Christian Donocik

Ein Animationswerkzeug zur Visualisierung sozialer Handlungen

Projektbetreuung:

Carola Eschenbach, Felix Lindner

Universität Hamburg, Dept. Informatik

AB Wissens- und Sprachverarbeitung

WiSe 2010/2011

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Inhaltsverzeichnis

Abbildungsverzeichnis

I. Einleitung

A. Die Mensch-Roboter-Interaktion

Dieser Bericht wurde im Rahmen des Projekts „Ein Animationswerkzeug zur Visualisierung sozialer Handlungen“ an der Universität Hamburg, Department Informatik im Wintersemester 2010 / 2011 erstellt.

Untersuchungsgegenstand des Projekts ist die Interaktion zwischen Maschinen (hier: Robotern) und Lebewesen (hier: Menschen). Menschen reagieren unterschiedlich stark, sobald sie sich in einer Interaktion mit einem Roboter befinden. Gründe für die verschiedenen Reaktionen können sowohl auf der Seite des Roboters, als auch auf der des Menschen liegen. Auf Seite des Roboters wären hier das Aussehen sowie die Verhaltensweise zu nennen. Aber auch die persönlichen Eigenschaften eines Menschen beeinflussen das Verhältnis Mensch-Roboter. So reagieren einige Menschen wesentlich stärker auf ein bestimmtes Roboteraussehen bzw. –verhalten, als andere. Dabei spielen menschliche Eigenschaften wie das Alter und das Geschlecht der Person eine entscheidende Rolle.[1]

Im Rahmen der Informatik als Wissenschaft ist die Mensch-Computer-Interaktion in vielen Zweigen relevant. Im Zuge fortschreitender Technologie nimmt ihre Bedeutung zudem rasant zu. So sind es heutzutage nicht nur die „klassischen“ Bereiche wie die Automobilindustrie (die Roboter u.a. zur Fertigung ihrer Produkte verwendet), sondern auch Organisationen wie Krankenhäuser, die beispielsweise Medikamente von einem Ort im Krankenhaus zu einem anderen Ort transportieren oder Operationen durchführen[2]. Hier werden in verschiedenen Bereichen bereits Roboter zur Durchführung der anfallenden Aufgaben verwendet.

Neben diesen exemplarisch dargestellten Anwendungsgebieten lassen sich noch diverse weitere aufzählen. Dabei konzentrieren sich die Forschungen im Wesentlichen auf die Gestaltung und das Verhalten eines Roboters. Beispiele hierfür sind die Software- und die Filmindustrie: Wie sollte sich ein Fußballspieler im Computerspiel „Fifa 2011“ optimaler Weise *verhalten*, damit der menschliche Spieler an seinem Spielcontroller das Maximum an möglicher Realität beim Spielen erfährt und wie sollte man im Rahmen eines Horrorfilmes den „Bösewicht“ optimaler Weise *gestalten*, damit er die gewünschten (Angst-)Gefühle beim Zuschauer hervorruft ?

[1] (Savicic, 2010), S. 43.
[2] Bsp: (Murphy & Schürmann, 2010), abrufbar im Internet unter: http://www.handelsblatt.com/technologie/forschung-medizin/forschung-innovation/roboter-laufen-aus-den-werkshallen/3550306.html (Stand: 4.März 2011).

B. Experimente zur Mensch-Roboter-Interaktion

In der Vergangenheit wurden bereits einige Experimente hinsichtlich bestimmter Faktoren durchgeführt, die die Beziehung zwischen einem Menschen und einem Roboter näher untersuchten. Aus diesen wurden bestimmte Rückschlüsse gezogen, die weitestgehend auch heute noch ihre Gültigkeit besitzen.

So zeigte bereits Albert Michotte in den 1940er Jahren durch Experimente, dass wahrgenommene Lebendigkeit hauptsächlich durch Bewegung erzeugt wird. Michotte ließ in seinen Experimenten ein Objekt in Form eines Kreises von links gegen ein ruhendes Objekt prallen. Die Versuchspersonen nahmen einen „launching effekt", also ein Anstoßen des bewegten Objekts gegen das unbewegte, wahr, obwohl sie wussten, dass es in der Realität keinen Zusammenhang zwischen beiden Objekten gab.[3]

Auch wurde in Experimenten das Aussehen eines Roboters im Hinblick auf dessen Wirkung bei einem Menschen untersucht. Der sog. „Uncanny Valley" Effekt stellt die Menschenähnlichkeit eines Roboters ins Verhältnis zur Vertrautheit eines Menschen. Dabei stellt man fest, dass ein Roboter einem Menschen immer vertrauter wird, je menschenähnlicher der Roboter gestaltet ist. Dies gilt jedoch nur bis zu einem gewissen Punkt, in dem sich neben der Wahrnehmung der Menschenähnlichkeit gleichzeitig das technische Element des Roboters bemerkbar macht. In diesem Punkt schlägt die Vertrautheit gegenüber dem Roboter stark ins Negative um (in das sog. „Uncanny Valley"), bis der Punkt einer vollendeten Menschenähnlichkeit vorliegt. Dieses Experiment dient bis heute als Grundlage für zahlreiche Experimente. So fand man bspw. im Jahre 2006 heraus, dass die (grafische) Form des „Uncanny Valleys" auch von ästhetischen Aspekten abhängt, „d.h. davon, welche Zwischenstufen die Transformation vom humanoiden zum androiden Roboter durchläuft".[4]

Ein für dieses Projekt besonders bedeutendes Experiment stellt die von Heider und Simmel erstellte Studie dar, die eine experimentelle Untersuchung von Attributen bei der Beobachtung von Bewegung einfacher geometrischer Figuren durchführt.

C. Die Heider-Simmel-Studie

Grundlegend für unser Softwaretool sind die im Jahre 1944 erstellten Animationsstudien von Fritz Heider und Marianne Simmel. Heider und Simmel erstellten einen kurzen Animationsfilm, der drei unterschiedlichen Gruppen jeweils zwei Mal vorgespielt wurde.

In dem Film werden drei bewegte geometrische Figuren gezeigt, welche sich mit unterschiedlicher Geschwindigkeit umherbewegen. Die Versuchspersonen sollten im Anschluss der Filme verschiedene Fragen beantworten.

[3] (Carroll, 1996), S. 170.
[4] (Decker, 2010), S. 59.

Beinahe alle Versuchspersonen deuten die Bewegungen als Handlungen von Lebewesen, in den meisten Fällen von Personen, in zwei Fällen von Vögeln. Einige interpretieren die gezeigten Handlungen als Kampf zwischen Objekten, zudem besteht die Auffassung, dass im Anschluss eine Jagdszene zwischen ihnen stattfindet. Die Versuchspersonen geben an, dass die Bewegungen der Objekte Ursachen haben und die Objekte ein bestimmtes Ziel erreichen wollen, „in der Wahrnehmung des Geschehens wurden den Personen also Motive und Absichten zugeschrieben“[5].

Außerdem wurden auch die äußeren Merkmale verschieden interpretiert. So wird behauptet, dass das größere Objekt einen Mann darstellt (stark/bedrohlich), während das kleinere Objekt eine (schwache, hilfsbedürftige) Frau veranschaulicht.[6] Abb. 1 zeigt die in den Heider-Simmel-Filmen dargestellten Objekte in Form von unterschiedlich großen Dreiecken und Kreisen.

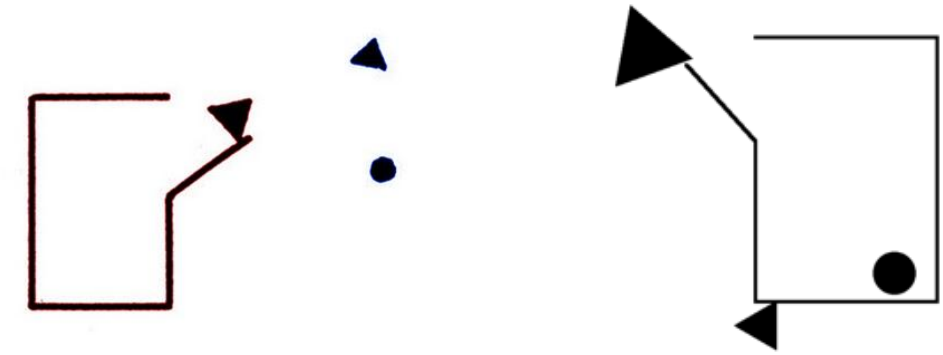

Abbildung 1: Momentaufnahmen des Heider-Simmel-Films

D. Vorgehensweise

Nachdem in den vorangegangenen Abschnitten eine kleine Einleitung zu unserem Softwaretool gegeben wurde, soll nun im folgenden Abschnitt das Simulationstool an sich vorgestellt werden. Ausgehend vom allgemeinen Aufbau und den verwendeten Bibliotheken fokussiert sich der Bericht auf den Bereich des Players.

In diesem Bereich stehen dem Nutzer mehrere Funktionalitäten zu Verfügung, wobei sich insbesondere der Export des Videos als etwas schwieriger gestaltet. Auf das zum Export verwendete Tool namens „Xuggler“ wird in diesem Abschnitt dann noch näher eingegangen. Abschließend findet noch eine Evaluation zur Exportfunktionalität statt.

[5] (Heidbrink, Lück, & Schmidtmann, 2009), S. 19.
[6] (Heider & Simmel, 1944), S. 248.

II. Das Simulationstool

A. Ziel

Da es äußerst kostspielig -und teilweise unmöglich- ist ein realitätsnahes Szenario zu erstellen, erscheint es sinnvoll ein (Software-)Animationstool zu entwickeln, welches Animationen im Sinne des Heider-Simmel-Film aufbauen kann.

Mit diesem Tool soll es möglich sein, auf grafische Weise Animationen zu editieren und individuell aufgebaute Szenarien abspielen zu lassen. Aus den dann zu beobachtenden Verhaltensweisen erhofft man sich Rückschlüsse auf die Realität ziehen zu können, sie also wieder in einen konkreten Kontext einbetten zu können. Dies würde helfen, sowohl vergangene Phänomene ex post zu betrachten (also abgeschlossene Experimente neu zu interpretieren) als auch Zukunftsperspektiven bei der sozialen Akzeptanz von Raumverhaltensmustern der Mensch-Roboter-Interaktion aufzuzeigen.

B. Kernkomponenten des Tools

Um ein individuelles Szenario aufbauen und sinnvoll bewerten zu können muss das Simulationstool gewisse Grundfunktionalitäten anbieten. Die praktische Umsetzung findet sich in drei Unterteilungen des Tools wieder: Dem Editorbereich (1), dem Animationsbereich (2) und dem sogenannten Player(-bereich) (3). Abbildung 2 zeigt einen Screenshot von der GUI des Simulationstools.

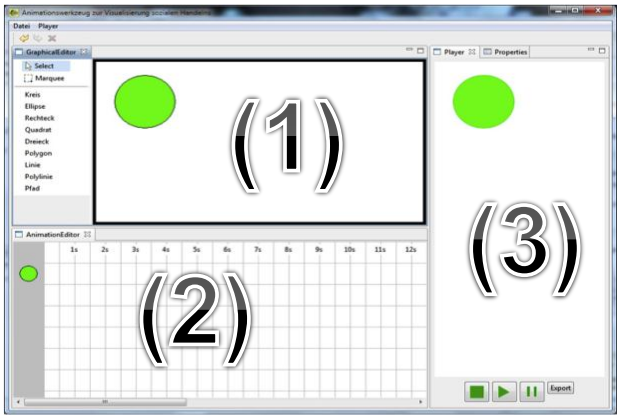

Abbildung 2: Aufbau des Simulationstools

Die einzelnen Bereiche des Softwaretools setzen unterschiedliche Funktionalitäten um. So ist es möglich, im Rahmen des Editorbereiches Szenarien aufzubauen und individuell zu gestalten. Hier können sowohl die Objekt*art* (Kreis, Dreieck, usw.), als auch deren *Eigenschaften* (z.B. Farbe) individuell verändert werden.

Zudem ist es hier möglich, Pfade zu erstellen, die „Wege der Objekte“ darstellen. Diese Funktionalität ermöglicht, Bewegungen grafisch darzustellen. Dabei kann es sich um gradlinige Pfade handeln, die den direkten Weg von einem Punkt A zu einem anderen Punkt B darstellen, oder aber auch um gekrümmte Pfade, bei denen das Objekt erst einen Bogen ablaufen muss bis das gewünschte Ziel erreicht ist.

Im Animationsbereich hingegen steht die Animation als solche im Mittelpunkt. Hier können die im Grafikeditor erstellten geometrischen Figuren einer oder mehrerer Animation(en) hinzugefügt werden. So lassen sich hier beispielsweise Start- und Endzeitpunkt der Animation festlegen. Zudem sind Abhängigkeiten zwischen Objekten in dieser Übersicht sehr gut erkennbar. Sobald ein Objekt im Editorbereich erstellt wird, erscheint es in einer vertikal angelegten Liste auf der linken Seite des Animationsbereiches. Durch Selektieren des Objekts lassen sich dann verschiedene Eigenschaften der Animation einstellen (Farbanimation, Rotationsanimation, etc.)

Das Ergebnis des mit Hilfe des Grafik- und Animationseditors aufgebauten Szenarios wird in der dritten Komponente, dem Player, in Form eines Videos dargestellt. Videos können hier gestartet, gestoppt, angehalten und exportiert werden.

C. Bibliotheken des Tools

1. Vorbemerkung

Da alle Komponenten nicht isoliert zueinander stehen, sondern ineinander greifen, bedarf es einer gemeinsamen Datenbasis. Dabei muss beachtet werden, dass die Informationen der verschiedenen Toolkomponenten persistent gehalten werden müssen, zugleich aber auch eine gewisse Performance gewährleistet sein muss.

Hierbei gibt es zwei prinzipielle Implementationsmöglichkeiten. Entweder tauschen die einzelnen Bereiche ihre jeweiligen Daten direkt untereinander aus, oder die Daten werden in einem zentralen Modell / Datei gespeichert, auf die alle Komponenten Zugriff haben.

Unser Softwaretool nutzt letztere Variante, also ein zentrales Szenenmodell in Form einer *.SVG Datei, in der alle Informationen des Aufbaus, der Objekte und der Animation hineingeschrieben werden. Die einzelnen Editoren greifen gezielt auf die jeweils benötigten Daten in der *.SVG Datei zu. Zudem wäre es bei der ersten Variante viel komplizierter eine erstellte Animationssequenz zu speichern und zu laden, da mehrere Dateien vorliegen.

Das zentrale Szenenmodell stellt somit eine vierte, auf der GUI nicht sichtbare, zentrale Komponente neben dem Grafik-, dem Animations- und dem Playerbereich dar.

2. SVG als zentrales Element

Die Abkürzung SVG steht für „Scalable Vector Graphics“. Hierbei handelt es sich um eine Spezifikation, die zweidimensionale Vektorgrafiken beschreibt. Das SVG Format ist ein Standardformat, welches von dem World Wide Web Consortium (W3C), einem Gremium zur Standardisierung des World Wide Webs, vorgeschlagen wurde und dem Anspruch gerecht wird, auf allen gängigen Browsern abspielbar zu sein. Diese Abspielbarkeit auf fast jedem Standardbrowser stellt einen zentralen Vorteil dieses Formates dar. Außerdem besteht mithilfe einer bereits existierenden BATIK-Bibliothek eine gute Möglichkeit für die Anbindung von SVG an Java. Für den Einsatz von SVG spricht zudem die Tatsache, dass die Darstellungsgröße unabhängig von der Informationsmenge, die man zur Beschreibung benötigt, immer dieselbe bleibt. Dies ermöglicht theoretisch eine hochkomplexe Darstellung von Grafiken mit geringen Dateigrößen, in die man beliebig hineinzoomen kann. Es basiert auf der „Extensible Markup Language“ (XML) und ist dadurch mit seinen Elementen und Attributen für jeden Menschen gut lesbar – auch ohne tiefergehende Programmierkenntnisse. Außerdem besitzt SVG eine hohe Kompatibilität zu bestehenden Technologien (CSS, etc.). Dadurch verringert sich der Lernaufwand, da die Verwendung von bereits bestehendem Wissen möglich ist.

Als Nachteile von SVG wäre zum einen zu erwähnen, dass SVG durch das W3C lediglich als „Empfehlung“ ausgesprochen wurde, nicht als fester „Standard“. Zudem gab es bis vor kurzem einige Browser, die keine native Unterstützung für die Anzeige von SVG Dateien enthielten. Hier ist die zusätzliche Installation von Plug-Ins erforderlich. Der Internet Explorer der Firma Microsoft beispielsweise, einer der gängigsten Browser überhaupt, besaß bis Version 8 keine SVG Unterstützung.

Die Vorteile von SVG überwiegen gegenüber den Nachteilen. Obwohl SVG lediglich eine „Empfehlung“ ist, gewinnt es rapide an Bedeutung.[7] Mittlerweile wird dies auch von fast allen bekannten Browserherstellern erkannt, so dass sich kaum ein aktueller Browser findet, der keine SVG Unterstützung anbietet[8] (der aktuelle Microsoft Internet Explorer, Version 9, beinhaltet nun auch eine entsprechende Unterstützung).

3. Exkurs: Die Extensible Markup Language

XML steht für Extensible Markup Language. XML besitzt eine gewisse Ähnlichkeit zu HTML. Im Gegensatz zu HTML wurde die XML Sprache jedoch nicht dazu entworfen, um Websites zu erstellen, sondern um Daten zu beschreiben. Sie stellt damit eine Meta-Sprache, also eine „Sprache über eine (andere) Sprache“, dar. Die Sprache besitzt eine Menge von

[7] (Gulbransen, Bartlett, Bingham, Kachur, Rawlings, & Watt, 2002), S. 438.
[8] Der aktuelle Netscape Navigator 4.x für Windows ist ein Beispiel für einen Browser, bei dem zusätzliche Plug-Ins installiert werden müssen, um SVG Dateien darzustellen.

Symbolen und eine Vielzahl von Regeln und ist damit sehr komplex. XML ist heutzutage weit verbreitet.[9]

Neben SVG existiert noch eine große Anzahl weiterer XML-Sprachen, so z.B. RSS, MathML, GraphML, XHTML u.v.m.[10]

Bsp.:

```
<Nachricht>
  <Ausruf> Hello World! </Ausruf>
  <Absatz> XML macht <Betonung>Spaß</Betonung>
  </Absatz>
</Nachricht>
```

Abbildung 3: Beispiel eines XML Codes

4. SMIL

Zudem wird in das Szenenmodell SMIL eingebunden. Die Abkürzung SMIL steht für Synchronized Multimedia Integration Language. SMIL ist ein vom W3C entwickelter Standard für eine Auszeichnungssprache für zeitsynchronisierte, multimediale Inhalte. Mit SMIL ist es möglich, Farbanimationen zu erstellen. Des Weiteren ermöglicht diese Bibliothek die Bewegung von Objekten auf einem festgelegten Pfad, wobei hierbei noch zusätzlich die Geschwindigkeit eingestellt werden kann, in der die Pfadbewegung erfolgen soll.

Abbildung 4 zeigt eine schematische Übersicht über die Kernkomponenten des Simulationstools:

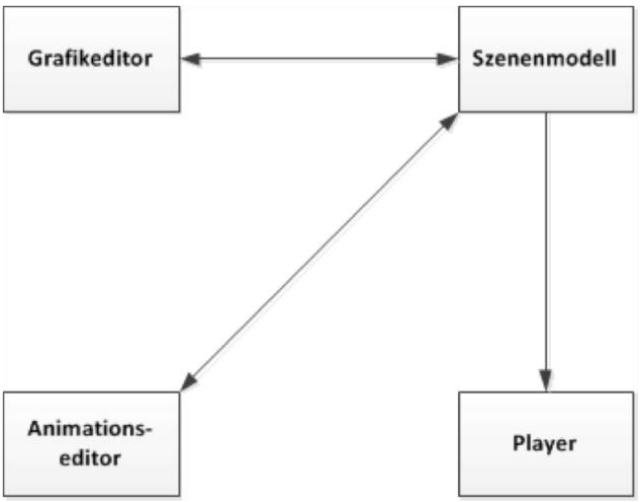

Abbildung 4: Komponentenübersicht der Software

Während der Grafik- und der Animationseditor auch Informationen in das zentrale Szenenmodell schreiben (daher an beiden Enden der Verbindungslinie eine Pfeilspitze), greift der (SVG-)Player lediglich die Daten aus dem SVG Dokument ab. Der Player selbst hingegen schreibt keine Daten in das SVG Dokument hinein.

[9] (Dunkel, Eberhart, Fischer, Kleiner, & Koschel, 2008), S. 195.
[10] (Rothfuss & Ried, 2003), S. 201.

5. Graphical Editing Framework

Kurz erwähnt werden sollte zudem das Graphical Editing Framework, kurz GEF[11]. Es dient dazu, grafische Editoren zu erstellen und ist frei im Internet erhältlich. So enthält GEF zum Beispiel ein Creation Tool, mit Hilfe dessen Objekte ausgewählt werden können.

Das GEF Framework basiert auf mehreren Design Pattern. Zu nennen ist hier primär das Model-View-Controller Pattern. Dieses Pattern (dtsch. Muster) hat drei verschiedene Rollen. Die Model Rolle wird durch ein Objekt repräsentiert, das die darzustellenden Informationen repräsentiert, einer View-Rolle für die Anzeige des Models in der Benutzerschnittstelle und einer Controller Rolle, die Eingaben entgegen nimmt und ggf. Veränderungen am Model veranlasst. Der Controller basiert auf dem EditPart und besitzt EditPolicies, die das Erstellen von Commands-Regeln gestattet. Weitere Patterns, die im Rahmen von GEF verwendet werden sind das Observer Pattern[12] und das Commands-Pattern.

Zusammen mit Draw2D[13] ermöglicht es GEF, geometrische Figuren mit dem zentralen Szenemodell zu verbinden. Draw2D ist eine Bibliothek zum Zeichnen von zwei-dimensionalen Objekten wie Linien, Kreisen oder Rechtecken. Sie hat als Grundlage das Standard Widget Toolkit (SWT), ein GUI Toolkit für Java. Ähnlich wie das Abstract Window Toolkit (AWT) verwendet SWT native Kontrollelemente.

Alle diese Bibliotheken werden im Tool an unterschiedlichen Stellen und für unterschiedliche Aufgaben verwendet. Die für den Player wichtigste Bibliothek ist SVG. Die anderen (insb. SMIL und GEF) sind eher für die Editor bzw. Animationsgruppe interessant und werden hier nicht weiter erläutert. Es sollte jedoch deutlich sein, dass sie wichtige Funktionen im Rahmen dieses Tools ermöglichen.[14]

III. Fokussierung: Der Playerbereich

A. Komponenten

Der Player spielt die im Editor und im Animationsbereich erstellten Animationen ab. Er befindet sich auf der rechten Seite des Simulationstools. Anders als der Grafikeditor und der Animationseditor ist der Player kein Editor. Das Abspielen der Videos ermöglicht die Durchführung einer umfangreichen Analyse der Animation.

Die Funktionalitäten lassen sich mit Hilfe von Buttons ausführen, an denen entsprechende Listener registriert sind. Die Buttons befinden sich im unteren Bereich des Playerfensters. Sie sind der Übersicht halber mit den gängigen Symbolen für Start, Stopp und Pause

[11] Nähere Ausführungen zu GEF Vgl. (Röhling, 2011).
[12] (Gruhn, Pieper, & Röttgers, 2006), S. 290.
[13] Eine sehr detaillierte Ausführung zu Draw2d findet sich in (Guojie, 2005), S. 463 ff.
[14] Auf GEF wird beispielsweise im Projektbericht von (Wilms, 2011) detaillierter eingegangen.

versehen. Der Export Button befindet sich rechts neben dem Pause Button, das eigentliche Video läuft in einem vordefinierten Bereich in der Mitte des Playerbereiches ab.

Im Playerbereich ist zudem eine Komponente untergebracht, die primär nicht der eigentlichen Playeraufgabe dient: Das Eigenschaftsfenster. (Properties). Es enthält sämtliche Eigenschaften zu dem im Editorfenster ausgewählten Objekt. Ist kein Objekt ausgewählt, so enthält das Eigenschaftsfenster auch keine Informationen und bleibt leer. Die folgende Abbildung 5 zeigt einen Screenshot des Playerbereichs.

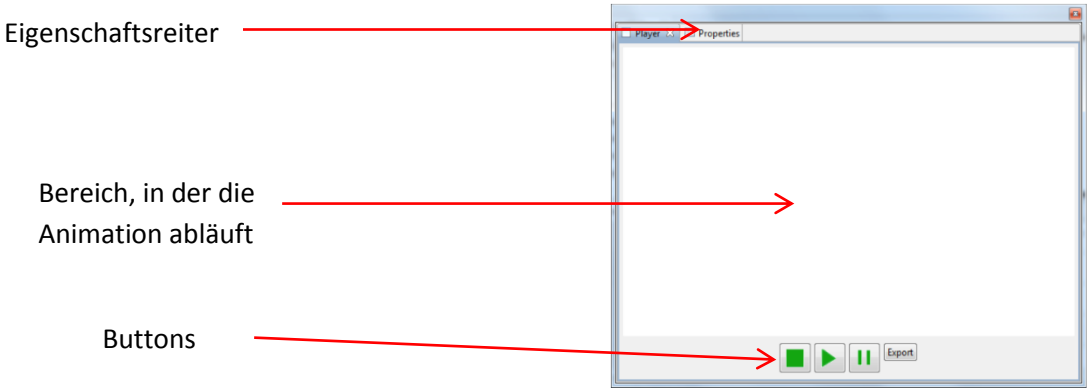

Abbildung 5: Der Playerbereich

Der Player umfasst im Einzelnen folgende Funktionalitäten:

- Start Button:
  Durch Betätigen des Start Buttons lassen sich Animationen starten.

- Stopp Button:
  Die Animation stoppt beim Drücken dieses Buttons und springt zum Start der Animation zurück.

- Pause Button:
  Die Animation wird in dem Punkt, in dem sie sich zum Zeitpunkt der Betätigung befindet, angehalten. Sie springt hier nicht zurück.

- Export Button:
  Durch Betätigung des Export Buttons öffnet sich ein Dateiauswahldialog, in dem man das Zielverzeichnis sowie das Zieldateiformat festlegen kann.

In der Menüleiste befindet sich während des Programmablaufes der Menüpunkt „Datei“, siehe Abbildung 6[15]. Hier besteht durch Anklicken die Möglichkeit eine Liste, das Drop-out-Menü, zu öffnen und die gewünschte Funktionalität auszuwählen.

Hier stehen folgende Funktionalitäten zur Verfügung:

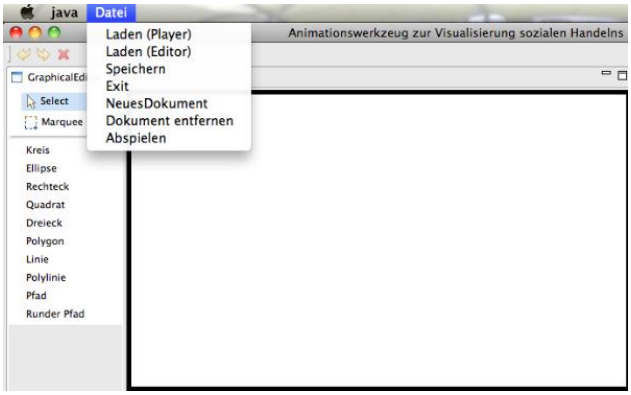

Abbildung 6: Das Datei Menü

- Laden:
  Einlesen von SVG Dateien in den Player bzw. Editor

- Speichern:
  Erstellte Animation im *.SVG Format abspeichern

- Exit:
  Beenden des Programms

- Neues Dokument:
  Hier wird ein neues, leeres *.SVG Dokument erstellt, in dem eine neue Animation erstellt werden kann

- Dokument entfernen:
  Hier kann eine geöffnete Animation geschlossen werden

- Abspielen:
  Dieser Menüpunkt ermöglicht das Abspielen der Animation.

[15] Hinweis: Hier wurde ein Apple Rechner verwendet, (optische) Abweichungen sind je nach Betriebssystem möglich.

Um die notwendigen Informationen der Animation zu erhalten, muss der Player in irgendeiner Weise auf die von den beiden Editoren erstellte SVG Datei zugreifen. Der Zugriff erfolgt über das sogenannte Batik Toolkit.

B. Die Technik des Players

1. Das Batik Toolkit

Batik[16] ist ein Java basiertes Tookit zur Erstellung von Anwendungen, die *.SVG Dateien anzeigen, generieren und manipulieren wollen. Batik wurde von der Apache Foundation entwickelt und enthält verschiedene Arten von Modulen. Unterschieden wird hier zwischen Application Modulen (SVG Browser (Squiggle), SVG Front Converter, SVG Rasterizer, SVG Pretty Printer, SVG Slideshow), Core Modulen (SVG Generator, SVG Dom, JSVGCanvas, Bridge, Transcoder) und Low Level Modulen (Graphic Vector Toolkit, Renderer, SVG Parser).

Application Module stehen auf der obersten Ebene und stellen die Anwendungen dar, die dem Endnutzer ohne Programmierkenntnissen zur Verfügung stehen. Ihre Funktionalität basiert auf Core Modulen. So ist der SVG Browser Squiggle ein Viewer, der zum Anzeigen von SVG Dokumenten dient. Neben dem Transformieren und Zoomen unterstützt er die Anzeige des Quellcodes und des DOM Baumes. Der SVG Front Converter ist ein Commandline-Programm zum Konvertieren von TrueType Fonts in SVG. Der Rasterizer hingegen unterstützt das Konvertieren von SVG-Dokumenten in Rasterformate (PNG, JPEG und TIFF). Schlussendlich dienen das Commandline-Programm SVG Pretty Printer zum Formatieren des Quellcodes von SVG Dokumenten und das SVG Slideshow Tool dem Verknüpfen von SVG Dokumenten zu einer Slideshow.

Core Module stellen das Herz des BATIK SVG Toolkits dar. Ihre Nutzung ist einzeln oder kombiniert möglich. Über JAVA und die Batik API kann auf die Core Module zugegriffen werden. Der SVG Generator beinhaltet SVGCanvas2D und hilft JAVA Programmen und Applets ihre Grafiken in SVG zu konvertieren. Das SVG DOM Modul ist eine Umsetzung der SVG DOM API. Über Java Programme können SVG Dokumente hier manipuliert werden. Die JSVGCanvas ist das UserInterface, welches der Anzeige von SVG-Dokumenten und der Interaktion mit dem User (Text markieren, Zoomen usw.) dient. Die Bridge konvertiert ein SVG Dokument in eine programminterne Repräsentation der Grafik (GVT, Graphic Vector Toolkit) und der Transcoder wandelt einen Input Stream oder ein Dokument in ein Ausgabeformat um.

Low Level Module hingegen dienen nicht dem Zugriff durch den Entwickler über die Batik API. Mit ihren Funktionen unterstützen sie die Core Module. So repräsentiert das Graphic Vector Toolkit die batikinterne Sicht auf den DOM Baum anhand von Java Objekten, der

[16] Weitere Ausführungen zu SVG und Batik, siehe (Geiger, 2011).

Renderer rendert die batikinterne Sicht und der SVG Parser dient zum Parsen von komplexen SVG Attributen wie *transform* oder *color*. Folgende Abbildung zeigt die Architektur des Batik Toolkits:

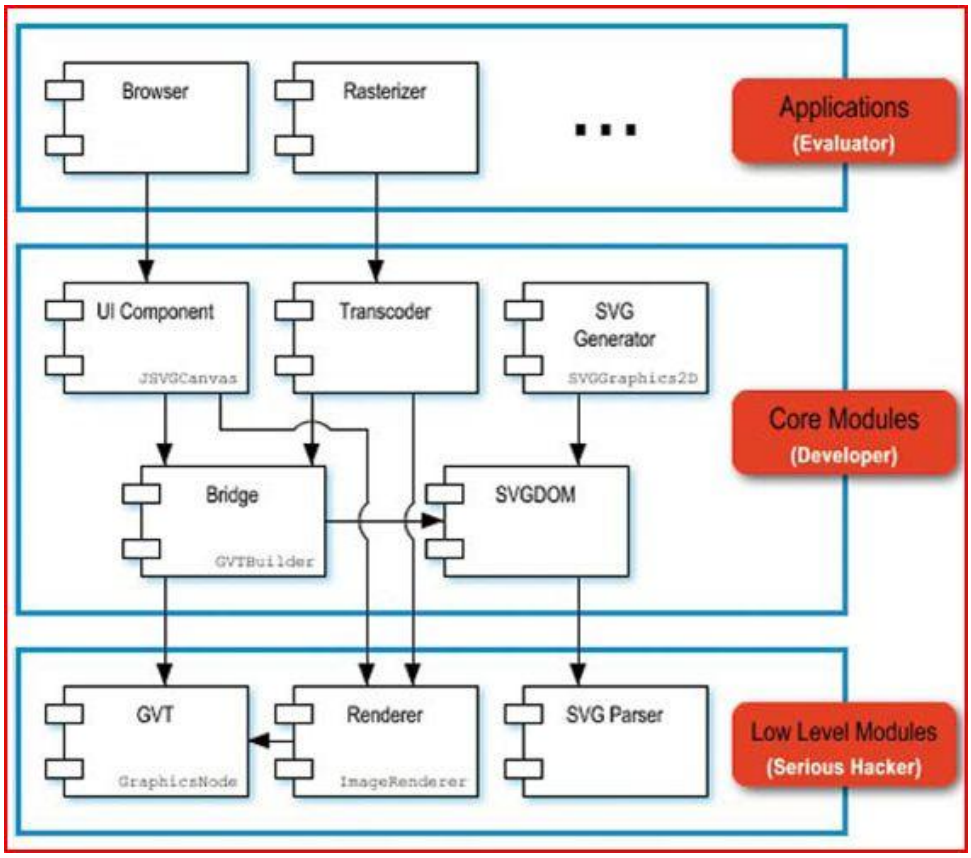

Abbildung 7: BATIK-Architektur, aus: http://XML.apache.org/

Die Java-Bibliothek Batik wird in unserem Tool als Bibliothek genutzt, da sie zum einen sehr umfassend ist, zum anderen ist Batik relativ einfach zu implementieren und zu verstehen.[17] Batik ist auch im Source-Code verfügbar, muss dann aber selbst kompiliert werden. Nachteil des Toolkits ist, dass es keine Methode für den Export bzw. die Konvertierung von SVG- in Videodateien beinhaltet. Nähere Erläuterungen zu diesem Punkt, siehe III.B.4.

Batik wird im Rahmen des Player Packages vorwiegend in den Klassen AnimCanvas und PUpdateManagerListener verwendet.

2. Implementation von Batik im Teilprojekt Player

Im Zusammenhang mit dem Player findet sich Batik in der AnimCanvas Klasse und in der PUpdateManagerListener Klasse wieder. Erstgenannte Klasse erbt von der JSVGCanvas

[17] Vgl. auch zu BATIK, (Geiger, 2011).

Klasse. Die SVGAnimationEngine hingegen wird benötigt, um den Animationsverlauf zu manipulieren.

Nachdem die AnimationsEngine ausgelesen wird („bridgeContext.getAnimationEngine()“) wird in der AnimCanvas Klasse festgelegt was in der Canvas, also dem Bildschirmbereich, in dem die Animation abläuft, passiert. Beispielsweise pausiert die ablaufende Animation durch den Aufruf „getAnimEngine().pause()“.

Im Rahmen der PUpdateManagerListener Klasse geht es primär darum, neue Updates im SVG Dokument zu registrieren, um diese dann entsprechend (im weiteren Verlauf) in der Canvas darstellen zu können. Dazu werden die UpdateManagerAdapter bzw. UpdateManagerEvent Klassen verwendet und ein Listener („PUdateManagerListener“) implementiert, der auf Ereignisse des UpdateManagers horcht.

3. Sonstiger (technischer) Aufbau des Players

Anders als der Grafikeditor und der Animationseditor ist der Player kein Editor und wird in einem View angezeigt. Um den Player mit seiner GUI aufzubauen wird die „Player“ Klasse verwendet. Die Elemente der GUI werden mit dem Gridlayout platziert, welches auf SWING basiert. Mit diesem Layout werden Komponenten in einem Gitter angeordnet, wobei jede Gitterzelle gleichgroß ist. Es kann spezifiziert werden, in wie vielen Spalten und Zeilen die Komponenten positioniert werden und wie groß der Abstand der Komponenten voneinander sein soll. Der Playerbereich ist in mehrere Komposites eingeteilt. Während in der „canvasComposite“ – wie der Name schon sagt- die Canvas festgelegt ist, werden im Komposit „comp“ die Buttons platziert. Das Canvas Bereichselement erlaubt das freie und starre Positionieren von Elementen über Koordinaten. Das englische Wort Canvas bedeutet Leinwand. Auf dieser werden die Elemente entsprechend angeordnet.[18] Beim Auslösen der Buttons reagieren dann entsprechende Listener (_pauseButton.addSelectionListener, _playButton.addSelectionListener, usw.). Zudem existiert ein Listener, der auf den Export Button reagiert. Es öffnet sich ein neuer Dateiauswahldialog, in dem das Zielverzeichnis und das gewünschte Zielformat für den Export des Videos festgelegt werden können (FileDialog fd = new FileDialog(shell, SWT.*SAVE*)), wobei als Zielformate *.mp4, *.mov und *.avi zur Verfügung stehen.

Um den Zustand der Anwendung zu verwalten existiert die ApplicationState Klasse. Sie erbt von der Observer Klasse. In ihr werden Informationen wie der SVGPfad gespeichert, also der Pfad in dem sich das vom Player gelesene SVG Dokument befindet (getCurrentSVGPath()).

Das Laden und Speichern einer Datei (erreichbar über den Menüpunkt Datei → Speichern bzw. Menü → Laden) ermöglicht der LoadSaveSVGHandler. Er implementiert das Interface IConstants und erbt von der Klasse AbstractHandler, die eine Superklasse für

[18] (Abramidis, 2009), S. 61.

HandlerListeners darstellt. Mit Hilfe der Methode selectFile() öffnet sich ein Dialog, in dem man – je nach Wunsch – eine SVG Datei in einem beliebigen Pfad ablegen oder aber auch ein bereits vorher abgespeichertes File öffnen kann. Lädt man eine Datei wird auch der aktuelle Zustand des ApplicationStates verändert (beispielsweise wird der dort festgehaltene Link entsprechend aktualisiert). Das Speichern einer SVG Datei erfolgt über die ImportExport Klasse, die dafür die Methode saveXML() anbietet.

4. Der Videoexport

Neben den Basisfunktionen „Start des Videos“, „Stopp des Videos“ und „Video-Pause“ besteht die Möglichkeit des Videoexports. Diese Funktion wird durch Betätigen des „Export“ Buttons ausgeführt. Da die Informationen in einer SVG Datei vorliegen, benötigen wir eine Möglichkeit, um SVG Dateien in entsprechende Videoformate zu konvertieren. Zudem muss folgender Punkt berücksichtigt werden: Da es sich um eine *.SVG Datei handelt und diese auf XML Notation beruht, liegen hier zunächst einfache Vektorgrafiken als Ausgangsformat vor. Hier wird demnach eine Animation als „zeitliche Änderung der Animation selbst oder eine Änderung der Position von Elementen im aktuellen Koordinatensystem“ verstanden.[19] Sowohl die Eigenschaften als auch die Attribute der in SVG enthaltenen Objekte können unabhängig voneinander animiert werden, wobei jeder dieser Animationen einen eigenen Zeitverlauf hat. Bei der in SVG vorliegenden (deklarativen) [20] Animationsform wird der Zeitverlauf formal in einem Dokument beschrieben und es wird ein Programm benötigt, welches aus dieser Beschreibung eine zeitabhängige Darstellung generiert. Eine alternative Variante stellt die rahmenbasierte Animation dar. Diese Animation wird durch Aufnehmen der gesamten Szenerie mit Hilfe von Einzelbildern, sog. Rahmen (engl. Frames) durchgeführt, die dann in der Darstellung mit fest definierten Anzeigedauern aufeinander folgen. Vorteil der deklarativen Animation ist insbesondere, dass eine Angabe der Interpolation über den gesamten Zeitverlauf ermöglicht wird. Die framebasierte Animation hingegen ist dann besser geeignet, wenn man keine sehr komplexen Szenerepräsentationen vorliegen hat, denn bei sehr komplexen Objekten ist die Erstellung einer exakten geometrischen Szenerepräsentation unter Umständen extrem arbeitsintensiv und mit wirtschaftlich vertretbarem Einsatz kaum möglich.[21]

Batik bietet in seinen Standardbibliotheken keine Möglichkeit an, eine direkte Umwandlung von SVG Dateien in eine Videodatei vorzunehmen. Potentielle Alternativen, wie „Quicktime for Java“ oder die javaeigene Klasse „JpegImagesToMovie“ sind aufgrund einiger Schwierigkeiten bei der Implementierung bzw. den –im Vergleich zu Xuggler- relativ spärlichen vorhandenen Umwandlungsmöglichkeiten nicht verwendet worden.

---

[19] Ein Beispiel für die Implementation über das *animateMotion-Element* ist in (Born, 2005), S. 371 ausführlich dargestellt.
[20] Vgl.. http://de.wikibooks.org/wiki/SVG/_Animation (Stand: 18.03.2011).
[21] Vgl. (Schmid, 2003), S. 57.

In unserem Projekt wird eine Umgebung namens Xuggler verwendet, welche als Open Source Applikation erhältlich ist. Xuggler ermöglicht über die Aufnahme von Screenshots, also framebasiert, einen Film zu erstellen und diesen dann in einem gewünschten Format in einem bestimmten Verzeichnis abzulegen. Xuggler besitzt diverse Vorteile, auf die in den folgenden Abschnitten näher eingegangen wird.

C. Xuggler als „Exporttool“

In diesem Abschnitt soll erläutert werden, warum Xuggler genutzt wird und welche Vorteile diese Umgebung hat ? Welche Nachteile werden dafür ggf. in Kauf genommen ?

Xuggler ermöglicht das Komprimieren bzw. Dekomprimieren, konvertieren und modifizieren von Mediendateien (oder Streams) aus Java heraus. Mit Hilfe von Xuggler kann man beispielsweise Videos mit zusätzlichem Untertitel versehen, oder auch Videos – wenn denn gewünscht - rotieren lassen. Die aktuelle Version von Xuggler ist die 3.4. – „Forrest“ Version, welche am 1. Februar 2010 veröffentlicht wurde. Xuggler wird in der Windows Umgebung durch bloße Ausführung einer *.exe Datei installiert, unter Linux basierten Systemen oder Mac OS X müssen weitere Schritte durchgeführt werden.

Vorteil dieser Umgebung ist zunächst die Tatsache, dass sie frei und kostenlos verfügbar ist. Somit kann sich jeder mit einem Internetzugriff Xuggler installieren. Zudem ist Xuggler leicht zu implementieren, dem Nutzer werden „lästige“ Details zu einzelnen Codecs, Bibliotheken oder Umwandlungsprozessen erspart und es wird ihm hier nur wenig Vorwissen abverlangt. Dennoch ist es ohne großen Aufwand möglich, sämtliche Einstellungen auch manuell einzustellen. So erkennt Xuggler die Parameter für das umzuwandelnde Quell(video-)format grundsätzlich automatisch, manuelle Eingaben sind aber in den entsprechenden Methoden des Quellcodes möglich. Ein weiterer großer Vorteil ist zudem die Mächtigkeit der Umgebung. Xuggler erlaubt die Bearbeitung einer sehr großen Anzahl an verschiedenen Video- und Audioformaten. So ermöglicht es unter anderem eine „on-the-fly Konvertierung“ oder beinhaltet diverse Modifikationsmöglichkeiten beim Streamen von Videos.

Größter Nachteil ist die unter Umständen schwierige Portierbarkeit der Xuggle-Umgebung. Während der Hersteller für Windows Nutzer eine ausführbare *.exe Datei zu Verfügung stellt, müssen unter MAC OS X oder Linux basierten Systemen zusätzliche Einstellungen durchgeführt werden, nach denen ggf. im Internet gesucht werden muss.(bspw. muss der genaue Pfad zur DYLD_Library_Path eingestellt werden, der je nach Rechner und Installation unterschiedlich sein kann).

Xuggler besteht aus einem Set von Java und (System-) Bibliotheken und beinhaltet zwei verschiedene APIs:

- MediaTools API
- Xuggler API

Die MediaTools API beinhaltet die Basisfunktionalitäten von Xuggler. Mit ihr lassen sich auf einfache Art und Weise Audio- und Videodateien kodieren bzw. dekodieren. Diese API wird in unserem Projekt verwendet.

Die Xuggler API ist eine Erweiterung der MediaTools API. Sie ist mehr für Fortgeschrittene gedacht und beinhaltet weitergehende Möglichkeiten bei der Video- und Bildverarbeitung. Von der MediaTools API kann man auf die Xuggler API zugreifen, falls der Bedarf weitergehender Einstellungsmöglichkeiten besteht. Diese wurde im Rahmen dieses Projekts nicht verwendet, da die MediaTools API ausreichend für unsere Anforderungen war.

Alle Funktionalitäten von Xuggler zu beschreiben würde den Rahmen dieses Berichts sprengen. Ich beschränke mich im Folgenden auf die für das Projekt relevanten Aspekte der Xuggler Umgebung. Dabei wird zunächst kurz auf die allgemeine Funktionsweise der bei uns verwendeten MediaTool API eingegangen, um dann die konkrete Umsetzung im Quellcode näher zu erläutern.

1. Allgemeine Funktionsweise der MediaTool API

Die MediaTool API stellt aus programmiertechnischer Sicht ein Interface dar. Ein Interface ist eine Klasse, welche ausschließlich einen Typ definiert und keine konkrete Implementation beinhaltet. Die MediaTool API erleichtert dem Nutzer die Bearbeitung von Mediendateien, da sie viele Details der dahinter liegenden Container(-formate) und Codes versteckt. Unter einem Containerformat lassen sich verschiedene Arten von Dateiformaten (also z.B. Bildformate ebenso wie Audio- oder Videoformate) verstehen. Das Containerformat fungiert dabei als Behälter (engl. Container) und legt fest, in welcher Art und Struktur seine Inhalte darin untergebracht werden.[22]

Kernelemente des MediaTool API sind im Wesentlichen die beiden Interfaces „IMediaViewer“ und „IMediaWriter“. Dabei verwendet die MediaTool API das Event Listener Konzept: Der IMediaWriter wird als „Listener“ zum IMediaReader hinzugefügt und erhält alle decodierten Medien. Die beiden Interfaces IMediaViewer und IMediaWriter implementieren das IMediaListener Interface und können dann weiter als Listener zu einem IMediaReader hinzugefügt werden. Die beiden folgenden Abbildungen veranschaulichen die Beziehungen:

[22] (Krischak, 2008), S. 8.

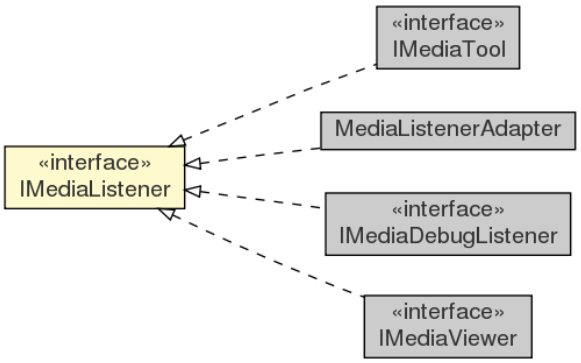

Abbildung 8: Die Interfaces IMediaViewer und IMediatool erben vom IMediaListener

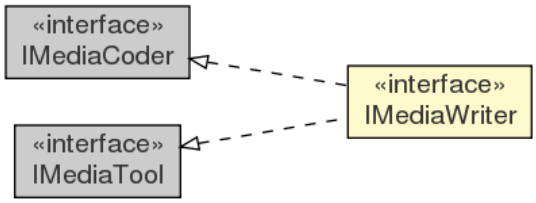

Abbildung 9: Der IMediaWriter erbt - über IMediaTool - vom IMediaListener

Um nun beispielsweise ein IMediaWriter Objekte zu erstellen, lässt sich die ToolFactory Klasse verwenden. Die beiden erstellten Objekte lesen und schreiben dann Daten in IContainer Objekte, ohne die Details bzgl. der Kodierung / Dekodierung dabei anzuzeigen. Sie erzeugen dabei Events, welche entsprechende Objekte, die beim IMediaListener registriert sind, wahrnehmen (und ggf. reagieren). Dieses Wahrnehmen erfolgt direkt zur Laufzeit, d.h. dass der IMediaWriter direkt aktiv werden kann, sobald der IMediaReader anfängt, eine Datei zu lesen. Dabei erkennen die beiden Objekte automatisch die für die Kodierung und Dekodierung der entsprechenden Mediendatei erforderlichen Parameter. Manuelle Anpassungen sind hierbei allerdings möglich. Um entsprechende Medien zu konvertieren verwendet Xuggler exklusiv das sogenannte FFmpeg.

2. FFmpeg

Das FFmpeg Projekt ist ein OpenSource Projekt und besteht aus einer Vielzahl von Programmbibliotheken, die das digitale Aufnehmen, konvertieren und streamen von Audio- und Videomaterial ermöglicht. Dazu verpackt es die Mediendateien in verschiedene Containerformate (z.B. *.avi, *.mkv, *.flv). FFmpeg lässt sich aufgrund seiner Kommandosteuerung über diverse Programmiersprachen wie ActionScript, Lingo, PHP und Java ansprechen und ist damit sehr flexibel einsetzbar. Es verwendet libavcodec als Bibliothek, welche eine sehr große Auswahl an ffmpeg-Audio- und –Video Encoder beinhaltet. Aufgrund der großen Anzahl an Formaten, mit denen FFMpeg umgehen kann, wird es auch als das „Schweizer Messer“ der Videokonvertierung bezeichnet.[23] FFmpeg wird in Xuggler zur Konvertierung eines Quellformats in ein entsprechendes Zielformat genutzt. Es ist in Xuggle als fester Bestandteil enthalten, wie in Abbildung 10 dargestellt.

[23] (Förster & Öggl, 2011), S. 99.

Abbildung 10 veranschaulicht den Umwandlungsprozess von einem Quellformat in ein bestimmtes Zielformat mit Hilfe von FFmpeg:

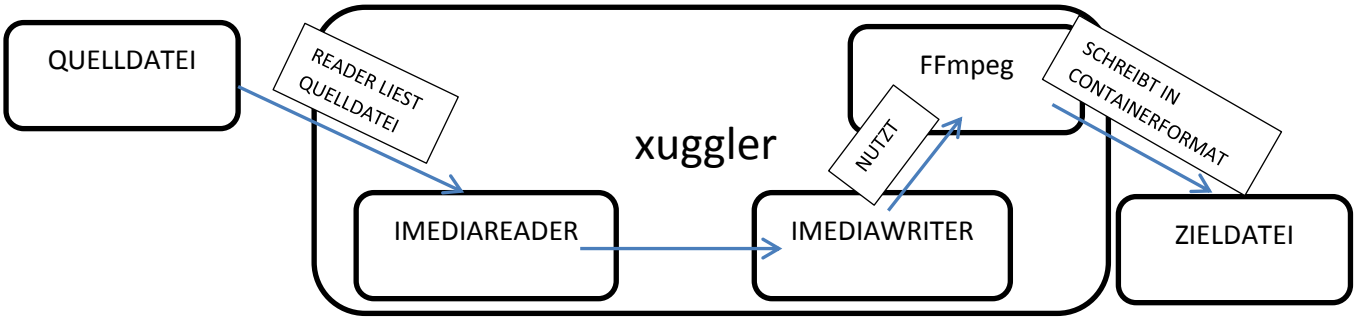

Abbildung 10: Der Umwandlungsprozess mit Xuggler

3. Implementierung von Xuggler im Projekt

Die konkrete Umsetzung von Xuggler findet in unserem Tool in der Klasse „CaptureScreenToFile“ statt. Diese Klasse befindet sich im Player Package des AnimationswerkzeugGEF Projekts. Sie enthält einen Programmcode, welcher über einen Zeitraum von 5 Sekunden 24 Screenshots pro Sekunde aufnimmt und diese dann im Rahmen einer Schleife speichert. Durch die Verwendung des Interfaces IMediaWriter wird ein Objekt mit Hilfe der „ToolFactory.makeWriter“-Methode erstellt. In unserem Projekt beschränken wir uns auf das Hinzufügen eines VideoStreams. Ein AudioStream, der optional möglich wäre, wird nicht genutzt. Mit Hilfe der Methode „writer.addVideoStream“ wird der Videostream mitsamt seiner Eigenschaften hinzugefügt. (bspw. die Frame Rate oder der Aufnahmebereich).

In dem Zeitraum von 5 Sekunden werden dann nacheinander insgesamt 24*5 = 120 Bildschirm Screenshots gemacht, konvertiert und zum neu erstellten Video hinzugefügt. Dies erfolgt in der folgenden Weise: Der aufgenommene Screenshot wird mit Hilfe der Methode „convertToType“ in das gewünschte Format konvertiert. Beim Betätigen des Export Buttons öffnet sich ein Fenster, in dem der Nutzer zwischen den (Video-) Zielformaten *.mp4, *.mov und *.avi per Drop Down Auswahlliste wählen kann. Zudem wird auch hier das Verzeichnis festgelegt, in dem das zu exportierende Video abgespeichert werden soll. Abbildung 11 zeigt das Dialogfenster, in dem ausgewählt werden kann, in welches Verzeichnis und in welchem Format das Video exportiert werden soll.

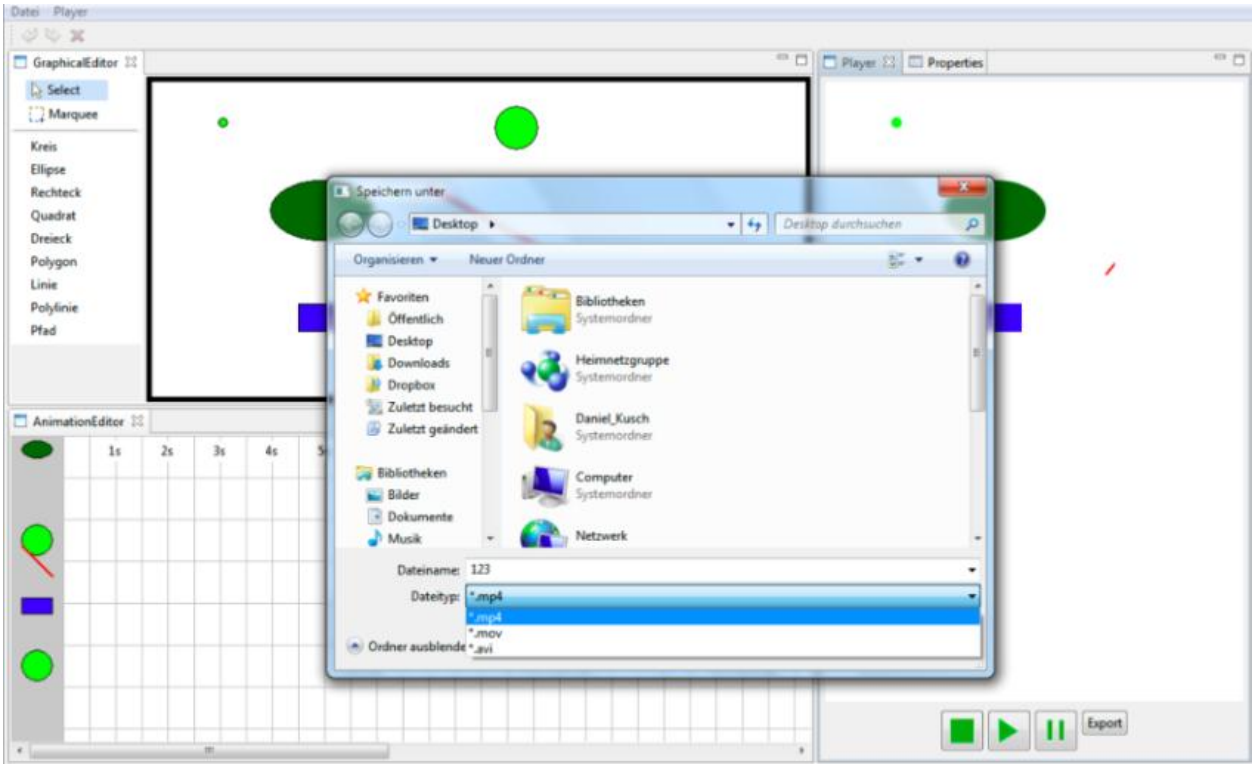

Abbildung 11: Dialogauswahlfenster beim Export

Der konvertierte Frame wird dann zum neu erstellten Container hinzugefügt. („writer.encodeVideo“- Methode). Deckt sich das ursprüngliche Format der Videodatei bereits mit dem gewünschten Zielformat ist natürlich keine Konvertierung notwendig.

IV. Evaluation

Eine umfangreiche Gesamtevaluation des Projektes ist im Projektbericht von Sergej Lehmann[24]aufgeführt. In dem hier enthaltenen Abschnitt wird daher nur eine kurze Evaluation hinsichtlich des Videoexports erfolgen.

Die Videoexport Funktionalität ist im Rahmen der Playerkomponente implementiert. Die Player Komponente ist essentiell, um das eigentliche Projektziel, nämlich die soziale Akzeptanz von Raumverhaltensmustern bei der Mensch-Roboter-Interaktion, zu untersuchen. Mit ihrer Hilfe lassen sich Bewegungen von Objekten optisch darstellen. Da jede Animation ihre spezifischen Eigenschaften (Objekte, Wege, etc.) hat, ist es sinnvoll, diese an einem Ort dauerhaft abzuspeichern. Es wäre sehr umständlich jedes Mal eine

[24] Vgl. (Lehmann, 2011).

Animation neu aufzubauen, zumal dies auch einen großen zusätzlichen Zeitaufwand als Konsequenz hätte.

Hier kommt die Export Funktionalität ins Spiel. Mit ihr ist es möglich, die erstellte Animation in einem gewünschten Videoformat abzuspeichern. Diese erfolgt, wie oben erwähnt, framebasiert.

Die Implementation dieser Funktionalität mit Hilfe von Xuggler besitzt zunächst den Vorteil, dass dieses Tool ziemlich mächtig ist. So lassen sich mit Hilfe von FFMpeg bei Bedarf noch (diverse) weitere Videoformate denken, in die die Bilder exportiert werden könnten. In unserem Tool wird dem Nutzer ermöglicht, sein gewünschtes Zielformat festzulegen, indem er die Auswahl zwischen drei verschiedenen Zielformaten, *.mp4, *.mov und *.avi Dateien, hat. Weitere Zielformate wären bei Bedarf implementierbar und machen das Tool somit flexibler.

Zudem wäre eine Erweiterung dahingehend denkbar, dass man die Filme mit zusätzlichen „Features“ ausstattet, beispielsweise mit einer Art textuellen Beschreibung während des Films oder auch einem Audiostream. Dies würde zwar über die Heider-Simmel-Animation hinausgehen, eröffnet jedoch zugleich neue Möglichkeiten, da ein Zusammenspiel von Audio und Video nun möglich wäre. Die Implementation des Audio Streams wäre mit Hilfe von Xuggler ohne viel zusätzlichen Code möglich.

Weiterhin spricht für die Anwendung des Tools, dass es frei und kostenlos im Internet verfügbar ist. Es wird nicht verlangt, dass man für den Export zusätzlich Geld bezahlt o.ä.

Ein drittes Argument pro Xuggler ist die leichte Implementation. Es wird lediglich die *.exe Datei benötigt, die auf der Xuggler Homepage heruntergeladen werden kann, vorausgesetzt man arbeitet auf einer Windows basierten Plattform. Die entsprechenden Einträge übernimmt das Programm dann von selbst, so dass es sich ohne fachmännisches Wissen installieren lässt, vorausgesetzt es wird eine Windows Plattform verwendet.

Durch den relativ einfachen Quellcode ließe sich zudem die Qualität der Aufnahme leicht erhöhen. In der aktuellen Version nimmt Xuggler 24 Screenshots/Sekunde auf. Dieser Wert ließe sich noch beliebig weiter erhöhen. Allerdings ist für den aktuellen Zweck die Größenordnung von 24 Screenshots/Sek. absolut ausreichend.

Ein Nachteil liegt hingegen darin, dass überhaupt eine externe Umgebung verwendet werden muss, die nicht zum „Java Standardpaket“ gehört. Dies erfordert erhöhten Aufwand, da Xuggler zunächst installiert werden muss. Zudem erfordert die Installation von Xuggler auf einer Linux basierten Plattform / Mac OS X basierten Plattform zusätzlichen Aufwand, da dort weitere Dateien manuell erstellt werden müssen, bevor Xuggler überhaupt verwendet werden kann. Dies wurde auch in unserem Teilprojekt Team deutlich: Während die Installation bei zwei Personen unter Windows problemlos funktionierte, dauerte die Installation bei den Macintosh Usern erheblich länger.

Eine Verbesserungsmöglichkeit besteht darin, das Auslesen der Animationslänge beim Videoexport dynamisch zu gestalten. Aktuell nimmt Xuggler beim Videoexport 24 Bilder pro Sekunde für 5 Sekunden lang auf. Sollte eine Animation länger als 5 Sekunden dauern, dann würde alles, was nach einer Länge von 5 Sekunden abgespielt wird nicht mit in das exportierte Video eingefügt, was ggf. zu einer unvollständigen oder gar falschen Analyse führen könnte.

Hierbei muss bei einem framebasierten Export allerdings beachtet werden, dass zunächst das gesamte Video durchlaufen werden muss, bevor es abgespeichert werden kann. Dies kann unter Umständen bei sehr langen Videos zu einer schlechteren Performance führen, da hierbei auch die Video(datei)größe entsprechend zunimmt.

Eine weitere Möglichkeit besteht in der Erweiterung der Zielformatliste. Da manche externe Player nur bestimmte Videoformate abspielen, ist es u.U. sinnvoll weitere Formate zu definieren. (bsp. *.flv, *.wmv, *.mkv, etc.)

Abschließend bleibt festzuhalten, dass der Export einer aufgebauten Animation mit Hilfe von Xuggler problemlos erfolgt. Da wir es bis zum aktuellen Zeitpunkt ausschließlich mit Animationen mit einer geringen Maximallänge zu tun haben, ist das drohende Performanceproblem bei einer langen Szene hier zu vernachlässigen. Xuggler ist eine einfach zu implementierende Umgebung, gleichzeitig stellt es aber auch aufgrund seiner vielzähligen Erweiterungsmöglichkeiten eine sehr umfangreiche Möglichkeit dar, Videos zu exportieren und eignet sich damit auch gut für Projekte, die zukünftig auf diesem Tool aufbauen.

Ob es sich rentiert, alternative Exportvarianten zu nutzen, hängt von mehreren Faktoren ab, z.B. von der Länge der Animation oder von zukünftigen Alternativen auf dem Markt. Ggf. wäre dann auch an eine deklarative Herangehensweise zu denken.

Für das zum aktuellen Zeitpunkt existierende Simulationstool mit dem Ziel (kurze) Animationen analog zu denen von Heider und Simmel zu erstellen und diese zu exportieren, stellt Xuggler eine gute und zuverlässige Möglichkeit dar.

V. Literaturverzeichnis

Abramidis, G. (2009). *Silverlight 2 - Professionelle Media Rich Anwendungen.* München: Addison-Wesley-Verlag.

Born, G. (2005). *XML - Der einfache Einstieg in den führenden Dokumenten- und Web-Standard.* München: Markt+Technik Verlag.

Carroll, N. (1996). *Theorizing the moving image.* New York : Cambridge University Press .

Decker, M. (2010). Ein Abbild des Menschen: Humanoide Roboter. *Information und Menschenbild*, S. 41-62.

Dunkel, J., Eberhart, A., Fischer, S., Kleiner, C., & Koschel, A. (2008). *Systemarchitekturen für verteilte Anwendungen.* München: Carl Hanser Verlag.

Förster, K., & Öggl, B. (2011). *HTML 5 - Leitfaden für Webentwickler.* München: Addison-Wesley-Verlag.

Geiger, A. (2011). *Batik und SVG.* Hamburg: Universität Hamburg.

Gruhn, V., Pieper, D., & Röttgers, C. (2006). *MDA: Effektives Software-Engineering mit UML2 und Eclipse* . Heidelberg: Springer Verlag.

Gulbransen, D., Bartlett, K., Bingham, E., Kachur, A., Rawlings, K., & Watt, A. (2002). *Using XML - second edition.* USA: Que.

Guojie, J. L. (2005). *Professional Java Native Interfaces with SWT/JFace.* Danvers: Wrox Verlag.

Heidbrink, H., Lück, H. E., & Schmidtmann, H. (2009). *Psychologie sozialer Beziehungen.* Stuttgart: Kohlhammer Verlag.

Heider, F., & Simmel, M. (April 1944). An Experimental Study of Apparent Behaviour. *The American Journal of Psychology, Vol. 57*, S. 243-259.

Krischak, M. (2008). *Empfehlungen für die Digitalisierung, Konvertierung und Publikation von Audio- und Videodokumenten.* Norderstedt: GRIN Verlag.

Lehmann, S. (2011). *xcy.* Hamburg.

Murphy, M., & Schürmann, H. (29. 09 2010). Roboter laufen aus den Werkshallen. *Handelsblatt*.

Röhling, S. (2011). GEF als Werkzeug für Editoren - Evaluation auf Basis einer Fallstudie.

Rothfuss, G., & Ried, C. (2003). *Content Management mit XML, 2. Auflage* . Berlin: Springer Verlag.

Savicic, A. (2010). *Gesprächsakzeptanz von Robotern - Am Beispiel von Actroid-DER2 und Leonardo* . Norderstedt: Books on demand GmbH.

Schmid, K. (2003). Animation mit bildbasierter Szenerepräsentation - Diplomarbeit . Weimar, Universität Weimar.

Wilms, C. (2011). *Ein Multi-Selection-Tool für GEF.* Hamburg.