Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Fachbereich Informatik
Wissens- und Sprachverarbeitung (WSV)

GEF als Werkzeug für Editoren

Evaluation auf Basis einer Fallstudie

Projektbericht

Animationswerkzeug zur Visualisierung sozialen Handelns

Sven Röhling

Matr.Nr. 5887082

sven.roehling@informatik.uni-hamburg.de

April 2011

Dozenten:

Dr. Carola Eschenbach
Felix Lindner

Inhaltsverzeichnis

Abkürzungsverzeichnis

GEF Graphical Editing Framework

IDE Integrated Development Environment

MVC Model-View-Controller

RCP Rich Client Platform

SMIL Synchronized Multimedia Integration Language

SVG Scalable Vector Graphics

SWT Standard Widget Toolkit

W3C World Wide Web Consortium

WSV Wissens- und Sprachverarbeitung

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Gegenstand dieses Projektberichts ist die Dokumentation der Konzeption, der Durchführung, sowie die Beurteilung der endgültigen Anwendungssoftware, die im Rahmen des Projekts „Animationswerkzeug zur Visualisierung sozialen Handelns“ entwickelt wurde. Die Veranstaltung fand innerhalb des Arbeitsbereich Wissens- und Sprachverarbeitung (WSV) des Fachbereichs Informatik der Universität Hamburg im Wintersemester 2010 / 2011 statt.

Dabei liegt der Schwerpunkt dieses Dokuments auf der Untersuchung eines Fallbeispiels, nämlich der Entwicklung eines Animationswerkzeug, einem grafischen Editor für einen definierten Anwendungszweck, mit Hilfe des Graphical Editing Framework (GEF). Anhand diesem wird der Produktionsprozess unter die Lupe genommen und Vorteile, aber auch Nachteile, die eine Arbeit mit dem Framework mit sich bringen zusammengetragen.

Beginnend in Kapitel 2 wird das Animationswerzeug an sich beschrieben. Das heißt, es wird ein Einblick in die Motivation für den Einsatz eines solchen Tools gegeben, als auch eine von seinen Anforderungen getriebene konzeptionelle Grundstruktur veranschaulicht. Das Kapitel 3 dient zur Einführung in die Grundstruktur von GEF und gibt einen Überblick über spezielle Konzepte und Features, die mit dem Nutzen des Frameworks einhergehen. Die tatsächliche Umsetzung einzelner dieser Features werden in dem Kapitel 4 dargestellt, indem konkrete Beschreibungen dieser zu ausgesuchten Anwendungsbeispielen näher erläutert werden. Die Evaluation im abschließenden Kapitel 5 trägt die daraus resultierenden Erkenntnisse zusammen, die sich in der Entwicklung mit GEF herauskristallisiert haben.

2 Das Animationswerkzeug

2.1 Motivation

Roboter dringen immer weiter in unsere Gesellschaft ein und finden dank der ständigen technischen Weiterentwicklung immer mehr Plätze, an denen sie eingesetzt werden können, um dem Menschen Aufgaben abzunehmen oder ihm bei der Verrichtung dieser zu unterstützen. Wo früher ein Roboter noch mit einer Fernbedienung gesteuert werden musste und somit sein Handeln von einem Menschen explizit veranlasst wurde, geht der Weg heute immer mehr hin zu selbst „denkenden“ Maschinen, die die Entscheidungen über ihr Handeln selbst treffen können.

Aus diesem Grund ist es wichtig, dass diese eigenständigen Agenten sich auch adäquat im Umgang mit den Menschen verhalten. Das heißt einerseits, dass sie von dem Menschen mit dem sie interagieren auch als diese vorgesehene unterstützende Maschine akzeptiert werden. Wissenschaftliche Untersuchungen haben ergeben, dass ein Roboter, der sich zu menschlich bewegt oder ihm ähnelt, von seinem Gegenüber, ungewollter Weise, durchaus als unangenehmen Partner empfunden werden kann. Diese Tatsache nennt man den „Uncanny Valley “ Effekt, der bereits 1970 von Masahiro Mori beschrieben wurde [6].

Ein weiterer Schwerpunkt in der Forschung ist die Bewegung von Robotern innerhalb von Räumen, in denen sich auch Menschen aufhalten. Auch hier soll sich der Roboter so verhalten, dass er vom Menschen als Mitglied der Gemeinschaft empfunden wird und nicht als störende Maschine. Der Artikel von Elena Pacchierotti et al. beschreibt den privaten Raum einer Person, wie die private Sphäre, die ein Roboter betreten würde, falls er der Person zu nahe kommt [7]. Dieses soll natürlich verhindert und aus diesem Grund weiterhin untersucht werden, wie hier in einem Experiment, in dem sich ein Roboter und eine Person auf einem Flur gegenüberstehen und eine für den Menschen bevorzugte Weise des einander Passieren untersucht wurde.

Diese Art von dem zuvor beschriebenen Experiment ist natürlich sehr kostspielig und erfordert eine Menge Raum, Zeit und Material, was gerade bei dem Testen mit Prototypen eher rar ist. Dieser Umstand erfordert eine Suche nach andersartigen Forschungsmethoden, die Ergebnisse nahe der Realität liefern. Einen Ansatz für eine Alternative liefern die Untersuchungen von Fritz Heider und Marianne Simmel aus dem Jahr 1944 [5]. In dem von ihnen dokumentierten Experiment wurde Probanden ein Animationsfilm, der aus gezeichneten zweidimensionalen Dreiecken, Kreisen und Linien bestand, die in bestimmter Weise auf der Leinwand angeordnet waren und sich auf dieser bewegten, vorgeführt und anschließend zu ihren Eindrücken befragt.

Die Große Mehrheit der befragten interpretierte in den gezeigten Ablauf von Bewegungen eine Geschichte mit eigenständigen Lebewesen, die soziale Handlungen ausführen, wie sich gegenseitig jagende Dreiecke oder ein Tür (Linie) öffnendes Dreieck. Aus diesen

Erkenntnissen kristallisierte sich eine Verbindung zwischen bestimmten Bewegungsabläufen der gezeichneten Objekten und einer ihr eindeutig zugeordneten Handlung heraus. Aufbauend auf diesen Ergebnissen haben auch Brian J. Scholl und Patrice Tremoulet [8] die Wirkung sich bewegender zweidimensionaler Formen untersucht und haben ebenfalls eine Vielzahl an Bewegungsabläufen isoliert, die von Menschen als eindeutig interpretierte soziale Handlungen angesehen werden können.

Diese Tatsache bildet die Motivation auf der die Durchführung des Projekts, in dem ein Animationswerkzeug zur Visualisierung sozialen Handelns entwickelt werden soll, fußt. Mit Hilfe dieses Animationswerkeugs soll die Möglichkeit geschaffen werden, Animationsfilme aus sich bewegenden zweidimensionalen geometrischen Grundformen benutzerfreundlich erstellen zu können.

Diese Sequenzen von Bewegungsabläufen können dann, aufgrund der zuvor betrachteten Forschungsergebnisse, als Grundlage für eine erste Einschätzung über die Einstufung von sozialem Verhalten nicht menschlicher Akteure dienen. Das heißt, bestimmte Akteure in den Animationsfilmen sollen Roboter darstellen und Probanten können das Verhalten, dass in den Filmen veranschaulicht wird, bewerten. Somit kann vor der Durchführung eines Experiments unter realen Bedingungen bereits eingeschätzt werden, wie die Testpersonen voraussichtlich reagieren werden und daraus gegebenfalls bereits Hinweise für eine Veränderung und damit verbundene Verbesserung von Testszenarien erhalten.

2.2 Anforderungen

Das zu entwickelnde Animationswerkzeug soll eine komfortable Erstellung von Animationsfilmen gewährleisten. Dazu zählt das Erstellen und Editieren von geometrischen Figuren, sowie die Möglichkeit diese zu animieren und schlussendlich muss das Abspielen der gesamten animierten Szene ermöglicht werden.

Die Bedienung soll dem Standard ähneln, der bei gängigen grafischen What You See Is What You Get (WYSIWYG) Editoren vorherrscht. Das heißt, Funktionen, wie zum Beispiel eine „Drag & Drop“ Unterstützung oder Steuerung mittels einer Werkzeugleiste, an die sich Benutzer weitestgehend gewöhnt haben, müssen vorhanden sein und eine intuitive Bedienbarkeit garantieren. Das sichert eine kurze Einarbeitungszeit der Anwender und damit mehr Zeit für das Wesentliche, nämlich das erstellen von Animationen.

Eine weitere Anforderung für die Umsetzung des Projekts, die Entwicklungsumgebung betreffend, ist die Umsetzung mit der Programmiersprache Java und der Eclipse Integrated Development Environment (IDE), was sich aufgrund der Vorkenntnisse der Projektteilnehmer anbietet und in diesem Punkt keine allzu umfangreiche Einarbeitungszeit von Nöten ist. Das Dateiformat, in dem die Szenen gespeichert werden, ist das SVG Format, welches in dem Kapitel 2.4 näher beschrieben wird.

2.3 Konzept der Grundstruktur

Das Konzept für das Animationswerkzeug besteht aus einer Grundstruktur, die sich in vier Hauptmodule unterteilt. Jedes Modul beinhaltet ein abgeschlossenes Aufgabengebiet für den Erstellungprozess einer Animationszene.

Das zentrale Dokument (hier Szenenmodell) muss natürlich jedem Anwendungsbereich zugänglich gemacht werden, woraus Kommunikationsströme zwischen den Modulen entstehen, die in Abbildung 2.1 veranschaulicht werden. Nachfolgend werden die Module und ihre Hauptaufgaben kurz vorgestellt.

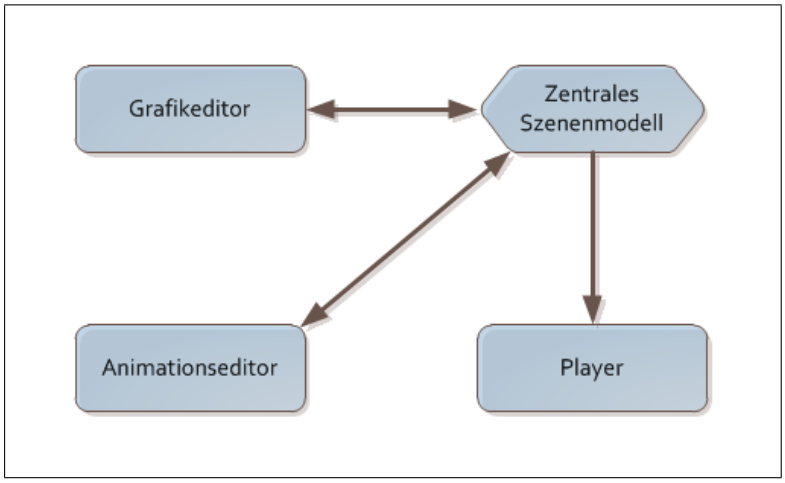


Abbildung 2.1 – Entwurf der Grundstruktur für das Animationswerkzeug

2.3.1 Zentrales Szenenmodell

Das zentrale Szenenmodell verwaltet das SVG-Dokument, welches die Gesamte Animationsszene enthält. Außerdem extrahiert es aus diesem die einzelnen enthaltenen geometrischen Figuren und Animationen als Java-Objekte, die ebenfalls den Bedarf nach einer konsistenten Verwaltung haben. Dieses Modul muss geeignete Schnittstellen besitzen, damit andere Module, wie die Editoren, problemlos an die in ihm enthaltenen Informationen heran kommen.

2.3.2 Grafikeditor

Der Grafikeditor dient zur Erstellung der Geometrische Formen, wie Kreise, Vierecke oder auch Pfade. Diese müssen nachträglich editiert werden und bei Bedarf auch wieder gelöscht werden können. Diese Funktionalität soll die zuvor genannten Anforderungen

für eine intuitive Benutzung erfüllen. Selbstredend muss eine Möglichkeit zum Manipulieren der im zentralen Szenenmodell gehaltenen Objekte, sowie das Anzeigen dieser gegeben sein.

2.3.3 Animationseditor

Im Animationseditor sind die im Grafikeditor erstellten Objekte aufgelistet und ihnen können Animationen zugeordnet werden, die das zeitliche und räumliche Verhalten dieser verändert. Animationen können in ihm erstellt, editiert und auch wieder gelöscht werden. Auch hier muss ein Zugriff auf das zentrale Szenenmodell gewährleistet sein und es gilt die geforderte Benutzerfreundlichkeit zu wahren.

2.3.4 Player

Der Player dient zum Abspielen des im zentralen Szenemodell gehaltenen SVG- Dokuments. Außerdem bietet er noch die Funktionalität zum Exportieren der Szene in weitere Videoformate. Dieses Modul muss zwar an das Dokument aus zentralen Szenenmodell heran kommen, verändert es aber selbst nicht, weswegen hier nur eine einseitige Kommunikation stattfinden wird.

2.4 SVG/SMIL

Scalable Vector Graphics (SVG) ist ein Vektorgrafikstandard, der im Jahr 2001 von dem World Wide Web Consortium (W3C) verabschiedet wurde. Er ist Extensible Markup Language (XML)-basiert und befindet sich momentan in der Version 1.1. Das Format ermöglicht mittels Textdateien verlustfreie Vektorgrafiken zu entwickeln, die mittlerweile in fast allen gängigen Internet-Browsern dargestellt werden können. Die Vorteile sind sehr kleine Dateigrößen, da sie nur aus Text bestehen und die Beibehaltung der Qualität bei der Skalierung der Grafiken. Eine ausführliche Dokumentation ist auf der W3C-Website [13] zu finden.

Ein weiterer Vorteil, der das SVG-Format von anderen Bildstandards abhebt, ist die Möglichkeit die beschriebenen grafischen Objekte zu animieren. Dieses geschieht mit Hilfe der eingebetteten Synchronized Multimedia Integration Language (SMIL) Spezifikation [14], die ebenso XML-basiert, momentan in der Version 3.0, direkt in die Beschreibung der SVG-Elemente einfließen kann und eine Vielzahl an Möglichkeiten zur Einbindung von Multimediainhalten, sowie deren Animation bietet.

Für das Animationswerkzeug werden genau diese Animationsmöglichkeiten, wie Bewegungsanimation, Farbanimation oder auch Transformationsanimationen benötigt und dadurch erfüllt die Kombination dieser beiden Standards genau die Anforderung an das Speicherformat für das Animationswerkzeug.

3 Das Graphical Editing Framework

Das Graphical Editing Framework (GEF) ist ein open source Java-Framework das eine große Bandbreite an Funktionalität für die Entwicklung von rich client Editoren bietet und somit die Entwicklungszeit von grafischen Editoren erheblich verkürzen soll. Es ist Teil des Eclipse Tools Project [11] und derzeit in der Version 3.6.2 verfügbar.

GEF setzt auf der Rich Client Platform (RCP) Eclipse auf und profitiert aus diesem Grund vorweg von Vorzügen, wie ausgereiften Basiskomponenten für GUI Anwendungen, die sich bereits in vielen Anwendungsfällen bewährt haben, die daraus resultierenden gefestigten Grundstrukturen, sowie die Modularität und Erweiterbarkeit durch das Plug-in Konzept, auf die die RCP ausgelegt ist. Eine grundlegende Einführung mit der Unterstützung von Tutorials zum Thema RCP bietet das Online Buch von Ralf Ebert [9] oder auch das Buch von Berthold Daum [1]. Die Abbildung 3.1 zeigt explizit die Plugin-Abhängigkeit von GEF.

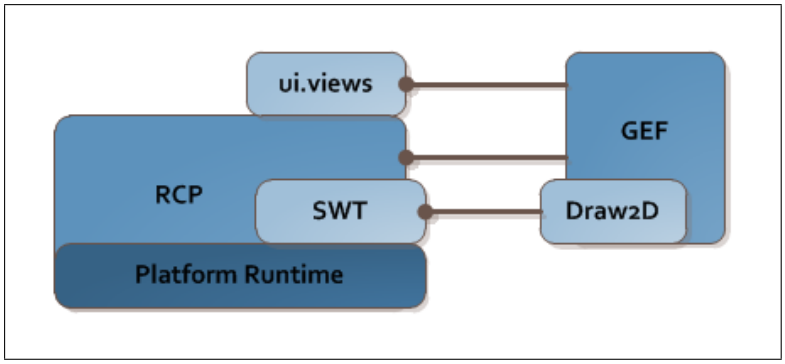


Abbildung 3.1 – GEF Plugin-Abhängigkeit

3.1 Draw2D

Die Grafikschicht von GEF setzt auf Draw2D auf, was wiederum auf dem Standard Widget Toolkit (SWT) aufbaut und den Umgang mit den von SWT benutzten Betriebssystemkomponenten vereinfacht. Anstatt direkt mit diesen arbeiten zu müssen, ist es mittels Draw2D möglich direkt mit den grafischen Objekten zu agieren. All diese Objekte basieren auf der Klasse `Figure`, welche das Interface `IFigure` implementiert und sämtliche Eigenschaften definiert.

Es existieren diverse vorgefertigte Klassen in Draw2D, wie zum Beispiel `Panel`, `Button` oder `Ellipse` und sofort einsatzfähig sind, das eigenständige Definieren von abgeleiteten Klassen oder das Bilden von Kompositionen ist genauso gut möglich. Zu den Eigenschaften des Erscheinungsbilds sind in den grafischen Objekten bereits die Unterstützung für die Verarbeitung von Ereignissen, die bei der Änderung von Eigenschaften oder Maus- und Tastatursteuerung auftreten, vorgesehen.

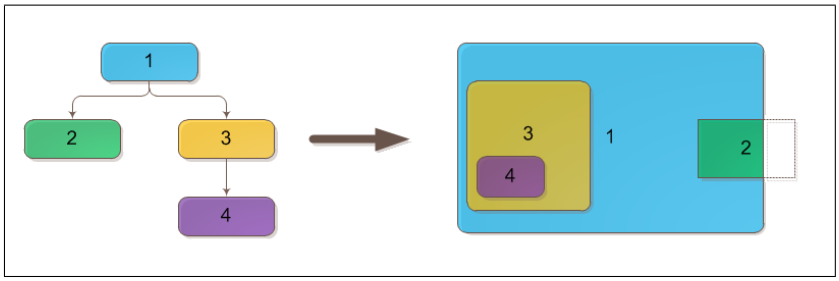


Abbildung 3.2 – Hierarchie von Draw2D Figuren

Eine weitere Eigenschaft aller grafischen Objekte in Draw2D ist ihr hierarchischer Aufbau. Das heißt, es gibt eine Wurzelfigur, der diverse Kinder, diesen wiederum Kinder und so weiter, zugeordnet werden können. Diese Kinder werden auch nur ausschließlich in ihren übergeordneten Figuren gezeichnet, was auch zu einem Abschneiden von Figuren führen kann. Abbildung 3.2 zeigt dieses Verhalten an der Figur 2. Die Eventverarbeitungen folgen ebenfalls dieser Hierarchie.

Die Positionierung von Figuren innerhalb ihrer Wurzel erfolgt mittels einem zugeordneten Layouts, das ihre Lage innerhalb ihres übergeordneten Behälters festsetzt. Beispiele sind bekannte Java Layouts, wie das `BorderLayout` oder das `FloatLayout`. Die Tabelle 3.1 zeigt die eingangs besprochenen Grafikschichten, die sich unterhalb von GEF befinden.

Schicht	**Aufgaben**
GEF	Benutzerinteraktionen Abbildung von Model zur Sicht Integration in die Workbench
Draw2D	Renderung Layout Skalierung
SWT Canvas	Nativer (SWT) Layer

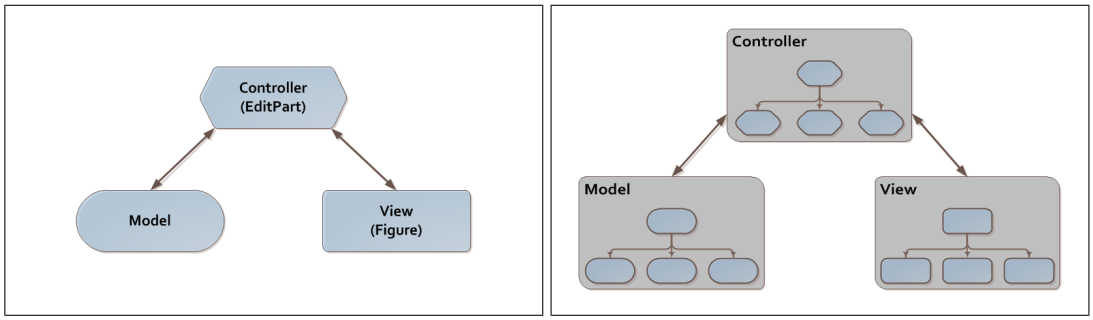
Tabelle 3.1 – Grafikschichten auf die GEF aufbaut und ihr Aufgaben

3.2 Architektur

GEF kombiniert eine Vielzahl an Entwurfsmuster aus der objektorientierten Programmierung, die eine einheitlich Struktur, was die Trennung einzelner Programmelemente und ihrer Aufgaben betrifft, herstellt und somit eine übersichtliche, leicht erweiterbare Grundarchitektur gewährleistet.

3.2.1 Model-View-Controller

Das MVC Architekturmuster spaltet die Grundaufgaben eines Softwaresystems in drei Einheiten. Erstens das Model, was für GEF eine beliebige Datenbasis sein kann. Einzige Bedingung ist, dass das Model in der Lage sein muss seine Zustandsveränderungen mitteilen zu können, sprich, dass man sich als Listener an dem Model anmelden kann, man spricht dann auch von einem Beobachter, der das Model beobachtet. Die zweite abgeschlossene Einheit ist die Sicht (View). Sie besteht ausschließlich aus den angezeigten Draw2D Figuren und enthält sonst keine weitere Anwendugslogik. Der Controller bildet eine Zwischenschicht zwischen den ersten beiden Einheiten. In GEF wird eine Controller-Klassen als `EditPart` bezeichnet. Sie ist einerseits für das Erzeugen der Figuren, die den jeweilige Zustand des Modelobjekts repräsentieren zuständig, als auch für die Entgegennahme von Benutzeraktionen (Requests), die über eine `EditPolicy` per Command-Objekt (siehe nächstes Entwurfsmuster) an das Model weitergegeben werden und somit dieses entsprechend verändert. Diese Dreiteilung wird natürlich auch auf eine Hierarchie, wie sie im Draw2D Abschnitt erwähnt wurde, abgebildet, wie die Abbildungen 3.3 a und b zeigen.



(a) MVC Einheiten (b) MVC Hierarchie

Abbildung 3.3 – MVC Architektur

3.2.2 Command

Das Command Entwurfsmuster ist ein Verhaltensmuster, bei dem in GEF anstatt der direkten Manipulierung des Models an dessen Schnittstellen ein Command-Objekt erzeugt wird, das die konkrete Sequenz von Aufrufen der Methoden des Models mit ihren Parametern und das Zielobjekt selbst enthält. Dieses Command-Objekt kann von beliebigen Aufrufern (`EditParts` bzw. `EditPolicies`) benutzt werden und gewährleistet somit die Wiederverwendbarkeit und die Möglichkeit einer leichten Erweiterung oder Veränderung von Befehlssequenzen an einer zentralen Stelle für eine bestimmte Aktion. Ein weiterer Vorteil dieses Entwurfsmusters ist es, dass die Command-Objekte gespeichert werden können und somit eine Undo/Redo-Funktionalität umgesetzt werden kann.

3.2.3 weitere Entwurfsmuster

Wie eingangs erwähnt benutzt GEF eine ganze Menge an Entwurfsmustern. Weitere sind Chain of Responsibility, State, Abstract Factory und Factory Method. Die genaue Erläuterung all dieser Muster ist nicht Gegenstand dieses Projektberichts und wird daher hier auch nicht weiter verfolgt.

Für nähere Informationen zu allen Entwurfsmustern wird auf die Standardliteratur zu Entwurfsmustern von Erich Gamma et al. [3], [4] oder Eric Freeman et al. [2] verwiesen.

3.3 Features

Nachdem zuvor eher auf die interne Architektur von GEF ein näherer Blick geworfen wurde, werden in diesem Abschnitt, auf einer höheren Ebene, Features zusammengetragen, die einem das Leben als Editorentwickler erleichtern sollen. Für die Umsetzung stellt GEF eine Vielzahl an vorgesehenen Klassen und Konzepte zu deren Anwendung zur Verfügung. Eine genaue Auflistung würde hier den Rahmen sprengen, also bleibt der Verweis für nähere Informationen auf die GEF Dokumentation [12]. Ein einführendes Tutorial für das Einsetzen vieler Features von GEF ist in dem Blog von Jean-Charles Mammana et al. [10] zu finden.

Die nachfolgende Tabelle 3.2 listet Features auf, die nach Abschluss des Projekts in dem Animationswerkzeug implementiert und einsatzbereit sind und auf dessen Umsetzung teilweise im nächsten Kapitel genauer eingegangen wird. Auf der anderen Seite sind weitere Features aufgelistet, die keine Anwendung in dem Animationswerkzeug fanden, aber als wichtige Komponenten in anderen Anwendungskontexten gelten.

implementierte Features	**weitere Features**
-Erstellen & Löschen von Figuren	-Cut & Paste
-Selektieren & Skalieren von Figuren	-Connections (Verbindungen)
-Drag & Drop	-Miniaturansicht
-Palette (Werkzeugleiste)	-Zoom
-Kontextmenu	-Animationen
-Property Sheet (Eigenschaftssicht)	-Shortcuts
-Grid (Raster)	-Drucken
	-Ruler & Guides (Lineale)
	-TreeView (Baumansicht)

Tabelle 3.2 – Featureübersicht von GEF

4 GEF im Einsatz

In diesem Kapitel werden nun ausgewählte Teilkomponenten des im Laufe des Projekts entwickelten Animationswerkzeugs näher unter die Lupe genommen. Dabei liegt der Schwerpunkt in der Umsetzung mittels GEF.

4.1 Die Architektur

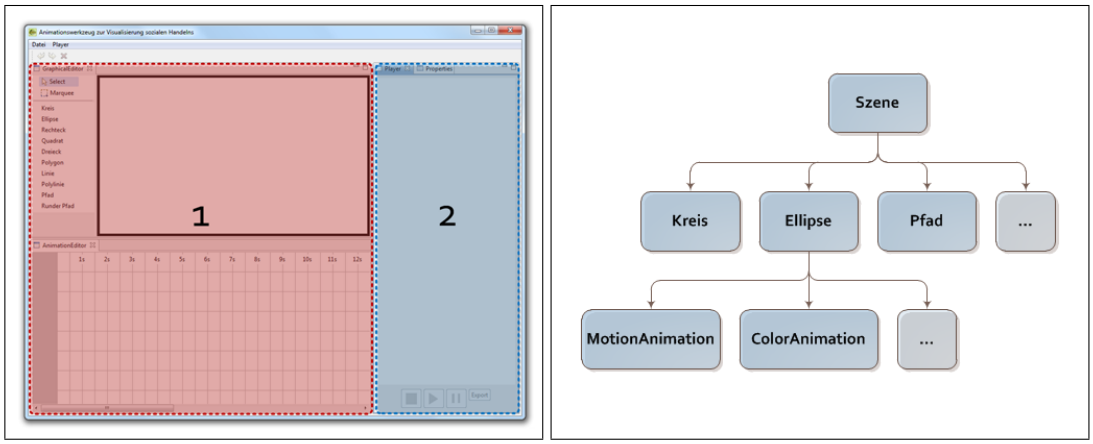
Die Grundstruktur aus Kapitel 2.3, in der das Animationswerkzeug in 4 Module aufgeteilt wird, dient als Ansatz für die Umsetzung in GEF. Die Basis für die Entwicklung des Animationswerkzeugs bildet ein RCP Eclipse Plug-in, das in zwei Hauptbereiche aufgeteilt ist. Auf der linken Seite (Bereich 1) die beiden Editoren innerhalb einer Multieditor Workbench `GraphicalEditor` und `AnimationEditor`, diese entsprechen, dem Namen leicht zu entnehmen, dem Grafikeditor und dem Animationseditor. Auf der rechten Seite (Bereich 2) eine View namens `Player`, auch hier wurde der Name aus der Anforderung beibehalten und eine View namens Properties, die zum Start der Anwendung nur als Tab zu erkennen ist, aber jederzeit den Platz in dem Bereich des Players einnehmen kann. Der Property Sheet taucht nicht explizit in den Anforderungen auf, sondern verrichtet eher eine Teilaufgabe für die Editoren und kann zu diesen gezählt werden. Er ist zuständig für das Anzeigen und Ändern von Eigenschaften des ausgewählten Objekts in dem jeweiligen Editor.

Die Abbildung 4.1 a zeigt die Aufteilung der Anwendung im Startzustand. Wie in jedem RCP Eclipse Plug-in ist es natürlich auch hier möglich, während des Betriebs Tabs zu verschieben oder zu skalieren. Die Editoren und Views werden als Extensions dem Plug-in bekannt gemacht.

Um auf die von GEF geforderte MVC Architektur zurückzukommen, sind die Editoren einerseits für das Verarbeiten von Benutzeraktionen zuständig. Sie halten also die Controller-Klassen (`EditParts`). Außerdem sind sie zuständig für die Sicht auf das Model, das heißt, sie zeigen in ihrem Viewer die zugehörigen Draw2D-Figuren. Damit haben zwei Elemente des Entwurfsmusters ihren Platz gefunden, bleibt nur noch das dritte, das Model.

Der Aufbau des Models ist an die Hierarchie eines SVG-Dokuments angelehnt, was bedeutet, dass es ein Wurzelobjekt namens Szene gibt, dem die grafischen Objekte als Kinder zugeordnet und diesen wiederum die Animationen als Kinder zugewiesen sind. Abbildung 4.1 b stellt diesen Sachverhalt schematisch dar. Die technische Umsetzung des Models und die damit verbundene Verknüpfung mit dem SVG-Dokument, hat jedoch eher weniger mit GEF zu tun, also wird hier auch nicht ausführlicher darauf eingegangen. Die genaue Spezifikation zu dem Model, das die geometrischen Objekte wiederspiegelt ist in dem Projektbericht von Tobias Staron zu finden [17] und die Modelspezifikation zu den Animationen beschreibt der Projektbericht von Stefan Schäfermeier [16].

Die Entwicklung des Players mit seiner Abspiel-, Speicher- und Exportfunktion für das SVG-Dokument, fand ebenfalls eher abseits der grundlegenden GEF Architektur statt und wird näher in dem Projektbericht von Christian Donocik spezifiziert.



(a) Hauptbereiche der RCP-Anwendung (b) Modelstruktur

Abbildung 4.1 – Grundstruktur des Animationswerkzeugs

4.1.1 Zwei Editoren ein Model

Normalerweise besteht ein in GEF entwickelter Editor, wie zum Beispiel ein UML-Editor, aus einer Editorumgebung, in dem ein Model grafisch bearbeitet werden kann. Das hat zur Folge, dass es zu jedem Modelobjekt eine Figur und den entsprechend zwischengeschalteten `EditPart` gibt. Das Animationswerkzeug sieht jedoch zwei Editoren vor, die beide mit den gleichen Modeldaten arbeiten und diese auch noch mit unterschiedlich aussehenden Figuren darstellen sollen. Daraus ergibt sich die Anforderung, dass es zu jedem Modelobjekt zwei unterschiedliche `EditParts` geben muss, die jeweils eine eigenständige Figur erzeugen müssen.

Um diesen Anforderungen gerecht werden zu können, besitzen beide Editoren einen gemeinsamen `MultiEditorInput`, der das Model hält und ihnen somit ermöglicht, auf dem gleichen Model arbeiten zu können. Des Weiteren gibt es eine `MultiEditDomain`, die beide Editoren benutzen und es somit einen einzelnen `CommandStack` für die gesamte Anwendung gibt, damit eine editorübergreifende und damit konsistente Undo-Funktionalität gegeben ist. Jeder der beiden Editoren besitzt eine eigene `EditPartFactory`, die aus dem Model für jeden Editor spezifische `EditParts` generiert, die wiederum die unterschiedlichen Figuren erzeugen.

Ein `Circle` Objekt im Model wird zum Beispiel für den `GraphicalEditor` zum `GCircleEditPart` der eine `CircleFigure` erstellt, im `AnimationEditor` wird aus dem `Circle` Objekt ein `ACircleEditPart` und die daraus generierte `ACircleFigure`.

Abbildung 4.2. zeigt den Aufbau der entwickelten MultiEditor Workbench, in der die `EditpartFactories` beider Editoren das Model als Input erhalten, die entsprechenden `EditParts` erzeugen und diese dann in der View des Editors als Draw2D Figuren gezeichnet werden. Dieses Vorgehen erfüllt die Anforderungen an das Animationswerkzeug, bringt aber auch einen erheblichen Overhead mit sich, der dadurch verursacht wird, dass die Mehrheit der `EditPart`- und `Figure`-Klassen für jedes Modelobjekt doppelt vorhanden sein müssen.

4.1.2 Requests

Requests werden von Benutzeraktionen ausgelöst, wie zum Beispiel das Erstellen eines neuen Kreises aus der Palette, die Veränderung des Layouts einer Figur, sprich das Verschieben oder Skalieren dieser innerhalb eines Editors, oder auch das Löschen einer Figur über das Kontextmenü bzw. Hauptmenü.

Abbildung 4.2 zeigt mittels gestrichelter Pfeile die Abarbeitung eines solchen Requests, hierbei wird als Szenario die Vergrößerung eines Vierecks im Grafikeditor gewählt. Der Ablauf anderer Requests erfolgt quasi analog zu dem Beispiel, nur das sich entsprechend die Akteure ändern.

Der Ausgangspunkt ist die Vergrößerung der `BoundingBox` des Vierecks innerhalb des `GraphicalEditors` mittels der Maus (**1**). Diese Funktionalität wird, wie bereits erwähnt von GEF standardmäßig mitgebracht und aus dieser Aktion entsteht das Request für die Änderung des Layouts des Vierecks. Dieses Request wird von der zuständigen `EditPolicy`, die in dem Viereck übergeordneten `SceneEditPart` installiert ist, aufgenommen (**2**). Der `EditPolicy` wird der `EditPart` zu der veränderten Figur, sowie das neue Layout als `Rectangle` Constraint mitgegeben. Aus diesen Informationen erstellt die `EditPolicy` ein Command-Objekt, das die spezifischen Methodenaufrufe zum Ändern des Models besitzt und nun ausgeführt wird. Dabei wird es an oberster Stelle auf den `CommandStack` gepackt (**3**) und verändert explizit das Layout auf der Modelebene (**4**). Das Model-Objekt vom Typ `Rectangle` feuert nun ein `PropertyChange` Event mit der Information, dass sich sein Layout geändert hat. Der `GRectangleEditPart`, der als Listener an seinem Model angemeldet ist, erhält diese Nachricht (**5**) und aktualisiert seine Figur mittels der Methode `refreshVisuals()` (**6**).

Dieser im Animationswerkzeug umgesetzte Ablauf ist absolut GEF konform und bietet somit auch alle Vorteile dieses Konzepts, wie die Trennung von Aufgabenbereichen und die daraus resultierende Möglichkeit der Austauschbarkeit oder auch Erweiterbarkeit von einzelnen Komponenten.

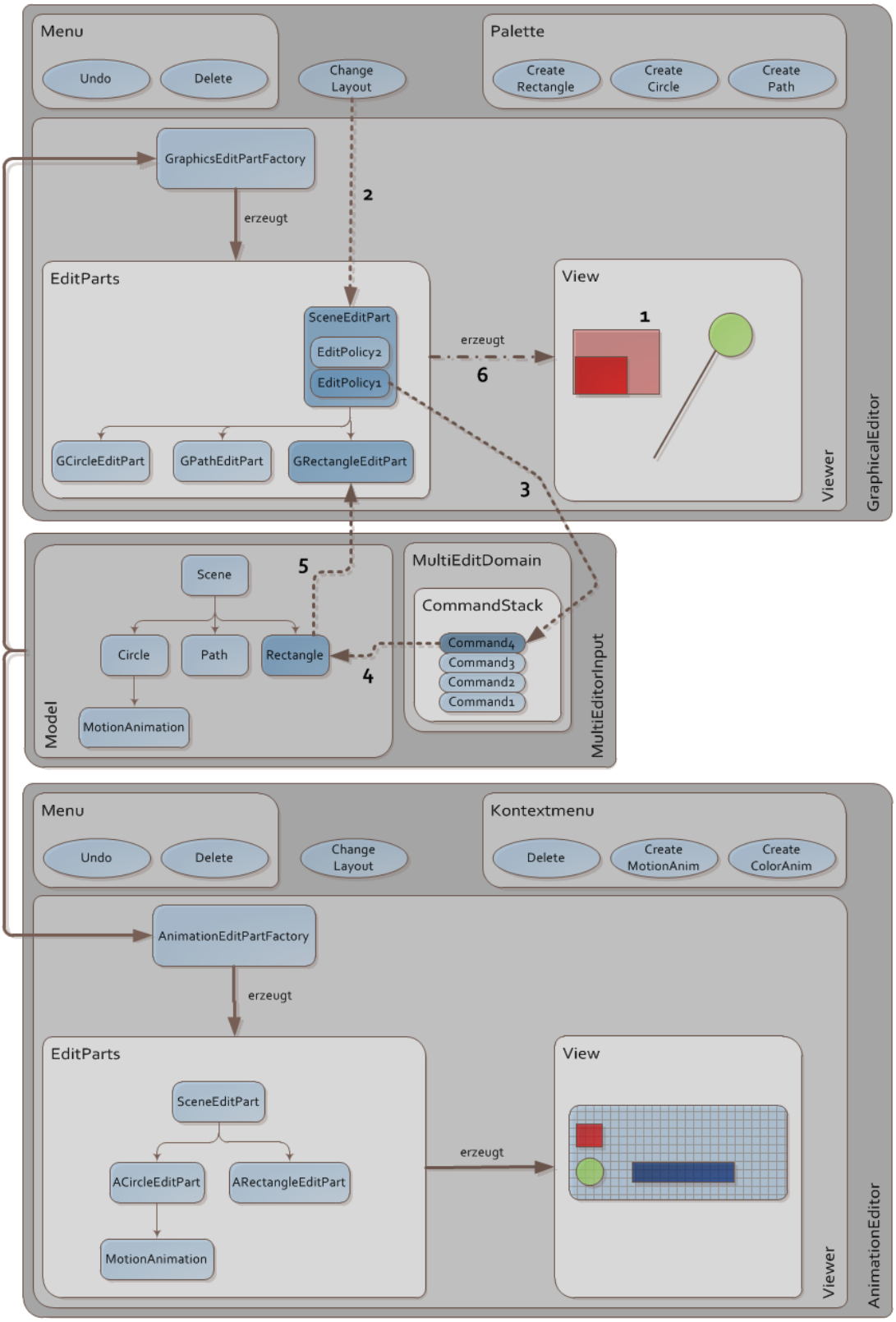


Abbildung 4.2 – Aufbau der Multieditor Workbench

4.2 Der Animationseditor unter der Lupe

Nachdem die Grundarchitektur der Multieditor Workbench im vorigen Abschnitt beschrieben wurde, wird dieser Abschnitt einem der beiden Editoren, den Animationseditor, etwas näher beleuchten und den Einsatz spezieller von GEF zur Verfügung gestellten Features darstellen. Umsetzungen von Features, die im Grafikeditor Anwendung finden, wie zum Beispiel die Palette, sind in dem Projektbericht von Tobias Staron [17] näher beschrieben.

4.2.1 Die Hierarchie der Figuren

Das Model ist hierarchisch aufgebaut und das bedeutet auch, dass die generierten `EditParts`, sowie die mit ihnen erzeugten Draw2D Figuren diese Hierarchie aufweisen. Konkret heißt das für den Animationseditor, dass die Figur des Wurzelobjekts, nämlich die Szene, den gesamten gezeichneten Bereich darstellt und nur innerhalb dieser Figur ihre Kinder, die grafischen Objekte, wie Kreise oder Vierecke gezeichnet werden. Des weiteren sind Animationen jeglicher Art Kinder von grafischen Objekten, was die Zeichnung dieser mit der gleichen Einschränkungen belegt.

Das führt dazu, dass die Figuren der grafischen Objekte, obwohl das sichtbare Objekt nur eine Abmessung von 30x30 Pixeln aufweist, einen transparenten Bereich, der die Breite der kompletten Szene hat, haben müssen. Nur so ist es möglich Animation an einer beliebigen Stelle der Szene, das heißt, zu einem beliebigen Zeitpunkt zu zeichnen bzw. sie dorthin verschieben zu können und dabei die vom Model geforderte Hierarchie beizubehalten. Bei zeitlich sehr langen Szenen resultiert daraus auch, dass es sehr breite Figuren gibt, die im Editor gehalten werden müssen, wie die `BoundingBox` einer `ACircleFigure` in Abbildung 4.3 bei einer 31 sekündigen Szene andeuten lässt.

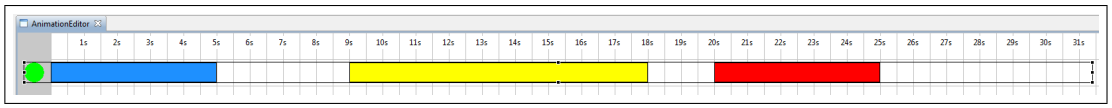


Abbildung 4.3 – Dimension einer Kreisfigur im Animationseditor

4.2.2 Das Grid

Die Sicht des Animationseditors spiegelt eine Zeitleiste wieder, die auf der horizontalen Achse in zeitliche Abschnitte und auf der vertikalen Achse in eine Liste der grafischen Objekte unterteilt ist. Um eine bessere Ausrichtung der Animationen in der gesamte Szene zu erhalten bietet sich eine Rasterunterteilung des gesamten Editors an.

Dieses ist ein von GEF vorgesehenes Feature, das sich Grid nennt. Es ist möglich den Viewer eines Editors in ein beliebig feines Raster zu kleiden, was mit einem sehr geringen Programmieraufwand möglich ist. Es benötigt lediglich das Setzen von drei

Eigenschaften innerhalb der `initializeGraphicalViewer()` Methode des jeweiligen Editors, was aufgrund der Kürze, nachfolgend beispielhaft aufgelistet ist.

```
1 viewer.setProperty(SnapToGrid.PROPERTY_GRID_ENABLED, true);
2 viewer.setProperty(SnapToGrid.PROPERTY_GRID_SPACING,
3                    new Dimension(25,40));
4 viewer.setProperty(SnapToGrid.PROPERTY_GRID_VISIBLE, true);
```

4.2.3 Das Kontextmenü

Wie in dem Abschnitt zu dem Verarbeiten von Benutzerinteraktionen erläutert, geschieht dieses mit Hilfe von Requests. Diese können per direkter Manipulation von Figuren in einem Editor, per Hauptmenü bzw. Property Sheet für die jeweils selektierte Figur oder aber auch im Animationseditor per Kontextmenu eingeleitet werden.

Für die Umsetzung des Kontexmenüs benötigt es den `AnimationEditorContextMenuProvider`, in dem die einzelnen Menüpunkte definiert und mit den aufzurufenden Aktionen verknüpft werden. Dieser Provider wird dem `AnimationEditor` bekannt gemacht und darüber hinaus werden die spezifischen `Actions` der `SelectionAction` Liste des Editors zugefügt. Über die `Actions` werden, wie gewohnt Request generiert und ausgeführt.

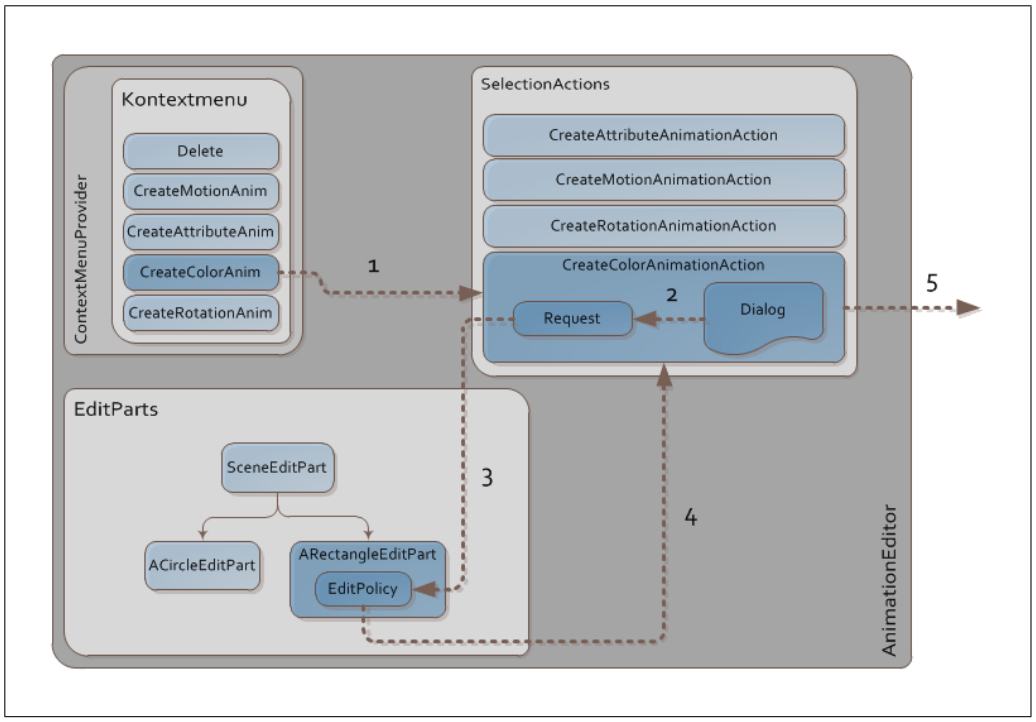


Abbildung 4.4 – Requestausführung über das Kontextmenü

Dieser Ablauf wird in der Abbildung 4.4 dargestellt. Bei dem selektieren eines grafischen Objekts und dem anschließenden Klicken auf einen Menüpunkt des Kontextmenüs wird die dazugehörige `Action` ausgeführt, hier die `CreateColorAnimationAction` (**1**). In ihr wird ein Request mit den nötigen Informationen erstellt.

Bei bestimmten `Actions`, wie es auch hier der Fall ist, ist ein Dialog zwischengeschaltet, der benötigte Informationen abfragt. Diese gesammelten Parameter werden ebenfalls mit in das Request-Objekt gepackt (**2**). Es gibt für verschiedene Animationen dementsprechende Dialoge, eine genaue Auflistung und die technische Umsetzung dieser erläutert der Projektbericht von Stefan Schäfermeier [16].

Das fertig geschnürte Request Paket wird nun an den selektierten `EditPart` geschickt (**3**) und in diesem kümmert sich die entsprechende `EditPolicy` um die Erstellung des Command-Objekts. Das Command-Objekt wird der Action zurückgegeben (**4**) und anschließend ausgeführt. Die Ausführung erfolgt, wie bereits bekannt, über den `CommandStack` zum Model (**5**). Es handelt sich hierbei um ein `CreateRequest`, das bedeutet, dass das Command-objekt eine neue `ColorAnimation` im Model anlegt. Der Weg zurück erfolgt über die `EditPartFactory`, die zu diesem neuen Objekt einen neuen `EditPart` erstellen muss und dieser kann dann eine neue Figur zeichnen.

4.2.4 Editorübergreifende Selektion

Das zuvor beschriebene Kontextmenu enthält auch die `Action` zum Erstellen einer `MotionAnimation`. Eine Bewegungsanimation braucht neben dem selektierten grafischen Objekt auch einen Pfad, an dem die Bewegung entlangläuft. In dem `AnimationEditor` werden Pfade nicht gezeichnet, nun bleibt die Frage, wie man den Pfad der Animation hinzufügt.

Das umgesetzte Vorgehen für das Erhalten der beiden Objekte, dem grafischen- und dem Pfad-Objekt, sieht vor, dass der im `GraphicalEditor` selektierte Pfad, dem im `AnimationEditor` ausgewähltem grafischen Objekt als Bewegungspfad für die `MotionAnimation` zugeordnet wird.

Glücklicherweise bietet GEF, durch die gewählte Architektur des Animationswerkzeugs mit einer Multieditor Umgebung, die Möglichkeit auf die gemeinsame `MultiEditDomain` zuzugreifen, der beide Editoren bekannt sind. Dadurch ist es möglich, an den im `GraphicalEditor` selektierten `EditPart` auch aus dem `AnimationEditor` heranzukommen und somit können die Objekte aus beiden Editoren in der `CreateMotionAnimationAction` als Parameter für das Request herangezogen werden.

4.2.5 Undo- & Redofunktionalität

Es wurde bereits mehrmals erwähnt das GEF automatisch eine Undo- und Redofunktionalität vorsieht. Die `EditDomain` hält einen so genannten `CommandStack` auf dem die ausgeführten Command-Objekte abgelegt werden und bei Bedarf (Undo) wieder herunter genommen werden können. Jedes Command-Objekt besitzt eine `undo()` Methode, die zu diesem Zeitpunkt ausgeführt wird und die Änderungen am Model rückgängig macht. Bei einem Redo wird das vorige Command-Objekt einfach wieder normal ausgeführt und auf den Stapel zurück gepackt.

Das Animationswerkzeug besitzt eine `MultiEditDomain`, die beide Editoren benutzen. Das bedeutet, es gibt auch nur einen gemeinsamen `CommandStack`, auf dem jegliche Command-Objekte gemischt vom Grafikeditor und Animationeditor gesammelt werden. Diese Vorgehen ist nötig, damit es zu keinen inkonsistenten Modelveränderung kommen kann. Hätten beide Editoren jeweils einen eigenen `CommandStack` zur Verfügung, könnte es zum Beispiel zu einer Situation kommen, indem ein grafisches Objekt (Kreis) und ein Pfad im Grafikeditor erstellt werden, anschließend eine Bewegungsanimation zu dem Kreis im Animationeditor hinzugefügt und nun ein Undo im Grafikeditor durchgeführt wird. Das hätte zur Folge, dass es im Model einen Kreis mit einer dazugehörigen Bewegungsanimation gäbe, die aber auf einen Pfad referenziert, den es dann gar nicht mehr gibt.

Aus diesem Grund ist eine gemeinsame Undofunktionalität für beide Editoren notwendig und es wird immer die zuletzt ausgeführte Aktion rückgängig gemacht, egal in welchem Editor sie stattgefunden hat.

4.2.6 PropertySheet

Der Property Sheet ist eine standard View, die die Modeleigenschaften, wie zum Beispiel die Größe oder Farbe, einer selektieren Figur als Tabelle mit dem Namen und dem zugehörigen Werde anzeigt. Es ist auch möglich diese Werte direkt in der View zu verändern. In dem Animationwerkzeug ist es möglich alle wichtigen Modelobjekte über den Property Sheet zu verändern, sogar die spezifischen Dialoge für die Animationen können über ihn aufgerufen und die eingegebenen Parameter übernommen werden.

Umgesetzt wird diese Funktionalität durch eine `SceneNodePropertySource` Klasse, die als Vermittler zwischen dem im Model festgelegten Eigenschaften und dem Property Sheet fungiert. Die genaue technische Umsetzung wird unter anderem in dem Projektbericht von Tobias Staron dokumentiert [17].

4.3 Was GEF nicht kann

Beinahe das gesamte Portfolio an Anforderungen, die an das Animationswerkzeug gestellt wurden, konnten mittels GEF mehr oder weniger umfangreich umgesetzt werden. Es gibt jedoch auch Bereiche, die zu spezifisch in ihrer Anwendung und ihrem Nutzen sind, als dass sie GEF für einen Standardeditor vorsieht.

Dazu gehören die Dialoge für das setzen von Animationsparametern, wie das Manipulieren einer Bézierkurve für die Geschwindigkeitsveränderung. Diese Dialoge mussten außerhalb des Frameworks entwickelt werden, wie der Projektbericht von Stefan Schäfermeier [16] erläutert.

Eine weitere sehr projektspezifische Anforderung war das Abspielen des SVG-Dokuments und dessen Export in weitere Videoformate. Dazu sind spezielle Bibliotheken notwendig, die natürlich keinen Bezug zu GEF haben. Der Projektbericht von Christian Donocik [15] befasst sich mit den benötigten Java-Bibliotheken Batik und Xuggler, mit dessen Hilfe diese Anforderungen umgesetzt werden konnten.

Das zeichnen von Pfaden über mehrere Punkte, was ebenfalls zu den Anforderungen des Animationswerkzeugs gehört, kann ebenfalls mittels der GEF Standardfunktionalität, die es vorsieht das Erstellen von Figuren über das Aufziehen ihrer `BoundingBox` zu realisieren, nicht umgesetzt werden. Für die Erfüllung dieser Anforderung gibt es ein eigens entwickeltes Multi-Selection-Tool, dessen genaue Entstehungsgeschichte in dem Projektbericht von Christian Wilms [18] dokumentiert ist.

5 Evaluation

Abschließend bleibt die Bewertung der Tauglichkeit von GEF für die Entwicklung eines Editors, wie das Animationswerkzeug.

Im allgemeinen hinterlässt GEF einen sehr positiven Eindruck, was einerseits die Klarheit seines Aufbaus und andererseits die Vielfältigkeit seiner Features, sowie die Möglichkeit der Anpassung an unterschiedliche Anwendungsbereiche angeht. Ein sehr großer Vorteil ist die Einbettung in Eclipse RCP, was dazu führt, dass sehr schnell eine lauffähige Grundanwendung zur Verfügung steht, die einen hohen Bedienkomfort besitzt und mit der bereits während früher Entwicklungsphasen getestet werden kann. Somit ist es möglich sich schnell mit den wichtigen Aufgaben zu beschäftigen, was die spezifischen Einsatzbereiche betrifft.

Ein unumgänglicher Nachteil ist jedoch die stark ansteigende Lernkurve, sobald man von dem Standard ein wenig abweicht. Trotz über zehn jährigem Bestehen dieses Eclipse Projekts ist die Dokumentation, sowie Literatur leider immer noch eher spärlich gesäht, was eine erhebliche Investition in Zeit für „learning by doing“ nach sich zieht. Diese Investion lohnt sich aber, da sich Vieles, was zuerst den Anschein hat nicht unterstützt zu werden, dann doch irgendwo versteckt ist.

Für die Nutzung von GEF im Rahmen eines Projekts, wie dem vorliegenden Fallbeispiel „Animationswerkzeug zur Visualisierung sozialen Handelns“ kann ich trotzdem eine Empfehlung aussprechen. In einem solchen Zeitrahmen müsste anderenfalls ein wesentlich größerer Anteil in die Entwicklung einer grundlegenden Architektur gesteckt werden. Das hätte zur Folge, dass das Endergebnis, was den Funktionsumfang eines Editors betrifft, wesentlich ärmer an Features ausfallen würde.

Mit Hilfe des Frameworks kann nun ein Produkt präsentiert werden, das den Grundanforderungen eines Animationswerkzeugs genüge trägt und bereits produktiv von Anwendern genutzt werden kann.

6 Literaturverzeichnis

Bücher & Zeitschriftenartikel

[1] Daum, Berthold:
Rich-Client-Entwicklung mit Eclipse 3.3.
dpunkt.verlag, 3. Auflage, 2008. 978-38986450346.

[2] Freeman, Eric, Elisabeth Freeman, Bert Bates und Kathy Sierra:
Head First Design Patterns.
O'Reilly Media, 1. Auflage, 2004. 978-0596007126.

[3] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides:
Design Patterns. Elements of Reusable Object-Oriented Software.
Addison-Wesley, 38. Auflage, 2010. 978-0201633610.

[4] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides:
Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.
Addison-Wesley, neue Auflage, 2010. 978-3827330437.

[5] Heider, Fritz und Marianne Simmel:
An Experimental Study of Apparent Behavior.
The American Journal of Psychology, 57(2):243–259, April 1944.

[6] Mori, Masahiro:
Bukimi no tani: The Uncanny Valley.
Energy, 7(4):33–35, 1970. übersetzt von Karl F. MacDorman und Takashi Minato.

[7] Pacchierotti, Elena, Henrik I Christensen und Patric Jensfelt:
Human-Robot Embodied Interaction in Hallway Settings: a Pilot User Study. In: *In Proceedings of the IEEE International Workshop on Robots and Human Interactive Communication*, Seiten 164–171, 2005.

[8] Scholl, Brian J. und Patrice D. Tremoulet:
Perceptual causality and animacy.
Trends in Cognitive Sciences, 4(8):299–308, August 2000.

Webseiten & Tutorials

[9] Ebert, Ralf:
Eclipse RCP Buch.
`http://www.ralfebert.de/rcpbuch/`. Stand: 14.04.2011.

[10] Mammana, Jean-Charles, Romain Meson und Jonathan Gramain:
GEF (Graphical Editing Framework) Tutorial, Version 1.1.
`http://www.psykokwak.com/blog/images/gef/GEF_Tutorial.pdf`, 2008.

[11] The Eclipse Foundation:
Graphical Editing Framework.
`http://www.eclipse.org/gef/`. Stand: 14.04.2011.

[12] The Eclipse Foundation:
GEF/Reference Documentation.
`http://wiki.eclipse.org/GEF/Reference_Documentation`. Stand: 14.04.2011.

[13] World Wide Web Consortium:
Scalable Vector Graphics (SVG) 1.1 (Second Edition).
`http://www.w3.org/TR/SVG11/`. Stand: 14.04.2011.

[14] World Wide Web Consortium:
Synchronized Multimedia.
`http://www.w3.org/AudioVideo/`. Stand: 14.04.2011.

Projektberichte

[15] Donocik, Christian:
Ein Animationswerkzeug zur Visualisierung sozialer Handlungen.
Projektbericht, Universität Hamburg, 2011.

[16] Schäfermeier, Stefan:
Animationseditor: Anforderungen & Lösungen.
Projektbericht, Universität Hamburg, 2011.

[17] Staron, Tobias:
Der Graphikeditor und GEF: die Umsetzung, das Modell und der Eigenschafteneditor. Projektbericht, Universität Hamburg, 2011.

[18] Wilms, Christian:
Ein Multi-Selection-Tool für GEF.
Projektbericht, Universität Hamburg, 2011.