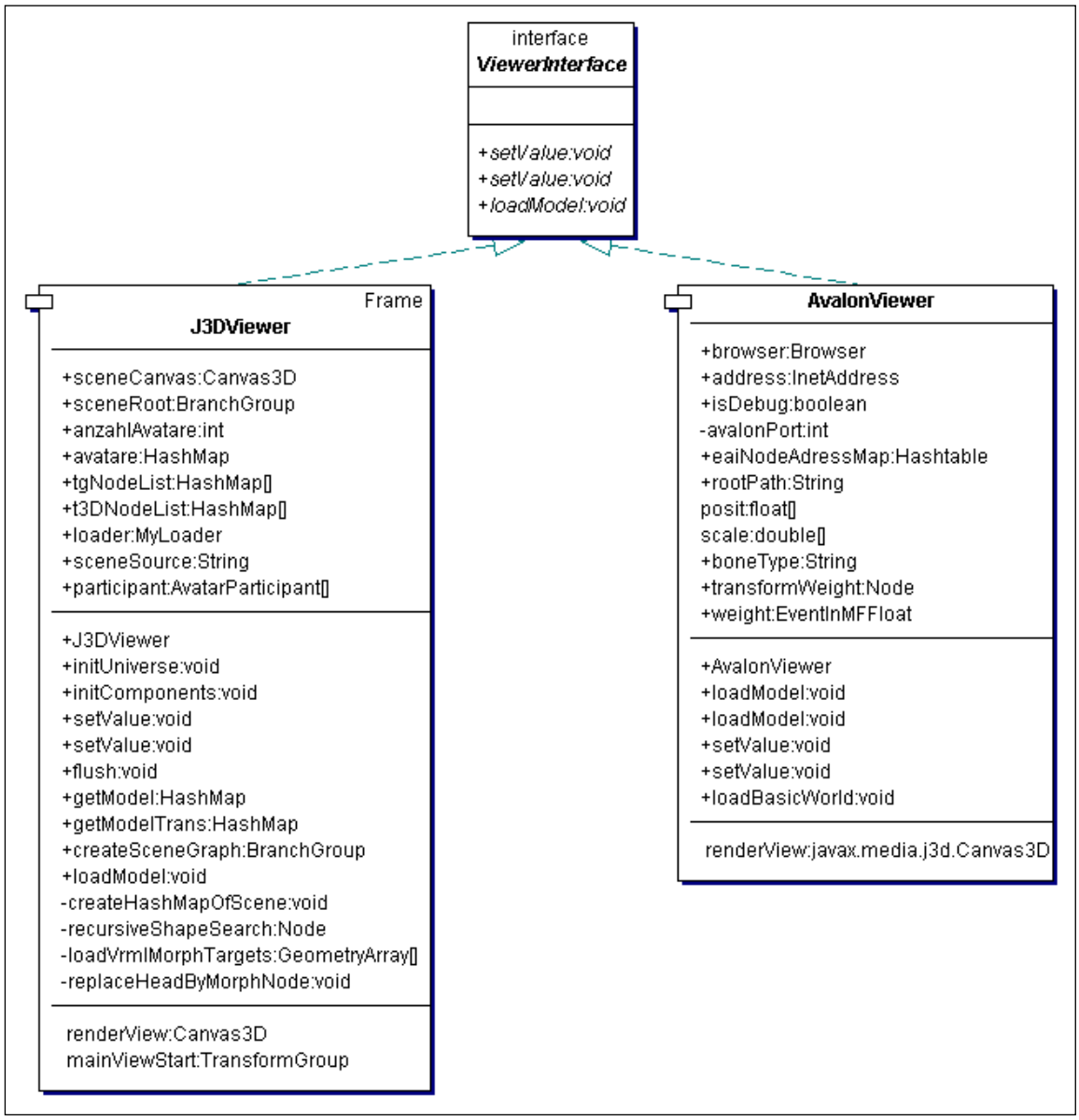


Abbildung 5.27: Klassendiagramm des Package exhibitionPlatform.viewer

5.6.2.1 Funktionsweise eines Morphknotens in Java3D

Zur Realisierung von Mimik und Lippenbewegungen im Morphknoten werden mehrere Geometrien eines Kopfes als VRML-Dateien benötigt. Insgesamt werden für jeden virtuellen Charakter 28 so genannte Morphtargets gespeichert, wobei acht Geometrien zur Darstellung von Lippenbewegungen verwendet werden. Die restlichen 20 Geometrien stellen Gesichtsausdrücke dar, wie z. B. Lächeln, offene oder geschlossene Augen, Angst und Wut (Abbildung 5.28). Zur Veränderung des Gesichtsausdrucks eines virtuellen Charakters wird die im VRML-Modell vorhandene Geometrie des Kopfes durch ein Objekt der Klasse Morph aus dem Package javax.media.j3d ersetzt. Die einzelnen Morphtargets werden mit der Methode loadVrmlMorphTargets() geladen und im Morph-Objekt in einem GeometrieArray gespeichert. Die Methode setWeight(double[ ] weights) des Morphknotens ermöglicht eine Gewichtung der verschiedenen Geometrien. Jeder Wert im Array weights entspricht dabei einer Geometrie im GeometrieArray. Alle Gewichte müssen in der Summe eins ergeben. Dazu wird die Summe aller Gewichte von eins subtrahiert und dem ersten Wert im Array zugewiesen. Dieser wird immer mit dem neutralen Gesichtsausdruck belegt.

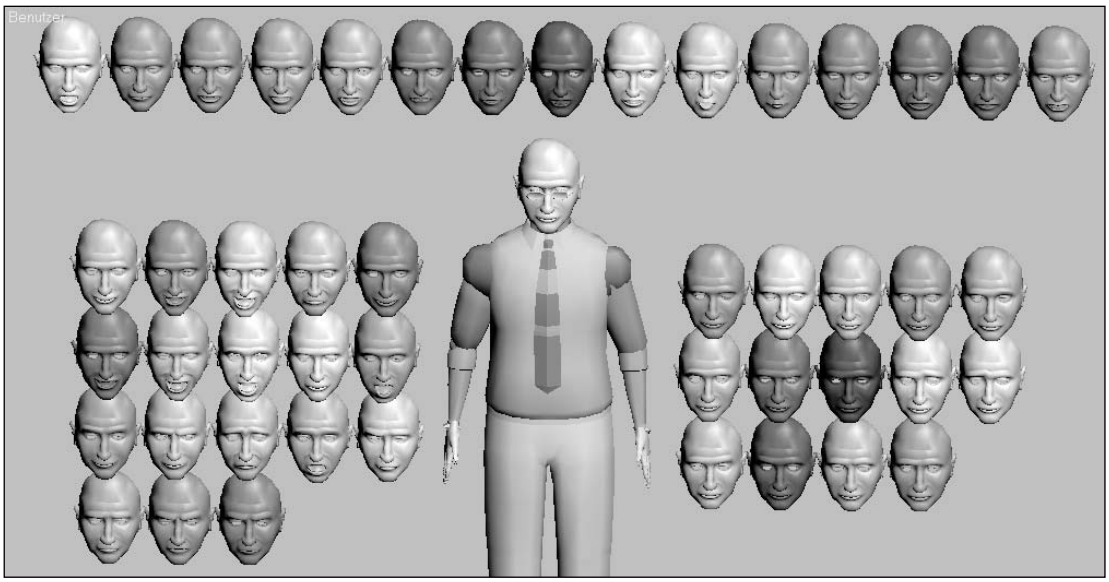


Abbildung 5.28: So genannte Morphtargets zur Darstellung der Mimik

5.6.2.2 Adressierung der Knoten

Um den Knoten des VRML-Modells Werte zuweisen zu können, wird für jeden virtuellen Charakter in der Methode createHashMapOfScene() die HashMap tgNodeList erstellt. In dieser HashMap werden alle Knotennamen und eine Referenz auf den Knoten des

VRML-Modells gespeichert (Tabelle 5.7). Das Scene-Interface stellt dazu die Methode getNamedObjects() zur Verfügung.

Knotenname	Referenz auf Objekt
prof_r_shoulder	TransformGroup
prof_l_shoulder	TransformGroup
prof_morphnode	Morph
…	…

Tabelle 5.7: HashMap zum Speichern der Knotennamen

Jeder Knoten wird durch die Java3D-Klasse TransformGroup dargestellt. Eine TransformGroup beinhaltet die Transformationswerte eines Knotens in Form einer Transformations-Matrix, die in Java3D durch die Klasse Transform3D dargestellt wird (Abbildung 5.29). Die Methoden setTransform() und getTransform() werden von der Klasse TransformGroup bereitgestellt um den Knoten eine Transformations-Matrix zuzuweisen oder diese auszulesen (Abbildung 5.29). Um einen möglichst schnellen Zugriff auf die Matrix eines Knotens zu erhalten, wird für jeden virtuellen Charakter eine zweite HashMap angelegt (t3DNodeList), in welche Kopien der Transform3D-Objekte der TransformGroups anhand der Knotennamen gespeichert werden. Dadurch entfällt das Auslesen der zu verändernden Matrix zur Laufzeit (Abbildung 5.30).

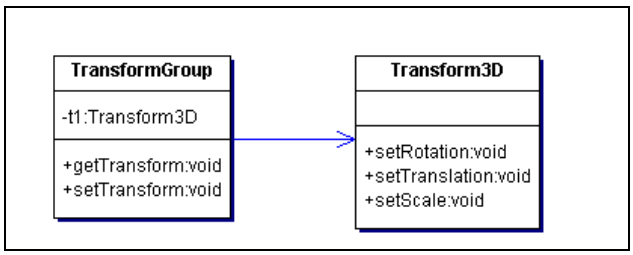


Abbildung 5.29: Vereinfachte Darstellung der Klasse TransformGroup

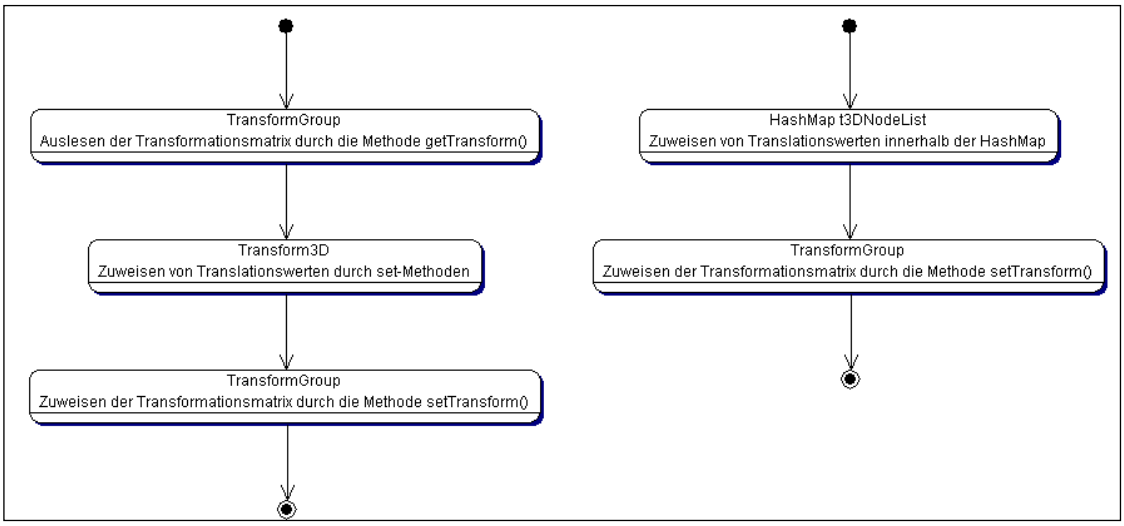


Abbildung 5.30: Veränderung der Transformationsmatrix in einer TransformGroup (links); Verkürzter Ablauf durch Verwendung einer HashMap zur Speicherung der Transformationsmatrizen (rechts)

Durch das ViewerInterface implementiert die Klasse J3DViewer die Methoden setValue(String avatar, String nodeName, Quat4f tempRot, float[] tempTr, double[] tempSca) und setValue(double[] weights). Diese werden dazu verwendet, Transformationswerte in den TransformGroups und die Gewichtung der Morphknoten zu verändern.

5.6.3 Avalon

Die Darstellungsplattform Avalon ermöglicht es im Gegensatz zu herkömmlichen VRML-Viewern, VRML-Dateien darzustellen, die einen Morphknoten verwenden.

Avalon stellt eine Implementierung des External Authoring Interface (EAI) zur Verfügung, das eine Verbindung zwischen Java und VRML-Viewern darstellt [11]. Ein VRML-Viewer wird darin als Browser-Objekt umgesetzt. Die Verbindung zwischen dem VRML-Viewer und Java wird über eine TCP/IP-Verbindung realisiert.

Beim Ausführen von Avalon wird eine VRML-Datei geladen, die nur einen Knoten mit dem Namen ROOT besitzt. Dieser Knoten wird benötigt, um über die EAI-Methode createVrmlFromURL() eine VRML-Datei einfügen zu können (Abbildung 5.31).

```
public void loadModel(String modelPath) {
    //Pfadangabe der zu ladenden VRML-Datei
    String[] url = {rootPath + modelPath};

    //Referenzieren des ROOT-Knotens des Browsers
    Node root = browser.getNode("ROOT");

    //Hinzufügen einer VRML-Datei zum ROOT-Knoten
    browser.createVrmlFromURL(url, root, "addChildren");
}
```

Abbildung 5.31: Methode zum Laden von VRML-Modellen

5.6.3.1 Funktionsweise eines Morphknotens in Avalon

Die Verwendung eines Morphknotens in Avalon wird durch das Erweitern der VRML-Datei eines 3D-Modells um den Knoten AvatarFaceMorpher ermöglicht. In diesen Knoten werden alle vorhandenen Morphtargets eines virtuellen Charakters kopiert.

Damit dem Knoten Werte zugewiesen werden können, wird über das EAI der EventIn des Knotens referenziert. Dieser EventIn stellt die Methode setWeights(float[ ] weights) zur Verfügung. Das double-Array mit den Gewichtungswerten für den Morphknoten muss deshalb vor dem Übertragen an das EventIn in ein float-Array konvertiert werden.

```
public Node transformWeight = null;
public EventInMFFloat weight = null;

public void setValue(String avatar, double[] weights) {
    if(transformWeight==null)
    transformWeight = browser.getNode(avatar+"_avatar_face_morpher");
    if(weights != null) {
        if(weight == null)
    weight=(EventInMFFloat)transformWeight.getEventIn("set_weights");
        float[] floatWeights=new float[weights.length];
        for(int k=0;k<weights.length;k++)
            floatWeights[k]=(float)(weights[k]);
        weight.setValue(floatWeights);
    }
}
```

Abbildung 5.32: Zuweisen von Gewichtungen an den Morphknoten des Avalon-Viewers

5.6.3.2 Zuweisen von Animationswerten

Das Zuweisen von Transformationswerten wird über EventIn-Referenzen der Knoten realisiert. Von jedem Knoten wird dabei eine EventIn-Referenz für Rotation, Translation und Skalierung verwendet. Nach den ersten Tests hat sich gezeigt, dass die Abfrage der EventIn-Referenzen zur Laufzeit zuviel Zeit in Anspruch nimmt. Die Lösung dieses Problems stellt die Speicherung der EventIn-Referenzen in einer HashMap dar, welche einen schnelleren Zugriff ermöglicht. Als Schlüsselwert wurde ebenfalls der Knotenname verwendet. Da es nicht möglich ist, in einer HashMap mehrere Einträge unter dem gleichen Schlüsselwert zu speichern, wurden drei HashMaps für Rotation, Translation und Skalierung verwendet.

Zur Übertragung von Rotationswerten an den EventIn eines Knotens ist in VRML ein float-Array vorgesehen. Die Werte stimmen hierbei mit dem Datenformat AxisAngle4f überein. Über die Methode set(Quat4f q1) werden dem Objekt der Klasse AxisAngle4f die Rotationswerte eines Quaternions zugewiesen. Anschließend kann das benötigte float-Array über die Methode get(float[ ] rotation) ausgegeben werden.

Die in Avalon dargestellten Animationen erwiesen sich mit nur 10 Frames/sec als zu langsam, was auf ein Problem bei der EAI-Implementierung in Avalon zurückzuführen war. Dieses Problem konnte im Rahmen dieser Diplomarbeit nicht behoben werden.

5.7 Sprachsynthese

5.7.1 Einleitung

In diesem Kapitel wird die Realisierung der Sprachausgabe durch so genannte Text-to-speech-Programme (TTS) beschrieben. Im Rahmen der Diplomarbeit wurde eine bereits bestehende Java-Implementierung der TTS-Software Mbrola verwendet, die an die Anforderungen der Plattform angepasst wurde. Zusätzlich wurde eine Anbindung für die TTS-Software AT&T Natural Voices™ vorgesehen, die jedoch im Rahmen dieser Arbeit noch nicht eingesetzt werden konnte [18]. Für beide Programme musste eine einheitliche Schnittstelle definiert werden, um sie innerhalb der Animationsplattform einsetzen zu können. Die Klassenstruktur der Sprachsynthese wurde im Package characterPlatform.speech realisiert.

5.7.2 Funktionsweise der TTS-Software Mbrola

Mbrola ist eine frei erhältliche Sprachsynthese, stellt jedoch keine direkte Umsetzung einer Text-to-speech-Software dar, weil als Eingabeformat Phoneme verwendet werden [14]. Mbrola verwendet Sprachdatenbanken, um verschiedenen Landessprachen und weibliche und männliche Stimmen umzusetzen. Die von Mbrola verwendeten Phoneme stellen eine textbasierte Beschreibung einzelner Laute dar. Zum Erzeugen von Phonemen werden Hilfsprogramme benötigt, die dazu die Sprachdatenbanken von Mbrola verwenden. Zum Erzeugen von deutschen Phonemen wurde die Software txt2pho verwendet [13]. txt2pho liest den zu konvertierenden Text aus einer Textdatei ein und erzeugt eine neue Textdatei mit Phonemen. Zu jedem Phonem werden die Dauer und so genannte Pitch-Angaben gespeichert, die eine Veränderung der Tonhöhe eines Phonems angeben. Zusätzlich zu txt2pho wurde die Software Emofilt eingesetzt, um emotionale Veränderungen an der erzeugten Phonemedatei vorzunehmen. Die dabei verwendeten Parameter für die Emotionen Glücklich, Traurig, Wütend und Neutral waren von der Software vorgegeben. Mit der Software Mbrola werden aus den erzeugten Phonemedateien Sprachsamples generiert, die in einer Wave-Datei gespeichert werden. Die Verbindung der Programme txt2pho und Mbrola stellt eine TTS-Software dar.

5.7.3 Funktionsweise der TTS-Software AT&T Natural Voices™

Die TTS-Software AT&T Natural Voices™ erzeugt synthetische Sprache direkt aus einem eingegebenen Text. Dabei kann Sprache in einer Wave-Datei gespeichert, oder direkt gesprochen werden. Die Qualität der erzeugten Sprache ist sehr hochwertig, basiert jedoch nicht auf der Verwendung von Phonemen. Eine emotionale Veränderung der Sprache ist für diese Sprachsynthese nicht vorgesehen. Die TTS-Software ist nicht frei erhältlich. Jede Stimme in einer bestimmten Landessprache muss zusätzlich erworben werden. Der Umfang der Sprachdatenbanken unterscheidet sich erheblich von den Datenbanken für Mbrola. Die in Mbrola verwendeten Datenbanken für eine Stimme oder Landessprache haben einen Umfang von ca. 15 MB, die für die TTS-Software AT&T Natural Voices™ verwendeten Datenbanken benötigen ca. 500 MB an Speicherplatz.

5.7.4 Anbindung an die Plattform

Zur Anbindung der beiden TTS-Programme an die Plattform wurde das Interface Java-Speech-API (JSAPI) der Firma Sun-Microsystems® verwendet [16]. Die TTS-Software AT&T Natural Voices™ implementiert diese Schnittstelle bis auf die Methode phoneme(). Diese Methode ist vorgesehen, um aus einem Text Phoneme zu erstellen. Die Dauer eines Satzes wird bei der Verwendung der Software Mbrola über Phoneme berechnet, was bei

der Verwendung der Software von AT&T nicht möglich ist. Die Software von AT&T erzeugt jedoch während ein Text gesprochen wird eine Viseme-Notation, die eine Zuordnung der Dauer zu jedem Visem enthält. Zur Umsetzung einer einheitlichen Verarbeitung wurde zur Berechnung der Dauer eines Wortes die Methode getWordDuration() zur JSAPI hinzugefügt und als JSAPIXtendedInterface von den Klassen MbrolaSynthesizer und ATTSynthesizer der beiden Sprachsynthesen implementiert.

Die Sprachsynthese Mbrola sowie die Hilfsprogramme txt2pho und emofilt werden über ein Process-Objekt aus dem Package java.lang mit den Pfadangaben zu den Text- bzw. Phonemedateien und der verwendeten Sprachdatenbank gestartet. Zum Abspielen der von Mbrola erzeugten Wave-Dateien wurde ein Objekt der Klasse AudioClip aus dem Package java.applet verwendet. In der Klasse MbrolaSynthesizer wurde das Abspielen der AudioClip-Objekte in der Methode speak() umgesetzt, wobei die URL zu der jeweiligen Audiodatei angegeben wird.

Zum Sprechen von Texten wird in der Klasse ATTSynthesizer die Methode speakPlainText() verwendet, wobei der Text der gesprochen werden soll als Stringparameter an die Sprachsynthese übertragen wird.

Die Methoden speak() und speakPlainText() werden über das JSAPI implementiert.

Zur Anbindung an die Plattform muss für die TTS-Software AT&T Natural Voices™ das Java-Native-Interface (JNI) verwendet werden, um auf die Methoden der JSAPI innerhalb der TTS-Software zugreifen zu können. Die Umsetzung des JNI und dessen Einbindung in die Klasse ATTSynthesizer war nicht Bestandteil der vorliegenden Arbeit.

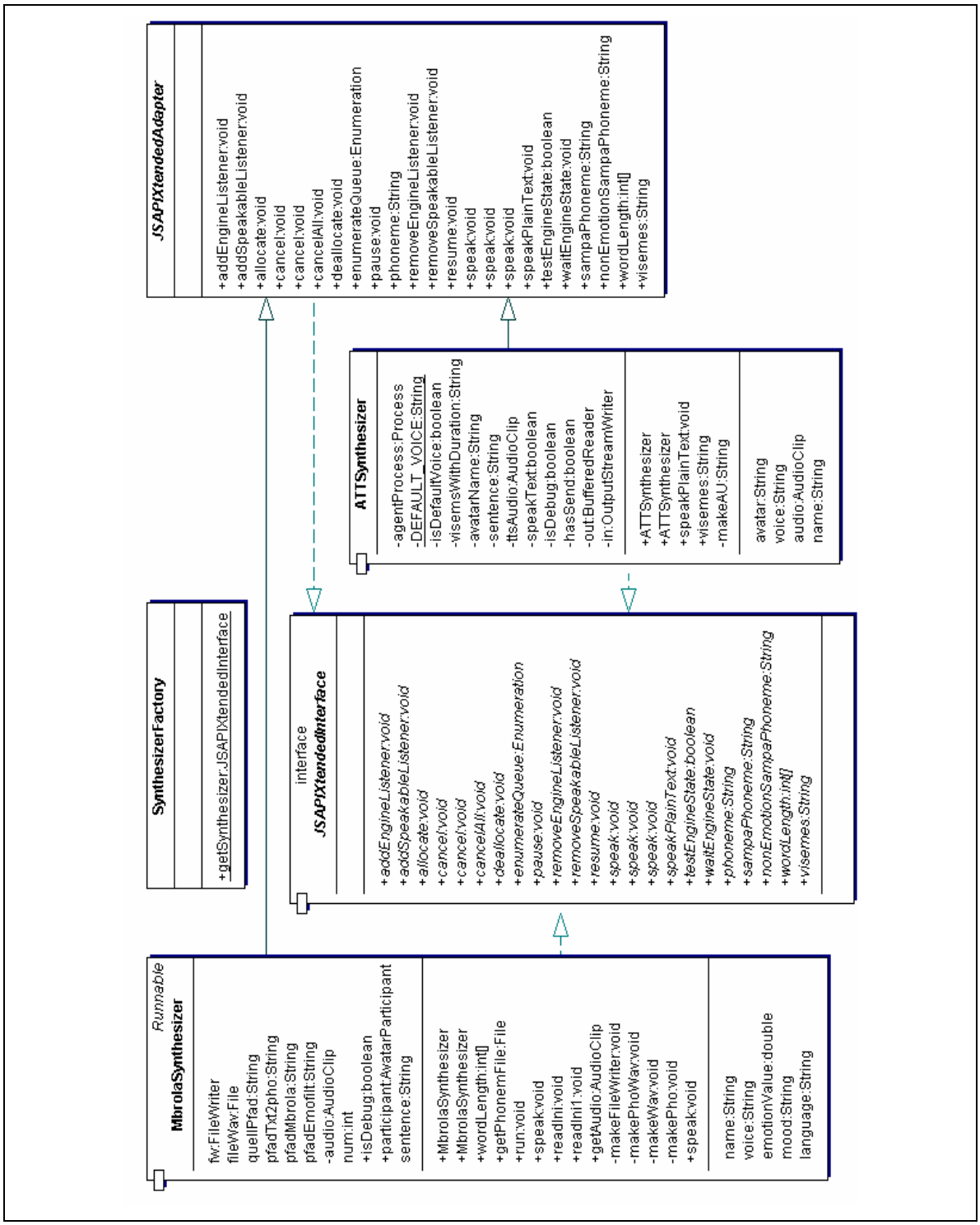


Abbildung 5.33: Klassenstruktur des package characterPlatform.speech

5.8 Konvertieren von XML-Dokumenten

Innerhalb der Animationsplattform werden in vielen Klassen XML-Dokumente in Form von XML-Objekten und Text verwendet. Beispielsweise wird zum Versenden von RRL von der Klasse RRLEditor zur Klasse Player ein XML-Dokument in Form eines String übertragen. Zur weiteren Verwendung in der Klasse Player wird jedoch ein XML-document-Objekt benötigt. Zur Konvertierung des XML in die benötigte Form wurde die Klasse XMLConverter erstellt (Abbildung 5.34). Im Folgenden werden die einzelnen Möglichkeiten der Konvertierung beschrieben.

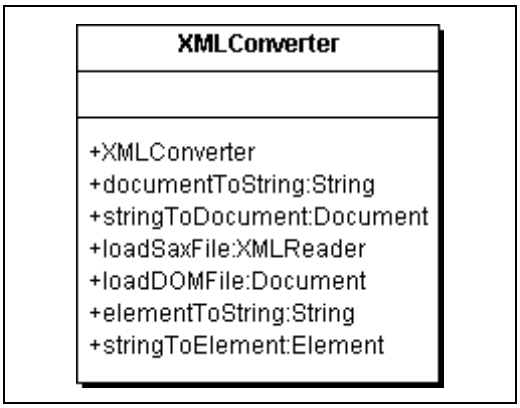


Abbildung 5.34: Der Klasse XMLConverter aus dem Package shared

Zum Laden einer XML-Datei wird in der Methode loadDOMFile() ein DOM-Parser verwendet. Ein DOM-Parser wird durch ein Objekt der Klasse DocumentBuilder aus dem Apache-Xerces-Package umgesetzt. Die Methode parse() der Klasse DocumentBuilder erstellt aus einem Eingabe-Stream ein Document-Object. Der Eingabe-Stream wird durch eine Instanz der Klasse InputSource erzeugt. Als Quelle wird der Klasse InputSource eine Referenz in Form eines Reader-Interface übergeben. Das Reader-Interface wird von den Klassen StringReader und FileReader implementiert, die im Java Package java.io enthalten sind. Zum Laden einer Datei wird dem FileReader die URL der Datei im Konstruktor übergeben (Quellcode 5.6). Zur Konvertierung eines XML-String in ein document-Objekt, wird in der Methode stringToDocument() das gleiche Verfahren angewendet, wobei der FileReader durch eine Instanz der Klasse StringReader ersetzt wird. Der zu konvertierende XML-String wird dem StringReader im Konstruktor übergeben.

```
Document xmlDocument = null;
try {
  //url – Pfad zur XML-Datei
  Reader reader = new FileReader(url);
  //Erzeugen eines Eingabe-Stream
  InputSource parserSource = new InputSource(reader);
  //Erzeugen eines Parsers
  DocumentBuilderFactory factory =
  DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = factory.newDocumentBuilder();
  //Erzeugen des Document-Objektes
  xmlDocument = builder.parse(parserSource);
}
catch (FactoryConfigurationError e) {
  System.out.println(e);// unable to get a document builder factory
}
```

Quellcode 5.6: Konvertieren einer XML-Datei in ein document-Objekt

Zum Konvertieren eines document-Objekts in einen String wurde in der Methode documentToString() ein Beispiel von der Apache-Xerces-Homepage verwendet. Zur Konvertierung wird das document-Objekt in einer Instanz der Klasse XMLSerializer serialisiert und in ein Objekt der Klasse StringWriter gespeichert. Über die Methode toString() der Klasse StringWriter wird das Stringobjekt erzeugt.

Zur Konvertierung einzelner XML-Elemente in einen String oder ein element-Objekt wurde ein XML-Dokument mit einem XML-Header und einem Wurzelknoten in einem String gespeichert (Abbildung 5.35). In der Methode stringToElement() wird der String des Elements unter Verwendung von Stringmethoden in den Wurzelknoten kopiert. Das String-Dokument wird anschließend an die Methode stringToDocument() übergeben. Aus dem dabei entstehenden document-Objekt wird mit der Methode getDocumentElement() der Wurzelknoten des document-Objektes zurückgegeben. Das im Wurzelknoten gespeicherte element-Objekt wird mit der Methode des Wurzelknotens getFirstChild() ausgelesen und von der Methode stringToElement() zurückgegeben.

Die Methode elementToString() erzeugt aus einem element-Objekt ein String-Objekt mit der Struktur des element-Objektes. Element-Objekte lassen sich jedoch nur durch das document-Objekt verändern, mit dem sie erzeugt wurden. Aus diesem Grund wird mit der

Methode cloneNode() des element-Objektes ein Duplikat erzeugt. Dieses Duplikat wird als Wurzelknoten in ein document-Objekt eingefügt, welches an die Methode documentToString() übergeben wird. Aus dem von der Methode documentToString() zurückgegebenen String wird der XML-Header entfernt, wonach das String-Objekt von der Methode elementToString() als Ergebnis zurückgegeben wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
</root>
```

Abbildung 5.35: String einer XML-Datei mit XML-Header und Wurzelknoten

5.9 Zusammenfassung

In diesem Kapitel wurden zunächst das RRL-Format zur Speicherung von Handlungen eines virtuellen Charakters und die Erzeugung des RRL-Formats anhand des textbasierten RRL-Editors beschrieben. Das RRL wird über eine Kommunikationsstruktur innerhalb der Animationsplattform verschickt. Die Kommunikation wird durch die Methoden receive() und idEvent() umgesetzt, wobei die Methode receive() zum Übertragen des RRL vorgesehen ist und über die Methode idEvent() Events zur Steuerung der plattforminternen Verarbeitung versendet werden.

Die Klasse Player verteilt die im RRL gespeicherten Handlungen an die virtuellen Charaktere und überwacht das zeitlich synchrone Ausführen der Handlungen. Jeder virtuelle Charakter wird durch zwei Klassen repräsentiert. Die Klasse AgentivePerson erzeugt Zufallsgesten, die den virtuellen Charakter in Bewegung halten, wenn keine vorgegebenen Handlungen ausgeführt werden sollen. Die Klasse ManifestPerson beinhaltet so genannte Modulatoren zur Berechnung von Animationen. Das Auswerten des RRL und das Verteilen der Animationen an die Modulatoren und die Koordination von Lippenbewegungen und synthetischer Sprachausgabe werden durch eine Instanz der Klasse Controller innerhalb der Klasse ManifestPerson vorgenommen. Die berechneten Animationswerte werden über ein Interface an die Darstellungsebene übertragen und dort dem virtuellen Charakter zugewiesen. Die Darstellung wurde in Java3D umgesetzt, wobei die Möglichkeit zur Verwendung der Renderumgebung Avalon vorgesehen wurde.

Das Unterkapitel 5.7 beschreibt die Verwendung der synthetischen Sprachausgabe anhand der Programme txt2pho und Mbrola, sowie die Schnittstellendefinition zur Einbindung der TTS-Software der Firma AT&T.

Das innerhalb der Animationsplattform häufig verwendete Konvertieren von XML-Objekten in String-Objekte sowie das Einlesen von XML-Dateien wurde in der Klasse XMLConverter verwendet.

Im nächsten Kapitel werden theoretische Ansätze für eine alternative Kommunikationsstruktur innerhalb der Animationsplattform und für ein Beschreibungsformat für Animationen vorgestellt.

6 Theoretische Ansätze

6.1 Einleitung

In diesem Kapitel werden zwei theoretische Ansätze beschrieben, deren Implementierung in die Animationsplattform im Rahmen der Diplomarbeit noch nicht sinnvoll war. Der erste Ansatz stellt eine alternative zur bisher implementierten Kommunikationsstruktur dar. Der zweite Ansatz beschäftigt sich mit einem Beschreibungsformat für Gesten, das den automatisierten Einsatz von Animationen innerhalb der Animationsplattform ermöglichen soll.

6.2 Kommunikation innerhalb der Animationsplattform

6.2.1 Einleitung

In diesem Kapitel wird ein alternativer Ansatz, der in Kapitel 5.3 beschriebenen Kommunikation innerhalb der Plattform vorgestellt. Dieser Ansatz soll es ermöglichen Events und RRL in der auf XML basierenden Art-E-Fact-Markup-Language (aefML) zu Versenden, die in der Abteilung Z5 des ZGDV entwickelt wurde. Bisher wurden Events und RRL über zwei verschiedene Interfaces versendet, indem die Daten in der Klassenhierarchie weitergegeben wurden. Der Einsatz der aefML soll durch die Verwendung einer Adressierung eine zielgerichtete Kommunikation zwischen virtuellen Charakteren ermöglichen, wodurch Reaktionen beispielsweise auf eine Geste oder die "Blickrichtung" eines anderen Charakters möglich werden.

6.2.2 Anforderungen

Die im Folgenden beschriebenen Anforderungen ergeben sich durch die Verwendung der aefML und der dadurch möglichen zielgerichteten Verarbeitung von Events.

Zur plattforminternen Kommunikation soll ein Event-Modell verwendet werden, welches in der Klassenhierarchie richtungsunabhängig eingesetzt werden kann. Klassen auf der gleichen Hierarchieebene, beispielsweise die Klassen AgentivePerson zweier virtueller Charaktere, sollen ebenfalls miteinander kommunizieren können (waagerechte Kommunikation).

Das Versenden eines Events darf die sendende Klasse nicht blockieren, d. h. nachdem der Adressat das Event erhalten hat, ist in der Senderklasse der Empfang neuer Events,

oder die Verarbeitung von Daten möglich. Diese blockierungsfreie Verarbeitung muss für jede Klasse zwischen Nachrichtenquelle und -senke gewährleistet sein.

Die Struktur der Events ist durch die aefML vorgegeben. Das Eventmodell muss die XML-Struktur der aefML auslesen und zum Erzeugen neuer Events erstellen können.

6.2.3 Lösungsansatz und Softwareentwurf

Die Verwendung der Beschreibungssprache aefML, welche sowohl Statusmeldungen sowie RRL-Dokumente enthalten kann, macht den Einsatz der bisher verwendeten Interfaces EventListener zum Versenden von Events und CommunicationInterface zum Versand von RRL überflüssig. Zum Versenden der aefML wird nur noch die receive()-Methode des CommunicationInterface verwendet, wobei das XML als String übertragen wird, was den Einsatz der Animationsplattform auf verteilten Systemen vereinfacht.

Das Hauptaugenmerk liegt bei der Umsetzung des Event-Modells auf der blockierungsfreien Verarbeitung. Jede Klasse muss die empfangenen Daten direkt verarbeiten, was jedoch nicht durch den Methodenaufruf des eingehenden Events geschehen darf. In diesem Fall müsste die aufrufende Klasse auf die Verarbeitung innerhalb der Empfängerklasse warten. Die Lösung dieses Problems ist die Verwendung von Threads. Jeder Thread führt nach dem Aufruf seiner start-Methode seine run()-Methode aus. Ist das Ende der run-Methode erreicht, "stirbt" der Thread. Die start-Methode eines Threads kann nur einmal ausgeführt werden, d. h. ist ein Thread gestorben, kann er nicht wieder reanimiert werden. Durch Verwendung einer while-Schleife kann die Verarbeitung innerhalb der run-Methode solange wiederholt werden, bis ein bestimmtes Ereignis eintritt.

Die Verwendung einer Endlosschleife innerhalb der Methode run() des Threads nennt man aktives Warten, was im allgemeinen jedoch vermieden werden sollte, da für den Fall, dass keine Veränderung im Programm auftritt, die gesamte Rechenzeit, die dieser Thread zur Verfügung hat, nur für die Ausführung der while-Schleife genutzt wird.

Im Folgenden werden zwei Möglichkeiten beschrieben, wie aktives Warten verhindert werden kann.

Eine Alternative zur Verarbeitung durch aktives Warten ist die Verwendung der Methoden wait() und notify() der Threadklasse. Die Methode wait() versetzt einen aktiven Thread in einen Wartezustand, welcher durch die Methode notify() wieder aufgehoben wird. Nur

die Klasse, die wait() für einen Thread aufgerufen hat, kann diesen Thread durch den Aufruf der Methode notify() aufwecken. Die Methode in der wait und notify aufgerufen werden, müssen gegen den gleichzeitigen Zugriff mehrerer Threads geschützt werden, was durch deklarieren der Methode als synchronized realisiert wird. Greifen zwei Threads auf eine synchronized-Methode zu, wartet ein Thread, bis die Verarbeitung der Methode beendet ist. Warten mehrere Threads auf eine Klasse, wird durch den Aufruf der Methode notify() irgendein Thread aus der Warteschlange geweckt. Es gibt keine Garantie, dass der Thread, der sich am längsten in der Warteschlange befindet, geweckt wird.

Eine Lösung zur Einhaltung der Reihenfolge innerhalb der Warteschlange ist die Methode notifyAll(). Hierbei werden alle Objekte in der Warteschlange geweckt. Welcher Thread mit der Verarbeitung fortfahren darf, muss durch eine zusätzliche Verwaltungsklasse geregelt werden.

Innerhalb der run-Methode des Threads kann nun ebenfalls eine Endlosschleife verwendet werden, da der reaktivierte Thread die Verarbeitung an der Stelle weiterführt, an der er durch die Methode wait() angehalten wurde.

Eine zweite Möglichkeit, aktives Warten zu Umgehen, ist das Erzeugen eines neuen Threads, sobald die Verarbeitung in einer Klasse gestartet werden soll. Der Vorteil hierbei ist, dass eine Klasse nur während der Verarbeitung aktiv ist.

Anhand eines Benchmark-Tests wurde ermittelt, welche Zeit der Durchlauf der Methode run() bei aktivem Warten und der Erzeugung eines neuen Threads benötigt.

Zu diesem Zweck wurde die Zeit für 100.000 Durchläufe für das jeweilige Verfahren gemessen. Aufgrund der Tatsache, dass unter Verwendung von aktivem Warten jeder Thread mindestens eine Millisekunde schlafen gelegt werden muss, um andere Prozesse nicht zu blockieren, ist die Durchlaufzeit in diesem Ansatz um den Faktor 5 langsamer als die Erzeugung neuer Threads.

Wegen des Verwaltungsaufwands, den eine Verwendung der Methoden wait() und notify() innerhalb der Animationsplattform bedeuten würde, wird die Erzeugung neuer Threads für das Event-Modell verwendet.

Zur Realisierung eines blockierungsfreien Versands von Events wird in der receive-Methode, die durch das CommunicationInterface implementiert wird, der String des Events in einem Vector gespeichert und ein neuer Thread gestartet. Zur Verarbeitung eines Events wird in einer while-Schleife das erste Element des Vectors in der run-Methode des Threads ausgelesen und ausgewertet. Nach dem Starten des Threads ist die Übertragung des Events abgeschlossen. Zur Verhinderung eines gleichzeitigen Aufrufs der Methode receive() durch zwei eintreffende Events, wird die Methode als synchronized deklariert. Wird während der Verarbeitung eines Events die Methode receive() aufgerufen, wir mit der Methode isActive() der Klasse Thread überprüft, ob der Thread aktiv ist. In diesem Fall wird das Event im Vector gespeichert, und es wird kein neuer Thread erzeugt. Ist die Verarbeitung des aktuellen Events in der run-Methode abgeschlossen, wird das erste Objekt im Vector gelöscht und abgefragt, ob sich weitere Objekte im Vector befinden.

Zur Auswertung und zum Erstellen von Events werden die Klassen EventService und EventFactory verwendet. Die Klasse EventService implementiert das Interface CommunicationInterface und Runnable und stellt die Methode receive() zur Verfügung, über die Events empfangen werden. Jede Klasse in der Kommunikationsstruktur der Animationsplattform erbt die Klasse EventService. Die zum Erstellen von Events verwendete Klasse EventFactory wird der Klasse EventService vererbt (Abbildung 6.1). Damit wird die gesamte Struktur zur Verarbeitung von Events von der Klasse EventService bereitgestellt. In der jeweiligen Klasse der Animationsplattform muss zur individuellen Verarbeitung der Events die Methode run() überschrieben werden. Zur direkten Adressierung von Klassen ist in der aefML ein Empfänger-Attribut vorgesehen. Zur Adressierung wird auf der Variablen className der Klasse EventService ein für die Plattform eindeutiger Name gespeichert. Beim Versenden von Events wird der Name im Attribut Sender der aefML gespeichert, um Antworten zu ermöglichen.

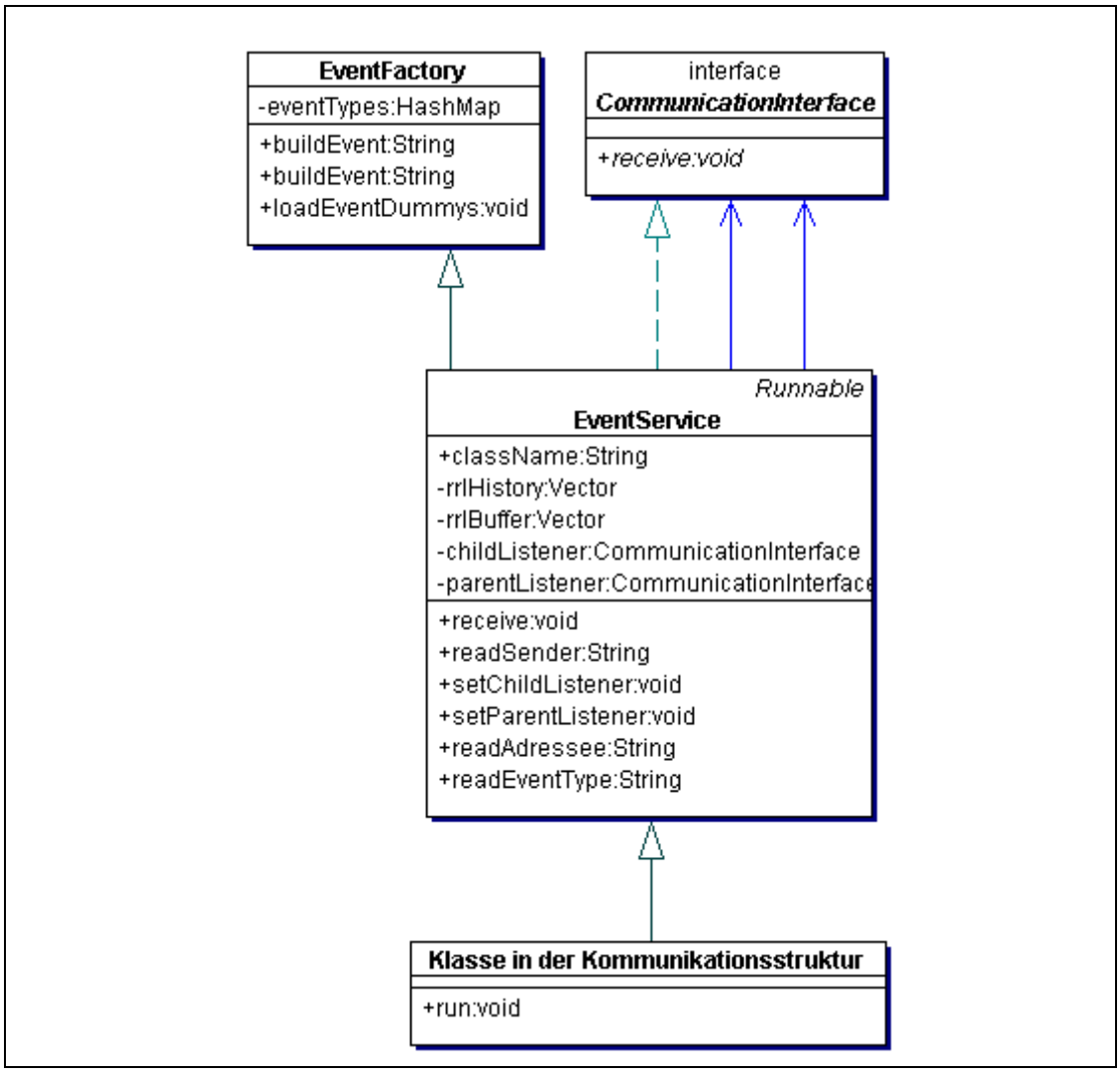


Abbildung 6.1: Mögliche Klassenstruktur zur Implementierung des Event-Modells

Zur Realisierung einer waagerechten Kommunikation wird in der Klasse Player eine Art Adressbuch erzeugt. Dazu werden alle in der Plattform verwendeten Namen der Klassen als Schlüsselwerte in einer HashMap gespeichert. Den Schlüsselwerten wird das CommunicationInterface der AgentivePerson-Klasse zugeordnet, die in der Hierarchie über der zu adressierenden Klasse steht. Soll beispielsweise ein Event von der Klasse Controller des Charakters A an den Controller des Charakters B versendet werden, leitet die Klasse Player das Event an die Klasse AgentivePerson des Charakters B weiter. Innerhalb der virtuellen Charaktere werden die Events an die nächst tiefere Ebene weitergegeben, bis in einer Klasse das Empfänger-Attribut des Events mit dem in der Klasse gespeicherten Namen übereinstimmt. Innerhalb der Klassenhierarchie wird das

einstimmt. Innerhalb der Klassenhierarchie wird das Event, abhängig von der Art des Events, nach oben oder unten weitergereicht.

Die Adressierung von Events könnte auch über eine Art Vermittlungsstelle umgesetzt werden, die alle Events empfängt und direkt an den Empfänger weiterleitet. Für diese Art der Umsetzung müsste das Erzeugen von Zufallsgesten in der Klasse AgentivePerson auf einem anderen Wege als über die Statusmeldungen der Klasse ManifestPerson gestartet werden. Eine Möglichkeit wäre ein zusätzliches Event zu verwenden.

Zum Erstellen von Events werden bei der Initialisierung der Klasse EventFactory XML-Dummy-Objekte für alle möglichen Formen von Events geladen und anhand des Eventnamens in einer HashMap gespeichert. Über set-Methoden werden der Klasse EventFactory die in den Dummy-Objekten fehlenden Parameter mitgeteilt und in das XML eingefügt. Das fertige Event wird in der Methode buildEvent() in ein Stringobjekt konvertiert und der aufrufenden Klasse übergeben.

Der in diesem Kapitel beschriebene Softwareentwurf soll den Einsatz der aefML zur zielgerichteten Kommunikation innerhalb der Animationsplattform ermöglichen. Durch die Verwendung von Threads wird ein blockierungsfreies Versenden von Nachrichten realisiert. Die Vererbung der Klasse EventService an die Klassen der Animationsplattform vereinfacht das individuelle Erzeugen und Versenden der aefML. Die Verwendung eines Adressbuches in der Klasse Player ermöglicht den zielgerichteten Versand von Events und RRL.

Im nächsten Kapitel wird eine Möglichkeit für die Umsetzung eines Beschreibungsformates für Animationen vorgestellt.

6.3 Beschreibungsformat für Gesten

6.3.1 Einleitung

Das Beschreibungsformat für Gesten Animation Description Language (animDL) soll die automatische Verarbeitung von Gesten innerhalb der Animationsplattform ermöglichen, sowie Kombinationen verschiedener Gesten und optisch verbesserte Übergänge zwischen einzelnen Gesten ermöglichen. Zur Speicherung der animDL wurde XML gewählt, da die

Verarbeitung von XML-Dokumenten bereits implementiert wurde. Im Folgenden werden verschiedene Eigenschaften der Gesten und die sich aus den Eigenschaften ergebende Struktur der animDL beschrieben. Weiterhin werden Ansätze zur plattforminternen Verarbeitung der animDL vorgestellt.

6.3.2 Lösungsansatz

Die animDL wird in XML realisiert. Für jede zu beschreibende Geste wird dem XML-Dokument ein Knoten mit dem Namen gesture hinzugefügt, welcher als Attribut den Dateinamen einer Geste enthält. Dem gesture-Knoten werden für die Eigenschaften einer Geste Knoten mit entsprechenden Attributen hinzugefügt. Zum Erzeugen und Abspeichern der im Folgenden aufgeführten Eigenschaften von Gesten, ist die Realisierung eines Editors, wie im Entwurf in Abbildung 6.9 dargestellt, nötig.

Zur Beschreibung der Aussage einer Geste werden einzelne Stichworte verwendet. Zur Eingabe der Stichworte wird ein Textfeld im Editor vorgesehen. Die eingegebenen Stichworte sollen zu einem späteren Zeitpunkt anstelle der Dateinamen im RRLEditor zur Auswahl einer Geste verwendet werden. Das Verwenden eines Stichwortes für mehrere Gesten ist dabei möglich, wobei das Stichwort nur einmal aufgeführt wird und die endgültige Auswahl der Geste von der Animationsplattform vorgenommen wird. Zum Speichern der Stichworte wird für jede Geste der Knoten "description" im XML-Dokument angelegt (Abbildung 6.2).

Gesten, die in der Klasse AgentivePerson als Zufallsgesten verwendet werden sollen, benötigen keine Beschreibung und können als solche durch ankreuzen der Checkbox "Noisegesture" gekennzeichnet werden. Soll eine Zufallsgeste zusätzlich im RRLEditor erscheinen, kann eine Beschreibung angegeben werden. In der animDL wird für Zufallsgesten der Knoten "noiseGesture" mit dem Attribut "noise" verwendet, dessen Werte true oder false sein können (Abbildung 6.2).

Die Skalierbarkeit von Gesten ist durch die Verwendung des RRL-Editors uneingeschränkt, was jedoch zu ungewollten Effekten führen kann, wenn eine Geste beispielsweise in der halben Zeit der Originaldauer abgespielt wird. Die Skalierbarkeit wird deshalb durch die Angabe eines minimalen und maximalen Skalierungswertes eingeschränkt. Zur Generierung dieser Werte muss im Editor ein Regler eingefügt werden, der eine Skalierung der Abspieldauer ermöglicht. Zur Speicherung der Werte für die Skalierbarkeit wird der animDL der Knoten "scaleability" hinzugefügt. Für Gesten, die uneingeschränkt skalierbar

sein sollen, wird das Attribut "restricted" mit den Werten "true" und "false" verwendet, das über eine Checkbox im Editor gesetzt werden kann (Abbildung 6.2).

```
<animDL>
  <gesture name="[dateiname]">
    <description>klatschen, applaudieren</description>
    <noiseGesture noise="[true/false]"/>
    <scaleability min="[float]" max="[float]"
      restricted="[true/false]"/>
  </gesture>
</animDL>
```

Abbildung 6.2: animDL mit einer Beschreibung der Geste, Angaben für Zufallsgesten und Skalierbarkeit

Die meisten Gesten die in der Animationsplattform verwendet werden, beginnen und enden in einer neutralen Ausgangsposition, in welcher der virtuelle Charakter aufrecht steht und die Arme seitlich neben dem Körper hängen lässt. Da Gesten wie beispielsweise Winken oder Zeigen meistens vor dem Körper ausgeführt werden, ist die erste Bewegung einer Geste das Anheben der Arme. Das Anheben zu Beginn und das Absenken der Arme am Ende einer Geste sind bei den meisten Gesten, bis auf wenige Abweichungen, ähnlich. Für das Erzeugen von nahtlosen Übergängen zwischen zwei Gesten durch Überblenden von einer Geste auf die folgende sind diese Bereiche gut geeignet, da die Bewegungen in diesen Bereichen auf den gleichen Achsen ablaufen (Abbildung 6.3). Für die Bezeichnung dieser Bereiche einer Geste wurden Onset und Offset aus dem Facial Action Coding System (FACS) übernommen, wo sie zur Beschreibung von Gesichtsanimationen verwendet werden. Onset beschreibt dabei den Übergang zwischen einem neutralen Gesichtsausdruck und dem Zeitpunkt, an dem die Hauptveränderung des Gesichtsausdrucks stattfindet. Der Bereich der maximalen Veränderung gegenüber dem neutralen Gesichtsausdruck wird in FACS mit Apex bezeichnet(Abbildung 6.4). Der Bereich nach dem Apex, in dem sich der Gesichtsausdruck wieder zu einem neutralen Gesicht verändert, wird mit Offset bezeichnet [1].

Zur Speicherung der Bereiche Onset und Offset werden in der animDL für jede Geste die Knoten "onset" und "offset" hinzugefügt. Für den Bereich Onset wird im Knoten ein Attribut vorgesehen, in dem das Ende des Onset als float-Wert gespeichert wird. Der Wert stellt den Frame der Geste am Ende des Onset dar. Der Beginn des Bereiches muss dabei

nicht gespeichert werden, da dieser mit dem Beginn der Geste übereinstimmt. Da der Bereich Offset an einem Punkt innerhalb der Geste beginnt und bis zum Ende der Geste dauert, wird für diesen Bereich ebenfalls ein Attribut vorgesehen, in dem der Beginn des Offset als float-Wert gespeichert wird.

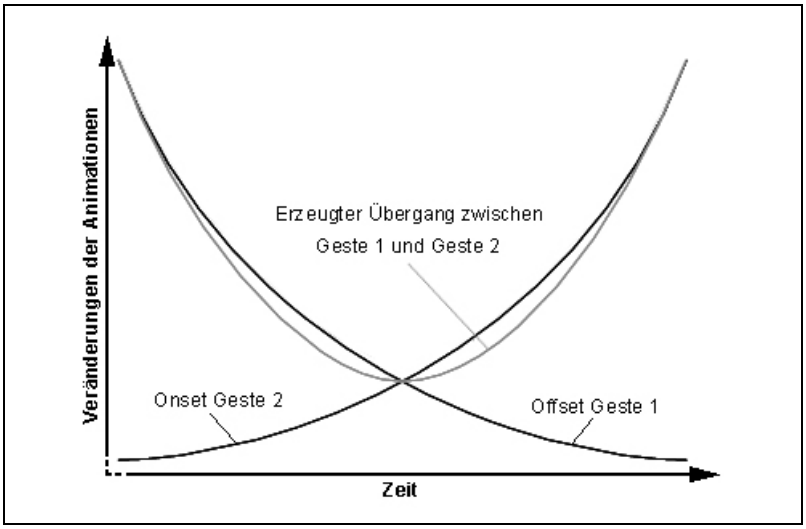


Abbildung 6.3: Überlappung zweier Gesten in den Bereichen Onset und Offset

Gesten wie beispielsweise das Deuten haben in ihrem Verlauf einen bestimmten Punkt in dem die Hauptaussage der Geste liegt. Für die Bezeichnung eines solchen Punktes wurde der Begriff Apex gewählt. Bei einer Verlängerung der Dauer einer solchen Geste wäre es sinnvoll, die Geste in der Originalgeschwindigkeit ablaufen zu lassen und sie für die zusätzliche Dauer im Apex anzuhalten. Durch zusätzliches Erzeugen von minimalen Bewegungen während des Anhaltens kann der Geste ein natürlicheres Aussehen gegeben werden. Zur Speicherung des Apex wird im XML-Dokument ein Knoten mit gleichem Namen erzeugt. Der Zeitpunkt des Apex wird als Attribut im Knoten gespeichert.

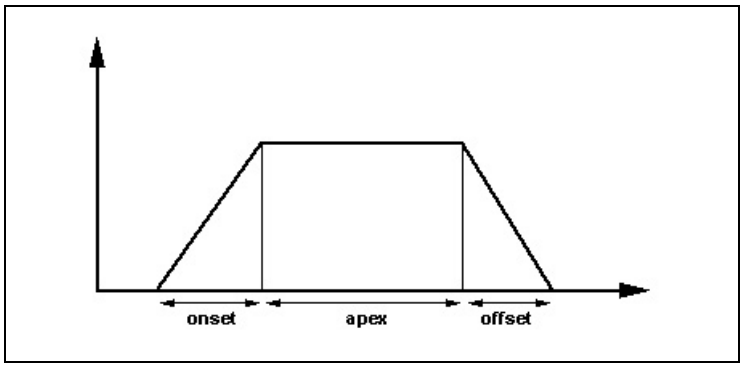


Abbildung 6.4: Darstellung der drei Phasen einer Animation

Bei Gesten wie z. B. dem Klatschen oder Winken werden bestimmte Bereiche der Gesten wiederholt. Werden diese Wiederholungen bereits beim Erzeugen der Gesten gespeichert, verändert sich durch eine Skalierung der Dauer lediglich die Geschwindigkeit der Geste. Wenn eine Bewegung für einen längeren Zeitraum wiederholt werden soll, wäre es sinnvoll, den zu wiederholenden Bereich nur einmal in der Geste zu speichern und innerhalb der Animationsplattform mehrmals abzuspielen. Zur Speicherung von Anfang und Ende eines zu wiederholenden Bereiches, wird dem XML der Knoten apexLoop hinzugefügt. Anfang und Ende des Bereiches werden als Attribute in dem Knoten gespeichert (Abbildung 6.5).

```
<animDL>
  <gesture name="[dateiname]">
    <description>klatschen, applaudieren, beifall</description>
    <noiseGesture noise="[true/false]"/>
    <scaleability min="[float]" max="[float]"
      restricted="[true/false]"/>
    <onset frame="[float]"/>
    <offset frame="[float]"/>
    <apex frame="[float]"/>
    <apexLoop begin="[float]" end="[float]"/>
  </gesture>
</animDL>
```

Abbildung 6.5: Struktur der animDL mit Angaben zu den Phasen einer Geste

Zum Erzeugen der Werte für Onset, Offset und Apex, sowie für den Anfangs- und Endwert der apexLoop, muss im Editor das Abspielen von Gesten ermöglicht werden. Das

Abspielen einer Geste sollte dabei anhand eines Schiebereglers umgesetzt werden, um das Auswerten der Gesten und Abspeichern von Werten an bestimmten Positionen innerhalb einer Geste zu ermöglichen (Abbildung 6.9).

In den meisten Gesten werden verschiedene Bereiche eines virtuellen Characters, wie z. B. beide Arme, der Oberkörper und der Kopf animiert, wobei die Hauptaussage der Gesten nur in einem Bereich liegt. Ein Beispiel dafür wäre eine verneinende Geste, wobei die Arme eine abwehrende Haltung signalisieren, was mit einem Kopfschütteln kombiniert wird. Soll diese Geste gleichzeitig mit einem Winken abgespielt werden, so wird eine einfache Kombination der beiden Gesten ein wenig ansprechendes Ergebnis darstellen. Bei einer solchen Kombination von verschiedenen Gesten kann es vorkommen, dass sich die Geometrien des virtuellen Charakters berühren oder durchdringen. Wünschenswert wäre bei der Kombination des Winkens mit der verneinenden Geste nur das Kopfschütteln mit dem Winken zu verbinden. Ein möglicher Ansatz zur Umsetzung dieser Kombination ist, den virtuellen Charakter in mehrere Bereiche zu unterteilen und diese Bereiche mit unterschiedlichen Gewichtungen zu versehen. Beim Abspielen der Gesten werden nur die Knoten aus dem Bereich mit der höheren Gewichtung in die Berechnung mit einbezogen. Die Wahrscheinlichkeit des Auftretens von gleichen Gewichtungswerten für einen Bereich wird durch die zusätzliche Verwendung eines Gewichtungswertes für jede Geste verringert, wodurch immer der Bereich der Geste mit der höheren Gesamtgewichtung verwendet wird. Bei gleichen Gesamtgewichtungswerten wird zufällig eine der beiden zu kombinierenden Gesten ausgewählt und komplett abgespielt.

Das Modell eines virtuellen Charakters wird in sechs Bereiche eingeteilt (Abbildung 6.7). Für jeden Bereich wird im Editor ein Textfeld vorgesehen in dem ein Gewichtungswert im Bereich von 0 – 1 vergeben werden kann (Abbildung 6.9). Zur Speicherung der Werte wird in der animDL der Knoten "dominance" hinzugefügt. Für jeden Bereich wird im Knoten dominance ein weiterer Knoten mit dem Namen des Bereichs erzeugt, dem ein Attribut weight hinzugefügt wird. Die Gesamtgewichtung der Geste wird im Attribut weight des dominace-Knotens gespeichert (Abbildung 6.6).

```
<animDL>
  <gesture name="[dateiname]">
    <dominance weight="[float]">
      <head weight="[float]"/>
      <torso weight="[float]"/>
```

```
      <left_arm weight="[float]"/>
      <right_arm weight="[float]"/>
      <left_leg weight="[float]"/>
      <right_leg weight="[float]"/>
    </dominance>
  </gesture>
</animDL>
```

Abbildung 6.6: animDL mit einem Knoten für Dominanzwerte

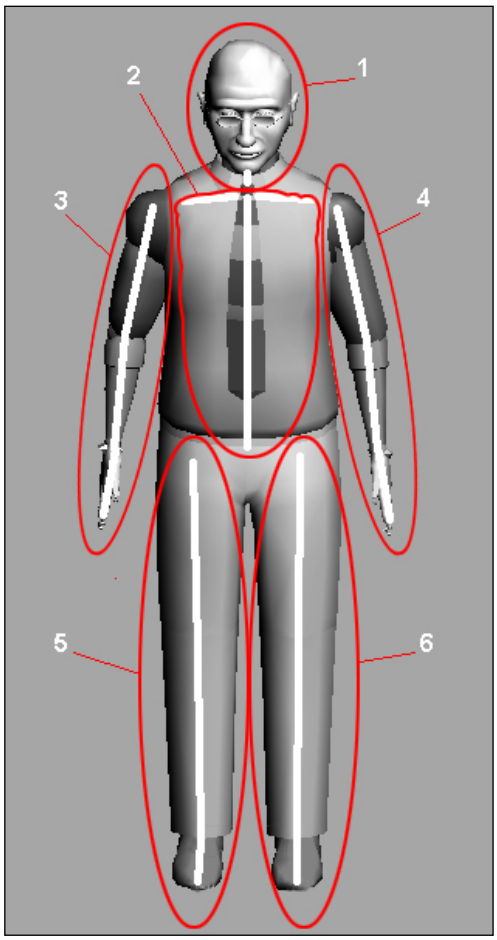


Abbildung 6.7: Einteilung eines 3D-Modells in 6 Bereiche. 1: Head, 2: Torso, 3: Right Arm, 4: Left Arm, 5: Right Leg, 6: Left Leg

Die Modellierung der Gesten in einer 3D-Software bietet die Möglichkeit, Gesten mit einer animierten Mimik in einer VRML-Datei zu speichern, was die Umsetzung von emotio-

nal vorbelegten Gesten erlaubt. Durch eine Veränderung der Animationsplattform, die nicht Bestandteil dieser Arbeit war, wurde das Abspielen der in den VRML-Dateien gespeicherten animierten Mimiken ermöglicht. Gesten, die mit Emotionen vorbelegt sind, dürfen durch Zuweisen von Emotionen im RRL-Editor nicht verändert werden. Aus diesem Grund werden die enthaltenen Emotionen in der animDL gespeichert, wozu der Knoten "emotion" verwendet wird. Über eine Checkbox im Editor wird dem Attribut des Emotionsknotens der Name der Emotion zugewiesen (Abbildung 6.8). Es kann jeweils nur eine Emotion ausgewählt werden. Enthält eine Geste eine Emotion, so wird der Name der Emotion im RRL-Editor in Klammern hinter dem Namen der Geste angezeigt.

Aufgrund der Nutzung der H-Anim-Spezifikation zur Normierung der Knotennamen eines humanoiden Skeletts können Gesten theoretisch auf alle Charaktere angewendet werden. Ausnahmen stellen lediglich Gesten dar, bei denen sich Teile der Geometrie eines virtuellen Charakters berühren, wenn z. B. ein virtueller Charakter die Hände an die Hüften legt oder die Arme verschränkt. Bei der Verwendung von Charakteren mit unterschiedlichen Geometrien, beispielsweise einem dünnen und einem dicken virtuellen Charakter, ist eine Geste wie das Verschränken der Arme auf den virtuellen Charakter beschränkt, mit dem die Geste in der 3D-Software erstellt wurde. Zum Testen, welche Geste mit welchen Charakteren abgespielt werden kann, werden im Editor alle verfügbaren virtuellen Charaktere geladen und können über eine Drop-Down-Liste ausgewählt werden. Standardmäßig werden alle Namen der zur Verfügung stehenden virtuellen Charaktere in der animDL im Knoten virtualCharacters gespeichert. Ist die aktuell ausgewählte Geste nicht mit dem aktuell angezeigten Modell abspielbar, kann der virtuelle Charakter anhand des Buttons removeCharacter aus der Liste in der animDL entfernt werden. Im RRL-Editor werden Gesten, die nur für bestimmte Charaktere geeignet sind, unter dem Namen des virtuellen Charakters aufgelistet. Wird eine solche Geste einem anderen virtuellen Charakter zugewiesen, wird diese in der Animationsplattform nicht abgespielt.

```
<animDL>
  <gesture name="[String]">
    <description>klatschen, applaudieren, beifall</description>
    <noiseGesture noise="[true/false]"/>
    <scaleability min="[float]" max="[float]"
      restricted="[true/false]"/>
    <onset frame="[float]"/>
    <offset frame="[float]"/>
    <apex frame="[float]"/>
```

```
    <apexLoop begin="[float]" end="[float]"/>
    <dominance weight="[float]">
      <head weight="[float]"/>
      <torso weight="[float]"/>
      <left_arm weight="[float]"/>
      <right_arm weight="[float]"/>
      <left_leg weight="[float]"/>
      <right_leg weight="[float]"/>
    </dominance>
    <virtualCharacters>name1, name2, name3, ... </virtualCharacters>
    <emotion type="[String]">
  </gesture>
</animDL>
```

Abbildung 6.8: Komplette Struktur der animDL für eine Geste

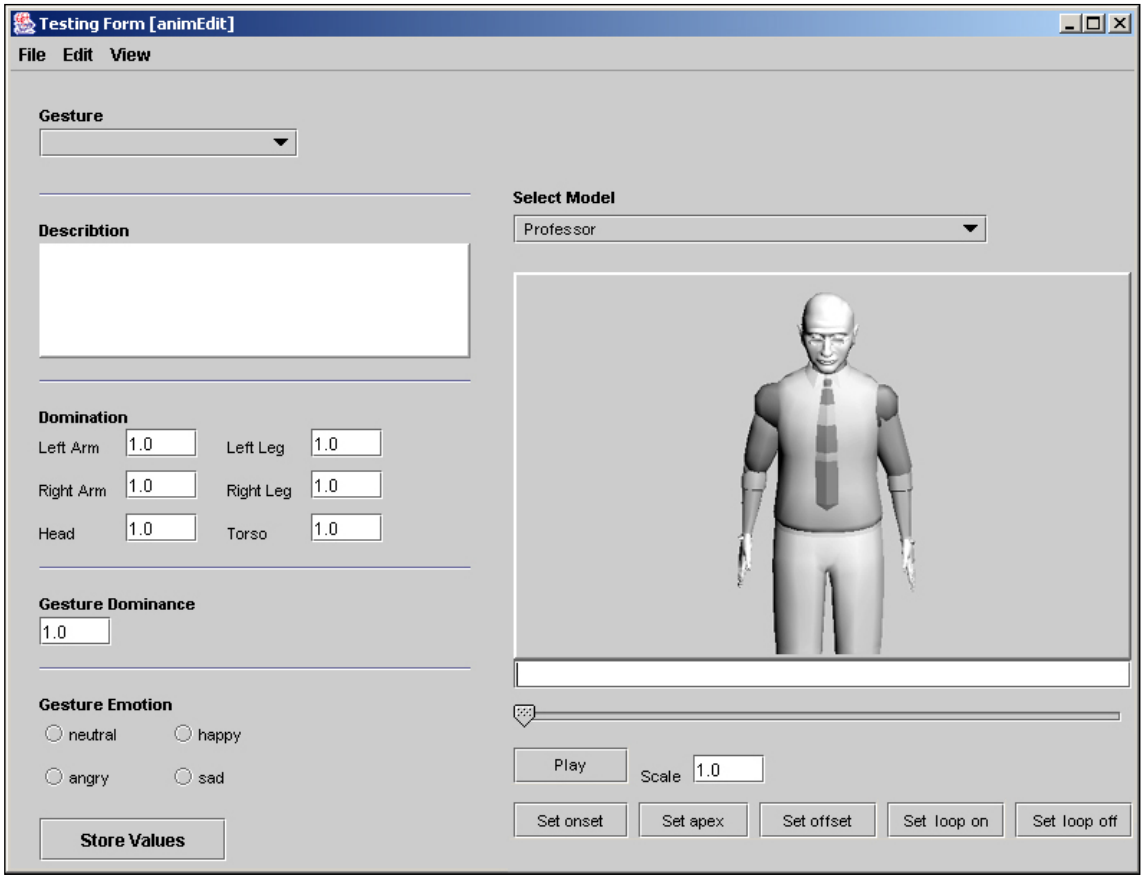


Abbildung 6.9: Entwurf eines Editors zum Erzeugen des Beschreibungsformates

7 Fazit

Die Aufgabenstellung der vorliegenden Diplomarbeit verlangte die Entwicklung einer Animationsplattform für virtuelle Charaktere, die Gestik, Mimik und eine synthetische Sprachausgabe beinhalten. Die Realisierung der Animationsplattform wurde in der vorliegenden Arbeit in der Reihenfolge der Verarbeitung beschrieben. Die dabei entstandenen Komponenten werden noch einmal zusammengefasst und es wird ein Überblick über mögliche Erweiterungen und Verwendungsmöglichkeiten gegeben.

Durch die mögliche Verwendung der Renderumgebung Avalon wurde die Auswahl der zur Abbildung von virtuellen Charakteren verwendbaren Formate auf VRML beschränkt.

Die in Java3D erstellte Renderumgebung ermöglicht das Laden und die Darstellung von VRML-Modellen. Durch die Verwendung von HashMaps zur Referenzierung der Knoten der VRML-Modelle anhand der Knotennamen der H-Anim-Spezifikation wurde ein schnelles System zur Zuweisung von Transformationswerten entwickelt.

Die Darstellung der Mimik wurde durch den Einsatz von Morphknoten ermöglicht, die anstelle der Geometrien der Köpfe der VRML-Modelle in den Java3D-Szenengraphen eingesetzt wurden. Morphknoten ermöglichen die Darstellung mehrerer Geometrien eines Objektes anhand unterschiedlicher Gewichtungen. Die Verwendung von Morphknoten bot sich an, da Avalon diese ebenfalls unterstützt. Die Renderumgebung Avalon wurde unter Verwendung des External Authoring Interface an die Animationsplattform angebunden. Der Einsatz von Avalon zur Darstellung der Animationsplattform war jedoch im Rahmen der Diplomarbeit aufgrund von Geschwindigkeitsproblemen noch nicht möglich. Die Klassen der Darstellungsebene wurden in die vorgegebene Klassenstruktur integriert.

Die virtuellen Charaktere werden in zwei Klassen abgebildet. Es wurde eine Klasse realisiert, welche alle "physikalischen Handlungen" eines virtuellen Charakters koordiniert. Alle Handlungen werden separat in Modulatoren berechnet und zeitlich synchron mit der Sprachausgabe ausgeführt. Der Modulator für Gesten setzt das Überblenden zwischen zwei Gesten um und wird durch eine Implementierung des theoretischen Ansatzes zur Beschreibung von Gesten einen automatisierten Einsatz der Gesten ermöglichen. In einer zweiten Klasse werden den virtuellen Charakteren Zufallsgesten zugewiesen, solange keine weiteren Handlungen vorliegen.

Für die synthetische Sprachausgabe wurde eine bereits bestehende Implementierung der Sprachsynthese Mbrola in Verbindung mit der Software txt2pho über die Definition des JSAPI an die Animationsplattform angebunden. Das JSAPI wurde erweitert, um die TTS-Software der Firma AT&T in die Plattform integrieren zu können.

Zur Festlegung von Handlungen einzelner virtueller Charaktere wurde ein Editor umgesetzt, der die Eingabe von Texten sowie das Zuordnen von Animationen zu markierten Textbereichen ermöglicht. Für die Zuordnung der Animationen wurde eine eigene Notation entwickelt, die eine Auswertung des zeitlichen Ablaufs der Handlungen ermöglicht. Zum Speichern der Handlungen und zur weiteren Verarbeitung innerhalb der Animationsplattform wurde die auf XML basierende Rich Representation Language (RRL) verwendet und zur Verwendung von Emotionen um entsprechende Tags erweitert. Bei der Konvertierung der Notation des Editors in RRL wird eine zeitliche Zuordnung von Lippenbewegungen, Gesten und Emotionen erzeugt und gespeichert. Bei der Verwendung von Mbrola zur Sprachausgabe werden Phoneme gespeichert. Für die Verwendung der Sprachsynthese AT&T wurde das Speichern des zu sprechenden Textes vorgesehen.

Die derzeitige Implementierung der Animationsplattform erlaubt die Darstellung mehrerer virtueller Personen und ein synchronisiertes Abspielen von Animationen mit gleichzeitiger Sprachausgabe (Abbildung 7.1). Durch die Manipulation des RRL ist auch das parallele Abspielen von Handlungen mehrerer virtueller Personen möglich. Die Verwendung einer Animation für alle virtuellen Charaktere wird durch die Verwendung der H-Anim-Spezifikation zur Benennung von Knoten möglich. Ein Problem stellen lediglich Animationen dar, bei denen sich ein virtueller Charakter berührt. Diese Gesten sind aufgrund unterschiedlicher Geometrien nicht auf andere Charaktere übertragbar. Aus diesem Grund wurde eine Zuordnungsmöglichkeit von Animationen zu Charakteren vorgesehen.

Abbildung 7.1: Parallele Animation zweier virtueller Charakter mit der in dieser Arbeit beschriebenen Animationsplattform.

Mit dem Beschreibungsformat für Gesten wurde ein Konzept vorgestellt, welches beim Abspielen von Gesten das Halten einer Pose und das Wiederholen eines Bereiches einer Geste, sowie nahtlose Übergänge zwischen zwei Gesten ermöglicht. Das Überblenden von einer Geste auf eine folgende wurde bereits implementiert, der Zeitraum der Überlappung ist jedoch abhängig von der Zuordnung der Gesten. Die Auswertung des Beschreibungsformates ermöglicht eine automatische Auswahl von Gesten, wobei die Kombination verschiedener Gesten anhand der vorgegebenen Gewichtungen ebenfalls möglich sein wird.

Das beschriebene Kommunikationsmodell ermöglicht das Erstellen und blockierungsfreie Versenden von Events und RRL unter Verwendung der aefML, sowie die Kommunikation von Klassen auf der gleichen Hierarchieebene. Durch die Verwendung von Strings zum Versenden von Nachrichten ist eine Übertragung der Events als ASCII-Text über ein Netzwerk möglich, was den Einsatz der Animationsplattform auf verteilten Systemen zulässt.

7 Fazit

Parallel zu dieser Diplomarbeit wurde ein Programm zum Erzeugen von interaktiven Geschichten erstellt, welches in Verbindung mit der Animationsplattform eingesetzt werden kann. Die einzelnen Bestandteile dieser Geschichten werden durch virtuelle Charaktere in der Animationsplattform wiedergegeben. Der Benutzer hat die Möglichkeit, über ein Textfeld Fragen zu stellen oder Antworten zu geben, was den Verlauf einer Geschichte verändert. Diese Art des Digital Storytelling bildet die Grundlage für den Einsatz der Animationsplattform im Projekt Art-E-Fact zur Vermittlung von Informationen. Durch diese Erweiterung ist es möglich, die Plattform als Informationsstand in Museen oder Firmen einzusetzen. Denkbar ist ebenfalls eine Verwendung der Plattform auf mobilen Geräten in Kombination mit einem Trackingsystem, um virtuelle Führer für Museen oder Städte zu verwirklichen, die eine kostengünstige und jederzeit verfügbare Alternative zu Personen darstellen.

8 Literaturverzeichnis

[1] BADLER, N., PHILLIPS, C., and WEBBER, B. "Simulating Humans: Computer Graphics Animation and Control", Oxford University Press, New York, NY, 1993

[2] PERLIN, K., "Real Time Responsive Animation with Personality", IEEE Transactions on Visualization and Computer Graphics, Vol. 1, No. 1, 1995

[3] GLEICHER, M., "Comparing Constraint-Based Motion Editing Methods", Graphical Models, Vol. 3, S. 107-134, 2001

[4] EKMAN, P., and FRIESEN, W. "Pictures of Facial Affect", Consulting Psychologists Press, Palo Alto, CA, USA, 1976

[5] EKMAN, P., and FRIESEN, W. "Facial Action Coding System", Consulting Psychologists Press, Inc., 1978.

[6] SHOEMAKE, K., "Animating rotation with quaternion curves", ACM SIGGRAPH Computer Graphics, v.19 n.3, p.245-254, Jul. 1985.

[7] WELLING, J., "Computer Graphics, Gimbal Lock" http://www-2.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15462/web.01s/notes/animation/sld026.htm, 02.04.2004

[8] Web3D Consortium, Humanoid Animation Working Group, www.h-anim.org, 02.04.2004

[9] PERLIN, K., "Violet", http://mrl.nyu.edu/~perlin/experiments/violet/, 31.03.2004

[10] AVALON, "An open X3D/VRML-Enviroment for Virtual and Augmented Reality Applications", http://www.zgdv.de/avalon/, 06.04.2004

[11] Web3D Consortium, "VRML97 - The Virtual Reality Modelling Language", http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All, 06.04.2004

[12] Web3D Consortium, "VRML97, Field and event reference, 5.8 SFRotation", http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/part1/fieldsRef.html#SFRotation, 06.04.2004

[13] Institut für Kommunikationsforschung und Phonetik, "IKP-Forschung: txt2pho", http://www.ikp.uni-bonn.de/dt/forsch/phonetik/hadifix/HADIFIXforMBROLA.html, 06.04.2004

[14] The Mbrola Project, http://tcts.fpms.ac.be/synthesis/mbrola.html, 06.04.2004

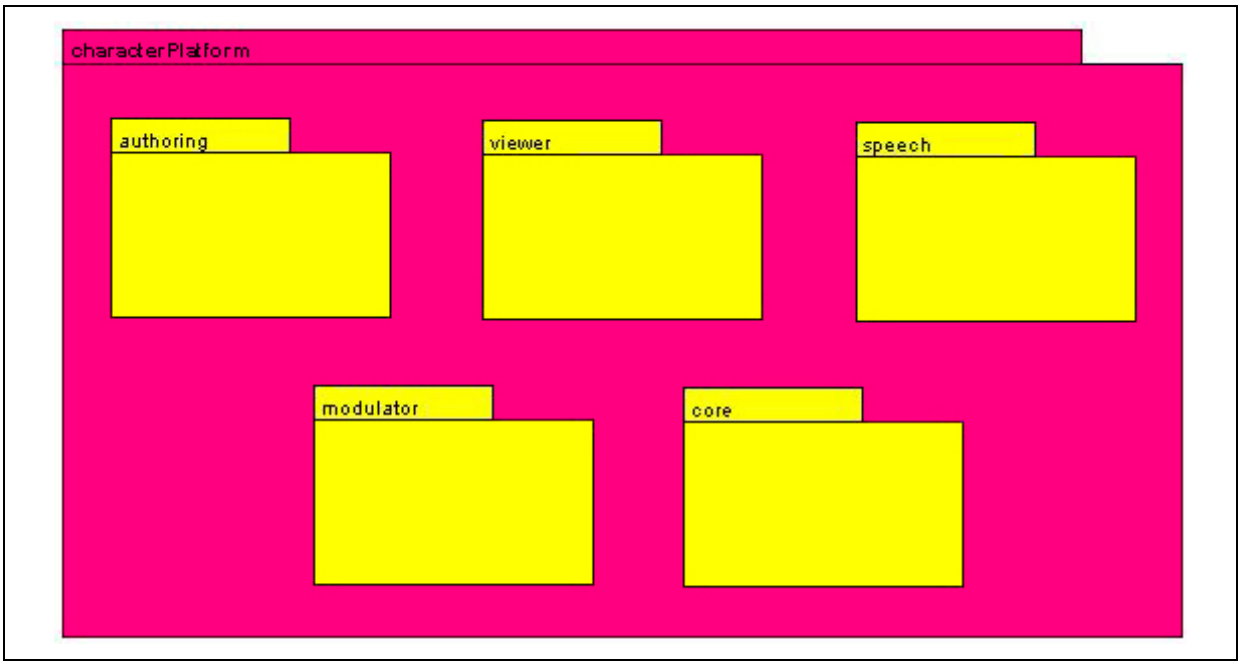
[15] Sun Microsystems®, "Java3D API 1.2", http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/j3dapi/, 06.04.2004

[16] Sun Microsystems®, "Java Speech API 1.0", http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-doc/, 06.04.2004

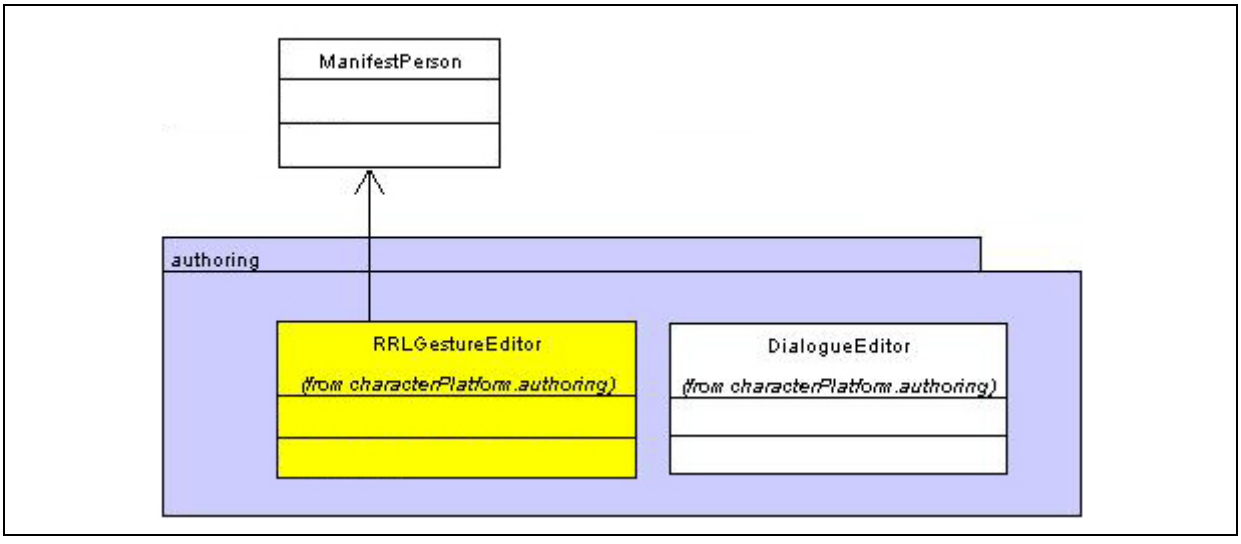
[17] The Apache Xml Project, http://xml.apache.org/xerces2-j/index.html, 06.04.2004

[18] AT&T Natural Voices™, http://www.naturalvoices.att.com, 07.04.2004

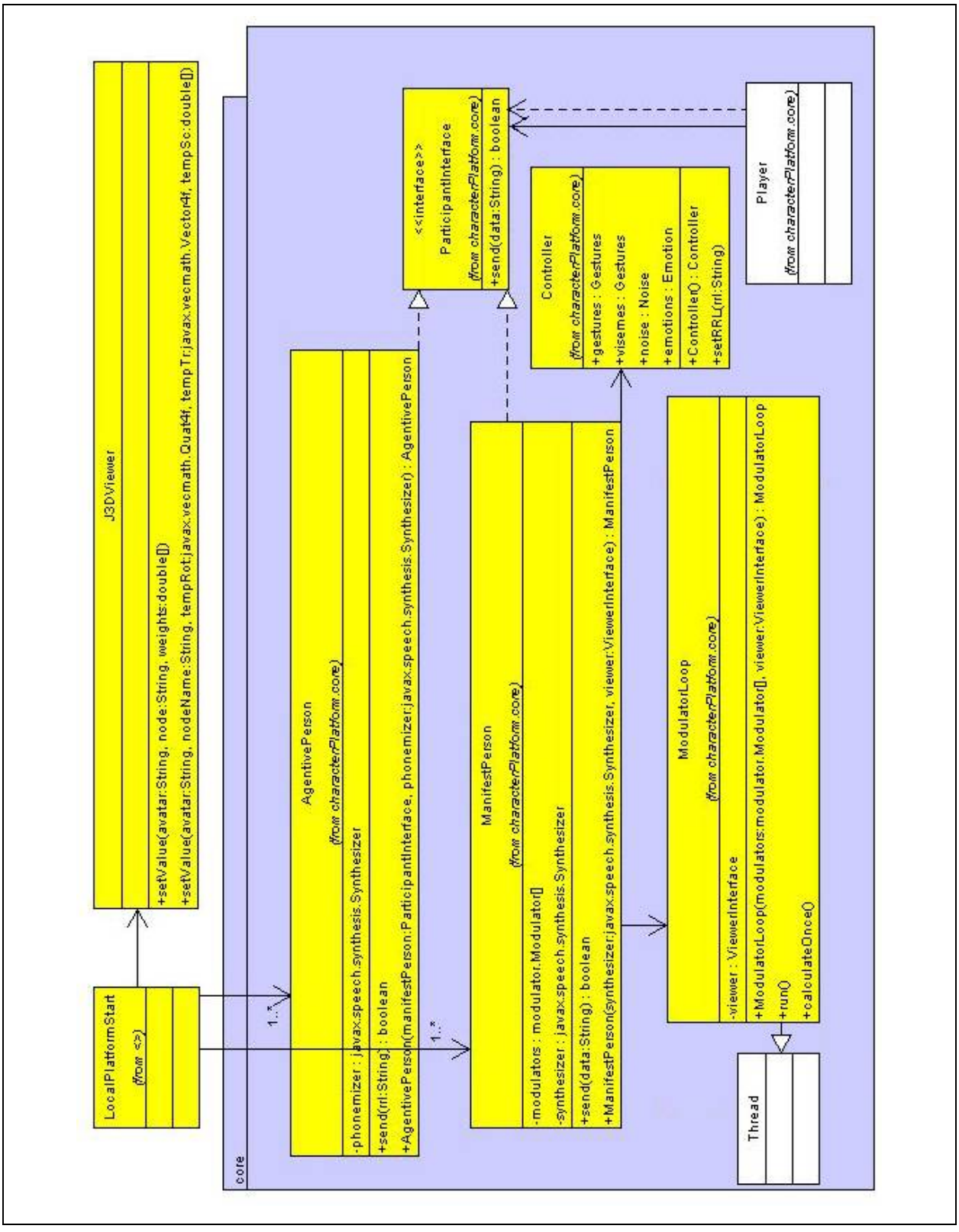
Anhang A: Vorgegebene UML-Klassendiagramme



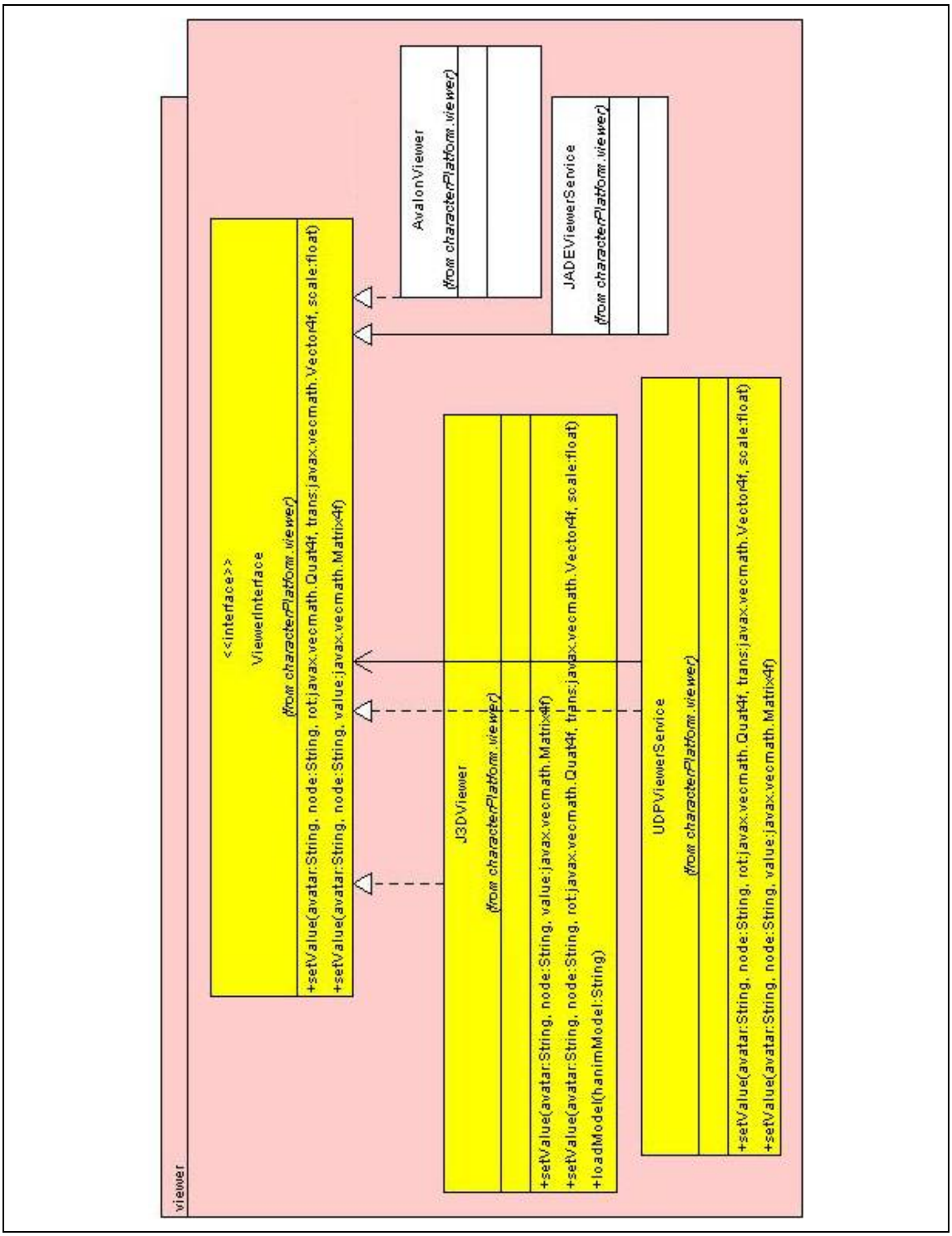
Vorgabe für das Package characterPlatform



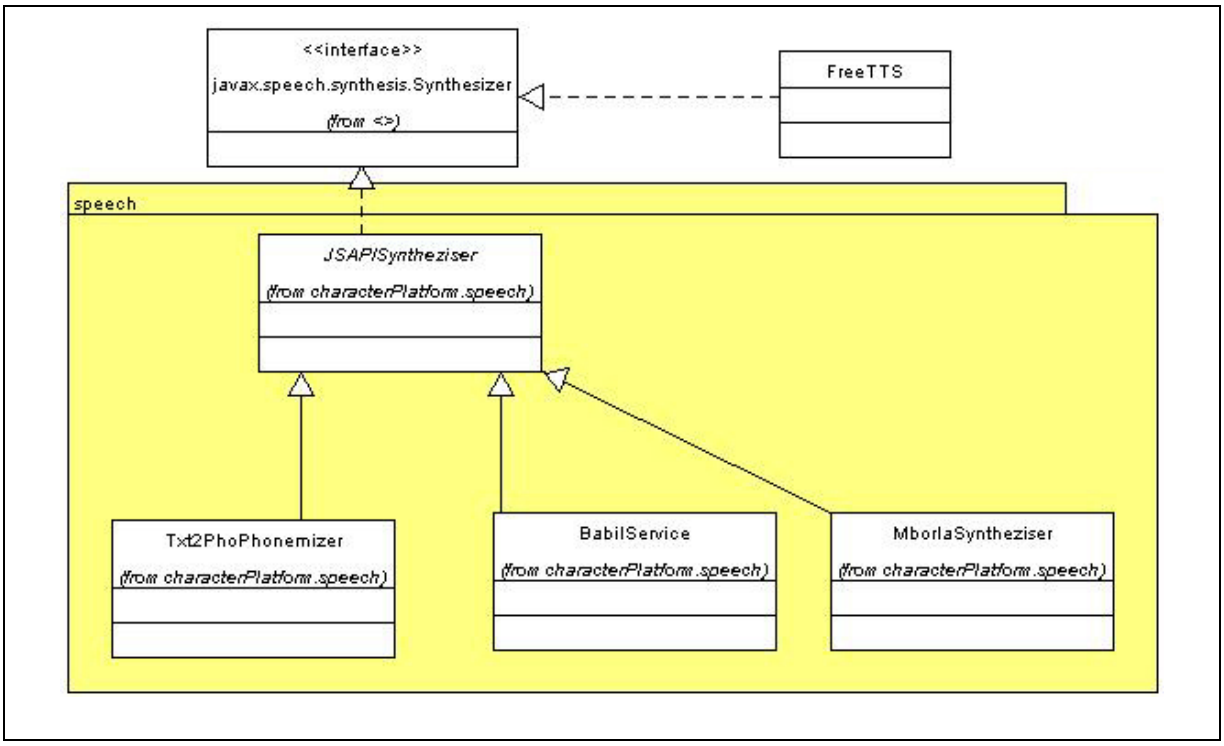
Vorgabe für das Package characterPlatform.authoring



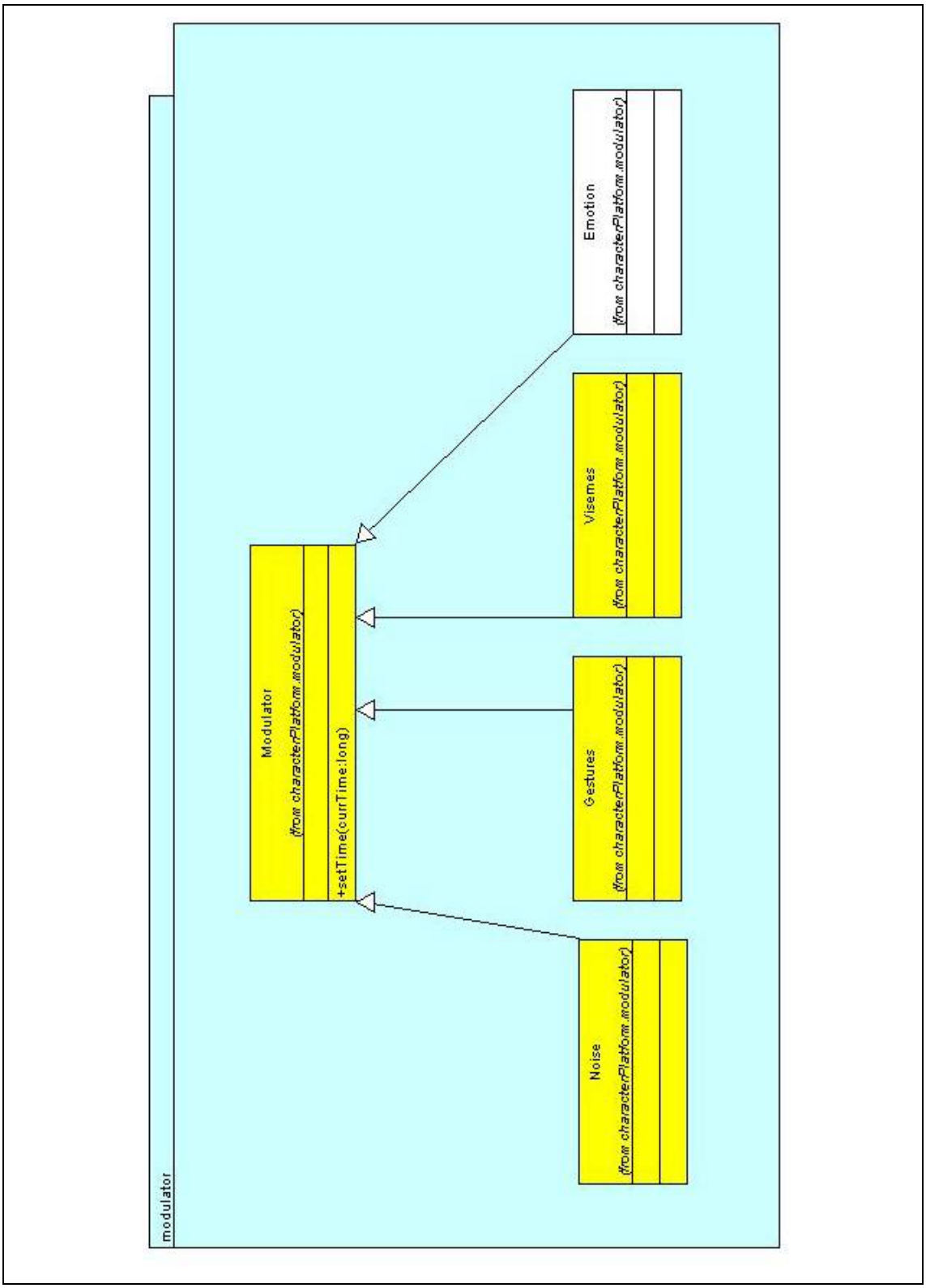
Vorgabe für das package characterPlatform.core



Vorgabe für das Package characterPlatform.viewer



Vorgabe für das Package characterPlatform.speech



Vorgabe für das Package characterPlatform.modulator

Anhang B: Inhalt der CD-Rom

Auf der CD-Rom, welche dieser Diplomarbeit beiliegt, befindet sich ein Film der die Animationsplattform in Verbindung mit einem interaktiven Geschichtsmodell demonstriert. Ebenfalls auf der CD enthalten sind eine digitale Version und eine Kurzfassung dieser Arbeit.

Verzeichnisstruktur der CD-Rom:

- Ordner Diplomarbeit
diplomarbeit.pdf – Digitale Version dieser Diplomarbeit

- Ordner Film
art-e-fact.avi – Film zur Demonstration der Animationsplattform

- Ordner Kurzfassung
kurzfassung.pdf – Kurzfassung der Diplomarbeit

index.html – Inhaltsverzeichnis der CD
autorun.inf – Datei zum automatisierten Öffnen des Inhaltsverzeichnisses der CD