Story Driven Web Applications

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

im Fachgebiet Software Engineering
Prof. Dr. Albert Zündorf
Fachbereich Elektrotechnik / Informatik
der Universität Kassel

vorgelegt von
Dipl. Inf. Nina Geiger MSc.
Disputation am 29.Juni 2016

Erklärung

„Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlich-materiellen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich hierfür nicht die Hilfe eines Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden."

Nina Geiger

Für meine Familie

Große Zeit ist's immer nur, wenn's beinahe schiefgeht, wenn man jeden Augenblick fürchten muß: jetzt ist alles vorbei. Da zeigt sich's. Courage ist gut, aber Ausdauer ist besser. Ausdauer, das ist die Hauptsache.
- *Theodor Fontane*

Zusammenfassung

Kern der vorliegenden Arbeit ist die Modellierung komplexer Webapplikationen mit dem Story-Driven-Modeling Ansatz. Ziel ist es hierbei, die komplette Applikation allein durch die Spezifikation von Modellen zu entwickeln. Das händische Erstellen von Quelltext ist nicht notwendig.

Die vorliegende Arbeit zeigt sowohl den Forschungsweg, der die angestrebte Modellierung von Webapplikationen ermöglicht, als auch die resultierenden Ergebnisse auf. Zur Unterstützung des Entwicklungsprozesses wird weiterhin ein modellgetriebener Softwareentwicklungsprozess vorgestellt, der die Modellierung einer Webapplikation von der Aufnahme der Anforderungen, bis zur abschließenden Erzeugung des Quelltextes durch Codegenerierung aus den spezifizierten Modellen, abdeckt. Für den definierten Prozess wird ferner Werkzeugunterstützung innerhalb der Fujaba Toolsuite bereitgestellt. Im Rahmen der vorliegenden Arbeit wurde die besteheden Toolsuite hierzu um alle zur Prozessunterstützung notwendigen Werkzeuge erweitert. Darüber hinaus wurden im Rahmen der vorliegenden Arbeit die in Fujaba bestehenden Werkzeuge erweitert, um neben den klassischen Möglichkeiten zur Modellierung komplexer Java-Applikationen auch die Erzeugung von Webapplikationen zu ermöglichen.

Neben der genauen Beschreibung des Entwicklungsprozesses werden im Rahmen dieser Arbeit die entstehenden Webapplikationen mit ihren spezifischen Eigenschaften genau beschrieben. Zur Erzeugung dieser Applikationen wird neben dem Entwicklungsprozess die Diagrammart der Workflowdiagramme eingeführt und beschrieben. Diese Diagramme dienen der Abbildung des intendierten Userworkflows der Applikation im Rahmen der Anforderungsanalyse und stellen im weiteren Entwicklungsverlauf ein dediziertes Entwicklungsartefakt dar. Basierend auf den Workflowdiagrammen werden sowohl die grafische Benutzerschnittstelle der Webapplikation beschrieben, als auch ein Laufzeitsystem initialisiert, welches basierend auf den im Workflowdiagramm abgebildeten Abläufen die Anwendung steuert. Dieses Laufzeitsystem wurde im Rahmen der vorliegenden Arbeit entwickelt und in der Prozessunterstützung verankert.

Alle notwendigen Änderungen und Anpassungen und Erweiterungen an bereits bestehenden Teilen der Fujaba Toolsuite werden unter dem Aspekt der Erstellung clientseitiger Datenmodelle einer Webapplikation genau beschrieben und in Verbindung mit den zu erfüllenden Voraussetzungen erläutert. In diesem Zusammenhang wird ebenfalls beschrie-

ben, wie Graphtransformationen zur Umsetzung von Businesslogik auf der Clientseite einer Webapplikation eingesetzt werden können und auf welche Weise Datenmodelländerungen zwischen unterschiedlichen Clients synchronisiert werden können.

Insgesamt zeigt die vorliegende Arbeit einen Weg auf, den bestehenden Ansatz des Story-Driven-Modeling für die Erzeugung von Webapplikationen einzusetzen. Durch die im Rahmen dieser Arbeit beschriebene Herangehensweise werden hierbei gleichzeitig Webbrowser zu einer neuen Klasse von Graphersetzungs-Engines erweitert, indem Graphtransformationen innerhalb der Ajax-Engine des Browsers ausgeliefert und ausgeführt werden.

Abstract

This thesis presents the modeling of complex web applications following the story-driven-modeling approach. Thereby it shows, how it is possible to develop the whole application by specification of models. Manual creation of sourcecode is not necessary.

The thesis shows both, the research path that allows the targeted modeling of web applications, and the resulting outcomes. To support the development of targeted applications, a model-driven software development process is presented, covering the whole story from requirements engineering to final code generation. Toolsupport for the defined process is provided in terms of the Fujaba Toolsuite. This thesis therefore has extended Fujaba with tools necessary to support the process. Furthermore, existing Fujaba tools have been extended and enhanced to support traditional modeling of Java-applications as well as modeling of web applications for this thesis.

In additon to the detailed description of the modeling process, specific properties of the resulting web applications are described in detail. The new diagram type of workflowdiagrams is introduced to support generation of web applications. These diagrams are used to depict the intended user workflow of the resulting application within requirements engineering and is afterwards used as dedicated development artefact. Based on this artefact both the graphical user interface of the web application is generated as well as a runtime system is initilized. This runtime system is also developed within this thesis and strongly associated to the modeling process and tooling.

This thesis describes in detail all necessary adaptions and extensions made to the Fujaba Toolsuite - having in mind the modeling process and preconditions to be met when engineering web applications. A major point here are graph transformations being used to run describe an run businesslogic on client side. This thesis describes in detail how to model these graph transformations in terms of web applications and also how client data from different clients can be synchronized and replicated.

Overall, this thesis presents a way to use the known approach of story-driven-modeling for the generation of web applications. By doing this, webbrowsers are introduced as one new class of graph transformation engines, making it possible to run complex graph transformations inside the browsers ajax-engine.

Danksagung

Endlich - meine Dissertation liegt vor mir - und auch vor Ihnen. Ein Ereignis, an welches ich selbst zeitweise nicht mehr zu glauben wagte. Viele Stunden Forschungsarbeit, Diskussionen, Programmieraufwand und nicht zuletzt Dokumentation liegen hinter mir. Viele Menschen haben direkten oder indirekten Anteil daran, dass ich jetzt glücklich vor einem fertigen Dokument sitze. Ich möchte hier die Gelgenheit nutzen, ihnen allen meinen aufrichtigen Dank auszusprechen.

Zunächst geht mein Dank an meinen Doktorvater Prof. Dr. Albert Zündorf. Albert, Du hast mich nach einem irreführenden Jahr in den Tiefen der Computergrafik mit offenen Armen an Deinem Fachgebiet aufgenommen und mir durch die Arbeit am FAST EU-Projekt die Möglichkeit und auch die Grundlage für die Idee gegeben, mich mit der Modellierung von Webapplikationen zu beschäftigen. Du hast mir stets mit Rat und Tat zur Seite gestanden und warst nie einer Diskussion neuer Ideen, und mochten sie auch noch so abgefahren sein, abgeneigt. Durch deine herzliche und offene Art und deinen immerwährenden Optimismus hast Du mir oftmals Wege durch die Untiefen der Softwaretchnik aufgezeigt. Auch gemeinsame Lehrveranstaltungen in meinen Jahren als wissenschaftliche Mitarbeiterin bei Dir waren immer ein Vergnügen und manches mal von großen Forschungsfragen geprägt.

Ein weiterer Dank geht an meinen Zweitgutachter Herrn Prof. Dr. Matthias Tichy für seine Bereitschaft und Zeit meine Dissertation zu lesen und zu bewerten.

Ein riesiges Dankeschön möchte ich meinen (nun mehr ehemaligen) studentischen Mitarbeitern Marcel Hahn, Christoph Eickhoff und Ingo Witzky, sowie Alexander Jahl aussprechen. Ohne euren unermüdlichen Einsatz wäre die Umsetzung meiner Ideen nicht möglich gewesen. Auch wart ihr immer bereit, meine von euch oftmals exotisch bewerteten Ideen mit mir zu diskutieren und mir so eure Sichtweisen aufzuzeigen.

Danke auch an meine Kollegen am Fachgebiet Software Engineering, Jörn Dreyer, Tobias George und Andreas Koch, sowie alle übrigen studentischen Mitarbeiter es Fachgebietes. Ihr habt jeden einzelnen Arbeitstag zu einem Erlebnis gemacht und mir ein familiäres Arbeitsumfeld geboten, in welches ich stets gern zurückkehre. Dies wurde sicherlich auch dadurch zum Ausdruck gebracht, dass ich nach Leos Geburt täglich zu einer kurzen Stippvisite vorbeigeschaut habe.

Ein besonderer Dank gilt Frau Rosemarie Biehlig. Sie standen mir immer bei allen administrativen Fragen mit Rat und Tat zur Seite und wurden oftmals zur weiblichen Verbündeten und Gesprächspartnerin in unserem männerdominierten Fachgebiet.

Ein ganz tiefer Dank geht an meinen Mann Leif. Du hast immer an mich und meine Fähigkeiten geglaubt. Selbst als ich selbst schon nicht mehr daran denken mochte, dass ich jemals mit meiner Dissertation fertig werden würde, hast Du Dich nie beirren lassen und nie Deinen Optimismus verloren. Danke für Deine Liebe und Unterstützung. Ich liebe Dich.

Vielen Dank an meine Familie, meine Eltern Rosi und Norbert und meine Schwiegereltern Marita und Hartmut. Ihr habt jederzeit bereit gestanden um mich zu unterstützen, wie immer ihr auch konntet. Viele Stunden Babysitting bei Leo, wochenlange Betreuung innerhalb der Kita-Ferien, es war jederzeit auf euch Verlass. Insbesondere noch Dir Papa, danke für die Rechtschreibkorrektur meiner Dissertation.

Astrid, Christina, Doreen, Irina ich danke euch dafür, dass ihr in meinem Chaos von Hausbau, Umzug, Neuanfang in Frankfurt, Arbeit, Haushalt, Kindererziehung und Dissertation stets da wart und mir die Gelegenheit gegeben habt, einfach mal Dampf abzulassen oder über völlig andere Themen zu sprechen. Auch tatkräftige Unterstützung im täglichen Leben war von euch jederzeit gegeben.

Last but not least geht ein riesiges Dankeschön an das gesamte Team der Firma Yatta Solutions. Ihr habt mir nicht nur eine genauso familiäre Arbeitsatmosphäre geboten, wie ich es aus der Uni gewohnt war, ihr habt es mir erst ermöglicht, meine Dissertation am Ende doch noch zu einem guten Ende zu bringen. Insbesondere danke Johannes, Du hast immer an mich geglaubt, auch wenn manche Diskussion haarig war; danke Andi fürs Lesen, Korrigieren und Deine Anmerkungen; danke Falk für viele ermunternde Gespräche; danke Sophie für authoriellen Rat zu jeder Zeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Internet hat sich im Laufe der vergangenen Jahre gewandelt. Vom usprünglichen Netzwerk untereinander verknüpfter statischer Webseiten ist im heutigen Internet nicht viel geblieben. Multimedialie Inhalte und komplexe Applikationen im Webbrowser sind gängige Praxis. Ohne eine rasante Weiterentwicklung der technologischen Grundbausteine, aus welchen das Internet sich zusammensetzt, wäre diese Wandlung nicht möglich gewesen. Vom Anzeigegerät für statisch repräsentierte Inhalte, dargesellt als HTML-Markup, haben sich Webbrowser zu einer Ausführungsumgebung für komplexe Applikationen entwickelt. Dieser Wandel - und die damit einhergehende höhere Komplexität der angezeigten Inhalte bzw. Anwendungen - macht ein Umdenken auch in der Entwicklung von Inhalten für das Internet notwendig. Konnten statische Inhalte noch weitgehend ohne Programmierkenntnisse durch Spezifizierung des HTML-Markup erstellt werden, so trifft dies für gängige dynamische Applikationen im Internet nicht länger zu. Die stete Weiterentwicklung des Internet und der dazugehörigen Technologien wird auch von großen Konzernen, wie beispielsweise Google, weiter vorangetrieben. So bietet der Internetkonzern auch im Bereich komplexer Applikationen mit den Google Docs [Goo] ein Angebot, welches als Konkurrenz zu etablierten Office Paketen angesehen werden kann, jedoch komplett innerhalb des Webbrowsers lauffähig ist.

Technisch basieren heutige Webbrowser noch immer auf der Anzeige von HTML Inhalten. Allerdings ist zusätzlich mit der sogenannten Ajax-Engine [G+05] ein Softwareanteil in aktuellen Webbrowsern enthalten, der es ermöglicht, Javascript Befehle und daraus resultierende Berechnungen direkt innerhalb des Browsers auszuführen. Zusätzlich ist die Ajax Engine dazu in der Lage, komplexe Datenstrukturen in Form von Graphen zu berechnen und als Grundlage weiterer Berechnung im Speicher zu halten. Die somit komplexeren Möglichkeiten, Anwendungen für den Webbrowser zu entwickeln, machen den Bereich der Webapplikation und deren Implementierung als Forschungsbereich für das Softwareengineering interessant. Es handelt sich hier nicht länger um kleine Seiten, die mit HTML und CSS Kenntnissen ohne Programmiererfahrung entwickelt werden können, sondern zunehmend um komplexe Softwareentwicklungsprojekte, die denen aus dem Bereich der Softwareentwicklung für Desktopanwendungen ähneln. Für die Entwicklung komplexer Webapplikationen sind demnach nicht länger nur Designer, sondern auch Softwareentwickler zuständig. Der vermehrte Einsatz von JavaScript Quelltext im Bereich der komplexen Webanwendungen macht zudem Instrumente der klassischen Software-

entwicklung, wie etwa Debugging oder ausgiebiges Testen vor dem finalen Deployment, auf einem Webserver notwendig. Für komplexe Datenmodelle und Business Logik ist es zudem wünschenswert, Typsicherheit und Checks zur Compilezeit zu haben, um fehlerfreie Software entwickeln zu können. Darüber hinaus machen Integrierte Software Entwicklungsumgebungen (IDEs) den Entwicklern ihre Aufgaben leichter. Hier werden Codecompletion, Refactorings sowie Unit Testing und Coverage Tools unterstützt. Alles Anforderungen, die in der klassischen Softwareentwicklung gang und gäbe sind.

Einen Schritt weiter gehen die Ideen des Model-Driven-Development [BBG+05], die durch die Einführung von Modellen in den Softwareentwicklungsprozess eine weitere Abstraktionsebene bieten. Die Entwicklung von Software mit Modellen, insbesondere auch die systematische Entwicklung von Applikationslogik unter diesem Aspekt, ist ein Kern des Story-Driven-Modeling Ansatzes [Zün01]. Zur Unterstützung des Story-Driven-Modeling wurde die Fujaba Toolsuite (Fujaba) [wwwe] entwickelt.

Die Arbeitsgruppe von Prof. Schäfer an der Universität Paderborn [Sch] und das Fachgebiet von Prof. Zündorf an der Universität Kassel [SE] haben grundlegende Werkzeuge von Fujaba wie UML Klassendiagramm Editoren, Code Generatoren, Werkzeuge für Graphersetzungsregeln (Story Pattern), Werkzeuge für UML Activity Diagramme zur Modellierung von Kontrollfluss (Story Diagramme), Werkzeuge zur Modellierung von Objektmodell basierten Tests (Story Boards) und Statechart Editoren entwickelt. Aus Kassel kamen dann insbesondere auch noch Unterstützung für das graphische Debuggen und ein UML Objektdiagramm basierter Browser für Laufzeitdatenstrukturen hinzu. In dem Fachgebiet von Prof. Schürr an der TU Darmstadt [TUD] wurden insbesondere Modell zu Modell Transformationen mit Hilfe von Triple Graph Grammatiken entwickelt. Das Fachgebiet von Prof. Giese am Hasso-Plattner-Institut Potsdam [HPI] hat einen Story-Diagramm Interpreter entwickelt und steuert Arbeiten im Bereich Model Checking bei. Der Lehrstuhl von Prof. Westfechtel an der Universität Bayreuth [Bay] entwickelte Techniken für die Versionierung und für Produktlinien mit Story Diagrammen und Fujaba. Die Gruppe von Prof. Assmann an der TU Dresden [Dre] hat einen OCL Compiler in die Fujaba Tool Suite integriert. An der University of Victora entwickelte die Forschungsgruppe um Prof. Jens Weber-Jahnke [Vic] mit Hilfe von Fujaba Helth Care Applications. Die Gruppe von Prof. Tichy an der Universität Ulm verwendet die Fujaba Tool Suite für die Entwicklung von Selbst-Adaptiven Systemen.

Die Forschung und Entwicklung von Fujaba wird auch sehr gut durch die Workshop Reihe der internationalen Fujaba Days widergespiegelt:

- 8th International Fujaba Days 2011 at the University of Tartu, Estonia. May 11th to 13th, vgl. [FD8]
- 7th International Fujaba Days 2009 at the Eindhoven University of Technology,

The Netherlands. November 16th to 17th, vgl. [FD7]

- 6th International Fujaba Days 2008 at the Technical University of Dresden, Germany. September 18th to 19th, vgl. [FD6]
- 5th International Fujaba Days 2007 at the University of Kassel, Germany. October 8th to 9th, vgl. [FD5]
- 4th International Fujaba Days 2006 at the University of Bayreuth, Germany. September 28th to September 30th, vgl. [FD4]
- 3rd International Fujaba Days 2005 at University of Paderborn, Germany. September 15th to 18th, vgl. [FD3]
- 2nd International Fujaba Days 2004 at Darmstadt Technical University, Germany. September 15th to 17th, vgl. [FD2]
- 1st International Fujaba Days 2003 at University of Kassel, Germany. October 13th to 14th, vgl. [FD1]

1.1 Forschungsgegenstand

Die vorliegende Arbeit beschäftigt sich mit der Kernfrage, in welcher Weise die von Fujaba bekannten Mechanismen des Story-Driven-Modeling auf die Entwicklung von komplexen Webanwendungen übertragen werden können. Hierbei liegt der Fokus auf der möglichst kompletten Modellierung der Gesamtapplikation, ohne die durch Modelle geschaffene Abstraktionsschicht verlassen zu müssen.

Aufbauend auf den mit Fujaba bestehenden Grundlagen für die Modellierung von Java-Applikationen wurde an der Universität Kassel, Fachgebiet Software Engineering unter meiner Mitwirkung, in einem schrittweisen Verfahren evaluiert, welche Anforderungen im Bereich der Webanwendungen bestehen und wie sie mit Hilfe von Modellen gelöst werden können. Vor allem im Rahmen des FAST EU Projektes [wwwd] wurden hier Ideen etabliert und im Folgenden im Rahmen von Fujaba umgesetzt. Als technischer Zwischenschritt wurde hierbei das von Google entwickelte Google Web Toolkit (GWT) [GWTa] gewählt.

Im Rahmen dieser Arbeit wird von mir ein dedizierter Entwicklungsprozess vorgeschlagen, der die Modellierung komplexer Webanwendungen mit dem Story-Driven-Modeling Ansatz ermöglicht und durch Fujaba, sowie von mir entwickelten Erweiterungen, toolgestützt durchgeführt werden kann. Der vorgeschlagene Prozess baut dabei auf dem bereits vorhandenen Fujaba Process (FuP) [Gei11] auf und erweitert diesen um notwendige

clientseitige Prozessanteile, wie ein Verfahren zur Entwicklung grafischer Benutzeroberflächen.

Im Rahmen der vorliegenden Arbeit werden neben dem Prozess und der entwickelten Toolunterstützung auch die während der Forschungsarbeit durchlaufenen Wegpunkte und Überlegungen beschrieben, sowie bei Bedarf deren technische Umsetzung. Die Definition der Applikationsklasse einer Fujaba Web Application spezifiziert im Detail die Eigenschaften, die eine mit dem vorgeschlagenen Prozess und Tooling entwickelte Webapplikation auszeichnen.

1.2 Aufbau des Dokuments

Die vorliegende Arbeit ist wie folgt strukturiert:

In Kapitel 2 werden zunächst alle Wegpunkte beschrieben, die zur Lösung der Forschungsfrage, der Modellierung komplexer Webapplikationen mit Story-Driven-Modeling Mechanismen, notwendig waren. Hierbei wird für jeden Wegpunkt die Fragestellung des jeweiligen Forschungsabschnittes umrissen, sowie die Grundidee zur Lösung vorgestellt, oder Argumente geliefert, die zur Verwerfung der Idee geführt haben. Nachdem durch Kapitel 2 ein Gesamtüberblick über die geleistete Forschungsarbeit gegeben wurde, werden die für den resultierenden Gesamtprozess notwendigen Bestandteile in den Kapiteln 3 bis 6 beschrieben.

Kapitel 3 definiert hierbei im Detail die Eigenschaften einer Fujaba Web Application, wie sie durch das in dieser Arbeit vorgestellte Vorgehen entwickelt wird. Kapitel 4 stellt alle Aspekte dar, die notwendig waren, um im Browser lauffähige, sowie zwischen Client und Server replizierte Datenmodelle zu entwickeln und geht im Detail auf die Anwendung von Graphtransformationen im Webclient ein. Kapitel 5 beschreibt die von mir entworfenen Workflowdiagramme zur Definition von Applikationsfluss und Benutzeroberfläche. Kapitel 6 schließlich stellt die mit dem Prozess ausgelieferte Laufzeitumgebung von Fujaba Web Applications und dessen korrekte Verwendung dar.

Als Kernergebnis der Forschungsfrage wurde von mir der bereits vorhandene FuP um notwendige Elemente erweitert. Kapitel 7 beschreibt den von mir definierten Web Fujbaba Process (WFuP), sowie die zugehörige Toolunterstützung mit Fujaba.

Kapitel 8 gibt einen Einblick in andere Arbeiten, die sich mit dem Thema der Modellierung im Web und Model Driven Web Engineering beschäftigen und grenzt den hier vorgestellten Ansatz von diesen ab. Schlussendlich zieht Kapitel 9 ein Fazit und geht auf Forschungsfragen ein, die sich an die vorliegende Arbeit anschließen.

2 Der lange Weg zum Ziel - Ideen und Schritte im Laufe der Entwicklung des WFuP

Ausgang der dieser Arbeit zugrunde liegenden Forschungsfrage war die Beteiligung der Forschungsgruppe um Professor Zündorf am EU Projekt *FAST:Fast and Advanced Storyboard Tools* [wwwd].

Bereits seit Jahren wurden innerhalb der Forschungsgruppe Aktivitäten im Rahmen von modellgetriebener Softwareentwicklung im Java-Umfeld durchgeführt. Der Story Driven Modeling Ansatz, wie ihn Professor Zündorf in seiner Habilitation [Zün01] vorschlägt, bildete hier seit Jahren das Grundgerüst weiterer Forschungen. Das Softwareentwicklungswerkzeug der Wahl stellt in diesem Zusammenhang die Fujaba Toolsuite (Fujaba) [wwwe], [wwwf] dar. Fujaba bietet die notwendige Toolunterstützung für alle relevanten Schritte des Story Driven Modeling und steht zur freien Nutzung und Weiterentwicklung zur Verfügung. Eine Erweiterung des Story Driven Modeling Ansatzes auf andere Gebiete als die Entwicklung klassischer Java Applikationen wurden am Fachgebiet bereits durch erste Ansätze im Bereich der eingebetteten Systeme mit Lego Mindstorms [Min] durchgeführt.

Die Beteiligung an oben genanntem EU Projekt führte die Forschungsgruppe an die Entwicklung von Applikationen im World Wide Web heran. Das gemeinsame Forschungsziel aller beteiligten Projektpartner war die Entwicklung einer neuen visuellen Programmiersprache und zugehöriger Tools für eine spezielle Art kleiner Applikationen, die im Projektkontext `Front-end Gadgets` genannt wurde. Diese Applikationen sollten dem Nutzer auf einfache Weise Zugang zu verschiedensten Webservices verschaffen und deren Benutzung durch geeignete View-Komponenten unterstützen. Daher auch der Name *Front-end Gadget*; eine Behandlung serverseitiger Logik durch die Applikationen war für *Front-end Gadgets* nicht vorgesehen. Durch die visuelle Programmiersprache des FAST Projektes sollten Benutzer in die Lage versetzt werden, ohne Programmierkenntnisse einzelne dieser Gadgets durch sogenannte *Ports* miteinander zu einem komplexeren Gadget zu kombinieren. Die *Ports* stellten an dieser Stelle Schnittstellen dar, über welche Informationen und Daten unterschiedlicher Webservices über die zugehörigen Gadgets miteinander ausgetauscht werden konnten. Die entstehenden komplexen Gadgets sollten anschließend innerhalb einer Mashup-Plattform [HSSJS08] ausgeführt werden können. Zum Zeitpunkt der Projektlaufzeit waren Mashup-Plattformen wie beispielsweise Yahoo

Pipes [wwwc], iGoogle [wwwb] oder Microsoft Popfly [wwwa] weit verbreitet. Die Gadgets stellten im Kontext der Mashup-Plattformen die elementaren Bausteine dar, welche auf den Mashup-Plattformen zu einem größeren Gesamtsystem kombiniert werden konnten. Auf diese Weise konnten beliebige Kombinationen von Webservices auf einer einzigen Webseite realisiert werden. Jedes Gadget kontaktierte für gewöhnlich einen dedizierten Webservice zur Erfüllung seiner Aufgabe und war selbst Ein- und Ausgabemedium für die benötigten Daten. Darüber hinaus war es auf manchen Mashup-Plattformen möglich, die Ein- und Ausgabedaten der unterschiedlichen Gadgets miteinander zu koppeln, sodass rudimentäre Abläufe abgebildet werden konnten.

Die für uns naheliegende Forschungsfrage beschäftigte sich mit der konkreten Entwicklung solcher Gadgets und in der weitergehenden Betrachtung allgemeiner mit der Entwicklung von Applikationen, welche innerhalb eines Webbrowsers ausgeführt werden konnten. Die bereits im Java Umfeld bewährten Fujaba Vorgehensweisen sollten entsprechend den neuen Anforderungen, die eine im Browser ausführbare Applikation mit sich brachte, untersucht und gegebenenfalls weiterentwickelt werden. Das FAST Projekt lieferte hier mit seinen *Front-end Gadgets* eine erste Art von Applikationen, deren Entwicklung betrachtet werden konnte. Eine Beschränkung auf Gadgets im FAST Kontext kam für mich zu keinem Zeitpunkt der Forschungsarbeiten in Frage. In Zusammenarbeit mit dem FAST Team der Forschungsgruppe, namentlich Ruben Jubeh, Dr. Jörn Dreyer und Prof. Albert Zündorf wurde daher unter Einbeziehung unserer wissenschaftlichen Hilfskräfte eine Referenzarchitektur entwickelt, welche die Appliaktionen deren Entwicklung im Rahmen der weiteren Forschung betrachtet werden sollte, definiert. Diese in der Referenzarchitektur beschriebenen Applikationen wurden für unsere weitere Forschungsarbeit *Web 2.0 Applikationen* genannt. Abbildung 2.1 stellt die intendierte Referenzarchitektur dar, wie sie im Rahmen von [ADJZ08] von mir der Fujaba Gemeinde vorgestellt wurde.

Wie aus der Referenzarchitektur ersichtlich, lagen Applikationen im Forschungsfokus, die auf Seite des Clients (in der Abbildung *Node*) mehrere sogenannte *WebGadgets* gleichzeitig innerhalb eines Browsers ausführen konnten. Hierbei unterschied sich der gezeigte Ansatz von den zu diesem Zeitpunkt verbreiteten Referenzarchitekturen für Webapplikationen dahingehend, dass bei den hier eingesetzten *WebGadgets* nicht nur der View Anteil eines Gadgets in Form eines Document Object Model (*DOM*) der gezeigten Webseite auf Clientseite vorliegt, sondern ebenfalls ein eigenes `Application Model`. Die Serverseite beinhaltet unterschiedliche *Services*, welche wiederum Komponenten für Persistenz (*Persistency*), sowie Modellanteile im *ServiceModel* zur Verfügung stellen, wie es auch von anderen Architekturen bekannt war. Klares Ziel bei der Entwicklung von Applikationen gemäß der vorgestellten Referenzarchitektur war die Etablierung der bewährten Fujaba-Konzepte aus dem Java Umfeld im Kontext von Web Applikationen.

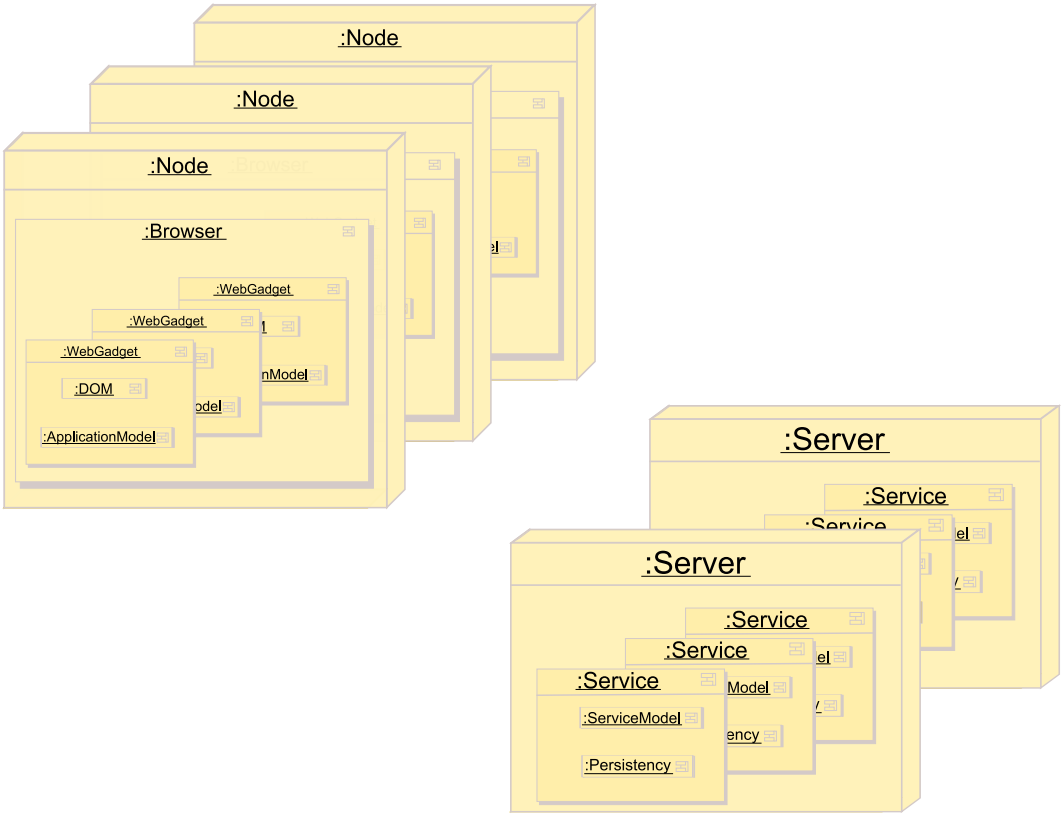

Abbildung 2.1: Referenzarchitektur für Web 2.0 Applikationen, wie sie von der Fujaba-Forschungsgruppe intendiert wurden.

Die so modellierten Webanwendungen sollten sich in der Benutzung möglichst wenig wie eine klassische Webseite, sondern vermehrt wie eine Desktopapplikation anfühlen. Web 2.0 Applikationen waren somit ganz klar als Ajax Applikationen[1] mit asynchroner Client-Server Kommunikation angelegt. Dies sollte den Benutzer in die Lage versetzen, mit der Applikation auch in Fällen, in welchen auf eine Antwort vom Webserver gewartet wird, zu interagieren. Eine Entwicklung solcher Applikationen stellte zum Zeitpunkt der Forschung kein großes Problem mehr dar, da alle gängigen Browser über die zu diesem Zweck notwendige Ajax-Engine zur Ausführung von JavaScript auf Clientseite verfügten. Auf Grundlage dieser vorhandenen guten technischen Möglichkeiten seitens der Webbrowser entstand die in der Referenzarchitektur wiedergespiegelte Idee des clientseitigen Datenmodells (*Application Model*). Neben diesen Anwendungsdaten sollte ein großer

[1] Ajax: Asynchronuous-JavaScript and XML; Ajax Applikationen zeichnen sich im Gegensatz zu traditionellen HTML-Seiten durch eine vermehrte „Intelligenz "auf Clientseite aus. Statt lediglich eines reinen HTML Dokuments wird vom Server bei einer Anfrage gleichzeitig JavaScript mit ausgeliefert, der Berechnungen auf Clientseite ermöglicht, während der HTML Inhalt der Seite vom Browser gerendert wird. Ajax Applikationen sind zudem in der Lage, auf Ereignisse - wie beispielsweise Tastenanschläge oder Mausklicks zu reagieren - und ermöglichen somit eine zeitnahe Reaktion auf Benutzereingaben.

Teil der Businesslogik auf die Clientseite verlagert und dort vom Browser ausgeführt werden. Einhergehend mit der Auslagerung des Datenmodells auf den Webclient sah die Referenzarchitektur weiterhin die vom Desktop bekannte Trennung von View und Modell sowie die Anwendung des Model-View-Controller-Patterns auf dem Webclient vor. Neben den grundsätzlichen Unterschieden zwischen einer mit Fujaba Technologien bereits beherrschbaren Desktopapplikation und einer Webapplikation, die eine Aufteilung des gesamten Applikationscodes in einen Server- und einen Clientanteil erforderlich machte, stellten die zuvor beschriebenen *Web 2.0 Applikationen* zudem die Anforderung, dass der Client-Teil (Datenmodell und Businesslogik), der in der Ajax-Engine des Webbrowsers ausgeführt werden sollte, als JavaScript Quelltext vorliegen musste. Mehr noch, es musste eine Möglichkeit gefunden werden, die benötigten Daten zwischen dem Server und potentiell mehreren Webclients replizieren zu können. All diese Anforderungen sollten im Fujaba-Kontext erfüllt werden. Die Webapplikation sollte mit Story-Driven Modeling Technologien modelliert werden. Dies galt sowohl für alle Serverteile, als auch für die JavaScript Anteile auf dem Client. Eine Erzeugung von JavaScript Quelltexten durch den Fujaba-Codegenerator war jedoch nicht möglich.

In einem ersten Schritt galt es zur Erfüllung der oben beschriebenen Anforderungen einen Weg zu finden, JavaScript Quelltexte aus Fujaba heraus zu erzeugen. Die Codegenerierung von Fujaba basiert auf Templates der Velocity-Template-Engine [Vel]. Für alle zu generierenden Artefakte der in Fujaba vorhandenen Diagramme sind jeweils eigene Templates vorhanden, die eine Erzeugung von Java Quelltexten aus diesen Diagrammen ermöglichen. Eine Möglichkeit zur Erzeugung des benötigten JavaScript Quelltextes für *Web 2.0 Applikationen* bestand in der kompletten Neuerstellung aller notwendigen Templates mit JavaScript als Zielsprache. Neben den zu erstellenden Templates für eine neue Zielsprache wäre es notwendig, Fujaba dahingehend anzupassen, dass die entsprechende Zielsprache ausgewählt werden kann. Im Fall der Webapplikationen wäre hier eine Trennung der Codegenerierung für Client und Server notwendig, da die Anforderung an JavaScript Quelltexte lediglich für die Clientseite zutreffend ist. Eine zweite Möglichkeit, die clientseitigen Anforderungen zu erfüllen, bot die Nutzung des Google Web Toolkit (GWT) [GWTa]. Das von Google frei zur Verfügung gestellte Toolkit verfügt über einen sogenannten Cross-Compiler, der es ermöglicht, Java Quelltext in browserspezifischen[2] JavaScript Quelltext zu übersetzen. Dies bot die Möglichkeit, die Codegenerierung von Fujaba weitgehend zu belassen und in einem ergänzenden Arbeitsschritt die notwendigen

[2] GWT stellt hier jeweils für alle auf dem Entwicklungssystem verfügbaren Browser eine eigene optimierte JavaScript Version der erzeugten Anwendung zur Verfügung. Ebenso verhält es sich mit gegebenenfalls mehrfach vorliegende Sprachen. Ist über Internationalisierung die Anwendung für mehrere Sprachen entwickelt, so wird pro Browser und pro Sprache eine eigene JavaScript Version erzeugt. Dieses Vorgehen soll den vom Server zu ladenden JavaScript Anteil minimieren, da der Browser nur das für den konkreten Nutzungsfall zutreffende JavaScript laden muss und nicht alle Teile der Anwendung.

Teile der Applikation mit GWT übersetzen zu lassen. Aufgrund der geringeren notwendigen Anpassungen an Fujaba fiel die Entscheidung auf die zweite genannte Möglichkeit zur Erzeugung des notwendigen JavaScript. Die Nutzung der GWT-Bibliothek allein war jedoch nicht ausreichend um alle Anforderungen zu erfüllen. Um clientseitige Datenmodelle zu erhalten, die zudem gleichzeitig auf mehreren Webclients verwendet werden konnten, waren weitere Schritte notwendig, die im Detail in Kapitel 4 nachzulesen sind. Die grundlegende Anforderung nach ausführbaren replizierten clientseitigen Datenmodellen konnte mit diesen Schritten gewährleistet werden.

Mit der angepassten Codegenerierung und durch Nutzung von GWT war es möglich, rudimentäre Applikationen zu entwickeln, die innerhalb des Webbrowsers lauffähig waren. Eine Behandlung grafischer Benutzerschnittstellen (GUI) war jedoch mit Fujaba nicht möglich. Zwar konnten innerhalb von Storydiagrammen grafische Elemente verwendet werden, ein Erstellen von grafischen Benutzerschnittstellen mit einer eigens dafür bestimmten grafischen Notation war jedoch in Fujaba nicht vorhanden. Bei der Forschung über Gadgets im Web blieb jedoch die Entwicklung soller GUIs nicht, wie dies im Bereich Java oft der Fall gewesen war, außen vor oder wurde von Hand erledigt - die GUIs sollten in den Gesamtansatz integriert werden. Auch hier bot das bereits zur Codegenerierung verwendete GWT ein adäquates Hilfsmittel. Das Toolkit verfügt neben dem bereits erwähnten Cross-Compiler über eine umfangreiche Widget-Bibliothek [GWTc] die zur Erzeugung komplexer GUIs verwendet werden konnte. Mit dieser Grundlage wurde als erster Schritt im Bereich GUI ebenfalls in [ADJZ08] der Ansatz erweiterter Storydiagramme vorgestellt. Ein Beispiel solcher Diagramme ist in Abbildung 2.2 dargestellt.

Die Abbildung zeigt das erweiterte Storydiagramm eines einfachen Login-Dialogs. Auf der linken Seite wurde hierbei in Storydiagramm-Notation die Erzeugung der notwendigen GUI modelliert, auf der rechten Seite des Diagramms wurde die entsprechende Oberfläche in konkreter Syntax dargestellt. Im Fall der Fujaba Modellierung bedeutete dies, zur Anzeige der konkreten Syntax wurden die entsprechenden Elemente der GWT-Widget Library in einem speziellen integrierten Browserfenster angezeigt. Die erweiterten Story-Diagramme waren ein erster Ansatz zur Erzeugung grafischer Benutzerschnittstellen mit Fujaba. Die für die Erzeugung einer kompletten Applikation notwendige Businesslogik wurde weiterhin über herkömmliche Fujaba Diagramme und die in Kapitel 4 beschriebene angepasste Codegenerierung für Businesslogik und Applikationsmodell erzeugt.

Nachdem die grundlegenden Möglichkeiten für browser-kompatible Datenmodelle in JavaScript sowie GUI Erzeugung mit den oben beschriebenen Ansätzen gegeben waren, fehlte zur Modellierung einer kompletten Anwendung mit Fujaba noch immer die Möglichkeit, Datenmodell, Serverseite und Businesslogik miteinander zu verbinden und geeignet zu synchronisieren. Hierzu wurde zunächst die Idee entwickelt, die Fujaba Storydiagramme dahingehend zu erweitern, `PropertyChange` basierte Event Mechanismen

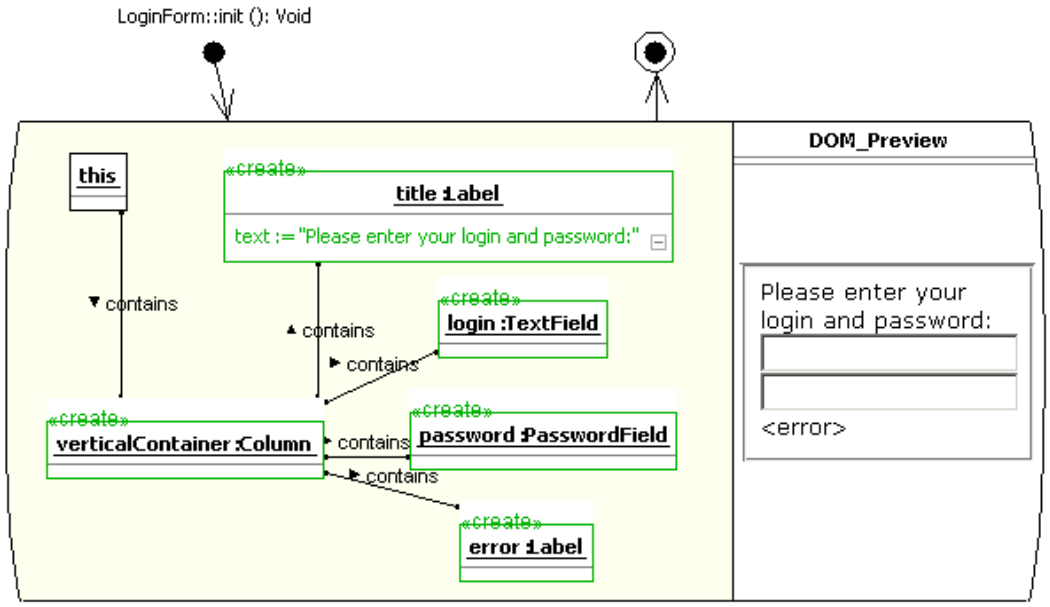

Abbildung 2.2: Erweitertes Story-Diagramm mit DOM Operationen in Fujaba

an Transitionen verarbeiten zu können. Diese neuen Transitionen wären beispielsweise dann vonnöten, wenn Änderungen innerhalb der GUI auftreten. Ein Beispiel für eine solche Änderung ist etwa ein Klick auf einen Button, wodurch gleichsam weitere Aktionen ausgelöst werden können, das Eingeben von Text in ein dafür vorgesehenes Textfeld oder aber das Auslösen beliebiger anderer Maus- oder Tastaturevents. Die Events werden vom Webbrowser zunächst lokal verarbeitet und an die laufende Webapplikation weitergeleitet. Ein Beispiel der angedachten erweiterten Storydiagramm-Transitionen ist in Abbildung 2.3 dargestellt. Hier wird ein Login-Vorgang modelliert.

Im Storydiagramm aus Abbildung 2.3 wird innerhalb der ersten Activity des Diagramms zunächst ein Dialog zur Eingabe der Login-Daten (*LoginForm*), sowie der zugehörige Service (*LoginService*)erzeugt und angezeigt (1). Innerhalb des Login-Dialog können die benötigten Informationen vom Benutzer eingegeben werden. Das Submit-Event des Formulars, das beim bestätigen des Login-Dialogs auftritt, ist an der ausgehenden Transition der entsprechenden Activity angegeben (2). Dies führt dazu, dass diese Transition bei Auftreten des entsprechenden Events behandelt wird. Der initiale Zustand des Storydiagramms wird verlassen und die zweite Activity betreten. Hier werden die eingegebenen Daten aus dem Login-Dialog ausgelesen (3). Das Überprüfen der User Credentials erfolgt erneut über eine Bedingung an der ausgehenden Transition (4). Ist die Bedingung erfüllt, wird der Login-Dialog verlassen und das Hauptfenster der Anwendung angezeigt (*MainMenu*) (5). Bei falsch eingegebenen User Credentials hingegen wird ein Fehler angezeigt und die Eingabe der Daten kann erneut erfolgen (6). Eine Umsetzung des

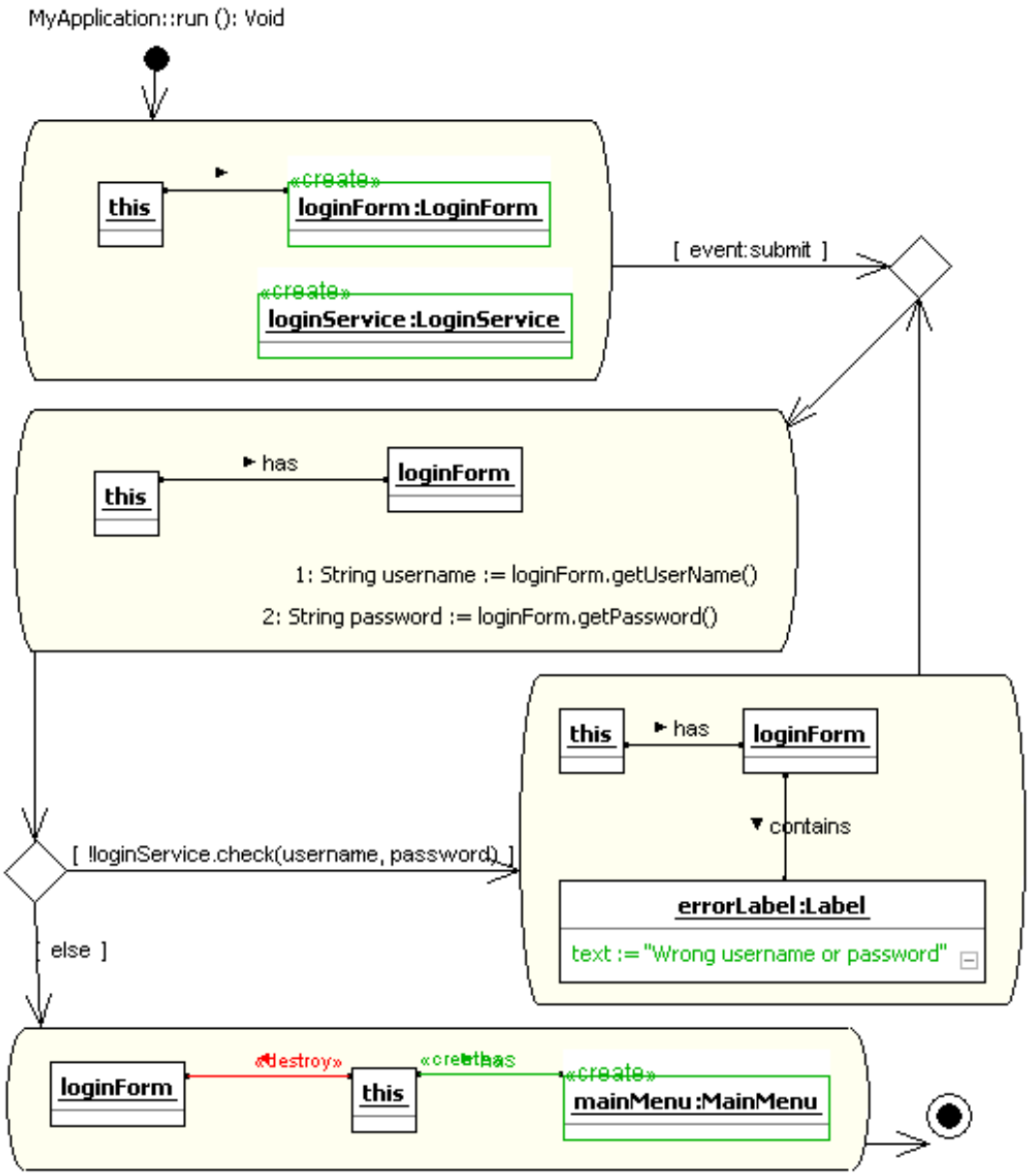

Abbildung 2.3: Definieren clientseitigen Verhaltens bei auftretenden Events.

reaktiven Verhaltens einer Appliaktion wurde im Folgenden jedoch nicht in Form von Storydiagramm-Transitionen, sondern in der Erweiterung und teilweisen Neu-Definition der bereits vorhandenen Fujaba-Statecharts als sogenannte *ActionCharts* [GGH+10]. Die Ideen dazu wurden erneut von mir gemeinsam mit Ruben Jubeh und Prof. Zündorf aus dem FAST-Kontext heraus entwickelt und von Tobias George in seiner Bachelorarbeit technisch umgesetzt. Durch eine Umsetzung des geplanten reaktiven Verhaltens mit Statecharts konnte eine größere Abstraktion in der Modellierung erreicht werden. Der Entwickler muss sich um die Behandlung asynchroner Abläufe nicht kümmern, eine geeignete Modellierung des Verhaltens innerhalb der *ActionCharts* sorgt automatisch für eine entsprechende Generierung der benötigten Verhaltensweisen. Bei einer Umsetzung mittels Storydiagrammen hingegen wäre die Modellierung oder Generierung verschiedener asynchroner Methoden und Rümpfe notwendig gewesen, vermehrte und unübersichtlichere Diagramme wären die Folge. Kaptitel 4.3.4 geht nochmals auf die Anbindung von Services, Events und Timern über *ActionCharts* ein.

Die grundlegenden Anforderungen zur Modellierung einer einzelnen Webapplikation waren so zwar erfüllt, die in Abbildung 2.1 gezeigte Referenzarchitektur ermöglichte es jedoch darüber hinaus, mehrere Clients gleichzeitig mit einer Serverkomponente synchron zu halten. Hierfür musste ein Mechanismus definiert werden, der diese Synchronisierung übernehmen und sich gleichzeitig mit dem Story-Driven Modeling Ansatz in Fujaba verbinden ließ. Im Technikrepertoire der Forschungsgruppe existierte hier bereits mit dem CoObRA Ansatz [Sch07] ein Mechanismus zur Synchronisation, Speicherung und Replikation von Daten. Die unterschiedlichen Anforderungen im Web-Bereich erforderten jedoch auch hier Anpassungen. Hierzu wurden unter dem Namen WebCoObRA [ADH+09] die notwendigen Anpassungen an CoObRA vorgenommen. Kapitel 4 beschreibt genauer, welche Anpassungen hierfür notwendig waren, und wie WebCoObRA in die Modellierung eingebunden werden kann.

Nicht nur eine Synchronisierung unterschiedlicher Clients mit der notwendigen Serverkompontente, sondern auch eine serverseitige persistente Speicherung von Anwendungsdaten waren für die komplette Erfüllung aller Applikationsanforderungen notwendig. Die Referenzarchitektur aus Abbildung 2.1 sieht daher auf der Serverseite sowohl ein Datenmodell, als auch eine Komponente zur persistenten Speicherung der Daten vor. In ganz einfachen Anwendungsfällen sind die Datenmodelle auf Client und Server identisch und können ohne weitere Einschränkungen aus Fujaba heraus generiert werden. Hier ist zudem lediglich die persistente Speicherung des auf den Server replizierten Datenmodells notwendig. Häufiger wird jedoch der Client lediglich einen Teil der kompletten Datenstruktur laden, sei es aus Gründen der Sicherheit, aus Datenschutzgründen oder aus Gründen der Performanz. In diesen Fällen können Operationen, welche auf der kompletten Datenstruktur arbeiten, nur auf Serverseite ausgeführt werden. Eine Trennung

von Client- und Servermodell ist an dieser Stelle notwendig. Desweiteren muss der Server darüber Kenntnis besitzen, welche Clients angemeldet sind und in welchem Status sich diese befinden. Im Falle einer Änderung am Datenmodell muss der Server dafür Sorge tragen, den aktuellen Datenstand über alle Clients hinweg konsistent zu halten. Auch diese Anwendungsfälle konnten mit dem WebCoObRA Framework abgedeckt werden. Hierfür wurden spezielle Annotationen eingeführt, welche im Klassendiagramm Lese- und Schreibrechte auf einzelnen Elementen des Datenmodells regeln konnten. Die notwendige Behandlung dieser Annotationen wurde von WebCoObRA sichergestellt und wird in Kapitel 4 genauer beschrieben.

Die Grundlagen für Story-Driven Modeling von Webapplikationen war mit den zuvor erwähnten Technologien gelegt. Die Forschung im Rahmen des FAST EU Projektes lieferte mir jedoch darüber hinaus weitere Erkenntnisse und Teilaspekte, die später in die Definition des Web Fujaba Process (WFuP) einflossen. Ein Ziel von FAST war, die Kopplung der einzelnen Gadgets innerhalb der Mashup-Plattformen an assoziierte Services, mit denen sie kommunizierten. Jedes Gadget war in der Lage, mit einem im Gadget definierten Webservice zu interagieren und Daten anzuzeigen, beziehungsweise an diesen weiterzuleiten. Darüber hinaus sollten die einzelnen Gadgets untereinander über sogenannte Data Ports verknüpft werden. Dies stellte das FAST Team der Universität Kassel, namentlich Ruben Jubeh, Jörn Dreyer, Prof. Zündorf und mich, vor die Aufgabe, Techniken in Fujaba zu entwickeln, mit denen diese Verbindung auf Modellierungsebene geschaffen werden konnte. Als fortlaufendes Forschungsbeispiel diente hier das ToDo Gadget aus Abbildung 2.4. Die Abbildung zeigt hier eine ToDo-Liste für ein bevorstehendes Meeting. Zu jedem einzelnen Punkt können der Status, ein assoziierter Webservice, ein Verantwortlicher, sowie Eingangsdaten und Ausgangsdaten für die entsprechenden Ports definiert werden. Das ToDo Gadget bot somit nicht nur die Möglichkeit, innerhalb eines einzelnen komplexen Gadgets gleich auf mehrere Services zuzugreifen, sondern war ebenfalls ein erstes Beispiel für die automatisierte schrittweise Ausführung einer Applikation. Die Beschreibung dieses schrittweisen Ausführungsverhaltens wurde von mir im weiteren Verlauf meiner Dissertation zu den Workflowdiagrammen (Kapitel 5) und der zugehörigen Ausführungslogik der Fujaba Web Applikationen (Kapitel 6) ausgebaut.

Wie später bei den Storydiagrammen konnte innerhalb des ToDo Gadget eine Anzahl an Schritten definiert werden, die zur Abarbeitung einer gegebenen Aufgabe notwendig sind. Im Falle des Gadgets war zudem jeder dieser Schritte mit einem externen Webservice verbunden. Diese Verbindung wurde über das konkrete Einbinden einer URL ins Gadget vorgenommen. Durch die Definiton einzelner Schritte innerhalb des Gadgets wird es ermöglicht, jeden assoziierten Service nacheinander, in der Reihenfolge der Schritte, zu besuchen. Hierbei kann wiederum jeder der Schritte mit Eingabe- und Ausgabeparametern verknüpft werden. Das Beispiel aus Abbildung 2.4 definiert als Parameter etwa

Have a meeting

No.	Action Item	State	Documents	Responsible	Input Data	Output Data
1	Book flights	☑	http://www.expedia.de/fluege/linienfluege	Nina	dates, airport of departure, destination airport	e-tickets, departure times, arrival times
2	Book hotel	☑	http://www.expedia.de/hotels/default	Ruben	dates, location, estimated price	Booking confirmation
3	Have presentations	☑	http://docs.google.com/Doc?docid=xxxxxxxxxxxx3	Albert	work done, ideas	-
4	Have social dinner	☑	-	UPM	date, time, number of persons	a lot of fun
5	Reimbursement for travel expenses	☐	http://www.uni-kassel.de/pvabt3/download/reisekosten.ghk	Nina, Ruben, Albert	dates, bills	money

Abbildung 2.4: ToDo Gadget - erstmals wurde schrittweise Ausführung einzelner Services innerhalb einer Applikation erprobt.

Angaben zu Reisedaten oder Abflug- und Zielflughafen. Das ToDo Gadget ermöglichte es erstmals, mehrere Services innerhalb eines gemeinsamen Workflows miteinander zu verbinden und war gleichzeitig die erste von uns entwickelte Web 2.0 Applikation nach der Referenzarchitektur aus Abbildung 2.1. Das ToDo Gadget war sowohl als Einzelapplikation innerhalb eines Webbrowsers, als auch als Gadget innerhalb einer Mashup Plattform lauffähig. Dabei wurden clientseitiges Datenmodell, Mehrbenutzerfähigkeit über Datenreplikation, automatisierte Ausführungslogik über ToDo Items und Datenpersistenz über die oben vorgestellten Technologien miteinander kombiniert, und das Gadget nahezu komplett mit Fujaba entwickelt. Anders als andere Gadgets, welche in Mashup Plattformen eingebunden werden konnten, benötigte das ToDo Gadget jedoch dedizierte eigene Services für Datenreplikation und Persistenz mittels CoObRA. Die Fujaba basierte Entwicklung des ToDo Gadgets war der erste Proof-of-concept für die von uns angestrebten Entwicklungsmechanismen, sowie gleichzeitig der erste Schritt hin zu webbasierter Workflow Ausführung.

Bereits bei der zuvor beschriebenen Einführung der erweiterten Story-Diagramme wurde die Entwicklung von grafischen Benutzerschnittstellen in die Gesamtbetrachtung mit einbezogen. Der dort vorgestellte erste Ansatz der erweiterten Storydiagramme erwies

sich jedoch bei weitergehenden Betrachtungen als nicht zielführend. Die Entwicklung der notwendigen komplexen GUIs machte eine große Menge an Diagrammen notwendig, die zudem schnell groß und damit unübersichtlich wurden. Eine prinzipielle Erfüllung der Anforderungen war zwar mit dem Ansatz der erweiterten Storydiagramme sowie der zuvor beschriebenen *ActionCharts* möglich, war für mich jedoch nicht ausreichend komfortabel. Zwar konnten in den erweiterten Story-Diagrammen bereits GUI Elemente erzeugt und über die erweiterten Transitionen auch an Events gekoppelt werden, eine Entwicklung komplexererer Benutzerschnittstellen mit diesem Ansatz stellte uns jedoch nicht zufrieden und war zudem dem Programmieren der GUI von Hand nicht wirklich überlegen. Eine Abstraktion der Sachverhalte, wie dies beim Modellieren von Businesslogik mittels Story-Driven Modeling der Fall ist, konnte durch die Modellierung der GUI nicht erreicht werden. Aus diesen Gründen entschloss ich mich, den Diagramm-Ansatz zur Erzeugung der grafischen Benutzerschnittstelle zu verwerfen und einen anderen Weg für die Umsetzung der GUI zu wählen. Als Hilfsmittel zur Erzeugung grafischer Benutzerschnittstellen werden auch grafische What-You-See-Is-What-You-Get Editoren eingesetzt, mit denen die Oberfläche aus grafischen Elementen zusammengesetzt werden kann. Diese Vorgehensweise erschien mir auch im Hinblick auf die für unsere Webapplikationen notwendigen GUIs sinnvoll. Ein geeigneter Editor musste in der Lage sein, eine Oberfläche aus GWT-Widgets zusammenzustellen und sich zudem in den Story-Driven-Modeling Ansatz einfügen lassen. Der einzige zum damaligen Zeitpunkt in Frage kommende Editor war der GWT-Designer der Firma Instantiations[3]. Dieser war zwar dazu in der Lage GWT-Widgets zu einer Oberfläche zusammenzufügen und konnte darüber hinaus weitere Widget-Bibliotheken, wie etwa SmartGWT [wwwg] einbinden, war jedoch als kommerzielles Werkzeug nicht für unsere Zwecke nutzbar, da es nicht den Story-Driven-Modeling Anforderungen gemäß angepasst werden konnte. Ich entschied mich daher, einen eigenen grafischen Editor für GWT in konkreter Syntax zu entwickeln [GEH+09]. Dies bedeutete, der Editor selbst wurde als GWT-Webanwendung geplant. Abbildung 2.5 zeigt den grafischen Editor in einem frühen Stadium der Arbeiten. Auf der rechten Seite befindet sich ein Panel welches, wie für GUI-Designer üblich, eine sortierte Palette mit verfügbaren User Interface Elementen enthält. Per Drag-and-Drop können diese Elemente auf die links befindliche Zeichenfläche übertragen werden. In Abbildung 2.5 wurde ein *TextField* aus der Palette auf die Zeichenfläche gezogen. Eine Umrandung zeigt dabei die Ausmaße des grafischen Elements an.

Durch die komplette Neuentwicklung des GUI-Designers war es mir möglich, das Tool bestmöglich in den angestrebten Modell-zentrierten Entwicklungsansatz auszulegen. Mein Ziel war, die komplette Webanwendung und auch die Anbindung der Businesslogik an die grafische Benutzerschnittstelle mit den bewährten Fujaba-Mitteln durchführen zu

[3] Der erwähnte Designer ist zum jetzigen Zeitpunkt nicht mehr unter diesem Namen verfügbar, sondern wurde in der Folge von Google zur freien Benutzung zur Verfügung gestellt [GWTb]

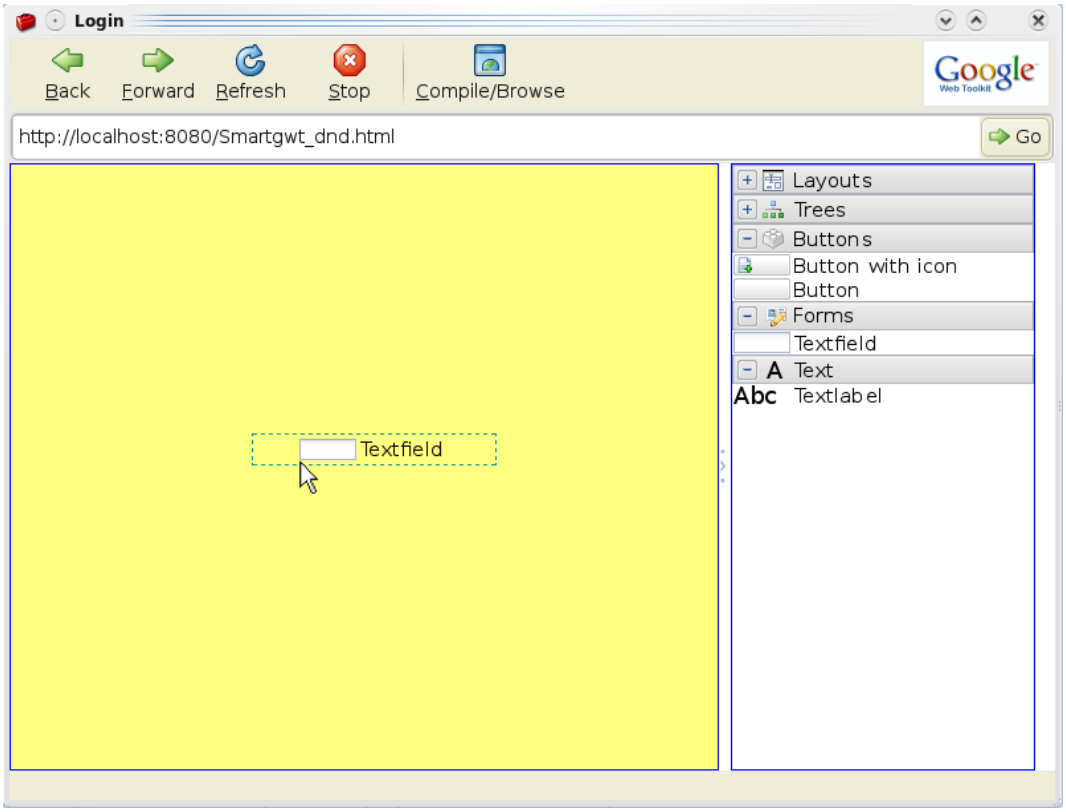

Abbildung 2.5: Der von mir entwickelte GUI Designer in einem frühen Stadium der Arbeiten

können. Ein wichtiges Ziel bei der Konzeption des GUI-Designers war es daher, einen möglichst einfachen Weg zu etablieren, die erstellten grafischen Benutzerschnittstellen und die in Fujaba modellierte Businesslogik miteinander zu verknüpfen. Ich wählte daher als Repräsentation der GUI keinen Quelltext, wie dies in anderen Tools der Fall ist, sondern eine Repräsentation als Fujaba Aktivitätsdiagramm. Der zur Ausführung notwendige Quelltext der GUI sollte im Codegenerierungs-Schritt mit Fujaba erzeugt werden. Diese Repräsentation hat den Vorteil, dass eine grafische Benutzerschnittstelle, welche als Aktivitätsdiagramm in Fujaba vorliegt, einfach über Story-Driven-Modeling Mechanismen mit der Businesslogik gekoppelt werden kann. Hierzu sind die bereits bei den erweiterten Story-Diagrammen verwendeten Mechanismen ausreichend. Für die Konzeption des GUI-Designers stellte sich nun die Herausforderung, die im Webbrowser laufende und somit als JavaScript vorliegende GUI-Designer Software dazu in die Lage zu versetzen, die entsprechenden Aktivitätsdiagramme innerhalb von Fujaba zu erzeugen.

Um den GUI-Designer nicht zu einer Einbahnstraße werden zu lassen, verfolgte ich weiterhin das Ziel, Änderungen, die an den entsprechenden Aktivitätsdiagrammen innerhalb von Fujaba vorgenommen werden könnten, in den GUI Designer zurück zu spiegeln.

Ich besann mich auf die Fujaba zugrunde liegenden Graphgrammatiken und traf den Entschluss, einen Abgleich zwischen GUI Designer und Fujaba Toolsuite mittels Model-Mapping Technologien [Mod15] vorzunehmen. Model-Mapping Techniken dienen dazu, Modelle unterschiedlicher Natur aufeinander abzubilden. In meinem Fall war für die Lösung der kompletten Problemstellung eine einseitig nutzbare Abbildung vom GUI Designer auf das Fujaba interne Metamodell nicht ausreichend. Ein Model-Mapping in beide Richtungen war notwendig, weshalb ich mich dafür entschied, Triple-Graph-Grammatiken [GW06] als Bindeglied zwischen dem GUI Designer und der Fujaba Toolsuite einzusetzen. Triple-Graph-Grammatiken basieren, wie der Name bereits andeutet, auf drei Graphen, zwischen denen Mappings definiert werden können. Aus diesen Mappings können anschließend Transformationsregeln generiert und auf den Graphstrukturen ausgeführt werden. Die Triple-Graph-Grammatiken bedienen sich hierbei einer Mapping-Struktur, welche den dritten Graphen zwischen Quelle und Ziel darstellt. Von diesen Mappingstrukturen können nun Transformationen sowohl ins sogenannte Quellmodell - in unserem Fall der GUI Designer - wie auch ins Zielmodell - das Fujaba Metamodell - definiert werden. Um ein geeignetes Modell der innerhalb des Webbrowsers laufenden GUI aufbauen zu können, war zunächst die Erzeugung einer Wrapper-Struktur für die GWT-Klassenbibliothek notwendig, die es ermöglichte, alle Widget-Klassen inklusive zugehöriger Attribute abzubilden. Die Wrapper-Struktur war hierbei nicht nur zum korrekten Aufbau des Model-Mappings über Triple-Graph-Grammatiken notwendig, sondern ersetzte ebenfalls den naturgemäß im Webbrowser fehlenden Mechanismus für Java Reflection. Ohne den Mechanismus der Reflection wäre es sonst unmöglich gewesen, Attribute und Funktionen einer User Interface Komponente während der Laufzeit automatisch aus der entsprechenden GWT-Bibliotheksklasse auszulesen. Für das von uns beabsichtigte Verhalten des GUI-Designers war dies jedoch notwendig und wurde so durch die aufgebaute Wrapper-Struktur ermöglicht. Die Kombination der Wrapper-Struktur, der Triple-Graphen, sowie die mit CoObRA bestehenden Replikationsmechanismen stellte sicher, dass alle Informationen über die GUI zu jeder Zeit in Fujaba zur Verfügung standen, um die notwendigen Aktivitätsdiagramme aufzubauen. Nur so wurde es möglich, das Model-Mapping mit den vorhandenen Informationen korrekt zu initialisieren, sowie im späteren Verlauf der Entwicklung die Businesslogik über geeignete Attribute und Methoden an die User Interfaces anzuknüpfen. Ich entschied mich weiterhin dafür, das Model-Mapping nicht auf der Seite des Webbrowsers - und somit auf der Clientseite der GUI-Designer Applikation - durchzuführen, sondern innerhalb von Fujaba, auf der Serverseite des GUI-Designers. Auf diese Weise konnten das schwergewichtige Fujaba-Metamodell ohne weitere Anpassung verwendet und übliche Fujaba-Funktionen zum Aufbau der Aktivitätsdiagramme genutzt werden. Die clientseitig aufgebaute Struktur der GUI wurde in Form der zuvor erwähnten Wrapper-Struktur über die WebCoObRA Mechanismen auf die in Fujaba eingebundene Serverseite repliziert. Durch die Verwendung von WebCoObRA waren alle im GUI-Designer erzeugten Elemente durch eine Re-

präsentation ihrer jeweiligen Wrapper-Klasse innerhalb der serverseitigen Datenstruktur vorhanden und ein Mapping zwischen GUI-Designer und Fujaba Aktivitätsdiagramm konnte in Echtzeit durchgeführt werden, ohne die erzeugten Graphical User Interfaces zuvor abspeichern zu müssen. Die Kombination von Triple-Graph- Grammatiken und Datenreplikation ermöglichte, so alle am GUI Designer vorgenommenen Änderungen sofort ins Fujaba Aktivitätsdiagramm zu übernehmen und umgekehrt. Abbildung 2.6 zeigt in einem Überblick die Gesamtarchitektur des von mir konzipierten GUI Designer Systems.

Auf Client-Seite sind hier die einzelnen User Interface Komponenten über PropertyChange Mechanismen an ihre zugehörige Wrapper-Klasse gebunden. So kann sichergestellt werden, dass alle in der konkreten Syntax auftretenden Änderungen auch modellseitig erfasst werden. Die Wrapper-Klasse wird mittels WebCoObRA auf den zugehörigen Server repliziert und dort mit Hilfe von Model-Mapping Mechanismen (Triple Graph Grammatiken) in die entprechenden Funktionen des Fujaba Aktivitätsdiagrammes übersetzt. Hier wird für jede grafische Benutzerschnittstelle (GUI) im Konstruktor der zugehörigen Klasse die konkrete Modellierung der erstellenden Funktionen vorgenommen. Nachdem die gewünschte Benutzerschnittstelle auf diese Weise erzeugt wurde, stellten die vorhandenen Aktivitätsdiagramme sicher, dass auch der Rest der Applikation mit Fujaba modelliert werden konnte. Die in Fujaba integrierte und gemäß der Beschreibung aus Kapitel 4 angepasste Codegenerierung für Java Code (CodeGen 2) übersetzt die so erzeugten Aktivitätsdiagramme in ausführbaren Java-Quelltext. Dieser kann einerseits mit GWT-Mechanismen auf Client-Seite debugged, oder aber über den GWT Compiler in JavaScript Quelltext übersetzt werden.

Die Verlagerung der Erzeugung der GUI über ein Designer-Tool, und somit auf dem für den Anwender komfortabelsten Weg, stellt im ersten Moment einen Bruch mit der Anforderung an eine komplett modellierte Anwendung dar. Die Vorteile der grafischen Erzeugung der GUI und die somit bestehende Abstraktion vom manuellen Erzeugen der Oberfläche, sei es per Hand in Quelltext, oder aber als Aktivitätsdiagramm, stellt jedoch in meinen Augen einen beträchtlichen Mehrwert dar. Zudem kann die Erzeugung der GUI auf diesem Weg einfach in den gesamten Modellierungsansatz eingebunden werden. Eine weitere Anforderung würde in der Folge jedoch sein, einen Mechanismus zu etablieren, um einzelne GUIs und ihre angeknüpfte Businesslogik (über EventHandler oder ähnliche Mechanismen) miteinander zu einer größeren Gesamtapplikation zusammenzufügen. Diese ist im Folgenden mit der von mir vorangetriebenen Entwicklung des WFuP geschehen.

Im Jahr 2008 wurde die erwähnte kommerzielle GWT-Designer Software von Google übernommen und als freies Tool mit offenen Schnittstellen der Entwicklergemeinde zur Verfügung gestellt. Es stand nun also ein komplett entwickeltes Tool zur Verfügung, das über Schnittstellen in den Gesamtansatz eingebunden werden konnte. Die Grund-

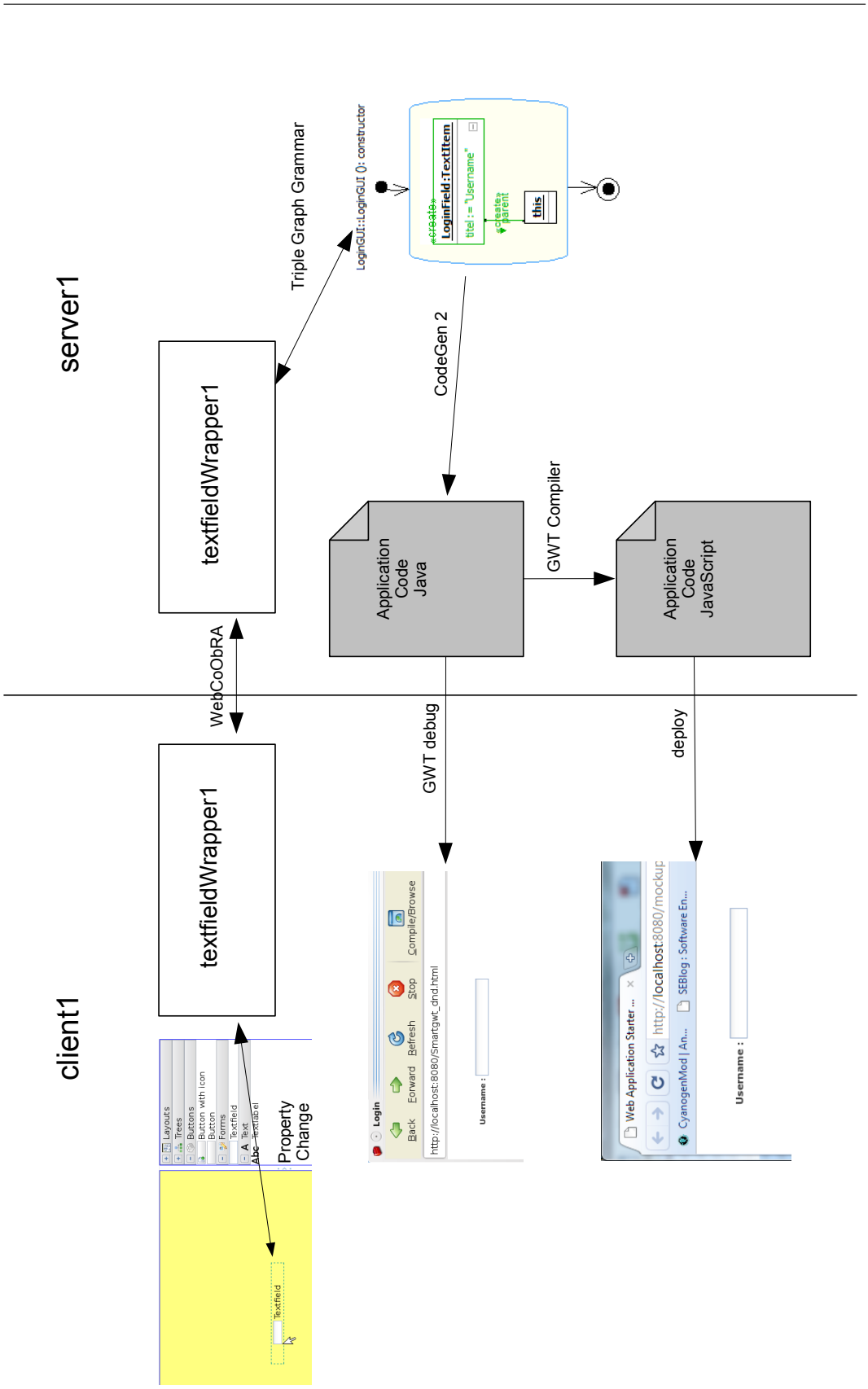

Abbildung 2.6: Gesamtarchitektur des GUI-Designers.

lage, die zur Entwicklung eines eigenen Tools geführt hatte, bestand nicht weiter. Ich untersuchte die Möglichkeiten des GWT-Designers und entschloss mich schließlich, das in Abbildung 2.6 gezeigte Konzept zu verwerfen und die bestehende Software in meine Arbeit zu integrieren. Im Gegensatz zu dem von uns konzipierten Ansatz verwendete der GWT-Designer als Repräsentation der Benutzerschnittstelle Quelltext. Es bedurfte also eines neuen Konzeptes, um die erstellte GUI mit Fujaba und somit mit der Businesslogik zu kombinieren. Ein Schritt, diese Kombination zu ermöglichen, war die Einbindung von Logik direkt über die Menüs des GWT-Designers. Hier entwickelte ich zunächst Zusatzfunktionalität, die eine alternative Anbindung von UI-Handlern und Databindings an das User Interface ermöglichte (vgl. Kapitel 7.2.6). Da jedoch zur Modellierung der Businesslogik in Fujaba Informationen über die GUI notwendig werden konnten, entschloss ich mich, einen Mechanismus zur Rückgewinnung des User Interface Quelltextes und zur anschließenden Überführung in ein Diagramm zu entwickeln. Zur tatsächlichen Anbindung der Logik und zur Verwendung innerhalb logischer Operationen ist lediglich Wissen über die strukturelle Zusammensetzung der grafischen Benutzerschnittstelle notwendig. Die Übersetzung in konkrete Aktivitätsdiagramme zum Aufbau der GUI, wie dies mit dem zuerst verfolgten Konzept realisiert werden sollte, ist für die Anbindung der Logik nicht zwingend notwendig und wurde daher verworfen. Das Konzept sah nun vielmehr vor, in Fujaba ein zusätzliches Klassendiagramm für alle UI-Komponenten anzulegen und die strukturellen Informationen der Views dort abzulegen. Mittels einer in Fujaba verankerten Aktion kann das Einlesen des Quelltextes und das Erzeugen dieses View-Klassendiagrammes mit allen die GUI betreffenden Informationen angestoßen werden. Der Wechsel auf den komplett entwickelten GWT-Designer hatte den Nachteil, dass Änderungen an der GUI nun nicht mehr, wie zuerst geplant auch innerhalb von Fujaba vorgenommen werden konnten, brachte aber den enormen Vorteil mit sich, die komplexe GUI-Designer Software nicht selbst entwickeln zu müssen. Des weiteren konnte eine Anbindung der GUI an den gesamtheitlichen Modellierungsansatz über das Erzeugen der Klassendiagramme weiterhin gewährleistet werden. Die prinzipielle Möglichkeit der Anbindung an die modellierte Businesslogik stand für mich hier klar im Vordergrund. Hierfür genügten die erzeugten View-Klassendiagramme, die die exakte Struktur der Benutzerschnittstelle wiedergeben konnten. Diese Diagramme, zusammen mit den von uns erzeugten Methoden zum Anbinden von UI-Handlern und Databinding Mechanismen komplettierten den von mir vorgesehenen Ansatz und ermöglichten uns ein komplett auf Modellierung gestütztes Vorgehen. In Kapitel 7 wird im Bereich der Fujaba Webtools genauer auf die vorgenommenen Erweiterungen des GWT-Designers, sowie auf die Mechanismen zur Rückgewinnung des UI-Quelltextes in Fujaba eingegangen. Der User Interface relevante Teil meiner Arbeit war mit den zuvor beschriebenen Entscheidungen abgeschlossen.

Die Forschung war jedoch weiterhin nicht allein auf die Entwicklung von grafischen Be-

nutzerschnittstellen, sondern auf die gesamtheitliche Modellierung von Webapplikationen ausgelegt. Vor allem durch die im Rahmen des FAST-Projektes angestrebte Kombination einzelner Gadgets durch die Verwendung von Ports wurde ich gehäuft mit Applikationen konfrontiert, die darauf aufbauen, dass einzelne Aufgaben in einer mehr oder weniger definierten Reihenfolge nacheinander abgearbeitet werden, um eine größere Gesamtaufgabe zu erfüllen. Dies ist im zuvor erwähnten Todo-Gadget aus Abbildung 2.4 aufgrund der Eigenschaft einer ToDo Liste besonders ausgeprägt, fand sich aber bei allen Appliaktionen im FAST Kontext. Diese definieren über die zur Verfügung gestellten Ports eine Reihe an Services, die nacheinander besucht werden müssen, um die Gesamtaufgabe - im Fall des ToDo-Gadget "Have a meeting" - durchzuführen. Dieses Verhalten entstand bei den Applikationen im FAST Kontext implizit - durch die Kombination unterschiedlicher Gadgets im Rahmen einer Mashup-Plattform - bei der Entwicklung des Todo-Gadget wurde dieser Umstand jedoch explizit bei der Entwicklung bedacht. Die Kombination der Services bildet eine Art Workflow, der von der Applikation während ihrer Laufzeit schrittweise abgearbeitet wird. Auch viele der sogenannten Rich Internet Applications (RIAs) zeigten ähnliches Verhalten. Hier war der Kontrollfluss der Applikation so gestaltet, dass diese nur eine explizite Reihenfolge der Bearbeitung einzelner Prozessschritte zulässt. Dieses Verhalten der RIAs kann dann zum Problem werden, wenn sich der Kontrollfluss der Applikation und der zugrunde liegende Workflow nicht decken.

Solche Diskrepanzen wollte ich bei den mit meinem Ansatz entwickelten Applikationen möglichst vermeiden. Komplexe Web Applikationen meiner Art sollten systematisch anhand der bereits im Rahmen von Requirements Engineering Tätigkeiten festgehaltenen intendierten Nutzerworkflows entwickelt werden und somit nach Möglichkeit genau so funktionieren, wie erwartet. Um dies zu gewährleisten schien mir ein methodisches Vorgehen nötig, mit dem die grafischen Benutzerschnittstellen, die der Anwender sieht, und der Workflow, welcher der Applikation zugrunde liegt, auf sinnvolle Weise kombiniert werden können. Dieses Vorgehen sollte gleichzeitig das Abstraktionslevel während der Applikationsentwicklung anheben und Implementierungsdetails verstecken. Zu diesem Zweck entwickelte ich die Workflowdiagramme als gleichzeitigen Spezifikationsmechanismus für den Kontrollfluss der Applikation und die Abfolge der einzelnen grafischen Benutzerschnittstellen. Die Workflowdiagramme sind in Kapitel 5 genauer definiert. Dieser letzte gedankliche Schritt zur Spezifikation der Applikation über Workflowdiagramme versetzte mich schließlich in die Lage zur Definition des Gesamtprozesses des WFuP als Erweiterung des bereits bestehenden Fujaba Process. Um den WFuP in der Folge nicht zu einem reinen Vorschlag für das Entwicklungsvorgehen verkommen zu lassen, entschied ich mich zur Umsetzung einer möglichst kompletten Toolunterstützung mit der Fujaba Toolsuite innerhalb der Eclipse Entwicklungsumgebung. Zu diesem Zweck war im Folgenden die Implementierung einiger Editoren sowie Projektwizards notwendig. Eine detaillierte Beschreibung des Web Fujaba Process inklusive der entstandenen Toolunter-

stützung zur modellgetriebenen Entwicklung komplexer Fujaba Web Applications (vlg. Kapitel 3) ist in Kapitel 7 zu finden.

3 Fujaba Web Application - Definition und Eigenschaften

Der im Rahmen dieser Arbeit vorgestellte WFuP (Kapitel 7) bildet den von mir gesteckten Rahmen für die Modellierung komplexer Webapplikationen mit Story-Driven Modeling Methodiken. Bei Definition und Erarbeitung des Prozesses lag der Fokus neben der komplett modellzentrierten Entwicklung der Applikation auch darauf, Applikationen mit dedizierten Eigenschaften zu erstellen und im Webbrowser verfügbar zu machen. Applikationen mit diesen Eigenschaften werden hier als Fujaba Web Applications definiert und zeichnen sich vor allem durch folgende Eigenschaften aus:

- **Umfangreiche Umbauten am Document Object Model** - Fujaba Web Applications zeichnen sich dadurch aus, dass sie auf Ajax-Technologien aufbauen. Es handelt sich hierbei explizit nicht um statische Seiten-basierte Applikationen, deren Infrastruktur über Links aufgebaut werden und die häufig wechselnde HTML-Seiten beinhalten. Wir forcieren einen Applikationsaufbau mit möglichst wenigen einzelnen Seiten. Notwendige Änderungen innerhalb einer Seite werden durch Umbauten am Document Object Model realisiert, ohne dabei die Seite zu verlassen. Dieses Verhalten ist bereits im Rahmen sogenannter Rich Internet Applications bekannt.

- **Datenmodell und Logik auf dem Client** - Fujaba Web Applications lagern so viel des Datenmodells und der Businesslogik wie nur möglich auf den Client aus. Die im Webbrowser integrierte Ajax Engine übernimmt in diesem Fall die Berechnungen der Businesslogik und die Änderungen am Datenmodell. Fujaba Web Applications kontaktieren den Server nur in Spezialfällen, in welchen der Client allein die Berechnung nicht durchführen kann oder die Logik aus Sicherheitsgründen auf den Server verlagert werden muss.

- **Grafische Benutzeroberfächen ähnlich einer Destop-Applikation** - Fujaba Web Applications besitzen grafische Benutzerschnittstellen, die denen einer Desktopapplikation ähneln. Sie werden aus sogenannten Widgets zusammengesetzt und beinhalten klassiche Bedienelemente, wie beispielsweise Listen oder Dropdown-Boxen über welche Änderungen am zugrunde liegenden Datenmodell ausgelöst werden können.

- **Minimaler serverseitiger Code** - Fujaba Web Applications reduzieren die serverseitigen Berechnungen soweit wie möglich. Lediglich sicherheitsrelevante Bereiche der Applikation werden auf dem Server berechnet. Eine persistente Speicherung des Datenmodells auf dem Server, sowie eine Replikation der Daten zwischen Server und Clients ermöglichen den Multi-Client-Betrieb.

- **Workflow getrieben** - Fujaba Web Applikationen folgen einem zugrunde liegenden Workflow. Der Benutzer wird in der Abarbeitung von Aufgaben optimal von der Applikation unterstützt, da bereits bei der Entwicklung der dazu notwendige Workflow definiert wurde. Hierbei verstehen wir als Workflow nicht zwingendermaßen einen durch ein Business Process Model definierten Prozess, sondern vielmehr eine Abfolge (wiederkehrender) Aufgaben.

Die genaue Definition der Art von Applikationen, die durch Anwendung und Einhaltung des Prozesses entstehen, ist notwendig, um dem Anwender des Prozesses einen Leitfaden zu geben, der ihm hilft abzuwägen, ob die Entwicklung der von ihm angedachten Applikation mittels des WFuP für ihn passend ist. Über die oben beschriebenen allgemeinen Eigenschaften einer Fujaba Web Application hinaus werden durch die Abarbeitung des WFuP mit dem in dieser Arbeit vorgestellten Tooling (Kapitel 7.2) einige weitere Eigenschaften festgelegt. Sind diese - durch das vorhandene Tooling indizierte - Eigenschaften nicht gewünscht, können sie durch entsprechende Anpassung im Tooling nach eigenem Ermessen abgeändert werden. Eine dieser Eigenschaften ist die Nutzung des Google Web Toolkit (GWT) [GWTa] zur Implementierung der Applikation. Die Nutzung von GWT wird durch entsprechende Trigger innerhalb von Fujaba zur anschließenden Codegenerierung eingestellt (siehe Kapitel 4). Eine Anpassung der Codegenerierung kann hier zu einer Änderung des gewählten Frameworks, oder aber zur direkten Generierung von JavaScript Quelltext für den Client durchgeführt werden. Wird auf die Modellierung der Applikation mit Hilfe von Story-Driven Modeling verzichtet, so können Grundideen des in Kapitel 7 beschriebenen Entwicklungsprozesses ebenso mit anderen Tools oder Entwicklungsumgebungen angewandt werden. Die hier dargestellten und sowohl durch Tooling, als auch durch den Prozess vorgegebenen Eigenschaften einer Fujaba Web Application wurde erstmals in [EGHZ12] vorgestellt.

4 Fujaba Web Graphs - Replizierte Laufzeitdaten und angewandte Graphtransformationen auf dem Webclient

In den vorangegangenen Kapiteln wurde bereits darauf eingegangen, wie die Etablierung und Ausweitung des Internets, sowie die immer potenter werdenden Webbrowser die Angebote von Applikationen im Internet verändert haben. Kapitel 3 definiert die Eigenschaften der im Rahmen dieser Arbeit entwickelten Fujaba Web Applications. Dort wird als eine der wichtigen Eigenschaften die möglichst vollständige Verschiebung der relevanten Datenmodell-Anteile in den Client beschrieben. Zur Erfüllung dieser Anforderung ist es im Rahmen des in Kapitel 7 beschriebenen Entwicklungsprozesses zwingend notwendig, aus Fujaba heraus Datenmodell erzeugen zu können, die innerhalb der Ajax-Engine des Webbrowsers nicht nur geladen, sondern auch verändert werden können. Die zur Veränderung der Modelle notwendige Businesslogik soll gemäß des verfolgten Story-Driven-Modeling Paradigmas mit Fujaba modelliert, generiert und anschließend im Webclient ausgeführt werden. Hierbei ist zu beachten, dass innerhalb von Fujaba Graphtransformationen die Grundlage aller Operationen auf Datenmodellen darstellen. Da nach Möglichkeit weiterhin die in Fujaba vorhandenen Möglichkeiten zur Modellierung der Businesslogik verwendet werden sollen ist es notwendig, die Graphtransformationen so anzupassen, dass sie auch innerhalb der Ajax-Engine des Webbrowsers lauffähig sind.

Für die persistente Speicherung von Datenmodellen kommt im Rahmen des Story-Driven-Modeling mit Fujaba traditionell das von Christian Schneider entwickelte CoObRA Framework [Sch07] zum Einsatz. Dieses Framework kann gleichsam zur Verteilung von Datenmodellen zwischen mehreren Rechnerinstanzen verwendet werden. Die zur Modellierung der Businesslogik benötigten Graphtransformationen werden in Form von Activity-Diagrammen in Fujaba spezifiziert und anschließend als ausführbarer Java-Quelltext generiert. Diese beiden Grundbausteine einer mit Fujaba modellierten Applikation sollen auch für Fujaba Web Applications bestehen bleiben. Zunächst werden deshalb das CoObRA Framework, sowie Graphtransformationen näher beschrieben.

4.1 Beispiel - Kanban-Board

Zum leichteren Verständnis der relevanten Sachverhalte wird im weiteren Verlauf dieses Kapitels das Beispiel eines Kanban-Boards als zu modellierende Applikation eingesetzt.

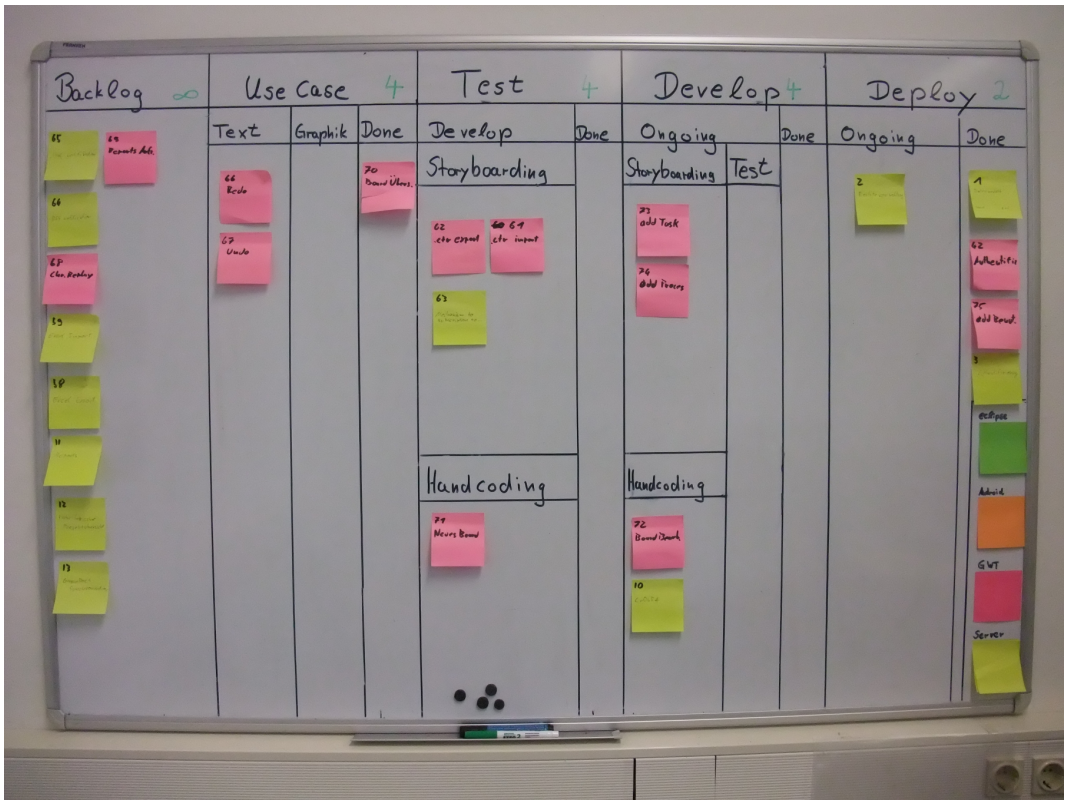

Abbildung 4.1: Kanban-Board eines Test-First Entwicklungsprozesses

Kanban wurde in seiner ursprünglichen Form von Toyota erarbeitet, um die Verfügbarkeit und den Fluss von Materialien in den Fertigungsstraßen der Toyota Fabriken zu kontrollieren und zu steuern, vgl. [CMCM11]. Das zur Produktionssteuerung erdachte Verfahren fand jedoch - in angepasster Form - bald auch seinen Platz im Rahmen von Softwareentwicklungsprozessen. Als Steuerungs- und Überwachungsintrument werden bei Kanban sogenannte Kanban-Boards eingesetzt.

Abbildung 4.1 zeigt das Kanban-Board für einen Test-First Softwareentwicklungsprozess. Jeder einzelne Prozesschritt wird auf dem Kanban-Board als Spalte dargestellt. Die einzelnen zu entwickelnden Features werden auf Karten geschrieben und durchlaufen anschließend den Entwicklungsprozess, indem die Karten von Spalte zu Spalte geschoben werden. Im Fall des Test-First Softwareentwicklungsprozesses treten die Prozessschritte *Use Case*, *Test*, *Develop* und *Deploy* auf. Ergänzt werden die Prozessschritte durch eine

Spalte namens *Backlog*. Dieses *Backlog* befindet sich ganz links im Kanban-Board und enthält alle zu entwickelnden Features der Software. Die Spalte dient gewissermaßen als Sammelbecken für zu entwickelnde Features. Die nachfolgenden Prozessschritte können nun sukzessive aus diesem Backlog befüllt werden. Hierbei ist zu beachten, dass bei einem Vorgehen nach Kanban die einzelnen Features immer vom nachfolgenden Prozessschritt gezogen werden, niemals werden Features auf dem Kanban-Board vorwärts geschoben. Zur besseren Ablaufplanung sind aus diesem Grund die Prozessschritte im Beispiel nochmals in Spalten aufgeteilt, die nacheinander durchlaufen werden. Lediglich Features, die sich innerhalb eines Prozessschrittes innerhalb der *Done*-Spalte befinden, können vom nächsten Schritt gezogen werden. Das in Abbildung 4.1 dargestelle Kanban-Board visualisiert einen Test-First Prozess, in welchem zum einen handgeschriebener Quelltext produziert, zum anderen jedoch auch Quelltext mit Fujaba generiert wird. Aus diesem Grund beinhalten die Prozessschritte *Test* und *Develop* jeweils parallele Unterprozesse für *Storyboarding* und *Handcoding*. Durch die parallele Anordnung dieser Unterprozesse wird symbolisiert, dass ein Feature nur einen dieser Unterprozesse durchlaufen muss, um die Spalte *Done* zu erreichen. Zur Steuerung der Entwicklungsaktivitäten können bei Kanban einzelne Prozessschritte mit Zahlen versehen werden, die angeben, wie viele Features mindestens und wie viele Features maximal in einem Prozessschritt enthalten sein dürfen. Ist die angegebene Obergrenze eines Prozessschrittes erreicht, dürfen keine weiteren Features in den Schritt nachgezogen werden, solange nicht vom nachfolgenden Prozessschritt seinerseits Features geholt und somit Platz freigemacht wurde. Die so beschriebene Überwachung von Entwicklungsprozessen mittels Kanban eignet sich insbesondere für die Entwicklung innerhalb von Teams. Aus diesem Grund ist eine der Grundvoraussetzungen für Kanban, dass alle Teammitglieder jederzeit in der Lage sind, das Kanban-Board zu sehen. Auf diese Weise kann sichergestellt werden, dass jeder Anteil an der Überwachung des Gesamtfortschritts nimmt und bei etwaigen Problemen im Prozessablauf eingreifen kann.

Traditionelle Kanban-Boards, wie etwa das in Abbildung 4.1 gezeigte, stellen sich immer dann als problematisch dar, wenn das Team über mehrere Orte verteilt ist. Ein gleichzeitiger gesicherter Zugriff auf das Kanban-Board kann nicht jederzeit gewährt werden, zudem können Änderungen am Kanban-Board nur von dem Teil des Teams vorgenommen werden, welches tatsächlich Zugriff auf die Tafel und die Feature-Karten hat. Als Lösung bietet sich eine verteilte online Version des Kanban-Boards an. Das Kanban-Board erfüllt hierbei gleichsam alle in Kapitel 3 gelisteten Eigenschaften einer Fujaba Web Applikation. Die Entwicklung eines Kanban-Boards stellt somit ein gutes Beispiel für den Einstieg in die Erläuterung der Entwicklung und die notwendigen Grundlagen CoObRA und Graphtransformationen dar. Das digitale Kanban-Board als verteilte Webanwendung kann in hohem Maße von der mit WebCoObRA ermöglichten Datenreplikation profitieren. Alle Datenmodellanteile werden auf den Client ausgelagert und mittels des WebCoObRA

Frameworks zwischen den unterschiedlichen angemeldeten Clients repliziert. Auf diese Weise kann jeder Client in Echtzeit alle Änderungen nachverfolgen. Hierfür verwendet das digitale Kanban-Board graphbasierte Datenmodelle und Graphtransformationen zur Spezifikation der Businesslogik.

4.2 Hintergrundwissen

4.2.1 CoObRA Framework

Das CoObRA Framework, wie es als Grundlage der vorliegenden Arbeit eingesetzt wird, wurde von Christian Schneider bereits im Rahmen seiner Diplomarbeit entworfen und im Rahmen der Dissertation [Sch07] in vollem Funktionsumfang vorgestellt. Das Framework stellt im Kontext der vorliegenden Arbeit die grundlegende Funktionalität für persistentes Speichern und Datenreplikation bereit. Das CoObRA Framework basiert auf der Protokollierung atomarer Aktionen, welche Modelländerungen darstellen. Die protokollierten Aktionen können zur persistenten Speicherung im `ctr`-Format in einer Textdatei abgelegt werden. Listing 4.1 stellt den Auszug einer solchen Textdatei dar. Einzelne protokollierte Aktionen können außerdem sowohl vorwärts (Redo) als auch rückwärts (Undo) erneut auf das zugrunde liegende Datenmodell angewendet werden. Diese Vorgehensweise macht es möglich, Undo/Redo-Operationen auf Objektmodellen durchzuführen, dient aber auch als Grundlage für die Synchronisation der Datenmodelle auf unterschiedlichen Clients. Zur Kapselung mehrerer atomarer Aktionen in sogenannten Change-Sets implementiert CoObRA zudem ein Transaktionskonzept. Bei der Speicherung der atomaren Aktionen innerhalb der `ctr`-Datei werden für alle Objekte eindeutige IDs und für die Art der Operationen verschiedene Marker verwendet. Bei Verwendung von Fujaba werden automatisch ab der initialen Operation, also dem Anlegen eines Projektes in Fujaba alle durchgeführten Aktionen in der Projektdatei abgespeichert. Das erneute Laden eines mittels CoObRA gespeicherten Objektmodells kann somit durch ein erneutes Ausführen aller im Protokoll enthaltener Operationen umgesetzt werden.

```
1  h;CoObRA2 Change stream - Version ;0;3
2  h;Encoding;windows-1252
3  h;ApplicationModel;Fujaba
4  h;Application Name;Fujaba Tool Suite
5  h;Fujaba Version;5.3.2
6  h;reuseFAM;true
7  [...]
8  t;4UVDW#8C;editClass;1447080470391;-;;
9  c1;;v::de.uni_paderborn.fujaba.uml.structure.UMLClass;i:4UVDW#9C;-;i:4UVDW#8C;
10 c3;;i:4UVDW#9C;declaredInPackage;i:9U_s_#92;-;-;i:4UVDW#8C;
11 c1;;v::de.uni_paderborn.fujaba.uml.common.UMLFile;i:4UVDW#AC;-;i:4UVDW#8C;
```

```
12 c3;;i:4UVDW#AC;contains;i:4UVDW#9C;-;v:java.lang.Integer:0;i:4UVDW#8C;
13 c3;;i:4UVDW#9C;file;i:4UVDW#AC;-;-;i:4UVDW#8C;
14 c3;;i:4UVDW#9C;declaredInPackage;-;i:9U_s_#92;-;i:4UVDW#8C;
15 c3;;i:4UVDW#AC;contains;-;i:4UVDW#9C;v:java.lang.Integer:0;i:4UVDW#8C;
16 c3;;i:4UVDW#9C;file;-;i:4UVDW#AC;-;i:4UVDW#8C;
17 c3;;i:4UVDW#9C;name;v::QBan;-;-;i:4UVDW#8C;
18 c3;;i:4UVDW#9C;declaredInPackage;i:9U_s_#92;-;-;i:4UVDW#8C;
19 c3;;i:4UVDW#AC;contains;i:4UVDW#9C;-;v:java.lang.Integer:0;i:4UVDW#8C;
20 c3;;i:4UVDW#9C;file;i:4UVDW#AC;-;-;i:4UVDW#8C;
21 c3;;i:4UVDW#9C;declaredInPackage;i:9U_s_#13;i:9U_s_#92;-;i:4UVDW#8C;
22 c3;;i:4UVDW#9C;stereotypes;i:9U_s_#F1;-;v:java.lang.String:JavaBean;i:4UVDW#8C;
23 c3;;i:9U_s_#F1;increments;i:4UVDW#9C;-;-;i:4UVDW#8C;
24 c3;;i:9U_s_#A2;elements;i:4UVDW#9C;-;-;i:4UVDW#8C;
25 c1;;v::de.uni_paderborn.fujaba.asg.ASGUnparseInformation;i:4UVDW#BC;-;i:4UVDW#8C;
26 c3;;i:4UVDW#9C;unparseInformations;i:4UVDW#BC;-;i:9U_s_#A2;i:4UVDW#8C;
27 c1;;v::de.uni_paderborn.fujaba.asg.ASGInformation;i:4UVDW#CC;-;i:4UVDW#8C;
28 c3;;i:4UVDW#BC;aSGInformation;i:4UVDW#CC;-;v:java.lang.String:entry;i:4UVDW#8C;
29 c3;;i:4UVDW#CC;information;v::401;-;v:java.lang.String:location_X;i:4UVDW#8C;
30 c3;;i:4UVDW#CC;information;v::119;-;v:java.lang.String:location_Y;i:4UVDW#8C;
31 c1;;v::de.uni_paderborn.fujaba.asg.ASGInformation;i:4UVDW#DC;-;i:4UVDW#8C;
32 c3;;i:4UVDW#BC;aSGInformation;i:4UVDW#DC;-;v:java.lang.String:attributePanel;i:4UVDW
       #8C;
33 c3;;i:4UVDW#DC;information;v::false;-;v:java.lang.String:collapsed;i:4UVDW#8C;
34 c1;;v::de.uni_paderborn.fujaba.asg.ASGInformation;i:4UVDW#EC;-;i:4UVDW#8C;
35 c3;;i:4UVDW#BC;aSGInformation;i:4UVDW#EC;-;v:java.lang.String:methodPanel;i:4UVDW#8C
       ;
36 c3;;i:4UVDW#EC;information;v::false;-;v:java.lang.String:collapsed;i:4UVDW#8C;
37 c1;;v::de.uni_paderborn.fujaba.asg.ASGInformation;i:4UVDW#FC;-;i:4UVDW#8C;
38 c3;;i:4UVDW#BC;aSGInformation;i:4UVDW#FC;-;v:java.lang.String:signalPanel;i:4UVDW#8C
       ;
39 c3;;i:4UVDW#FC;information;v::false;-;v:java.lang.String:collapsed;i:4UVDW#8C;
40 c3;;i:4UVDW#9C;diagrams;i:9U_s_#A2;-;v:java.lang.Integer:0;i:4UVDW#8C;
41 c3;;i:4UVDW#9C;defaultIcon;v::;-;-;i:4UVDW#8C;
```

Listing 4.1: Auszug aus dem Coobra Protokoll zum Erzeugen der Qban-Klasse aus dem Klassendiagramm in Abbildung 4.2

Listing 4.1 zeigt einen Ausschnitt aus der Protokolldatei zur Erstellung des Klassendiagramms aus Abbildung 4.2. Dieses Klassendiagramm setzt das in Abbildung 4.1 vorgestellte Kanban-Board für einen Test-First Softwareentwicklungsprozess um. Im Klassendiagramm in Abbildung 4.2 stellt die Klasse `QBan` die Hauptklasse des Datenmodells dar. Sie bildet den Einstiegspunkt zur Replikation des Datenmodells mit dem adaptierten WebCoObRA Framework. Die Klasse `Board` stellt die Repräsentation des gesamten Kanban-Boards im Datenmodell dar. Sie verfügt über ein Attribut für den Namen des Boards. Die Klasse `ProcessStep` entspricht einem Prozessschritt auf dem Kanban-Board und kann über die Assoziation `processSteps` ins `Board` eingefügt werden. Zur Mo-

dellierung verschachtelter Prozessschritte ist die Selbstassoziation `parentProcessStep` vorhanden. Die Reihenfolge der Prozessschritte wird mit der Assoziation `nextProcess` modelliert. Einzelne abzuarbeitende Features eines Kanban-Boards werden im Klassendiagramm durch die Klasse `Task` modelliert. Die Besonderheiten der Klasse `QBan`, sowie die dort verwendeten Stereotypen und die im Klassendiagramm aufgeführten Annotationen werden im weiteren Verlauf des Kapitels in Abschnitt 4.3.1 genauer beschrieben. Gleiches gilt für die Klassen `CObject` und `ModelRootEventListener`. Das Listing 4.1 zeigt die in der Protokolldatei gespeicherte Transition zum Erzeugen und benennen der Klasse QBan. In Zeile 8 des Listings wird die Transition gestartet und die ID `4UVDW#8C` zugewiesen. Die Markierung `c1` in Zeile 9 zeigt eine erzeugende Operation an. In diesem Fall wird ein Objekt vom Typ `UMLClass` erzeugt. In den folgenden Zeilen wird die UML-Klasse einem `Package` zugewiesen, sowie ein `File` für die Klasse angelegt. In Zeile 17 schließlich wird über eine mit dem Marker `c3` gekennzeichnete Editieroperation der name der Klasse auf `QBan` gesetzt. Eine genaue Definition der CoObRA Semantik ist [Sch07] zu entnehmen.

Neben vielen vor allem für die Bedürfnisse des vorgestellten Entwicklungsvorgehens dargestellten Vorteilen bei der Nutzung des CoObRA Frameworks, gibt es auch kleine Einschränkungen. Bei der Verwendung von CoObRA ist es zum jetzigen Zeitpunkt notwendig, alle zu bearbeitenden Daten im Hauptspeicher zu halten. Auf allen 32-bit Systemen führt dies zwangsläufig zu einer Beschränkung der Datenmodellgröße auf 2 Gigabyte. Um diese Einschränkung vor allem im Hinblick auf größer angelegte Systeme zu minimieren ist eine Kombination der CoObRA-Mechanismen mit einer Datenbank (relational oder nosql) als Lösung für die Zukunft eine Option. Eine umfangreiche Definiton und Funktionsbeschreibung des CoObRA-Frameworks, sowie des zugehörigen Textprotokolls ist aus [Sch07] zu entnehmen. Im Rahmen der vorliegenden Arbeit wurde die in Fujaba verwendete Implementierung des CoObRA Frameworks leicht modifiziert und den speziellen Anforderungen der Fujaba Web Applications angepasst, vgl. Abschnitt 4.3.1.

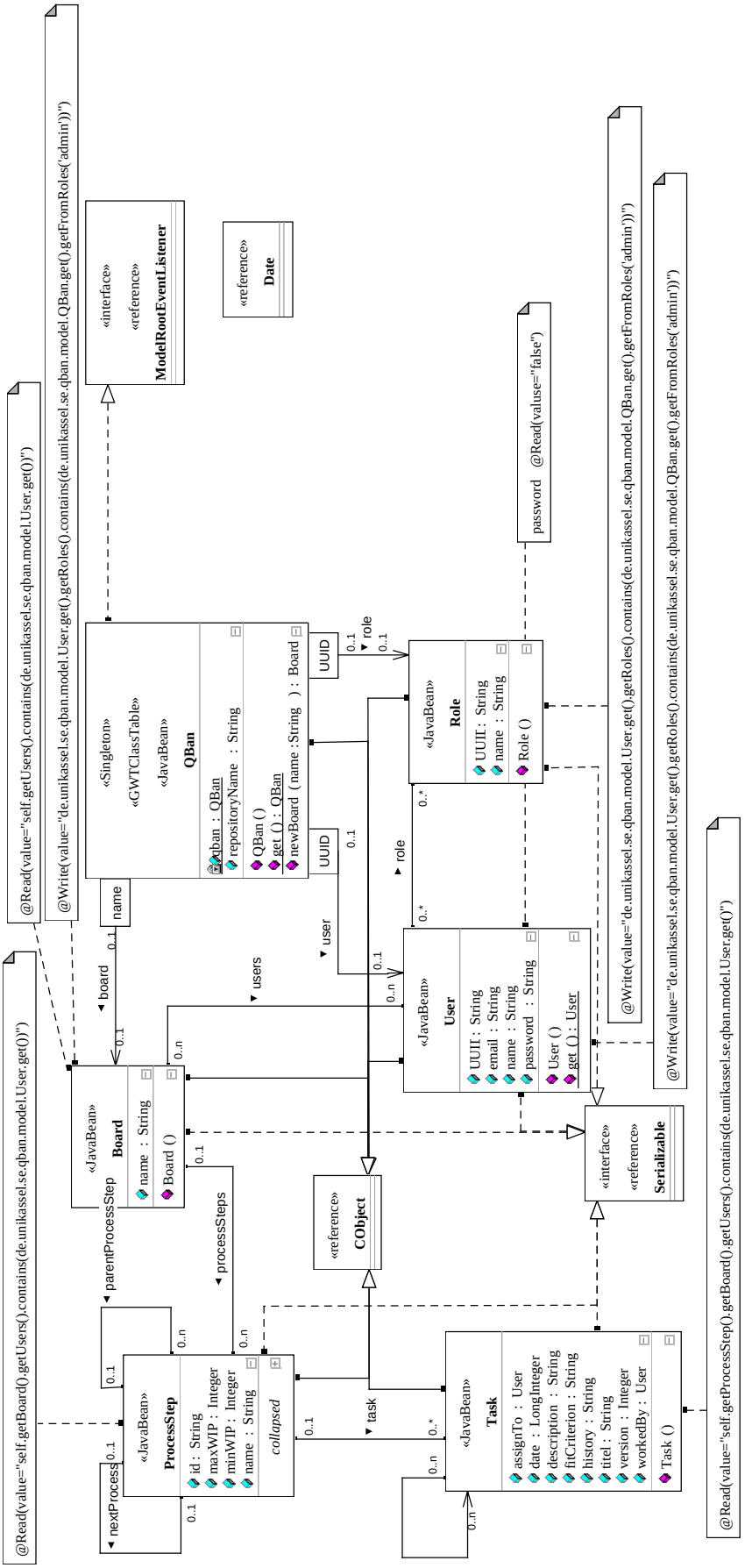

Abbildung 4.2: Klassendiagramm des digitalen Kanban-Board

4.2.2 Graphtransformationen

Graphtransformationen, wie sie im Rahmen dieser Arbeit für die Ausführung von Businesslogik auf dem Webclient benutzt werden, basieren auf den innerhalb der Fujaba Toolsuite eingesetzen Implementierungen. Diese wiederum bauen auf den Graphkonzepten, welche in [Zün01] vorgestellt werden auf. Hier wird die theoretische Grundlage für alle mit Fujaba realisierten Diagrammarten und Ausführungsmechanismen von Graphgrammatiken gelegt. Die im Rahmen dieser Arbeit verwendeten Ansätze verwenden Fujaba als Modellierungswerkzeug der Wahl und bauen somit ebenfalls auf den in [Zün01] beschriebenen Mengentheoretischen Grundlagen auf. Basis aller verwendeten Graphen und Operationen auf diesen bilden die gerichteten, attributierten, knoten- und kantenmarkierten Graphen (gakk-Graphen) wie sie analog zu [Zün01] in Definition 4.1 eingeführt werden.

Definition 4.1 (Graph) *Ein gakk-Graph definiert sich wie folgt:*

$\mathtt{G} := (\mathtt{NL}, \mathtt{EL}, \mathtt{A}, \mathtt{N}, \mathtt{E}, \mathtt{l}, \mathtt{av})$ *mit*

$\mathtt{NL}$ *endliche Menge von Knotenmarkierungen / −typen / −labeln*

$\mathtt{EL}$ *endliche Menge von Kantenmarkierungen / −typen / −labeln*

$\mathtt{A}$ *endliche Menge von Attribut(nam)en*

$\mathtt{N}$ *endliche Menge von Knoten(bezeichnern)*

$\mathtt{E} \subseteq \mathtt{N} \times \mathtt{EL} \times \mathtt{N}$

$\mathtt{l} \quad \mathtt{N} \rightarrow \mathtt{NL}$

$\mathtt{av} \quad \mathtt{N} \times \mathtt{A} \rightarrow \{\mathtt{true}, \mathtt{false}\} \cup \mathtt{N} \cup \mathtt{CHAR}* \cup \ldots \cup \mathtt{P(N)} \cup \mathtt{P(CHAR}*\mathtt{)} \cup \mathtt{P}(\ldots)$

Gemäß Definition 4.1 bestehen gakk-Graphen immer aus einer endlichen Menge an Elementen. Mit Mengentheoretischen Operationen wird hier sowohl das Metamodell des betreffenden Graphen beschrieben, wie auch die konkrete Ausprägung. Zum Bereich des Metamodells gehören die Knotentypen (NL) und Kantentypen (EL). Diese geben die Typen von Knoten und Kanten an. Die Attributnamen (A) geben an, welche Attribute innerhalb der Knotentypen vorhanden sein können. Die Knotenbezeichner (N) stellen die konkreten Auspräfungen von Knoten innerhalb des Graphen dar. Durch die Abbildung einer Funktion (l) wird anschließend jedem Knoten des Graphen der entsprechende Typ zugeordnet. Die Kantenverbindungen werden in der Menge E festgehalten, hier wird zu jeweils zwei konkreten Knoten ein Kantentyp angegeben. Die Belegung der Attribute schließlich wird über die Funktion av als Menge von Knoten und Attribut abgebildet auf den entsprechenden Wert angegeben. Mit der in Definition 4.1 dargestellten Mengentheoretischen Schreibweise können beliebige Graphen abgebildet werden. Diese Abbildung der Graphen spiegelt die in Fujaba enthaltenen internen Graphstrukturen wieder und bildet somit auch die Grundlage der innerhalb der Fujaba Web Applications verwendeten Graphen (Datenmodelle). Auf den zuvor definierten gakk-Graphen können nun

Graphtransformationen definiert werden. Dies geschieht auf theoretischer Ebene gleichsam mittels Mengentheorie, in Fujaba sind die Mengentheoretischen Grundlagen in Form von Diagrammen implementiert. Die Definition von Graphtransformationen geschieht nach [Zün01] als Definition von Graphersetzungsregeln auf der zugehörigen Graphklasse. Hierbei ist die Graphklasse nach Definition 4.2 eine Zusammenfassung von gakk-Graphen mit gleichem Metamodell. Die Graphklasse gibt über die Mengen NL, EL und A genau die Modellanteile des gakk-Graphen an. Alle Graphen, die zu einer Graphklasse gehören, besitzen die gleichen Mengen für Knotentypen, Kantentypen und Attribute. Ist die Graphklasse und somit das Metamodell eines Graphen bekannt, reicht zur weiteren Definition des Graphen eine Angabe aller konkreten Knoten N, Kanten E, Knotenmarkierungen l und Attributfunktionen av gemäß Definition 4.1.

Definition 4.2 (Graphklasse) *Die Graphklasse eines gakk-Graphen definiert sich wie folgt:*

- $\texttt{GraphClass}(\texttt{NL}, \texttt{EL}, \texttt{A})$ *bezeichnet die Menge aller Graphen* G *über* $\texttt{NL}, \texttt{EL}, \texttt{A}$
- *Ein Graph* $\texttt{G} \in \texttt{GC}$ *wird angegeben als* $(\texttt{N}, \texttt{E}, \texttt{l}, \texttt{av})$
- *Sei* $\texttt{G} \in \texttt{GC}$, *dann bezeichnet*
  $\texttt{NG} := \texttt{N}$ *die Knotenmenge von* G
  $\texttt{EG} := \texttt{E}$ *die Kantenmenge von* G
  $\texttt{lG} := \texttt{l}$ *die Knotenmarkierungsfunktion von* G
  $\texttt{avG} := \texttt{av}$ *die Attributierungsfunktion von* G

Die Graphklasse von Graphen gemäß des in Abbildung 4.2 angegebenen Klassendiagramms für das Beispiel QBan definiert sich laut 4.2 wie in Listing 4.2 angegeben.

```
1 GraphClass( NL, EL, A ) mit
2 NL := {QBan, Board, ProcessStep, Task, User, Role}
3 EL := {board, processSteps, parentProcessStep, nextProcess, task, users, user, role}
4 A  := {qban, repositoryName, UUID, name, email, password, id, maxWIP, minWIP, date,
     description, fitCriterion, history, titel, version}
```

Listing 4.2: Graphklasse für Graphen gemäß des im Klassendiagramm aus Abbildung 4.2 angegebenen Modells für Kanban-Boards

Als Graphersetzungsregel wird nach Definition 4.3 schließlich ein Paar von Graphen (LG und RG) derselben Graphklasse (GC) bezeichnet, die zudem über gleiche Knotenmarkierungen (l) verfügen. Trifft dies zu, so definiert die Graphersetzungsregel exakt die Mengen, in welchen sich die beiden Graphen unterscheiden. Es werden innerhalb der Graphersetzungsregel exakt die Knoten (DelN) und Kanten (DelE) angegeben, die bei

einem Vergleich von LG und RG gelöscht wurden. Zudem werden alle Knoten angegeben, die gleich bleiben (CoreN), diese Knoten bilden den Kontext der Graphersetzungsregel. Desweiteren werden alle Knoten angegeben, die in RG eingefügt werden müssen (AddN). Die hinzuzufügenden (AddToAddE, AddToCoreE, CoretToAddE) und die gleichbleibenden Kanten (CoreToCoreE) werden ebenfalls durch Angabe mehrerer Mengen in der Graphersetzungsregel festgehalten. Ein exaktes Abbilden der in der Graphersetzungsregel angegebenen Mengen auf einen Ausgangsgraphen der korrekten Graphklasse bildet so eine Änderung der Graphstruktur ab. Im Rahmen von Fujaba werden Graphersetzungsregeln innerhalb der Aktivitätsdiagramme verwendet, um Businesslogik zu modellieren.

Definition 4.3 (Graphersetzungsregel)
Eine Graphersetzungsregel ist ein Paar von Graphen $(\mathtt{LG}, \mathtt{RG})$ *mit* $\mathtt{LG}, \mathtt{RG} \in \mathtt{GC}$ *und* LG *und* RG *sind konsistent markiert, d.h.* $\mathtt{l}_{\mathtt{LG}} \mid \mathtt{N}_{\mathtt{LG}} \cap \mathtt{N}_{\mathtt{RG}} = \mathtt{l}_{\mathtt{RG}} \mid \mathtt{N}_{\mathtt{LG}} \cap \mathtt{N}_{\mathtt{RG}}$
Sei $\mathtt{grr} := (\mathtt{LG}, \mathtt{RG})$ *gegeben, dann bezeichnen*
$\mathtt{DelN}_{\mathtt{grr}} := \mathtt{N}_{\mathtt{LG}} - \mathtt{N}_{\mathtt{RG}}$ *die Menge der von* grr *zu löschenden Knoten*
$\mathtt{DelE}_{\mathtt{grr}} := \mathtt{E}_{\mathtt{LG}} - \mathtt{E}_{\mathtt{RG}}$ *die Menge der von* grr *zu löschenden Kanten*
$\mathtt{CoreN}_{\mathtt{grr}} := \mathtt{N}_{\mathtt{LG}} \cap \mathtt{N}_{\mathtt{RG}}$ *die Kern − oder Kontextknoten von* grr
$\mathtt{AddN}_{\mathtt{grr}} := \mathtt{N}_{\mathtt{RG}} - \mathtt{N}_{\mathtt{LG}}$ *die Menge der von* grr *neu zu erzeugenden Knoten*
$\mathtt{AddToAddE}_{\mathtt{grr}} := \mathtt{E}_{\mathtt{RG}} \cap (\mathtt{AddN}_{\mathtt{grr}} \times \mathtt{EL} \times \mathtt{AddN}_{\mathtt{grr}})$
$\mathtt{AddToCoreE}_{\mathtt{grr}} := \mathtt{E}_{\mathtt{RG}} \cap (\mathtt{AddN}_{\mathtt{grr}} \times \mathtt{EL} \times \mathtt{CoreN}_{\mathtt{grr}})$
$\mathtt{CoreToAddE}_{\mathtt{grr}} := \mathtt{E}_{\mathtt{RG}} \cap (\mathtt{CoreN}_{\mathtt{grr}} \times \mathtt{EL} \times \mathtt{AddN}_{\mathtt{grr}})$
$\mathtt{CoreToCoreE}_{\mathtt{grr}} := \mathtt{E}_{\mathtt{RG}} \cap (\mathtt{CoreN}_{\mathtt{grr}} \times \mathtt{EL} \times \mathtt{CoreN}_{\mathtt{grr}})$

Die in Fujaba implementierten Aktivitätsdiagramme abstrahieren von den zuvor aufgeführten Mengentheoretischen Repräsentationen. Durch die Verwendung einer visuellen Repräsentation für die Aktivitätsdiagramme werden die Angaben der in Definiton 4.3 angegebenen Mengen teilweise zusammengefasst oder durch unterschiedliche Farben gekennzeichnet. Gleichwohl bleibt die interne Funktionsweise der Mengentheorie in Graphersetzungsregeln erhalten.

In Fujaba werden Graphersetzungsregeln - namentlich Aktivitätsdiagramme - verwendet, um Änderungen an der zugrunde liegenden Graphstruktur abzubilden und auf diesem Weg Businesslogik zu implementieren. Die Ausführung der Graphersetzungsregeln erfolgt über die Erzeugung von ausführbarem Java-Quelltext aus den modellierten Regeln. Hierbei ist es während der Ausführung der Regeln zudem notwendig, Mappings der tatsächlich vorliegenden Datenmodelle auf die anzuwendenden Graphersetzungsregeln zu berechnen. Zur Berechnung eines Mappings sind die in Definiton 4.3 angegebenen Kontextknoten und Kontextkanten notwendig. Sie stellen den unveränderlichen Kern der Graphersetzungsregel dar.

Abbildung 4.3 zeigt das Aktivitätsdiagramm zum Löschen eines Prozessschrittes aus dem Kanban-Board. Hier kann die zuvor beschriebene Abstraktion über visuelle Elemente nachvollzogen werden. Es ist zur Spezifikation einer Graphersetzungsregel mit Fujaba nicht notwendig, zwei Graphen zu spezifizieren. In Abbildung 4.3 zeigt Schritt 0 einen Knoten `this` vom Typ `ProcessStep`, sowie einen weiteren Knoten `child` vom selben Typ. Beide Knoten sind über die Kante `parentProcessStep` miteinander verbunden. Alle in Schritt 0 abgebildeten Knoten und Kanten sind schwarz markiert. Sie bilden den Kontext der Graphersetzungsregel und werden nicht weiter verändert. In Schritt 1 des Aktivitätsdiagramms ist die Kante `parentProcessStep` rot eingefärbt und zudem mit dem Schlüsselwort «`destroy`» markiert. Dies repräsentiert einen Löschvorgang der entsprechenden Kante. Von der eigentlich notwendigen Angabe der Knoten- und Kantenmengen einer Graphersetzungsregel nach Definition 4.3 wird durch die visuelle Markierung abstrahiert. Ähnliche Abstraktionen werden in Schritt 2 des Aktivitätsdiagramms durch die Markierung in grün vorgenommen, welche ein Hinzufügen der entsprechend markierten Elemente im Rahmen der Graphersetzungsregel repräsentiert.

Allgemein werden Aktivitätsdiagramme innerhalb von Fujaba verwendet, um Businesslogik zu modellieren. Dies geschieht über die konkrete Implementierung einzelner Methoden der Datenmodell-Klassen. Der in Abbildung 4.3 dargestellte Löschvorgang eines `ProcessStep` wird über das Implementieren der Methode `removeProcessStep()` in der Klasse `ProcessStep` vorgenommen. Die Methode führt in Schritt 1 des Aktivitätsdiagrammes zunächst eine Prüfung auf mögliche vorhandene Unterschritte durch. Werden diese gefunden, wird dort ebenfalls die Methode `removeProcessStep()` aufgerufen. Sind keine Unterschritte vorhanden wird geprüft, ob der zu löschende Schritt selbst als Unterschritt im Kanban Board vorhanden ist. Dies geschieht über das Löschen der Kante `parentProcessStep`. Die Rotfärbung der entsprechenden Kante, sowie die Markierung «`destroy`» kennzeichnen den Löschvorgang. Ist der Löschvorgang erfolgreich, war ein Vaterknoten vorhanden, und alle im zu löschenden Prozesschritt vorhandenen Tasks müssen in den Vaterschritt verschoben werden. Dies wird durch das Löschen der entsprechenden Kante sowie den Aufruf der Methode `addTask(task)` auf dem Vater realisiert. Sind alle Kanten gelöscht und alle Tasks verschoben, kann über einen Aufruf der Methode `removeYou()` am zu löschenden Prozessschritt der Löschvorgang abgeschlossen werden. Ist der zu löschende Prozessschritt nicht als Unterschritt in einem Vaterknoten enthalten, so muss er aus der bestehenden Reihenfolge der Prozessschritte entnommen und alle vorhandenen Tasks in den zuvor zu bearbeitenden Schritt verschoben werden. Zu diesem Zweck werden wiederum die bestehenden Kanten `nextProcessStep` zwischen dem zu löschenden Schritt sowie Vorgänger und Nachfolger gelöscht und eine neue Kante (gekennzeichnet durch Grünfärbung und Markierung «`create`») zwischen Vorgänger und Nachfolger gezogen. Das Zuordnen der Tasks geschieht, wie bereits im Fall zuvor beschrieben, über Aufruf der Methode `addTask(task)` auf dem entsprechen-

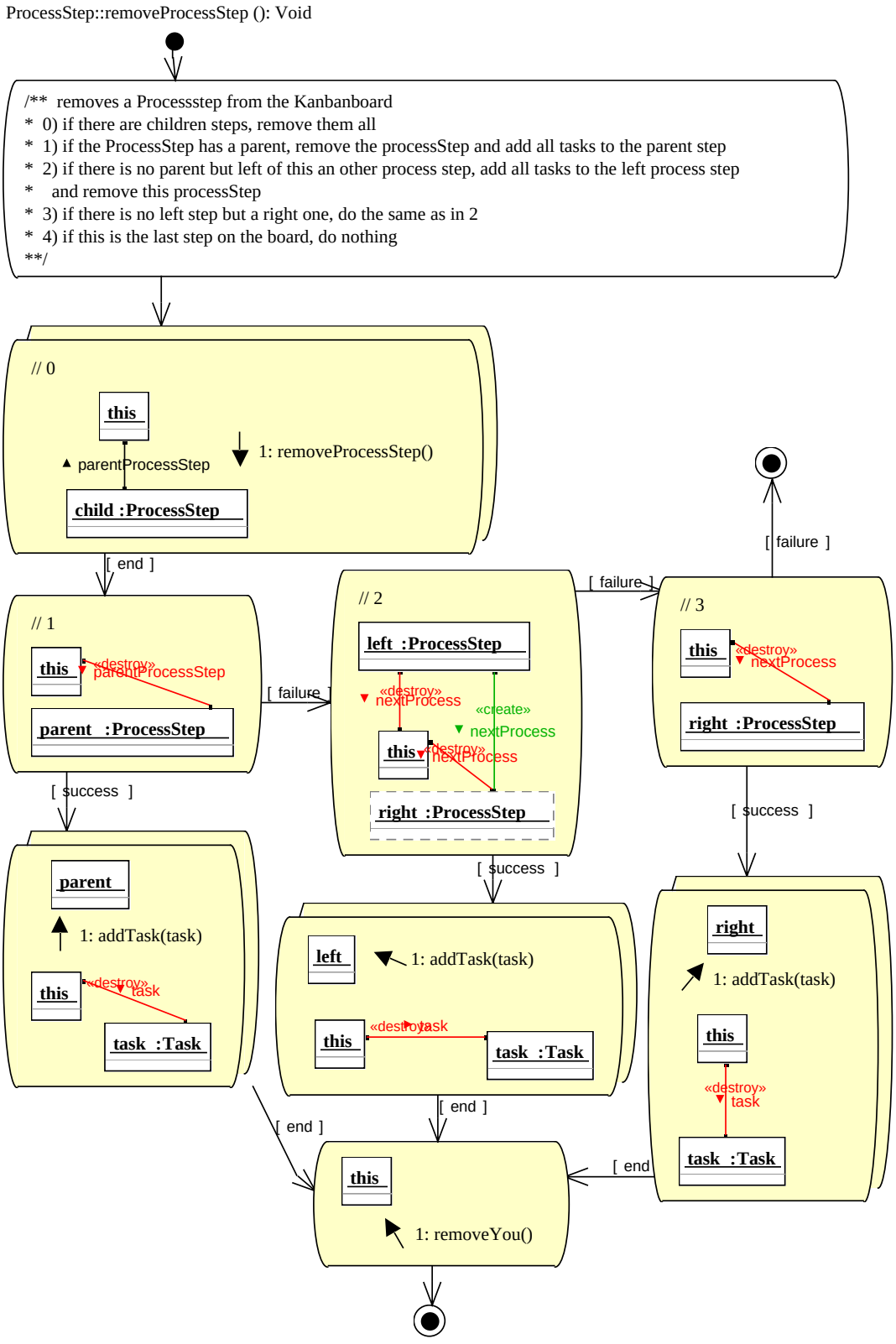

Abbildung 4.3: Aktivitätsdiagramm zum Löschen eines Prozessschrittes innerhalb des digitalen Kanban-Board

den Prozessschritt. Sollte kein Vorgänger vorhanden sein, werden alle Tasks stattdessen in den Nachfolger verschoben.

Abbildung 4.3 gibt einen Überblick über die Möglichkeiten, mit Fujaba Businesslogik zu modellieren. Auch wird an diesem Beispiel die von Fujaba vorgenommene visuelle Abstraktion von den in den Definitionen 4.1, 4.2 und 4.3 eingeführten theoretischen Grundlagen der Graphtransformationen dargestellt. Zur Nutzung der hier vorgestellten Techniken zur Modellierung clientseitigen Verhaltens im Webbrowser sind vor allem im Rahmen der zur Ausführung notwendigen Codegenerierung Anpassungen notwendig geworden. Die prinzipielle Verhaltensweise des im Beispiel gezeigten Kanban-Boardes lässt sich ohne weiteres mit den von Fujaba zur Verfügung gestellten Graphersetzungsregeln modellieren. Die vorgenommenen Anpassungen betreffen vor allem die von Fujaba verwendeten Laufzeitklassen zum Ausführen der Graphersetzungsregeln, der Codegenerierung, sowie für kompliziertere Sachverhalte eine leichte Modifikation der von Fujaba zur Verfügung gestellten Diagrammarten. Die weiteren Abschnitte dieses Kapitels werden detaillierter auf die bereits in Kapitel 2 angesprochenen Änderungen eingehen.

4.2.3 Google Web Toolkit

Das Google Web Toolkit (GWT) [GWTa] ist ein Projekt, das im Jahr 2006 von Google zur freien Nutzung zur Verfügung gestellt wurde. Unter dem Namen Google Web Toolkit vereinen sich hierbei unterschiedliche technologische Aspekte, die in ihrem Zusammenspiel das Entwickeln von Ajax-basierten Webapplikationen erleichtern sollten. Einen wichtigen Teil des Gesamtpaketes stellt hierbei der **Crosscompiler** dar. Hierbei handelt es sich um einen Compiler, der aus bestehenden Java Quelltexten keinen Bytecode, sondern ausführbaren JavaScript Quelltext erzeugt. Der Einsatz des Crosscompilers sollte es Entwicklern ermöglichen, eine Webapplikation möglichst in einer bereits bekannten Programmiersprache (Java) zu erstellen, ohne sich zunächst in neue Programmierumgebungen oder Sprachen einarbeiten zu müssen. Als besonderer Seiteneffekt werden vom Crosscompiler - je nach erkannten Browsertypen und im Sourcecode enthaltenen Sprachen (Internationalisierung) - dedizierte JavaScript Sourcen erzeugt, die auf den jeweiligen Browsertyp optimiert sind. Dies soll die pro Browser zu ladende Dateigröße minimieren und eine schnellere und stabilere Applikation zur Folge haben. Zur automatischen Übersetzung der Java Quelltexte mit dem Crosscompiler ist eine bestimmte Paketstruktur innerhalb der zu implementierenden Applikation einzuhalten. Die Applikation wird in drei wesentliche Pakete aufgeteilt, die wiederum beliebige Unterpakete enthalten dürfen. Am Beispiel der Applikation des beschriebenen Kanban-Boards wären dies beispielsweise die Pakete:

- `kanban.client`

- `kanban.server`
- `kanban.shared`

Das Paket `kanban.client` enthält alle Quelltexte, die ausschließlich auf dem Client laufen sollen, hier finden sich beispielsweise die Implementierungen für die GUI. Das Paket `kanban.server` beinhaltet alle rein serverseitigen Anteile der Applikation. Das Paket `kanban.shared` enthält alle Anteile der Applikation, die Client und Server teilen, beispielsweise das Datenmodell. Eine automatische Übersetzung nach JavaScript wird beim Kompilierprozess mit GWT lediglich für die Pakete `*.client` und `*.shared` sowie aller Unterpakete durchgeführt. Dies ist bei der Konzeption der Applikation zu beachten.

Der Crosscompiler spielt eng mit der in GWT integrierten **JRE Emulation** [JRE] zusammen. Diese Bibliothek emuliert einen Teil der in Java verfügbaren Standard-Klassenbibliothek und stellt deren Funktionalität auf Clientseite - sprich im JavaScript zur Verfügung. Die JRE Emulation stellt hierbei zum gegenwärtigen Zeitpunkt Subsets aus den folgenden Paketen bereit:

- `java.lang`
- `java.lang.annotation`
- `java.math`
- `java.io`
- `java.sql`
- `java.util` sowie `java.util.logging`

Die zur Verfügung gestellten emulierten Anteile haben sich hierbei in der Vergangenheit deutlich erhöht. Die Eingrenzung der JRE Emulation auf die genannten Pakete hat direkte Auswirkungen auf die in der Entwicklung der Zielapplikation zu verwendenden Anteile der Java-Standardbibliothek. Hier sollte stets sichergestellt sein, dass alle verwendeten Klassen und Pakete unterstützt werden und entsprechend in der JRE Emulation enthalten sind. Trifft dies nicht zu, kann der Java Quelltext nicht automatisch vom Crosscompiler in JavaScript Quelltext übersetzt werden.

Zur Vereinfachung der Client-Server-Kommunikation bietet GWT das Konzept der **Remote-Procedure-Calls** (RPC). Die grundlegende Architektur einer Kommunikation über GWT-RPC ist in Abbildung 4.4 dargestellt.

Die Kommunikation ist hierbei in Client- und Serveranteile aufgeteilt. Auf dem Server läuft ein (Java) Service, der aus dem Client durch Aufrufe der entsprechenden Methoden

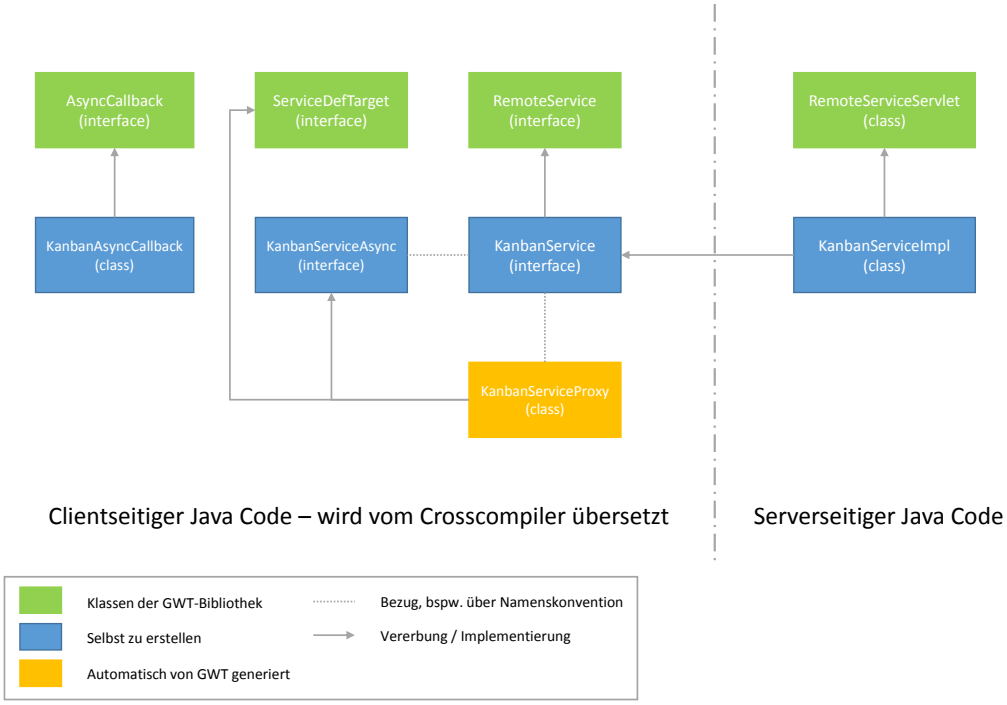

Abbildung 4.4: Kommunikation über GWT-Remote Procedure Calls

angesprochen werden kann. Hierbei kapselt GWT die komplette HTTP-Kommunikation und ermöglicht ein einfaches Senden von Java Objekten vom Client zum Server und umgekehrt. Aufrufende Clients müssen hierbei stets ein Callback-Objekt mit an den Server übergeben, über welches die asynchrone Antwort des Servers verarbeitet wird. Auf diese Weise kann realisiert werden, dass die im Client laufende Applikation während der Bearbeitung der Anfrage auf dem Server nicht blockiert und vom Benutzer weiter verwendet werden kann. Zur Implementierung der Kommunikation über RPC stellt GWT die in Abbildung 4.4 in rot dargestellten Interfaces `ServiceDefTarget` und `RemoteService`, sowie die Klasse `RemoteServiceServlet` bereit. Zur Etablierung der RPC-Kommunikation sind im clientseitigen Quelltext zwei Interfaces zu erzeugen. Im Beispiel aus Abbildung 4.4 sind dies die Interfaces `KanbanService`, sowie `KanbanServiceAsync`.

- `KanbanService` leitet vom GWT-Interface `RemoteService` ab und listet alle über die Kommunikation zur Verfügung gestellten Serveraufrufe als Methoden auf.
- `KanbanServiceAsync` beinhaltet alle im `KanbanService` angegebenen Methoden und ergänzt diese um das oben erwähnte Callback-Objekt.

Die Implementierung des als Callback angegebenen Objektes kann durch eine beliebige Klasse, die das von GWT zur Verfügung gestellte Interface `AsyncCallback`

implementiert, realisiert werden. Die realisierende Klasse - in Abbildung 4.4 `KanbanAsyncCallback` - implementiert die im Interface `AsyncCallback` definierten Methoden `onSuccess` und `onFailure` und wird so in die Lage versetzt, auf Antworten vom Server zu reagieren. Hierbei ist die Terminologie der Namensgebung zu beachten. Das definierte Async-Interface muss zwingend mit *ServiceName* gefolgt vom Suffix `Async` benannt werden.

Basierend auf den auf Clientseite definierten Interfaces wird automatisch durch das GWT eine Klasse `KanbanProxy` erzeugt, die `KanbanServiceAsync` und das GWT-Interface `ServiceDefTarget` implementiert und wiederum über den verwendeten Namen mit dem `KanbanService` verbunden ist. Die generierte Klasse `KanbanProxy` ist für die tatsächliche Ausführung der RPC-Kommunikation mit dem Server verantwortlich. Durch die automatische Erzeugung werden alle Details der Kommunikation vom Entwickler versteckt. Auf Serverseite besteht die RPC-Kommunikation aus einer assoziierten Klasse `KanbanServiceImpl`, die alle im `KanbanService` angegebenen Methoden enthält und implementiert. Nach Ausführung der Methode auf dem Server wird automatisch, je nach Ergebnis, die `onSuccess` oder `onFailure` Methode des im `KanbanServiceAsync` Interfaces angegebenen Callback-Objektes aufgerufen.

Zur Entwicklung grafischer Benutzeroberflächen liefert GWT eine eigene **Widget-Library**, welche ein Erstellen grafischer Benutzerschnittstellen für Webapplikationen in Java ermöglicht. Die Widget-Library enthält Elemente für alle gängigen grafischen Elemente, wie Buttons, Menüs, Texteingabefelder, Listen etc. Ähnlich wie die Entwicklung von Oberflächen mit Swing soll auf diese Weise das programmatische Erstellen von grafischen Benutzerschnittstellen durch Java-Entwickler erleichtert werden. Die grafischen Elemente können darüber hinaus mit Event-Mechanismen, wie beispielsweise `onClick` mit der Applikationslogik verbunden werden. Die Einbindung eigener CSS-Sheets und Templates ermöglicht ein weiteres grafisches Styling der entwickelten grafischen Elemente.

Neben den zuvor genannten großen Bausteinen, bietet GWT durch einen eigens entwickelten Mechanismus zum Debuggen die Möglichkeit, die Applikation innerhalb des Browsers zu debuggen. Hierfür sind speziell für den Browser entwickelte Plugins notwendig, die den so genannten `Development Mode` der Webapplikation unterstützen und so ein Debugging des Quelltextes erlauben.

4.3 CoObRA und Graphtransformationen im Kontext der Fujaba Web Application

Die innerhalb heutiger Webbrowser integrierten Ajax-Engines sind ohne Probleme in der Lage Graphtransformationen, wie im vorigen Abschnitt beschrieben, direkt auf dem Webclient auszuführen. Die einzige Einschränkung stellt hierbei die Notwendigkeit dar, die Graphtransformationen und Laufzeitinformationen als JavaScript beschreiben und an den Client ausliefern zu müssen.

Der von mir zur Modellierung von Fujaba Web Applications angestrebte Story-Driven-Modeling Ansatz lässt sich durch die interne Implementierung der Fujaba Werkzeuge hervorragend mit der Definition und Ausführung von Graphtransformationen kombinieren. Es ist daher naheliegend, mit Fujaba spezifizierte Graphtransformationen zur Modellierung der Applikationslogik auf den Webclient zu übertragen und dort als JavaScript auszuführen. Ein weiterer Vorteil der clientseitigen Berechnungen ist eine Verringerung der Serverlast, sowie der auf dem Server benötigten Rechenleistung, da diese mehr und mehr vom Webclient zur Verfügung gestellt wird. Rechenintensive Vorgänge auf der grafischen Benutzerschnittstelle, wie etwa Drag and Drop können so beschleunigt werden. Alle notwendigen Datenmodelländerungen können direkt auf dem Client berechnet und druchgeführt werden. Desweiteren wird die offline Verfügbarkeit einer Applikation deutlich vereinfacht, wenn sowohl Datenmodell als auch benötigte Logik direkt auf den Client übertragen und im Anschluss dort ausgeführt werden können.

Allen genannten Vorteilen der clientseitigen Berechnung von Datenmodellen zum Trotz bleibt die Problematik sicherheitskritischer Anwendungsteile. So ist es beispielsweise notwendig, den einzelnen Clients nur Zugriff auf jene Teile des Datenmodells zu gewähren, die für die Berechnung der eigenen Inhalte notwendig sind. Alle Teile des Gesamtmodells, die allein andere Clients betreffen, müssen vom Server gesichert werden. Ebenso sind Anwendungsteile denkbar, die allein vom Server verwaltet und in keinem Fall auf einen der angemeldeten Clients übertragen werden dürfen. Ein weiterer wichtiger Punkt für komplexe Webapplikationen ist die persistente Speicherung von Applikationsdaten. Zwar ist es mit dem HTML5 LocalStorage möglich, kleinere Datenmengen auch direkt auf dem Client abzuspeichern, die Speicherung des kompletten notwendigen Datenmodells sollte jedoch weiterhin auf dem Server erfolgen, schon allein um dem Benutzer die Möglichkeit offen zu lassen, seine Applikation von unterschiedlichen Rechnern aus gleichermaßen nutzen zu können. Handelt es sich um eine Mehrbenutzerapplikation, stellt die Synchronisierung von Datenmodellen über mehrere gleichzeitig angemeldete Clients eine weitere Herausforderung dar. Gleiches gilt für die Synchronisierung von offline-Daten mit dem Server. Dies wird zur Herausforderung, wenn ein Client im Offline-Modus Änderungen am Datenmodell vorgenommen hat und bei der anschließenden Anmeldung am Server

auf diesem bereits weitere Änderungen eines anderen Clients vorliegen. In diesem Fall müssen die verschiedenen Änderungen miteinander gemischt und potentielle Konflikte aufgelöst werden.

Für alle die Synchronisation von Modellen betreffenden Aufgaben, sei es der Abgleich zwischen mehreren gleichzeitig angemeldeten Clients und dem Server, oder das Auflösen von Konflikten bei gleichzeitig auftretenden Änderungen am Datenmodell, werden durch den Einsatz und die Adaption des im Fujaba Umfeld bewährten CoObRA Frameworks [SZN04] Lösungen zur Verfügung gestellt. Die notwendigen Grundlegenden Eigenschaften bringt CoObRA hierbei mit, durch die im Rahmen von WebCoObRA durchgeführten Anpassungen (vgl. 4.3.2) werden die bestehenden Funktionalitäten für Webclients nutzbar gemacht. Gleiches gilt für die persistente Speicherung von Datenmodellen auf dem Server. Der Einsatz von CoObRA im Kontext von Webanwendungen wurde erstmals in [ADH+09] vorgestellt.

Die mit Fujaba modellierte Applikationslogik wird zur Laufzeit der Applikation innerhalb des Webclients ausgeführt. Um dies zu ermöglichen waren Anpassungen an der Codegenerierung für Graphersetzungsregeln (Aktivitätsdiagramme), sowie der zur Ausführung notwendigen Laufzeitbibliothek notwendig. Abschnitt 4.3.1 stellt die notwendigen Änderungen dar und zeigt auf, wie Applikationslogik für Webclients mit Fujaba modelliert werden kann.

Zur Modellierung von Graphtransformationen im Webclient können neben traditionellen Aktivitätsdiagrammen, wie in Abbildung 4.3 dargestellt, auch erweiterte Statecharts - Action Charts - zum Einsatz kommen. Diese Action Charts wurden durch eine Neudefinition der Transitionen innerhalb der Statecharts und eine daran adaptierte Codegenerierung realisiert. Abschnitt 4.3.4 stellt die Möglichkeiten der Action Charts dar und erläutert dabei die neu eingeführte Transitionssemantik.

Zur Modellierung von Zugriffsrechten auf Datenmodellanteile werden die in Abbildung 4.3 zu sehenden Annotationen am Datenmodell verwendet. Diese Annotationen wurden im Rahmen der Entwicklungen zum WebCoObRA Framework umgesetzt und sind genauer in Abschnitt 4.3.3 beschrieben.

Die folgenden Abschnitte gehen auf die einzelnen genannten Datenmodell-relevanten Aspekte einer Fujaba Web Application genauer ein und definieren so die Fujaba Web Graphs als innerhalb des Webclient lauffähige, mit Fujaba modellierte, Graphtransformationen .

4.3.1 Clientseitige Datenmodelle für Anwendungsdaten

Die Realisierung clientseitiger Datenmodelle im Rahmen von Fujaba Web Applications hat zum Ziel, Datenmodelle zu ermöglichen, die komplett innerhalb der Webclients gehalten, verändert und bei Bedarf synchronisiert werden können. Hierbei ist zu beachten, dass die gewünschten Datenmodelle auf Clientseite als Javascript vorliegen müssen, um von der Ajax-Engine des Browsers verarbeitet werden zu können. Abbildung 4.5 stellt den dabei zugrunde liegenden Ablauf zur Erzeugung der gewünschten Datenmodelle dar.

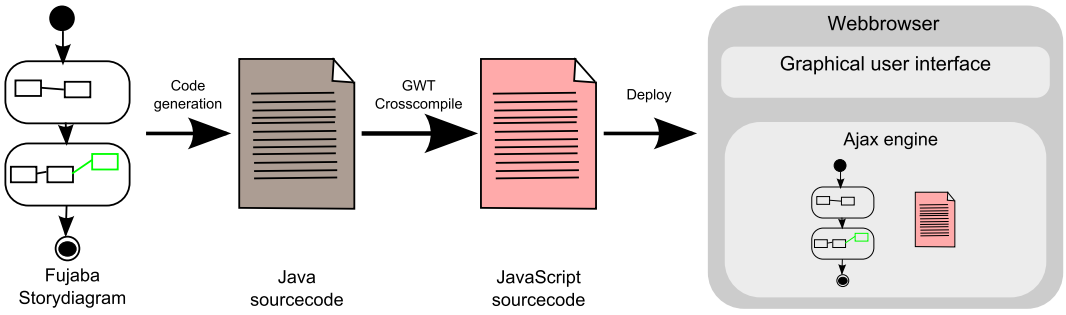

Abbildung 4.5: Umwandlung von klassischen Fujaba Graphtransformationen zu Fujaba Web Graphs in JavaScript. Die JavaScript Repräsentation der Graphtransformationen kann mit Hilfe der Fujaba Laufzeitbibliothek innerhalb der Ajax-Engine des Browsers ausgeführt werden.

Alle Datenmodelle, sowie die damit assoziierte Applikationslogik werden in Fujaba modelliert. Dies bedeutet, innerhalb von Fujaba existiert eines - oder auch mehrere - Klassendiagramme, welche das Datenmodell der Applikation genau beschreiben. Die im Datenmodell abgebildeten Klassen werden anschließend mittels der in Fujaba implementierten, Template basierten Codegenerierung in ausführbaren Java Quelltext umgewandelt. Zur Erzeugung des notwendigen JavaScript Quelltextes wird das Google Web Toolkit eingesetzt. Mit Hilfe des GWT-Crosscompilers werden alle generierten Java Quelltexte in Browser spezifischen JavaScript Quelltext umgewandelt.

Zur Übersetzung von Java Applikationen nach JavaScript verwendet der GWT-Crosscompiler intern die sogenannte JRE Emulation [JRE]. Dies hat zur Folge, dass lediglich Subsets aus der Java Standard Bibliothek zur Verfügung stehen. Klassen mit Inhalt aus nicht unterstützten Paketen können nicht automatisch vom Crosscompiler übersetzt werden. Fujaba verwendet jedoch intern zur Implementierung von Kanten im Klassendiagramm, sowie für Teile der Aktivitätsdiagramme die Fujaba-Runtime-Tools. Hierbei handelt es sich um die mit Fujaba mitgelieferte Laufzeitbibliothek, die dafür sorgt, dass sich die in Fujaba erstellten Graphtransformationen zur Laufzeit korrekt ver-

halten. Die Runtime-Tools beinhalten zu diesem Zweck Eigenimplementierungen von Java-Collections, die auf die von Fujaba gestellten Anforderungen abgestimmt sind. Die Klassen der Fujaba Runtime-Tools stellten sich als nicht GWT-kompatibel heraus. Es musste demnach eine Spezialbehandlung für die Erzeugung GWT-kompatibler Datenmodelle in Fujaba integriert werden.

Zu diesem Zweck wurde in Fujaba zur Verwendung in Webapplikationen ein eigener Stereotyp [Rum13] eingeführt. Stereotypen können an beliebige Fujaba-Modellelemente angefügt werden und bedingen im Rahmen der Codegenerierung eine Spezialbehandlung der assoziierten Elemente. Der neu eingeführte Stereotyp `GWT` muss in Fujaba an das Wurzelelement des Datenmodells gehängt werden und wird anschließend auf alle Kinder vererbt. So wird sichergestellt, dass alle zum Modell gehörigen Teile GWT-kompatibel gehalten werden. Zur Verwendung mit dem `GWT`-Stereotyp wurde die Codegenerierung von Fujaba erweitert.

Die Fujaba Codegenerierung basiert auf Templates [GSR05] und der Velocity Template Engine [Vel]. Zur Erzeugung der GWT-kompatiblen Java Quelltexte wurden eigene Templates für Klassen, Assoziationen, Methoden, sowie zur Erzeugung einer reflektiven Zugriffsschicht für Modellattribute erzeugt. Diese Templates werden durch die Verwendung des `GWT`-Stereotyps zur Codegenerierung ausgewählt und sorgen dafür, dass alle Klassendiagramm-Inhalte mit den gewünschten Eigenschaften generiert werden. Listing 4.3 zeigt einen Ausschnitt des Templates zur Erzeugung GWT-kompatibler Klassen.

```
1  #* *##parse("$lang/default:classDiag/class/fileImport.vm" )
2  #* *#/*
3  #* *# * GWT compliant generated by Fujaba - CodeGen2 version 03.12.2009
4  #* *# */
5  #* *#
6  #* *#$!comment##
7  #* *##if( $package )
8  #* *#package $package;
9  #* *#
10 #* *##end
11 #* *##foreach( $classToImport in $file.iteratorOfImportedClasses() )
12 #* *##if ( ${classToImport.InheritedCodeStyle} )
13 #* *##set( $importStyle = "$lang/${classToImport.InheritedCodeStyle.Name}:" )
14 #* *##else
15 #* *##set( $importStyle = "$lang/default:" )
16 #* *##end
17 #* *##parse("$!{importStyle}classDiag/class/addClassImport.vm" )
18 #* *##end
19 #* *##foreach( $import in $imports.iteratorOfImports() )
20 #* *##if(!$import.startsWith("de.upb.tools") && !$import.equals("java.beans.
       PropertyChangeSupport") && !$import.equals("java.beans.PropertyChangeListener"
       ))
```

```
21  #* *#import $import;
22  #* *##end
23  #* *##end
24  #* *##foreach( $import in $file.iteratorOfImportedPackages() )
25  #* *#import ${import.FullPackageName.toLowerCase()}.*;
26  #* *##end
27  #* *#import fujaba.web.runtime.client.reflect.*;
28  #* *#import fujaba.web.runtime.client.*;
29  #* *#import java.util.*;
30  #* *#import com.google.gwt.event.dom.client.ClickHandler;
31  #* *#import com.google.gwt.event.dom.client.ClickEvent;
```

Listing 4.3: Velocity Template zur Erzeugung von GWT kompatiblem Modellquelltext.

Die Codegenerierung umfasst jedoch neben den Elementen eines Klassendiagrammes, also dem reinen Datenmodell, ebenfalls alle in Aktivitätsdiagrammen modellierten Teile der Applikationslogik. Diese Teile verwenden die oben bereits eingeführte Laufzeitbibliothek mit GWT-inkompatiblen Anteilen. An dieser Stelle wurde aus Gründen der geringeren Komplexität entschieden, nicht die Templates zur Codegenerierung an GWT anzupassen, sondern eine zweite Laufzeitbibliothek zu erzeugen, welche alle Klassen der Runtime-Tools in einer GWT-kompatiblen Weise neu implementiert. Diese Laufzeitbibltiothek wird automatisch bei der Auswahl des `GWT`-Stereotyps in die entsprechenden Klassen importiert und kann so innerhalb der Logikanteile verwendet werden. Auf diese Weise ist eine Reimplementierung der komplexen Codegenerierung für Applikationslogik obsolet. Eine Übersicht über die Laufzeitbibliothek `fujaba.web.runtime` ist in Abbildung 4.6 dargestellt. Die im Klassendiagramm dargestellten Laufzeitklassen bilden ausgehend von der Klasse `CClass` die reflektiven Eigenschaften der modellierten Graphersetzungsregeln nach. Auf dieser Weise werden reflektive Zugriffe aus dem Webbrowser ermöglicht. Die Klasse (`CClass`) kann über den `packageName` identifiziert werden und verfügt über die Assoziation `cAttributes` über eine Referenz auf alle innerhalb der Klasse vorkommenden Attribute. Über die Referenz kann auf die Attribute zugegriffen werden. Neben der Darstellung der strukturellen Eigenschaften der innerhalb der Graphersetzungsregeln modellierten Elemente über die Laufzeitbibliothek sind ebenfalls Klassen enthalten, die vor allem in Verbindung mit dem Replikationsframework WebCoObRA (vgl. Kapitel 4.3.2) zum Einsatz kommen. Hierbei handelt es sich um die Klasse `ModelRoot`, sowie das Interface `ICobject` mit der Standardimplementierung `CObject`. Diese Klassen werden zur Modellierung der Eigenschaften des Instanzmodells verwendet. Alle Modellanteile, die zwischen Servern und Client(s) repliziert werden sollen, implementieren daher stets das Interface `ICObject`.

Der gezeigte Template-Text aus Listing 4.3 erzeugt die für Klassen notwendige Datei und bildet den Rumpf der Klasse ab. Attribute, Assoziationen, Methoden etc. werden durch

den Aufruf weiterer Templates in die hier generierte Datei erzeugt. Zunächst wird in Zeile 1 des obigen Listings die Template-Datei zur Erzeugung von imports eingelesen und so die zu importierenden Elemente ermittelt. Ist ein Paket für die zu erzeugende Klasse angegeben, so wird in Zeile 7 und 8 die Paketdeklaration vorgenommen. Zeilen 19 und 20 des Listings sorgen dafür, dass alle Klassen der Fujaba-Laufzeitbibliothek Runtime-Tools nicht importiert werden (`de.upb.tools`). Stattdessen werden in den Zeilen 28 und 29 die Pakete der neu entwickelten Laufzeitbibliothek (`fujaba.web.runtime`) importiert. Die Laufzeitbibliothek für Webapplikationen implementiert hierbei alle Klassen unter gleichem Namen wie in der Fujaba-Laufzeitbibliothek nach. Auf diese Weise reicht im Rahmen der Codegenerierung ein Austausch der importierten Klassen, um die Lauffähigkeit des generierten Quelltextes zu ermöglichen.

Neben der Sicherstellung der Übersetzbarkeit mit dem GWT-Crosscompiler stellen die neu implementierten Velocity-Templates für GWT die Erzeugung einer reflektiven Zugriffsschicht innerhalb der generierten Modelle sicher. Bei der Verwendung von GWT - insbesondere auf Clientseite - sind die mit der Java-Bibliothek gelieferten Klassen des Paketes `java.lang.reflect` nicht verfügbar. Es existiert somit bei GWT Modellen per se keine Möglichkeit, reflektiv auf dessen Elemente zuzugreifen. Das fehlende Applikationsverhalten wird durch die Einbindung des Paketes `fujaba.web.runtime.client.reflect` innerhalb der Templates gelöst. Dieses Paket enthält Klassen, die einen reflektiven Zugriff auf die generierte Datenstruktur ermöglichen und so diese Verhalten in den Client übertragen. Zusätzlich werden in jede Modelldatei generische `get()` und `set()` Methoden generiert, die das Auslesen oder Setzen eines Attributwertes unter Angabe einer Zeichenkette ermöglichen. So kann außerhalb der reflektiven Laufzeitbibliothek ebenfalls das Auslesen und Setzen von Attributwerten ermöglicht werden.

Das Zusammenspiel der hier beschriebenen angepassten Codegenerierung und Laufzeitbibliothek mit den in den nächsten Abschnitten beschriebenen Aspekten der Datenmodellsynchronisation mit WebCoObRA, sowie der Modellierung von Abläufen mit Action Charts ermöglichen auf Datenmodellseite eine umfassende Modellierung mit Fujaba und eine automatische Transformation der modellierten Modelle und Verhaltensweisen in Webapplikationen.

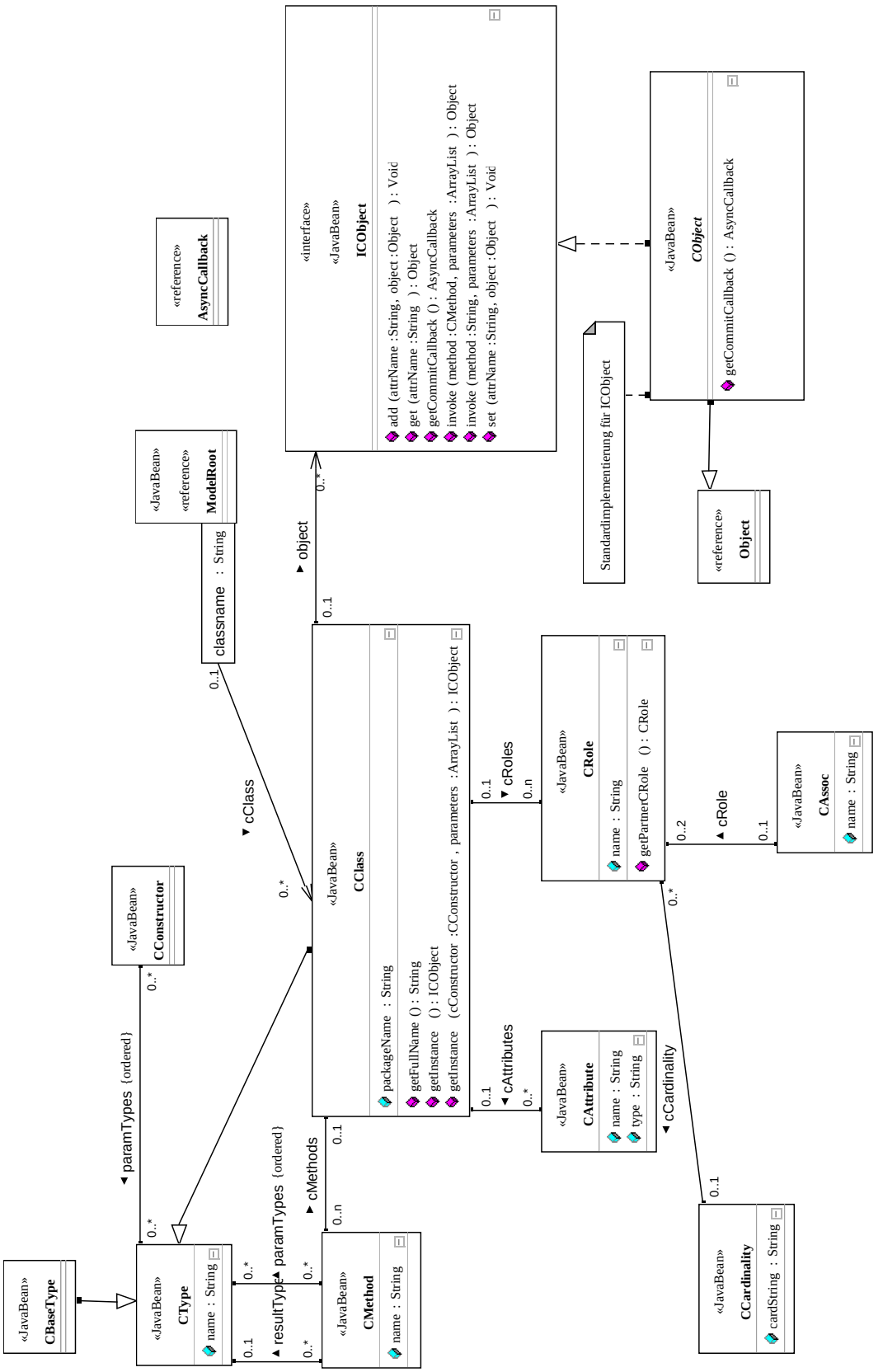

Abbildung 4.6: Klasendiagramm der Neuimplementierung der Laufzeitbibliothek für Webanwendungen - fujaba.web.runtime

4.3.2 WebCoObRA - Datensynchronisation und persistente Speicherung

WebCoObRA ist das im Zusammenhang mit Fujaba Web Graphs verwendete Datenreplikationsframework. Es wurde geschaffen, um eine einfache Möglichkeit zur Verteilung von Datenmodellen innerhalb der Fujaba Web Applications (vgl. Kapitel 3) bereit zu stellen. Ziel ist hier, bei der Modellierung einer Fujaba Web Application möglichst alle Details der Datenreplikation vor dem Anwender zu verbergen und diese automatisch über die Nutzung des WebCoObRA Frameworks zur Applikation hinzuzufügen.

Fujaba Web Applications stellen aufgrund ihrer Eigenschaften verteilte Applikationen dar. Im Gegensatz zu anderen Webapplikationen, die nicht darauf setzen Ajax-Technologien einzusetzen, nutzen Fujaba Web Applications die Clientseite neben der Darstellung der GUI auch für Berechnungen der Businesslogik auf dem zugrunde liegenden Datenmodell. Fujaba Web Applications sind darauf angewiesen, trotz der verteilten Architektur, innerhalb des Webclient durchgeführte Änderungen am Datenmodell auf dem Server persistent abzuspeichern. Hier können verteilte Applikationen nicht, wie dies im Rahmen der Entwicklung von Applikationen mit mehreren Threads aber einer Laufzeitumgebung (Mutlti-Thread Applikationen) der Fall ist, auf gemeinsame Speicherbereiche zur Persistierung der Daten zurückgreifen. Gemeinsame Speicherbereiche ermöglichen ohne weiteres ein komplexes modelliertes Datenmodell zu verwenden und dieses gleichsam von allen Applikationsthreads zu benutzen und Änderungen darauf durchzuführen. Im Rahmen von Multi-Thread Applikationen kann hier durch den Einsatz von Semaphoren zur Synchronisierung der Lese- und Schreibzugriffe unterschiedlicher Threads auf dem Datenmodell gearbeitet werden. Bei verteilten Anwendungen müssen hier andere Ansätze gefunden werden, um viele Prozesse, die auf unterschiedlichen Rechnern auf heterogenen Plattformen und vor allem mit jeweils eigenem Speicher laufen, zu einem gemeinsamen Datenbestand zu assimilieren.

Die Heterogenität unterschiedlicher Plattformen wird von Ansätzen, wie beispielsweise Corba [Cor], [OPR95] und anderen Techniken für Service Oriented Architectures (SOA) [Erl08], [NL04] bereits addressiert, das Problem der separierten Speicherbereiche jedoch bleibt bestehen. Fujaba stellt intern bereits einen Mechanismus bereit, der es erlaubt die in Fujaba modellierten Datenmodelle zu versionieren und zu speichern. Gleichzeitig ist es in Fujaba dank dieses Mechanismus möglich, mit mehreren Benutzern gleichzeitig am selben Datenmodell zu arbeiten und die Änderungen miteinander zu synchronisieren. Der in Fujaba verwendete Versionierungs- und Speichermechanismus baut auf dem von Christian Schneider entwickelten CoObRA Framework auf (vgl. Kapitel 4.2.1). Die von CoObRA zur Verfügung gestellten Funktionalitäten zur Replikation von Daten zwischen verschiedenen Instanzen, zur Auflösung von Konflikten, sowie zur persistenten Speicherung der Datenmodelle als Change-Stream mit dem *CoObRA Transaction*

Repository-Protokoll (CTR) stellen die Basis des WebCoObRA Frameworks dar, welches im Rahmen der verteilten Fujaba Web Applications die oben beschriebene Problematik getrennter Speicherbereiche löst.

Neben der clientseitigen Ausführung von Businesslogik in Form von Graphersetzungsregeln nutzen auch Fujaba Web Applications den Webbrowser zur Darstellung der GUI. Wie im Webbrowser üblich dient die einer Webseite zugrunde liegende DOM-Struktur (Document Object Model) der Darstellung der GUI. Änderungen an der Benutzeroberfläche spiegeln sich in Änderungsoperationen auf der DOM-Struktur wieder. Der Browser rechnet die Änderungen automatisch in die neu anzuzeigende Benutzeroberfläche um. Fujaba Web Applications benötigen Mechanismen, um die auf Clientseite berechnete Applikationslogik und dort durchgeführte Änderungen am Datenmodell, sowie die in Verbindung mit der GUI vorherrschenden DOM-Transformationen zu verarbeiten. Als Lösung werden `PropertyChange` Mechanismen eingesetzt, die eine Verbindung von DOM-basierter GUI und Datenmodell schaffen. Die `PropertyChange` Mechanismen werden von der angepassten Codegenerierung für GWT (vgl. Abschnitt 4.3.1) automatisch aus den modellierten Datenmodellen mit Fujaba generiert und anschließend von GWT in ausführbaren JavaScript Quelltext übersetzt.

Bei der Adaption des CoObRA Frameworks zur Nutzung mit Webapplikationen mussten sowohl die Anforderungen seitens GWT an clientseitigen Quelltext erfüllt werden (vgl. 4.2.3), als auch die Kommunikation zwischen den angemeldeten Clients und dem zentralen Server entsprechend der Gegebenheiten abgeändert werden. Die bereits zur Nutzung der Laufzeitbibliothek angepasste Codegenerierung für Webapplikationen wurde im Rahmen der Arbeiten an WebCoObRA daher erweitert. Das in Listing 4.1 dargestellte Textformat zur Speicherung und Übertragung atomarer Änderungen (CTR) wurde grundlegend beibehalten. Die Übertragung der Änderungen zwischen Client und Server wurde jedoch auf GWT-RPC umgestellt. Die serverseitigen Anteile zur Persistierung der Daten (insbesondere das CoObRA Repository) wurden beibehalten und über den Serveranteil der RPC Kommunikation mit WebCoObRA verknüpft. Die von CoObRA gelieferten Mechanismen zum Auflösen von Konflikten bei konkurrierenden Datenmodelländerungen wurden beibehalten, eine Behandlung erfolgt ausschließlich auf Serverseite.

4.3.3 WebCoObRA am Beispiel Kanban

Die in Abschnitt 4.1 eingeführte Software Version eines Kanban-Boards soll als Webaplikation im Browser lauffähig sein. Sie verfügt zu diesem Zweck sowohl über ein graphbasiertes Datenmodell, wie auch über modellierte Businesslogik zum Hinzufügen, Löschen und Reorganisieren einzelner Prozessschritte und Tasks in Form von Graphersetzungsregeln. Hierbei soll der Hauptteil der notwendigen Graphtransformationen innerhalb der

Ajax-Engine des Webbrowsers ausgeführt werden und nur zur Synchronisation mit dem Server, sowie mit den übrigen verbundenen Clients kommuniziert werden. Die Kanban-Applikation ist gemäß der zuvor beschriebenen Anforderungen als GWT-Applikation angelegt. Die Applikation verfügt über eine Startseite im HTML-Format, welche vom Webbrowser geladen wird. Diese includiert den benötigten JavaScript Quelltext und veranlasst auf diese Weise den Browser dazu, die modellierten Logik- und Modellanteile der Applikation vom Server nachzuladen. Beim Starten der Applikation wird durch das im Client laufende JavaScript eine Verbindung zur Serverkomponente des Kanban-Boards aufgebaut. Die Serverkomponente ist für die persistente Speicherung und Replikation der Modelldaten mit WebCoObRA verantwortlich. Dies hat zur Folge, dass beim Starten der Applikation initial eine Verbindung zum CoObRA-Repository aufgebaut wird und der aktuell gültige Zustand des Datenmodells vom Server an den Client übertragen wird. Anschließend führt der Nutzer Änderungen am Kanban-Board durch, indem die virtuellen Klebezettel auf dem Kanban-Board verschoben, einzelne Spalten des Workflows umstrukturiert oder bearbeitet werden. All diese Änderungen werden zunächst lokal am Datenmodell des Clients durch Anwendung der zugehörigen Graphersetzungsregeln durchgeführt. Die Benutzeroberfläche reagiert über die automatisch generierten `PropertyChange`-Mechanismen automatisch auf Änderungen des Datenmodells und wird in der Folge entsprechend angepasst. Abbildung 4.7 zeigt einen Ausschnitt des Klassendiagramms von WebCoObRA bei der Verwendung mit der Klasse `QBan` aus dem Klassendiagramm der Beispielapplikation aus Abbildung 4.2.

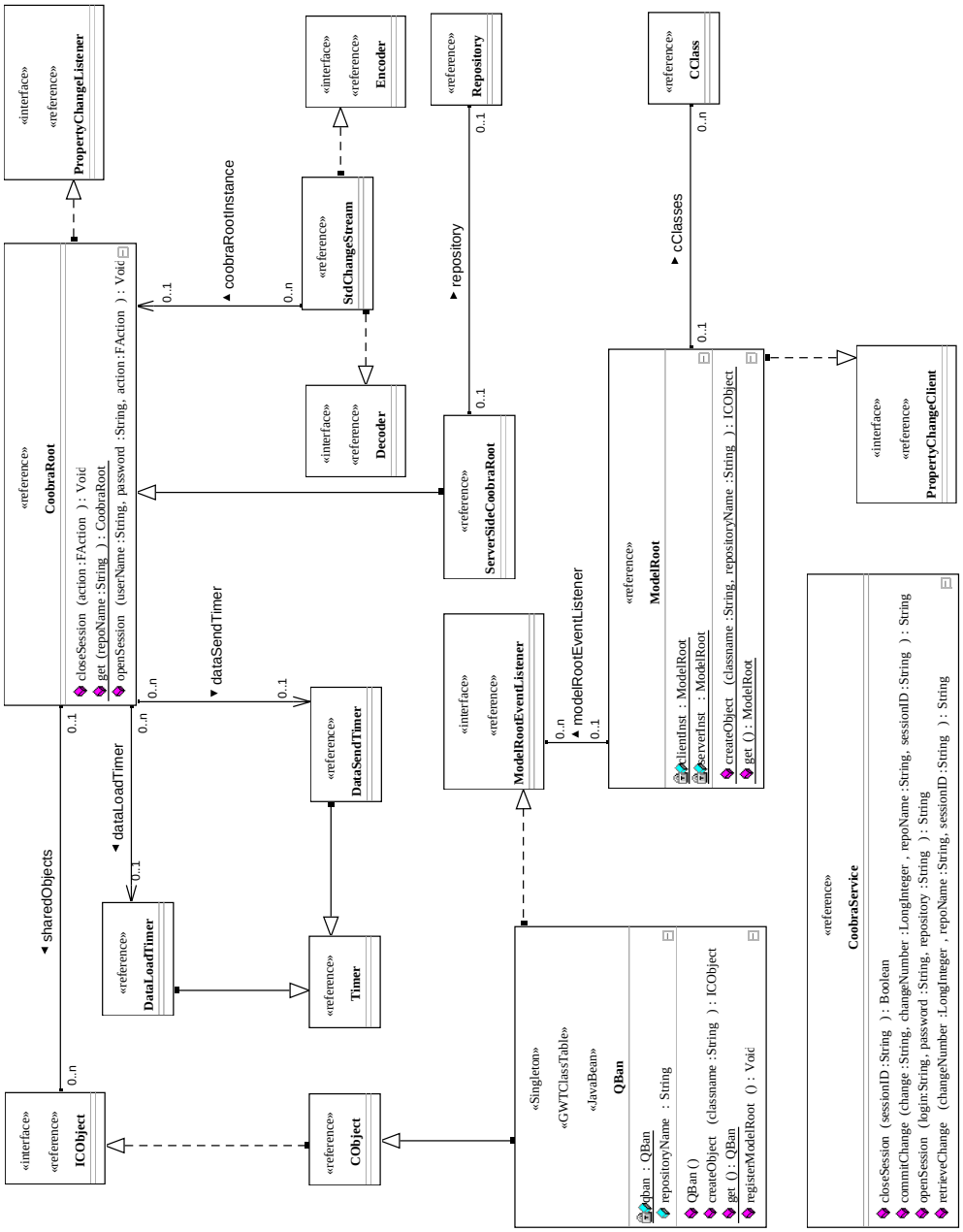

Abbildung 4.7: Ausschnitt aus WebCoObRA zur Replikation und Speicherung der Applikationsdaten anhand der Kanban Beispielapplikation.

Abbildung 4.7 zeigt die Klasse QBan in Zusammenhang mit den für die Datenreplikation und Speicherung zuständigen Klassen von WebCoObRA. Die Klasse QBan verfügt über den Stereotyp «GWTClassTable» und leitet von der Laufzeitklasse CObject (vgl. Klassendiagramm 4.12) ab. Der Stereotyp ist im Zusammenhang mit der Datenreplikation zwingend notwendig und sorgt für die Erstellung einer Klassentabelle. Über die bereits in Kapitel 4.3.1 beschriebenen generischen Methoden zum Setzen und Auslesen von Attributen hinaus wird bei Verwendung des Stereotyps «GWTClassTable» zusätzlich eine Klassentabelle aller im Paket enthaltener Modellklassen generiert. Diese Klassentabelle wird innerhalb der Methode registerModelRoot() der Klasse QBan erzeugt und am ModelRoot des WebCoObRA angemeldet. Für jede im Paket vorhandene Klasse wird die in Abbildung 4.12 dargestellte Struktur aller Klassen, Methoden, Attribute etc. erzeugt und alle Klassen in die cClasses Assoziation des ModelRoot eingehängt. Gleichsam wird automatisch innerhalb der createObject()-Methode der Klasse QBan die reflektive Erzeugung aller verfügbaren Klassen der Klassentabelle generiert. Basierend auf dem übergebenen Klassennamen werden so die korrekten Objekte erzeugt. Die Klasse QBan verfügt nicht nur über die Klassentabelle und Erzeugungsoperationen aller relevanten Modellklassen sondern registiert sich darüber hinaus als ModelRootEventListener beim ModelRoot. Auf diese Weise kann das Erzeugen neuer Klasseninstanzen von ModelRoot an die dafür dedizierte Klassentabelle (QBan) weitergereicht werden. Änderungen am Datenmodell werden generell über PropertyChange-Mechanismen verbreitet. Die Klasse CoobraRoot fungiert als PropertyChangeListener aller registrierten Objekte. Um über Änderungen informiert zu werden und selbst Änderungen weiter zu verteilen ist es notwendig, dass alle Instanzobjekte der in der Klassentabelle vorhandenen Klassen sich bei CoobraRoot registrieren. Dies geschieht automatisch im jeweiligen Konstruktor der Klasse. Leitet eine Klasse von CObject ab, so wird von der Codegenerierung im Konstruktor die in Listing 4.4 dargestellte Quelltextzeile eingefügt und so das Instanzobjekt bei CoobraRoot angemeldet.

```
1  CoobraRoot.get(QBan.get().getRepositoryName()).addToSharedObjects(this);
```

Listing 4.4: Anmeldung eines Objektes zur Datenreplikation mit WebCoObRA

Die Implementierung der Klasse CoobraRoot teilt sich in einen clientseitigen und einen serverseitigen Anteil. Die Serverseite verfügt mit der Klasse ServerSideCoobraRoot über eine Verbindung zum Repository. Hierbei handelt es sich um die Repository-Implementierung des ursprünglichen CoObRA-Frameworks. Auftretende Änderungen am Datenmodell werden über die Klasse StdChangeStream sowohl encodiert, als auch decodiert. Hier wird die Umsetzung der Modelländerungen in atomare Transaktionen des CoObRA-CTR-Protokolls vorgenommen. Die Klassen DataLoadTimer und DataSendTimer sind für die zyklische Überprüfung und Kommunikation mit dem Server zuständig. Beide implementieren zu diesem Zweck einen GWT-Timer. Dies stellt sicher,

dass die innerhalb der Klasse spezifizierte Funktionalität zeitgesteuert wiederholt wird. Basierend auf dem von den beiden `Timern` vorgegebenen Zeitintervall werden Verbindungen zum Server aufgebaut. Die Serverkommunikation erfolgt hierbei über GWT-RPC, wie in Kapitel 4.2.3 beschrieben. Die für die Kommunikation verantwortliche Klasse ist `CoobraService`. Diese Klasse verfügt über Methoden zum Aufbau (`openSession()`) und zum Schließen (`closeSession()`) einer Verbindung, sowie für die Übertragung von Datenmodelländerungen (`commitChange()`, `retrieveChanges()`). Diese werden jeweils zeitgesteuert aufgerufen. Objekte werden innerhalb des Change-Streams des CoObRA Protokolls durch eine eindeutige ID referenziert. Um jederzeit auf Clientseite das korrekte Objekt referenzieren zu können, existieren einfache Tabellen, welche Objekte und ihre zugehörige ID verwalten. Diese Tabellen dienen innerhalb des Encodier- und Decodiervorgangs als Referenz.

Alle auf Clientseite vorgenommenen Änderungen werden im lokalen Speicher vorgehalten und erst durch erfolgreiche Ausführung der `commitChange()`-Methode auf dem Server persistiert. Wird der Client jedoch geschlossen, ohne dass die Änderungen zuvor an den Server übertragen wurden, gehen diese verloren. Eine Anbindung an den mit der Einführung des HTML5-Standards [htm] zur Verfügung stehenden lokalen Speicher des Browsers existiert bislang nicht.

Zugriffsrechte über Modellannotationen

Zur Absicherung einzelner Modellelemente wurden Annotationen auf dem Datenmodell definiert. Diese Annotationen können, wie im Klassendiagramm in Abbildung 4.2 gezeigt an einzelne Modellklassen angehängt werden. WebCoObRA definiert zum jetzigen Zeitpunkt Annotationen für lesenden und schreibenden Zugriff - `Read` und `Write`. Die am Klassendiagramm vermerkten Annotationen werden in den Modellquelltext übernommen und während der Ausführung mit Hilfe eines Beanshell-Interpreters [Bea] überprüft. Der Interpreter erlaubt die Angabe komplexer Ausdrücke innerhalb der Annotationen. Alle `Write` Annotationen werden behandelt, sobald eine Transformation den Server erreicht, Bedingungen aus `Read` Annotationen werden überprüft bevor eine Transformation den Server verlässt. Hierbei werden alle Änderungen, wie bei WebCoObRA üblich, als Change Stream über GWT-RPC zwischen Client und Server versendet und vom Server innerhalb einer CTR-Datei abgespeichert. Alle eingehenden Änderungen werden zeilenweise decodiert und so in entsprechende Modelländerungen übersetzt. Für alle eingehenden Änderungen werden anschließend die `Write` Annotationen an Klasse oder Attribut überprüft. Ist die Änderung aufgrund der Auswertung der Annotation ungültig wird ein Rollback durchgeführt. Das Prüfen der `Read` Annotationen geschieht analog während des zeilenweisen Encodierens aus der auf dem Server befindlichen CTR-Datei in den Change

Stream. Jede Transformation wird einzeln überprüft, hat der anfragende Client kein Recht ein gewünschtes Attribut zu lesen wird dieses nullwertig in den Change Stream geschrieben und anschließend übertragen. Fehlt dem Client hingegen das Leserecht für ein komplettes Objekt so wird das Senden der entsprechenden Transformation an den Client abgebrochen.

4.3.4 Action Charts

Bei Fujaba Action Charts handelt es sich um eine Verfeinerung der bereits in Fujaba implementierten Statecharts [GZ05a] zur Vereinfachung der Modellierung von Webapplikationen. Action Charts kommen dabei vor allem zur Modellierung reaktiven Verhaltens von Webapplikationen (Fujaba Web Applications) zum Einsatz. Bei der Realisierung von Fujaba Web Applications sind, wie im gesamten Bereich der Rich Internet Applications [FRSF10] zahlreiche Callback Methoden und Listener für die Realisierung des asynchronen, reaktiven Verhaltens der Applikation notwendig. Im Rahmen der asynchronen Kommunikation mit der Serverseite werden diese Callbacks benötigt, um auf die zeitverzögert eintreffenden Antworten des Servers reagieren zu können, ohne die Applikation in der Wartezeit zu blockieren (vgl. Abschnitt 4.2.3 zu GWT-RPC). Aktive Elemente der GUI hingegen benötigen Eventhandler, die durch Interaktion des Benutzers ausgelöste Events wie etwa Mausklicks behandeln. Bei der Verwendung von zeitgesteuerten Applikationsanteilen über Timer sind ebenfalls geeignete Eventhandler notwendig, um die zeitlich auftretenden Events abzuarbeiten.

Action Charts wurden entwickelt, um die Erstellung aller zuvor genannter Applikationsteile zu vereinfachen. Als Erweiterung der in Fujaba implementierten UML Statecharts sind sie speziell auf die Modellierung von Callbacks und Eventhandlern zugeschnitten und durch semantische Änderung der Transitionsübergänge, sowie geeignete Codegenerierung auf die Anforderungen von Webapplikationen abgestimmt. Die Grundlagen für die Action Charts wurden von Tobias George in der von mir mitbetreuten Bachelorarbeit [Geo10] entwickelt.

Notwendige Handler, Callbacks oder Timer werden in Fujaba mittels grafischer Statechart-Notation modelliert. Jeder State wird im anschließenden Codegenerierungs-Schritt den Anforderungen gemäß in Handler, Callback oder Timer umgewandelt. Die Transitionen innerhalb eines Action Charts bilden den Ausführungsfluss der Applikation ab. Durch den Einsatz von Variablen, die über alle Actions geteilt werden, kann in einfacher Weise der aktuelle Status der Applikation zwischen den unterschiedlichen Teilen der Applikation weitergereicht werden. Konform zu den Fujaba Web Applications (vgl. Kapitel 3) wird aus den Action Charts Quelltext generiert, der konform zu allen von GWT gestellten Anforderungen ist (vgl. Kapitel 4.3.1). Dieser Quelltext wird anschließend vom

GWT-Crosscompiler übersetzt und kann dann im Webbrowser ausgeführt werden.

Die Nutzung von Action Charts mit Fujaba ermöglicht das Einbinden von Graphersetzungsregeln in die Aktivitäten eines Action Charts. Semantisch werden die dort in Fujaba Notation erstellten Regeln immer dann ausgeführt, wenn die entsprechende Aktivität des Action Charts aktiv wird. Statecharts in Fujaba - und damit auch die erweiterten Action Charts - verfügen über einen Ausführungsmechanismus der

1. einen Thread pro Statechart startet. Ein Statechart ist immer einer bestimmten Klasse fest zugeordnet, dies bedeutet, dass immer ein Statechart pro Klasseninstanz ausgeführt wird.

2. auf das Auftreten eines externen Events wartet und dann asynchron die entsprechenden Transitionen zwischen States aktiviert. Gewöhnlich wird dies durch den Aufruf einer externen Methode durchgeführt, welche gleichsam vom Entwickler modelliert werden muss.

Der Vorteil der Modellierung mit Statecharts ist die Möglichkeit, auf diese Weise asynchrones Programmverhalten zu Modellieren, wie es beispielsweise für die Benutzung von GWT-RPC notwendig ist. Die Action Charts erweitern die in Statecharts traditonell vorhandenen Transitionen und führen anstelle einer einheitlichen Transitionssemantik unterschiedliche Typen von Transitionen ein. Jeder dieser Typen ist mit dedizierten Events verknüpft, die automatisch aus den Action Charts generiert werden können. Auf diese Weise werden asynchrone Verhaltensweisen wie etwa Callbacks komplett abstrahiert.

Beispiel - Einfache ToDo Liste

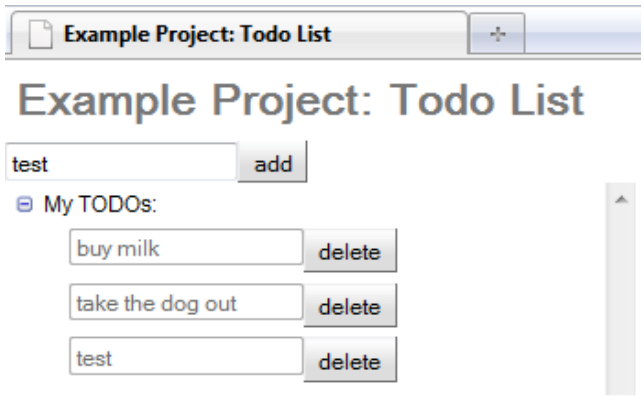

Abbildung 4.8: Grafische Benutzerschnittstelle einer einfachen Todo Liste

Zur Vereinfachung des Verständnisses wird für die Vertiefung der Action Charts vom Kanban-Beispiel aus Abschnitt 4.1 Abstand genommen. Das Kanban-Board verfügt über

ein clientseitiges Datenmodell, welches mittels der WebCoObRA Replikationsmechanismen zwischen einem Server und mehreren Clients synchron gehalten wird. Die Änderungen am Modell, welche über Drag-and-Drop Mechanismen vollzogen werden, werden automatisch zwischen den angemeldeten Clients verteilt. Eine weitere Definition von Events oder RPC-Calls, wie sie mit Action Charts modelliert werden können, ist für das Kanban-Beispiel nicht notwendig. Aus diesem Grund wird zur Verdeutlichung der Möglichkeiten zur Modellierung reaktiven Applikationsverhaltens mit Action Charts im Folgenden eine einfache Todo Liste, wie sie in Abbildung 4.8 zu sehen ist, verwendet.

Die Todo Liste aus Abbildung 4.8 besteht aus einem Textfeld zur Eingabe einzelner Aufgaben. Rechts neben dem Textfeld befindet sich ein Button (`add`) über welchen die definierte Aufgabe in die Liste eingefügt werden kann. In einer darunter befindlichen Übersicht wird die definierte Todo Liste (`My TODOs`) mit allen darin befindlichen Elementen angezeigt. Jedes Element besteht wiederum aus einem Textfeld, sowie einem Button `delete` zum Löschen des Eintrages aus der Liste. Über die Eingabe eines Textes in das Textfeld eines bestehenden Listeneintrages kann dieser geändert werden.

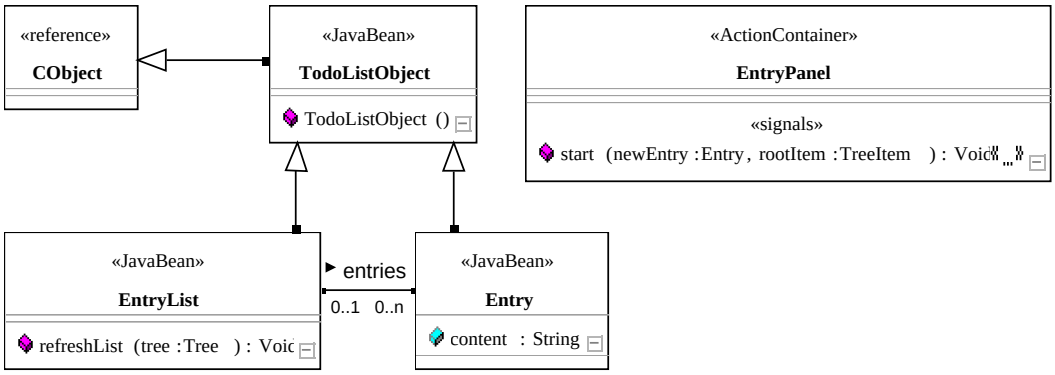

Abbildung 4.9: Example Application: TodoList Class Diagram excerpt

Das Klassendiagramm in Abbildung 4.9 stellt das der Todo Liste zugrunde liegende Datenmodell dar. Hauptelement der Todo Liste ist die Klasse `TodoListObject`. Diese Klasse fungiert als Vaterklasse für die Modellelemente der Todo Liste. Die Klasse `EntryList` modelliert die eigentliche Liste, die Klasse `Entry` modelliert die einzelnen Einträge innerhalb der Liste. Über die Kante `entries` zwischen `EntryList` und `Entry` können Elemente in die Liste eingefügt beziehungsweise aus ihr gelöscht werden. Das Attribut `content` der Klasse `Entry` stellt den textuellen Inhalt der Aufgabe in der Todo Liste dar. Zur Verteilung des Datenmodells über die in 4.3.2 beschriebenen WebCoObRA Mechanismen erben alle Modellelemente von der Klasse `CObject`. Die Klasse `EntryPanel` im Klassendiagramm ist mit dem Stereotyp «`ActionContainer`» versehen. Hierdurch wird das Action Chart der Todo Liste zugeordnet.

Eventhandling

Die Behandlung von Events mittels Action Charts ist beispielhaft in Abbildung 4.10 dargestellt. Die Abbildung zeigt einen Ausschnitt aus einem Action Chart zur Initialisierung der GUI auf Clientseite. Bei der Nutzung von GWT erfolgt die Initialisierung der GUI dadurch, dass einzelne Widgets erzeugt und zur Hauptkomponente der Applikation (`RootPanel`) hinzu gefügt werden. In Abbildung 4.10 werden die Widgets `panel:HorizontalPanel`, `contentBox:TextBox` und `deleteButton:Button` durch die in die Aktion `CreatePanel` eingebundene Graphersetzungsregel erzeugt. `contentBox` und `deleteButton` werden dem `panel` als Widget hinzugefügt. Das Einhängen des Widgets Panel in das `RootPanel` erfolgt implizit und muss nicht separat modelliert werden. Das Erzeugen von Objekten innerhalb einer Graphersetzungsregel wird durch die Angabe des ≪`create`≫ Stereotyps am entsprechenden Objekt realisiert. Im Beispiel ist dies für die drei vorgenannten Widgets der Fall. Auch das Löschen von Objekten kann durch die Angabe eines entsprechenden Stereotyps ≪`destroy`≫ modelliert werden.

Das in Abbildung 4.10 dargestellte Beispiel definiert neben der Initialisierung der GUI zwei weitere Aktionen. Hierbei handelt es sich um Eventhandler für zuvor erzeugte UI-Widgets. So wird innerhalb der Aktion `DeleteEntryHandler` das Verhalten der Applikation beim Klick auf den zuvor erzeugten `deleteButton` modelliert. Die zugehörige Aktion wird durch die Angabe `deleteButton.click` an der verbindenden Transition im Codegenerierungsschritt zu einem entsprechenden Handler umgewandelt. Die Angabe von Clickhandlern wird im Rahmen von Action Charts durch die Angabe von `<Objektname>.click` an einer Transition vorgenommen. Diejenige Aktion des Action Charts, die Ziel der entsprechend gekennzeichneten Aktion ist, fungiert nun als `ClickHandler`. Die in dieser Aktion eingebettete Graphersetzungsregel modelliert das beim Klick auf das entsprechende Objekt intendierte Verhalten. Wie in Graphersetzungsregeln üblich, ist es auch hier möglich, weitere Methoden über Collaboration-Statements aufzurufen. Auf diese Weise lassen sich auch komplizierte Verhaltensweisen mit Action Charts modellieren.

Auf gleiche Weise wie die `ClickHandler` können auch weitere UI-Handler modelliert werden. Selection Events werden etwa durch die Angabe von `<Objektname>.select` an der ausgehenden Transition erzeugt.

Die Codegenerierung von Fujaba erzeugt Zugriffsmethoden, die konform zu *JavaBeans* `PropertyChange` Events feuert, sobald eine Eigenschaft geändert wird. Die dritte Aktion, `EntryChangeListener` aus dem Action Chart in Abbildung 4.10 zeigt die Modellierung von Property Changes. Beim in Abbildung 4.10 definierten Objekt `newEntry` handelt es sich um ein Objekt konform zu der im Klassendiagramm aus Abbildung 4.9 de-

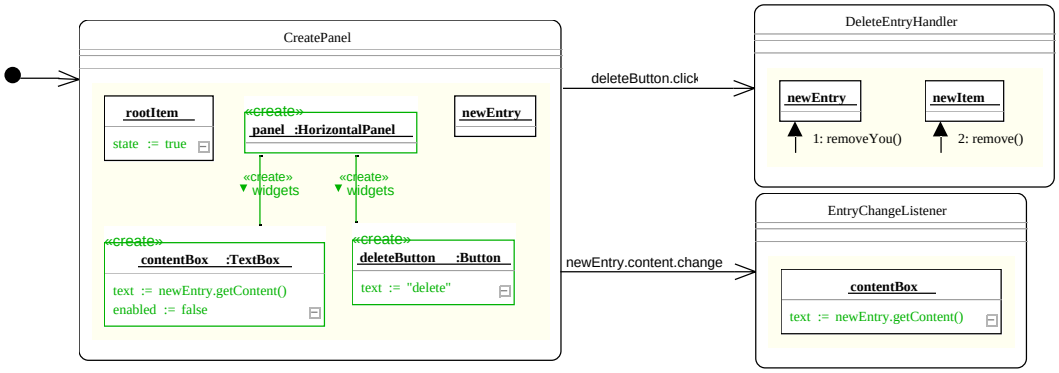

Abbildung 4.10: Action Chart zur Modellierung von Click- und Changeevents innerhalb der Beispielanwendung Todo Liste.

finierten Klasse `Entry`. Objekte dieses Typs verfügen über ein Attribut `content:String`. Die in Abbildung 4.10 angegebene Transition zeigt die Modellierung von Property Changes. Die Angabe an der Transition wird durch `<Attributname>.change` modelliert und erzeugt einen `PropertyChangeHandler` aus der Graphersetzungsregel, die innerhalb der Ziel-Aktion der Transition angegeben ist. Im Beispiel aus Abbildung 4.10 wird bei Auftreten eines Change-Events der Inhalt des UI-Elements `contentBox` auf den Wert des neu definierten Attributwertes gesetzt.

Durch die Nutzung von Timern lässt sich mit GWT-Applikationen quasi-parraleles Applikationsverhalten erzeugen. Wie bereits erwähnt ist de facto lediglich ein Thread zur Zeit innerhalb der clientseitigen Applikation lauffähig. Die Nutzung eines Timers ermöglicht an dieser Stelle die Verzögerung oder periodische Abarbeitung einzelner Operationen. Die Action Charts verfügen durch die Verwendung von Timern über die Möglichkeit, explizit zeitgesteuerte Aktion zu modellieren. Um ein Timer Objekt zu definieren kann die Bedingung `after <Integer>` an einer Transition angegeben werden. Im Ablauf des Action Charts bedeutet dies, dass bei Erreichen der Quell-Aktion der entsprechend gekennzeichneten Transition ein GWT-Timer gestartet wird, der die Ziel-Aktion der Transition als Timer-Callback Objekt übergeben bekommt. Die innerhalb der Ziel-Aktion angegebene Graphersetzungsregel definiert wie bei den zuvor angegebenen Events das Verhalten des gewünschten Timer Callbacks.

RPC Calls

Remote Procedure Calls (RPC) werden im Rahmen der Entwicklung von GWT-Applikationen verwendet um Methoden auf der Serverseite der Applikation aufzurufen (vgl. Abschnitt 4.2.3). Aufgrund des nicht-blockierenden Applikationsverhaltens der Ap-

plikationen ist ein Callback-Objekt notwendig, das als Ziel der Serverantwort beim Aufruf der Methode mit übergeben werden muss. Die Serverantwort wird unterschieden nach erfolgreichem oder nicht erfolgreichem Aufruf der Servermethode. Auch die Behandlung von RPC Verhalten kann mit Fujaba modelliert werden. Abbildung 4.11 zeigt exemplarisch die Modellierung von RPC Applikationsverhalten mit Action Charts. Gleichzeitig zeigt die Abbildung ein Beispiel für die Initialisierung der Datenreplikation über das WebCoObRA Framework (vgl. Abschnitt 4.3.2).

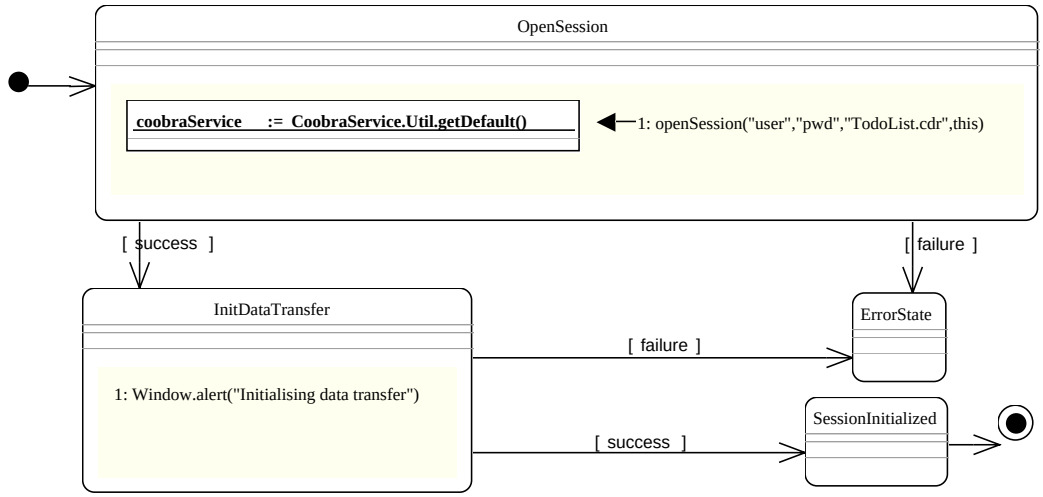

Abbildung 4.11: Beispiel für die Verwendung von Action Charts zur Modellierung von RPC Calls für die Beispielanwendung Todo Liste.

Die erste Aktion - `OpenSession` - des in Abbildung 4.11 dargestellten Action Charts zeigt den Aufruf einer Servermethode auf dem `CoobraService`. Der Aufruf geschieht, wie im Collaboration Statement Nummer 1 angegeben durch den Aufruf der Methode `openSession`. Der letzte Parameter dieser Methode kennzeichnet das assoziierte Callback-Objekt. Die Angabe `this` an dieser Stelle referenziert die zur Laufzeit erzeugte Klasse, die die `OpenSession` Aktion des Action Charts repräsentiert. Der Aufruf der `OpenSession` Aktion wird somit automatisch zum Callback. Die Callback-Behandlung wird über die beiden ausgehenden Transitionen der `OpenSession` Aktion modelliert. Diese beiden Transitionen sind mit `success` und `failure` markiert. Ein erfolgreicher RPC-Aufruf führt zur Ausführung der `success` Transition, sowie zur Aktivierung der entsprechenden Ziel-Aktion. Im Beispiel aus Abbildung 4.11 führt ein erfolgreicher Aufruf der Methode `openSession` zur Initialisierung des Datentransfers für die Todo Liste. Ein fehlgeschlagener RPC-Aufruf veranlasst die Ausführung der `ErrorState` Aktion. Dies kann beispielsweise verwendet werden, um Fehlermeldungen anzuzeigen.

Umsetzung der Anforderungen von Webapplikationen an Action Charts

Die zuvor beschriebenen Anforderungen zur Abstraktion asynchronen Applikationsverhaltens über Action Charts wurde durch mehrere ineinander greifende Maßnahmen realisiert. Zunächst wurde die Semantik der Transitionen, welche die Zustandsübergänge innerhalb eines Statecharts - und somit auch innerhalb der Action Charts - modelliert, grundlegend verändert. Anstatt der bislang vorhandenen Transitionen, die einen synchronen Übergang zwischen unterschiedlichen Zuständen eines Fujaba-Statecharts modellierten, wurden verschiedene Typen von Transitionen eingeführt.

- Success und Failure Transitionen zur Behandlung von RPC-Calls (vgl. Abschnitte 4.2.3 und 4.3.4)
- *auto* Transition zur Modellierung synchroner Applikationsübergänge
- Eventbasierte Transitionen zur Modellierung von z.B. Interaktionen mit der GUI
- PropertyChange basierte Transitionen zur Modellierung von Reaktionen auf Modelländerungen
- Boolesche Transitionen zur Modellierung von synchronen Applikationsübergängen, welche abhängig von einer gegebenen Bedingung sind

Eine globale Nutzung von Variablen über alle Zustände des Statecharts hinweg sorgt dafür, dass jeder Zustand jederzeit auf notwendige Informationen zugreifen kann, ohne dass diese zwischen den einzelnen Zuständen verteilt werden müssen.

Die Action Charts modellieren nicht länger eine strikt sequentielle Abarbeitung der gegebenen Zustände. Vielmehr werden die Zustandsübergänge basierend auf Events geschaltet. Gleichsam wurde die Integration zeitgesteuerter Abläufe realisiert, indem Timer innerhalb eines Zustandes definiert werden können. Wird der Timer nun innerhalb eines Zustandes des Action Charts gestartet, führt dies dazu, dass die vom Timer modellierte Funktionalität mit dem gegebenen Intervall immer wieder ausgeführt wird. Die vom verwendeten Zustand ausgehenden Transitionen modellieren das Verhalten der Gesamtapplikation, die parallel zur Ausführung des Timers weiterläuft.

Die beschriebene Änderung der Transitionssemantik allein reicht zur Realisierung des gewünschten Verhaltens nicht aus. Die für im Rahmen von Kapitel 4.3.1 erwähnte Laufzeitbibliothek für Webapplikationen wurde gemäß des Klassendiagramms in Abbildung 4.12 erweitert.

Die Klasse `FAction` des gezeigten Klassendiagramms stellt die grundlegende Implementierung der einzelnen Zustände (Actions) eines Action Charts dar. Die Klas-

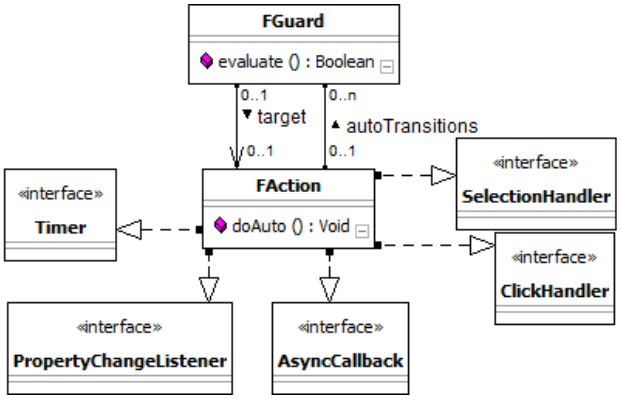

Abbildung 4.12: Klassendiagramm mit für Action Charts relevanten Anteilen der Fujaba Laufzeitbibliothek

se `FAction` implementiert zahlreiche Interfaces zur Behandlung von GUI Events (`SelectionHandler`, `ClickHandler`), zur Behandlung von Property Change Events (`PropertyChangeListener`), zur Erzeugung von Timern (`Timer`) und zur Kommunikation über GWT-RPC (AsyncCallback). Daneben führt die Klasse `FAction` die Methode `doAction()` ein, welche von den Actions eines Action Chart implementiert werden. Die Laufzeitbibliothek stellt sicher, dass immer die zur spezifischen Action gehörige `doAction()` Methode aufgerufen wird. Die Klasse `FGuard` des Klassendiagramms aus Abbildung 4.12 modelliert die Bedingung an einer Transition. Zur Auswertung verfügt diese Klasse über eine `evaluate()` Methode. Ist die Bedingung erfüllt wird die Transition geschaltet und die nächste Action des Action Charts ausgeführt.

Zur korrekten Umsetzung der geänderten Transitionssemantik wurde die Fujaba Codegenerierung für Statecharts angepasst. Bei verwendetem Stereotyp `GWT` (vgl. Kapitel 4.3.1) werden die für Webapplikationen bereitgestellten Codegenerierungsmechanismen für Action Charts angestoßen. Jedes Action Chart gehört, wie bereits bei traditionellen Fujaba Statecharts üblich, zu einer Klasse. In die zum Action Chart gehörige Klasse wird nun für jede innerhalb des Action Charts modellierte Action eine eigene innere Klasse eingefügt. Jede dieser inneren Klassen leitet von der Laufzeitklasse `FAction` ab und implementiert die `doAction()` Methode. Die Implementierungsdetails der `doAction()` Methode finden sich hierbei in der zugehörigen Graphersetzungsregel des Action Charts.

```
1 private $upName $downName;
2 public class $upName extends FAction
3 {
4 public void doAction()
5 {
6 ###TypeOf[ ${activityDiag} ] := de.uni_paderborn.fujaba.uml.behavior.
      UMLActivityDiagram
7 #set( $activityDiag = $elem )
```

```
8  #parse("$lang/GWT:statechart/state/activityDiagBody.vm" )
9  #*##if($elem.sizeOfContains() != 0)
10 #* *##set( $stateChart = $elem.iteratorOfContains().next() )
11 ${utility.downFirstChar("$stateChart.getInitialState()")}.doAction();
12 #*##else
13 #*scan for event listeners to be added*#
14 #* *##foreach( $eventTrans in $elem.iteratorOfExit() )
15 #* *##set ( $eventName = $eventTrans.toString() )
16 #* *##if ( $eventName.endsWith(".click") )
17 #* *##set ( $objectName = $eventName.substring(0,$eventName.indexOf(".click")))
18 ${objectName}.addClickHandler($utility.downFirstChar("$eventTrans.getTarget()"));
19 #* *##elseif ( $eventName.endsWith(".select") )
20 #* *##set ( $objectName = $eventName.substring(0,$eventName.indexOf(".select")))
21 ${objectName}.addSelectionHandler($utility.downFirstChar("$eventTrans.getTarget()"))
       ;
22 #* *##elseif ( $eventName.endsWith(".change") )
23 #* *##set ( $objectName = $eventName.substring(0,$eventName.indexOf(".change")))
24 #* *##set ( $typeName = "no_type" )
25 #* *##set ( $typeName = $statechartVars.get($objectName) )
26 #* *##if ("TextBox" == $typeName || "TextArea" == $typeName)
27 ${objectName}.addChangeHandler($utility.downFirstChar("$eventTrans.getTarget()"));
28 #* *##else
29 ${objectName}.addPropertyChangeListener($utility.downFirstChar("$eventTrans.
       getTarget()"));
30 ...
```

Listing 4.5: Auszug aus dem Velocity Template zur Erzeugung der inneren Klassen für Actions eines Action Chart.

Listing 4.5 zeigt einen Ausschnitt des Velocity Templates zur Generierung der inneren Klassen das Action Charts. Für jede Action des Action Charts wird in Zeile 1 des Listings ein Feld angelegt. Anschließend wir eine innere Klasse mit dem Namen der Action erzeugt, die von der Laufzeitklasse `FAction` erbt (Zeile 2). Für das zum generieren aktive Element (die Action, die gerade generiert wird), wird in Zeile 8 des Listings 4.5 der Inhalt der enthaltenen Graphersetzungsregel eingelesen und als Inhalt der `doAction()` Methode in die Klasse generiert. Zeilen 13 - 29 sind für die Erzeugung der verschiedenen EventListener innerhalb der Methode `doAction()` zuständig. Hier werden die von der Action ausgehenden Transitionen überprüft und für die auftretenden Events Handler am spezifizierten Objekt angemeldet. Die Anwendung des Templates aus Listing 4.5 auf die Transition `deleteButton.click` aus Abbildung 4.10 resultiert im Quelltext aus Listing 4.6.

```
1 private CreatePanel createPanel;
2 public class CreatePanel extends FAction
3 {
4 public void doAction()
5 {
```

```
6 ...
7 deleteButton.addClickHandler(deleteEntryHandler);
8 ...
```

Listing 4.6: Generierter Quelltext der Transition `delteButton.click` aus Abbildung 4.10 bei Anwendung des Templates aus Listing 4.5.

Zustätzlich zu den inneren Klassen für jede der definierten Actions wird gleichsam eine innere Klasse für alle mit einer Bedingung behafteten Transitionen generiert. Jede der erzeugten inneren Klassen leitet von der in Abbildung 4.12 dargestellten Laufzeitklasse `FGuard` ab. Das Template aus Listing 4.7 zeigt den für die Erzeugung der Bedingungen zuständigen Ausschnitt des Codegenerierungs-Templates.

```
1  ...
2  #*generate inner FGuard classes*#
3  #***#set ( $amount = 1 )
4  #***#foreach( $autoTrans in $elem.iteratorOfExit() )
5  #* *##set ( $autoTransName = $autoTrans.toString() )
6  #* *##if ( $autoTransName.equals("auto") )
7  #* *##set ( $guardExpr = $autoTrans.getGuardExprText())
8  #* *##if ( $guardExpr.equals("NONE") || $guardExpr.equals("") )
9  #* *##set ( $fGuardName = "autoGuard"+${elem.toString()}+${amount})
10 #* *##else
11 #* *##set ( $fGuardName = $autoTrans.getGuardExprText().replaceAll("[' '|.|(|)]","")
       )
12 #* replace *#
13 #* *##set ( $fGuardName = $fGuardName.replace("==","Equals") )
14 #* *##set ( $fGuardName = $fGuardName.replace("!=","NotEquals") )
15 #* *##set ( $fGuardName = $fGuardName.replace(">","GreaterThan") )
16 #* *##set ( $fGuardName = $fGuardName.replace(">=","GreaterThanEquals") )
17 #* *##set ( $fGuardName = $fGuardName.replace("<","LessThan") )
18 #* *##set ( $fGuardName = $fGuardName.replace("<=","LessThanEquals") )
19 #* replace operands*#
20 #* *##set ( $fGuardName = $fGuardName.replace("+","Plus") )
21 #* *##set ( $fGuardName = $fGuardName.replace("-","Minus") )
22 #* *##set ( $fGuardName = $fGuardName.replace("*","MultipliedWith") )
23 #* *##set ( $fGuardName = $fGuardName.replace("/","DevidedBy") )
24 #* *##set ( $fGuardName = $fGuardName.replace("%","Modulo") )
25 #* *##end
26 #* *##set ( $fGuardUpName = $utility.upFirstChar($fGuardName) )
27 private $fGuardUpName $fGuardName;
28 private class $fGuardUpName extends FGuard
29 {
30   #* *##if ( $guardExpr.equals("NONE") || $guardExpr.equals(""))
31   #* *##else
32   public boolean evaluate()
33   {
34     return $guardExpr;
```

```
35   }
36 ...
```

Listing 4.7: Auszug aus dem Velocity Template zur Erzeugung der inneren Klassen für bedingungsbehaftete Transitionen eines Action Chart.

Im Rahmen des Codegenerierungsprozesses werden hier alle ausgehenden `auto` Transitionen einer Action ermittelt und zu jeder dieser Transitionen die daran im Action Chart modellierte Bedingung ermittelt (Zeile 4 - 8). Die daran anschließenden Zeilen des Listings 4.7 dienen der Berechnung des zu generierenden Klassennamens. Zeilen 27 - 35 schließlich sorgen für die Erzeugung eines Feldes für die Bedingung, sowie zur Definition der tatsächlichen Klasse. Ist eine Bedingung an der `auto` Transition vorhanden, so wird mit den Zeilen 32- 35 des Listings die Methode `evaluate()` in die Klasse eingefügt. Ist die Bedingung leer oder nicht vorhanden erzeugt das Template aus Listing 4.7 eine leere Klasse.

Um die gesamtheitliche Ausführung des Action Charts zu ermöglichen, werden in die Hauptklasse des Action Charts eine Initialisierungsmethode `initStatechart()`, sowie eine `start()` Methode generiert. Die generierte Initialisierungsmethode erzeugt zur Laufzeit eine Objektstruktur, welche die einzelnen Klassen und Transitionen des Action Charts instantiiert. Diese wird von der Laufzeitumgebung zur Abarbeitung des Action Charts verwendet. Die generierte `start()` Methode veranlasst die tatsächliche Ausführung des Action Charts. Hierzu wird zunächst die Initialisierungsmethode `initStatechart()` aufgerufen und so die benötigte Objektstruktur erzeugt. Anschließend wird die `doAction()` Methode der initialen Action des Action Charts aufgerufen und so der Ablauf des Action Charts tatsächlich gestartet.

Ist die Ausführung einer Action des Action Charts beendet, also die Methode `doAction()` am Ende ihrer Ausführung, wird zunächst automatisch die generierte Methode `doAuto()` aufgerufen. Diese Methode wird automatisch in alle Unterklassen von `FAction` eingefügt und behandelt im Action Chart vorkommende Transitionen vom Typ `auto`. Diese Transitionen sind zur Modellierung von synchronem Applikationsverhalten vorgesehen. Die Methode `doAuto()` überprüft zunächst ob Transitionen des entsprechenden Typs an der aktiven Action vorhanden sind und wertet im Anschluss die zugehörigen Bedingungen aus. Dies geschieht über die `evaluate()` Methode der zur Bedingung gehörigen Klasse `FGuard`. Stellt die Auswertung eine gültige Transition fest, wird die `doAction()` Methode der auf die Transition folgenden Action aufgerufen.

Abgesehen vom oben beschriebenen synchronen Applikationsverhalten mit `auto` Transitionen bilden Action Charts keinen strikten zeitlichen Ablauf ab. Jede Action des Action Charts kann während der Laufzeit zu jedem Zeitpunkt Empfänger eines Events sein und so gestartet werden. Technisch können die einzelnen Actions eines Action Charts daher als

abkürzende Schreibweise für parallele And-States (vgl. Fujaba Statecharts in [GZ05a]), versehen mit einem vorgeschalteten Wartezustand und einem abschließenden reaktiven Zustand gesehen werden. Ein Action Chart welches über mehrere reaktive Zustände verfügt, verfügt somit gleichzeitig über mehrere versteckte Wartezustände. Hier werden Eventhandler als zusätzliche And-Substates behandelt. Dabei stellen alle parallelen Substates Listener für unterschiedliche Events dar - jeder einzelne Substate ist für sein eigenes dediziertes User Interface, seinen eigenen Timer oder ein eigenes RPC Event zuständig. Da die clientseitige Ausführung als JavaScript innerhalb des Browsers in einem einzelnen Thread abläuft, kann jeweils nur ein Event zur Zeit verarbeitet werden, trotz der Modellierung komplexer und vielschichtiger Substates können daher keine Nebenläufigkeitsprobleme zwischen einzelnen parallelen Substates auftauchen.

Neben Events und bedingungsbehafteten Transitionen stellt die Modellierung von RPC-Calls einen großen Anwendungsbereich der Action Charts dar. Hierzu ist es notwendig, die aktive Aktion des Action Charts als Callback-Objekt mit an den Server zu übergeben. Dies wird hierbei nicht vom Laufzeitsystem sichergestellt, da der Aufruf einer Servermethode als CollaborationStatement innerhalb der Graphersetzungsregel modelliert wird und nicht Teil der Action Charts an sich ist. Die Implementierung der Klasse `FAction` der Laufzeitbibliothek stellt jedoch innerhalb der Implementierung der `onSuccess` und `onFailure` Methode sicher, dass der Applikationsablauf gemäß der im Action Chart angegebenen `success` und `failure` Transitionen fortschreitet. Der Aufruf der Methoden `onSuccess()` oder `onFailure()` wird über die Standardmechanismen von GWT realisiert. Die Implementierung der Methode `onSuccess()` innerhalb der Klasse `FAction` berechnet alle `success` Transitionen, die die aktuelle Action verlassen, und ruft auf den jeweiligen Ziel-Actions die Methode `doAction()` auf. Auf gleiche Weise behandelt die Implementierung der `onFailure()` Methode `failure` Transitionen. Zusätzlich zum Aufruf der `doAction()` Methode wird der `resultValue` der Ziel-Action basierend auf dem Ergebnis des Serveraufrufs gesetzt.

5 Workflowdiagramme - Einordnung und Semantik

Die vergangenen Jahre haben die Beschaffenheit und Nutzung des weltweiten Internet verändert. Dies ist nicht zuletzt der wachsenden Bandbreite, Rechenleistung und vor allem anderen der erhöhten Akzeptanz für komplexe Anwendungen und Applikationen im Internet geschuldet. Kapitel 2 beschreibt die Schritte, die während der Forschungsarbeiten im Bereich Webapplikationen getan wurden und legt die Entwicklung der hier vorgestellten Workflowdiagramme dar.

Die im folgenden vorgestellten Workflowdiagramme sind Teil des Gesamtprozesses zur Entwicklung komplexer Webapplikationen (vgl. Kapitel 7). Durch Einhaltung des Prozesses und Nutzung der Workflowdiagramme als Teil der notwendigen Spezifizierung können komplexe Applikationen mit speziellen Eigenschaften (vgl. Kapitel 3) modelliert werden. Eine dieser Eigenschaften der entstehenden Fujaba Web Applications, nämlich dass diese Workflow getrieben sind, stellt die Grundlage der hier vorgestellten Diagramme dar.

Im Kontext dieser Arbeit bedeutet Workflow getrieben, dass die Applikation explizit vor dem Hintergrund entwickelt wird, dem Endnutzer bei der Bearbeitung bestimmter (in einer vorbestimmten Reihenfolge auftretender) Aufgaben zu helfen. Traditionell wird während der Entwicklung einer Applikation zwar im Rahmen der Anforderungsanalyse die Art und Reihenfolge der möglichen Funktionen einer Applikation aufgenommen, jedoch im weiteren Entwicklungsprozess nicht weiter verfolgt. Im Rahmen unserer Arbeit stellten wir fest, dass viele Applikationen schlussendlich einen anderen Arbeitsablauf verfolgen als der Endnutzer. Der Benutzer wird von ausgegrauten Menüs oder Buttons frustriert, da ihm nicht klar ist, warum die Applikation sich anders verhält als der Nutzer es erwarten würde. Die Lücke zwischen dem im Rahmen der Anforderungsanalyse aufgenommenen vom Endbenutzer verfolgten Arbeitsablauf und dem tatsächlich innerhalb der Applikation implementierten bleibt bei traditioneller Entwicklung bestehen, weil nach der Anforderungsanalyse keine weiteren Mechanismen bestehen, um die Einhaltung des einmal identifizierten gewünschten Nutzungsablaufs zu überwachen.

Die vorliegende Arbeit steuert im Rahmen des vorgeschlagenen Web Fujaba Process (WFuP, Kapitel 7) mit der strikten Entwicklung der Applikation entlang eines vordefinierten Workflows entgegen. Nach Definition von textuellen Szenarien und Usecasedia-

grammen stellt der erste Schritt der konkreten Applikationsentwicklung nach dem WFuP die Definition des Workflows dar. Dieser spiegelt wieder, welche Schritte in welcher Reihenfolge der Endnutzer durchführt. Um die im WFuP intendierte genaue Überwachung der anschließenden Entwicklungsschritte zu gewährleisten, sollten die der Applikation zugrunde liegenden Workflows mit den hier definierten Workflow Diagrammen abgebildet werden. Die Workflow Diagramme, wie sie im WFuP und seinem angeschlossenen Tooling in Fujaba definiert sind, haben im Gegensatz zu anderen Workflow-Tools direkten Einfluss auf die weitere Entwicklung der Applikation und bieten somit im Gegensatz zu reinen grafischen Abbildungen einen echten Mehrwert. Der WFuP verwendet die definierten Workflow Diagramme sowohl zur Generierung von Quelltext, als auch zur (automatischen) Kombination von UI-Komponenten und automatisiertem Applikationsablauf. Auf diese Weise soll erreicht werden, dass die modellierte Applikation sich exakt mit dem zugrunde liegenden Workflow deckt und Verwirrung und Frustration seitens des Nutzers vermindert werden.

Im Sinne des WFuP wird für jeden im Workflow definierten Schritt in der weiteren Entwicklung ein User Interface entwickelt. Auf diese Weise gibt bereits die Definition des zugrunde liegenden Workflows Aufschluss über die zu entwickelnden UI-Komponenten. Dem Softwareentwickler wird so ein Handwerkszeug gereicht, welches gleichzeitig die Entwicklung der Applikation lenkt und das Resultat überwacht. Neben der Definition der UI-Komponenten dienen die Workflow Diagramme der automatischen Generierung des Applikationsablaufes, der von der Laufzeitumgebung der Applikation (vgl. Kapitel 6) überwacht und gesteuert wird. Die hier definierten Workflowdiagramme sind somit eng mit Prozess und Tooling verbunden.

Abbildung 5.1 zeigt beispielhaft einen Workflow für Reisebuchungen. Der gezeigte Workflow beschreibt die einzelnen Schritte, die zur Buchung einer Reise durchgeführt werden müssen. Im gezeigten Beispiel beginnt die Reisebuchung stets mit der Entscheidung zwischen einer Pauschalreise (packaged tour) oder einer Individualreise (individual tour). Je nach Vorliebe wird der gewünschte Pfad des Workflows für die weitere Bearbeitung der Reisebuchung ausgewählt. Die Buchung einer Pauschalreise beinhaltet zunächst eine Hotelauswahl. Mit der Auswahl des gewünschten Hotels wird hierbei gleichzeitig das Reiseziel festgelegt. Abhängig vom gewählten Hotel - und somit Reiseziel - können Flüge gebucht werden, sowie anschließende optionale Pakete, wie etwa Mietwagen oder „Zug zum Flug“. Viele Reiseveranstalter bieten darüber hinaus, etwa zu Hochzeitsreisen, Spezialangebote für Hochzeitspaare an, welche jedoch nur dann gebucht werden können, wenn die passenden Bedingungen erfüllt sind. Zu diesem Zweck muss der Reisekaufmann die benötigten Informationen an den Reiseveranstalter übermitteln. Zur Buchung solcher Spezialangebote beinhaltet der in Abbildung 5.1 gezeigte Workflow einen eigenen Schritt innerhalb des Pfades für Pauschalreisen, welcher die Eingabe und Absendung

aller benötigter Daten ermöglicht. Möchte der Kunde eine Individualreise buchen, so ist der Reisekaufmann in der Reihenfolge der Abarbeitung der einzelnen Buchungsschritte völlig frei. Aus diesem Grund sind alle im Pfad „Individualreise“auftretenden Schritte des Workflows optional. Dies ermöglicht somit beispielsweise die Buchung von Flügen ohne Hotel oder weitere Optionen.

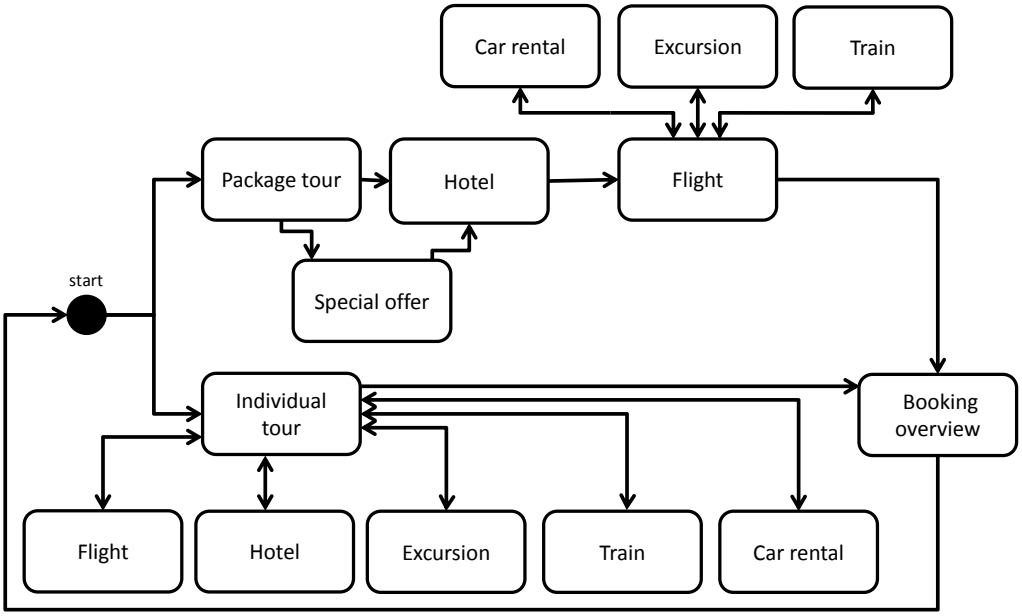

Abbildung 5.1: Skizzierter Workflow für Buchungen von Reisen.

Die in Abbildung 5.1 gezeigte Skizzierung des Workflows stellt eine erste Übersicht über die mit der gewünschten Applikation zu bearbeitenden Aufgaben dar. Hier soll sie als Grundlage für die Definition der Workflowdiagramme dienen. Eine Skizze wie in Abbildung 5.1 stellt für sich zunächst nur eine grafische Repräsentation dar. Abhängig davon, wer diese betrachtet wird, sind unterschiedliche Interpretationen der Grafik möglich. Ein Diagramm, welches zur weiteren Modellierung der Applikation verwendet werden soll, kommt indes nicht ohne weitere Definition aus. Eine genaue Semantik und die Definition aller zulässigen Diagrammteile sind notwendig, um einen gegebenen Workflow zweifelsfrei interpretieren und somit in einen eindeutigen Kontrollfluss für die zu entwickelnde Applikation übersetzen zu können. Im Beispiel kann etwa aus dem Zustand, dass der „Individual tour“mehrere ausgehende Kanten besitzt entweder gefolgert werden, dass alle Schritte optional sind, eine gültige Interpretation wäre jedoch auch, dass die Reihenfolge der Folgeschritte beliebig ist. Eine zwingend notwendige Definiton der Workflowdiagramme, wie sie vom WFuP interpretiert werden, wird im weiteren Verlauf dieses Kapitels dargestellt.

Die Entwicklung von Fujaba Web Applications nach Kapitel 3 mit dem in Kapitel 7 definierten Prozess baut auf den bereits vorhandenen Grundlagen des Story-Driven Modeling auf. Aus diesem Grund sind auch die Workflow Diagramme als Erweiterung und Unterstützung des gesamtheitlichen Story-Driven Modeling Ansatzes zu sehen. Mit Hilfe der Workflow Diagramme wird in diesem Rahmen die Möglichkeit geschaffen, User Interfaces implizit zu modellieren und somit die bisherigen Möglichkeiten von Activity-Diagrammen mit dedizierter UI Initialisierung abzulösen. In Kapitel 2 wurde bereits auf die Tatsache eingegangen, dass eine Modellierung von User Interfaces mit Hilfe von Activity-Diagrammen der handgeschriebenen Implementierung derselben nicht überlegen ist. Eine weitere Abstraktion, wie sie durch Workflow Diagramme und die implizite Erzeugung von User Interfaces und Applikationsfluss gegeben ist, bieten hier einen enormen Vorteil. Ausgehend vom Beispiel Workflow aus Abbildung 5.1 kann durch Anreicherung mit technischen Details ein Workflow Diagramm definiert werden. Das Ergebnis dieser Definition ist in Abbildung 5.2 dargestellt.

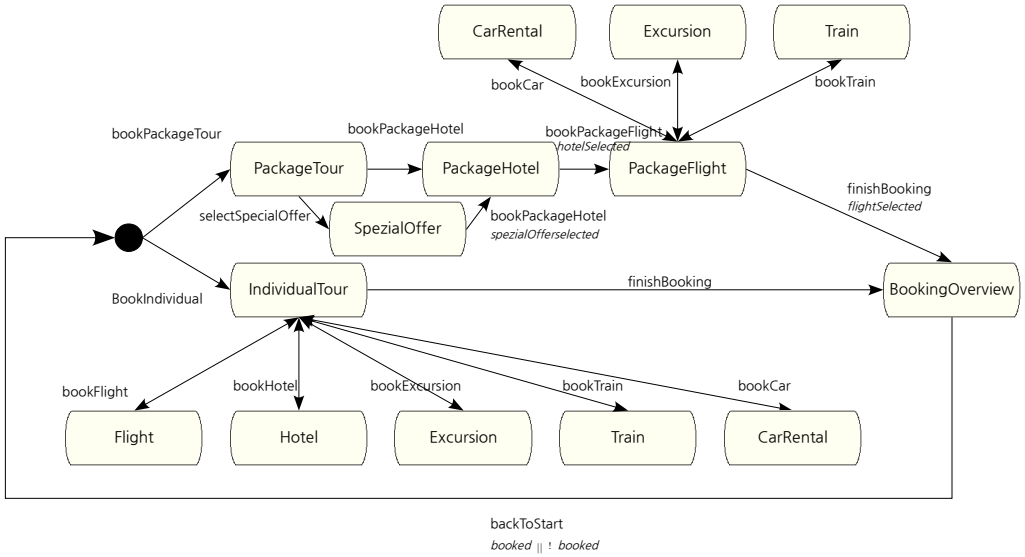

Abbildung 5.2: Valides `Workflowdiagramm` zur Reisebuchung aus Abbildung 5.1. Der Workflow wurde hier um technische Informationen angereichert.

Bei der Definition des Workflow Diagramms ist zu beachten, dass der grundsätzliche Prozessablauf soweit wie möglich bestehen bleibt. Es werden jedoch technische Details und unter Umständen Kantenbedingungen ergänzt. Desweiteren sollte im Hinblick der impliziten UI-Erzeugung darauf geachtet werden, dass alle Bereiche, in denen ein UI notwendig ist, auch als Prozessschritt vorliegen. Im obigen Workflow trifft dies beispielsweise auf den Startpunkt der Anwendung zu. Hier wird je nach gewünschter Buchung auf den Pfad für Pauschal- oder Individualreisen gewechselt. Eine Auswahl, welcher Pfad in der aktuellen Situation relevant ist, ist hier vonnöten. Aus diesem Grund ist eine Benutzer-

schnittstelle für den Startzustand notwendig. Je nach Toolimplementierung kann hier entweder automatisch ein User Interface für den Startzustand erzeugt werden, wenn dieser mehr als eine ausgehende Kante hat, oder aber es muss ins Diagramm ein Zwischenschritt eingefügt werden, welcher die Definiton der benötigten UI zulässt. Während der Workflow selbst noch mit beliebigen Zeichen- oder Business Process Modeling Notation (BPMN) -Tools gezeichnet werden konnte, trifft dies für die im Rahmen der Definiton von Workflow Diagrammen abgeleiteten Modellierung des Applikationsverhaltens nicht länger zu. Die Workflow Diagramme und ihre semantischen Eigenschaften sollen zur Codegenerierung verwendet werden. Deshalb ist es zu empfehlen, den im Fujaba-Tooling enthaltenen Editor für Workflow Diagramme zu verwenden, oder selbst sicher zu stellen, dass die semantische Interpretation des gezeichneten Workflows gemäß Definition 5.1 korrekt gehandhabt wird. Im dort implementierten Tooling muss für mehrere ausgehende Kanten am Startzustand ein weiterer Prozessschritt zur Definition der benötigten User Interfaces eingefügt werden. Eine Änderung hin zu automatisierter Erzeugung von UI kann im Rahmen weitergehender Entwicklungen am WFuP angedacht werden.

Definition 5.1 beinhaltet die semantischen Regeln der von mir für die Verwendung im WFuP definierten Workflow-Diagramme. Das implementierte Tooling hält sich exakt an diese Regeln.

Definition 5.1 (Workflowdiagramm) *Die Semantik eines Workflow Diagramms definiert sich dabei wie folgt:*

- *Jedes Workflow Diagramm besitzt einen dedizierten Startpunkt. Dieser wird durch einen gefüllten Kreis gekennzeichnet.*
- *Der Startpunkt des Workflow Diagramms muss mindestens über eine ausgehende Kante verfügen. Eine Beschränkung für die Anzahl ausgehender Kanten vom Startpunkt existiert nicht.*
- *Jeder Schritt innerhalb des Workflows wird durch ein abgerundetes Rechteck dargestellt, welches den Namen des Schrittes beinhaltet.*
- *Der Name jedes Schrittes stellt gleichzeitig den Namen der zugehörigen User Interface Klasse dar. Aus diesem Grund muss der gewählte Name jedes Schrittes den in der Zielprogrammiersprache gültigen Konventionen zur Benennung von Klassen folgen.*
- *Jeder Schritt im Workflow hat wenigstens eine eingehende Kante.*
- *Jeder Schritt im Workflow hat wenigstens eine ausgehende Kante.*

- *Jede Kante im Workflow besitzt einen Namen.*
- *Kanten können sowohl unidirectional als auch bidirectional angelegt werden.*
- *Für jede Richtung der Kante kann eine Bedingung angegeben werden.*
- *Workflows können anstatt eines Namens einen weiteren Workflow enthalten. Dies wird durch eine spezielle grafische Hervorhebung angezeigt. Im Fujaba-Toolign ist dies ein mit einem + versehenes Rechteck. Das eingebettete Workflow Diagramm folgt dabei den gleichen semantischen Voraussetzungen wie hier beschrieben.*

Die genaue Einhaltung der semantischen Definitionen (siehe Definition 5.1) eines Workflow Diagramms ist Grundvoraussetzung dafür, im Laufe der Codegenerierung einen vom Laufzeitsystem verwertbaren Applikationsablauf erzeugen zu können. Die genaue Funktionsweise dieses Laufzeitsystems wird in Kapitel 6 beschrieben. Die so gestaltete Nutzung von Workflow Diagrammen zur Spezifikation des Applikationsablaufs angelehnt an den Endnutzer-Workflow bietet die Chance, höhere Akzeptanz und einfacher nutzbare Applikationen zu erzeugen. Hierbei bleibt stets zu bedenken, dass das Vorgehen auf die in Kapitel 3 definierte Art von Applikationen beschränkt ist. Allein diese enge Fokussierung ermöglicht die Bereitstellung von Prozess und angekoppelten Tools für den Softwareentwickler und hilft so, die Anwendungsentwicklung so weit wie möglich zu automatisieren und das Abstraktionslevel so weit wie möglich anzuheben.

6 FujabaWebAI - eine Laufzeitumgebung für Fujaba Web Applications

Die im Rahmen der vorliegenden Arbeit vorgestellten Fujaba Web Applications (vgl. Kapitel 3) mit dem speziell abgestimmten Entwicklungsprozess (vgl. Kapitel 7), sehen als eines der Entwicklungsartefakte die in Kapitel 5 dargestellten Workflowdiagramme vor. Diese Diagramme beschreiben einen bestimmten Arbeitsablauf, für dessen Abarbeitung die Applikation entwickelt werden soll, und dienen dabei gleichzeitig der Generierung aller hierfür notwendigen assoziierten grafischen Benutzerschnittstellen. Jeder innerhalb des Workflowdiagramms definierte Schritt entspricht dabei einer grafischen Benutzerschnittstelle. Der Applikationsfluss und dadurch auch die Abfolge der Benutzerschnittstellen werden durch die Zusammensetzung des Workflowdiagramms beschrieben. Die Steuerung und Überwachung des modellierten Workflows wird durch die im Folgenden beschriebene Laufzeitumgebung - FujabaWebAI - sichergestellt.

6.1 Funktionsweise der FujabaWebAI Laufzeitumgbung

FujabaWebAI basiert auf einem clientseitigen interpretativen Mechanismus, der auf dem internen Modell (Metamodell) der Workflowdiagramme aufsetzt. Das zugrunde liegende Metamodell der modellierten Workflowdiagramme ist in Abbildung 6.1 dargestellt.

Hierbei stellt der `Workflow` gleichsam das Wurzelelement der Datenstruktur und das Diagramm dar. Ein `Workflow` wiederum setzt sich aus einer endlichen Anzahl an `WorkflowItems` zusammen. Hierbei handelt es sich entweder um `Start`-, `Stop`- oder `Activity`-Knoten. `Activity`-Knoten sind durch einen eindeutigen Namen gekennzeichnet. Dieser wird im `name`-Attribut abgespeichert und dient im Zusammenhang mit dem von mir entwickelten Tooling (vgl. Kapitel 7.2) ebenso zur Generierung der zugehörigen UI-Klasse. Eine Referenz auf die Fujaba-Repräsentation der UI-Klasse wird zusätzlich im Attribut `associatedFClass` als vollqualifizierter Name abgelegt. `Activity`-Knoten können wiederum `Workflows` enthalten. Auf diese Weise lassen sich komplexe Arbeitsabläufe strukturieren und in getrennten Workflowdiagrammen darstellen. Die Verbindung zwischen allen im Diagramm vorkommenden `WorkflowItems` wird über die Klasse `Transition` realisiert. Diese verfügt wiederum über ein Attribut `condition`, über wel-

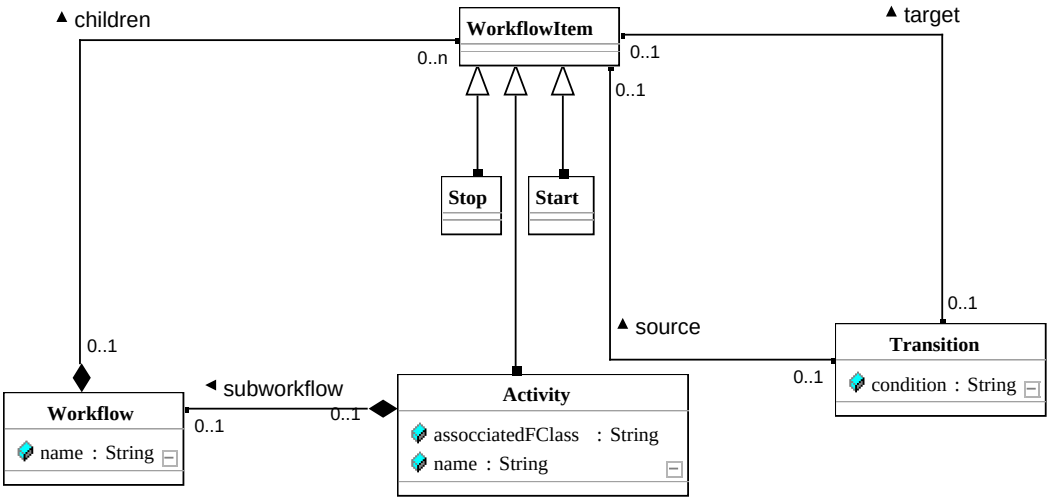

Abbildung 6.1: Internes Modell des Workflowdiagrammeditors

ches eine Bedingung, bzw. ein Event für den Übergang zwischen beiden Knoten angegeben werden kann. Das konform zum Klassendiagramm nach Abbildung 6.1 modellierte Workflowdiagramm wird in Form einer XML-Datei abgespeichert. Das der XML-Datei zugrunde liegende Schema kann der Document Type Definiton (DTD) aus Listing 6.1 entnommen werden.

```
1  <?xml version="1.0" encoding="UTF-16LE"?>
2  <!ELEMENT subworkflow (#PCDATA)>
3  <!ELEMENT start (name)>
4  <!ELEMENT stop (name)>
5  <!ELEMENT name (#PCDATA)>
6  <!ELEMENT associatedFClass (#PCDATA)>
7  <!ELEMENT activity (name,(subworkflow)?,(associatedFClass)?)>
8  <!ELEMENT activities (activity)+>
9  <!ELEMENT condition (#PCDATA)>
10 <!ELEMENT source (#PCDATA)>
11 <!ELEMENT target (#PCDATA)>
12 <!ELEMENT transition ((name)?,(condition)?,source,target)>
13 <!ATTLIST transition type CDATA "">
14 <!ELEMENT transitions (transition)+>
15 <!ELEMENT workflow (start,stop,activities,transitions)>
16 <!ATTLIST workflow name CDATA #REQUIRED>
17 <!ELEMENT workflowdiagram (workflow)+>
```

Listing 6.1: DTD Schema zur Speicherung der modellierten Workflowdiagramme zur Weiterverarbeitung in der Laufzeitumgebung.

Wie aus Listing 6.1 ersichtlich wird, unterscheidet sich das Schema der XML-Datei dahingehend vom in Abbildung 6.1 dargestellten Metamodell der Workflowdiagramme,

dass die Gesamtheit aller modellierten Workflows in einer einzelnen XML-Datei enthalten sind. Hierfür wird in der DTD in Zeile 17 des Listings 6.1 das Element `workflowdiagram` definiert, welches mehrere `workflow` Elemente enthalten kann. Auf diese Weise werden die im Rahmen mehrerer Diagramme definierten Workflows und Subworkflows, die den Arbeitsablauf der Applikation definieren in einer einzelnen XML-Beschreibung zusammengefasst. Dies ist zur einfacheren Interpretation des Gesamtablaufes der Applikation im Laufzeitsystem notwendig. Ein `workflow` Element enthält, basierend auf der DTD aus Listing 6.1, die Elemente `start`, `stop`, `activities`, sowie `transitions`. Hierbei handelt es sich um Beschreibungen der `WorkflowItems` und `Transitions` aus dem Metamodell (vgl. Abbildung 6.1).

Die Gesamtheit aller innerhalb eines Workflows definierten `Activitys` und `Transitions` wird im Rahmen der XML-Beschreibung durch die Einführung der Elemente `activities` und `transitions` realisiert. Diese Elemente beinhalten die eigentlichen Beschreibungen der im Workflow enthaltenen `Transitionen` und `Activities` und dienen somit als Sammelelement. Ein Element vom Typ `activity`, wie in Zeile 7 der DTD definiert, beinhaltet einen `name` und wahlweise eine `associatedFClass` oder einen `subworkflow`. Alle enthaltenen Eigenschaften werden durch die Angabe einer Zeichenkette beschrieben. So wird auch das Mapping zwischen einem in der `activity` enthaltenen `subworkflow` Element und dem dazugehörigen `workflow` Element über die Namensgleichheit vorgenommen. Gleiches gilt für die Realisierung der `transition` Elemente. Diese beinhalten ein Zeichenkette `condition`, sowie Angaben zu `source` und `target` der Transition. Aus dem Metamodell wird ersichtlich, dass es sich hierbei um `WorkflowItems` handelt. Innerhalb der XML-Beschreibung wird die Angabe des assoziierten Elementes durch Angabe des Namens und vorliegende Namensgleichheit realisiert.

Im Zuge des Codegenerierungsschrittes der Fujaba Web Applications werden alle erstellten Workflowdiagramme zu der gemeinsamen XML-Beschreibung zusammengefasst. Diese Beschreibung des Gesamtablaufes wird von der Laufzeitumgebung zur Interpretation des gewünschten Applikationsverhaltens verwendet. Zu diesem Zweck wird die XML-Beschreibung von der Laufzeitumgebung geladen und clientseitig zurück in eine Repräsentation des Datenmodells übersetzt.

Eine Übersicht über die Klassen der Laufzeitumgebung ist in Abbildung 6.2 dargestellt. Die Klasse `Statemachine` ist als Singletopn realisiert und nimmt den Hauptteil der Laufzeitumgebung ein. Über die Methode `initWithXML()` wird der Laufzeitumgebung die als XML-Beschreibung vorliegende Ablaufbeschreibung zur Verfügung gestellt. Die Datei wird dabei eingelesen und anschließend in eine clientseitige Repräsentation des Metamodells überführt.

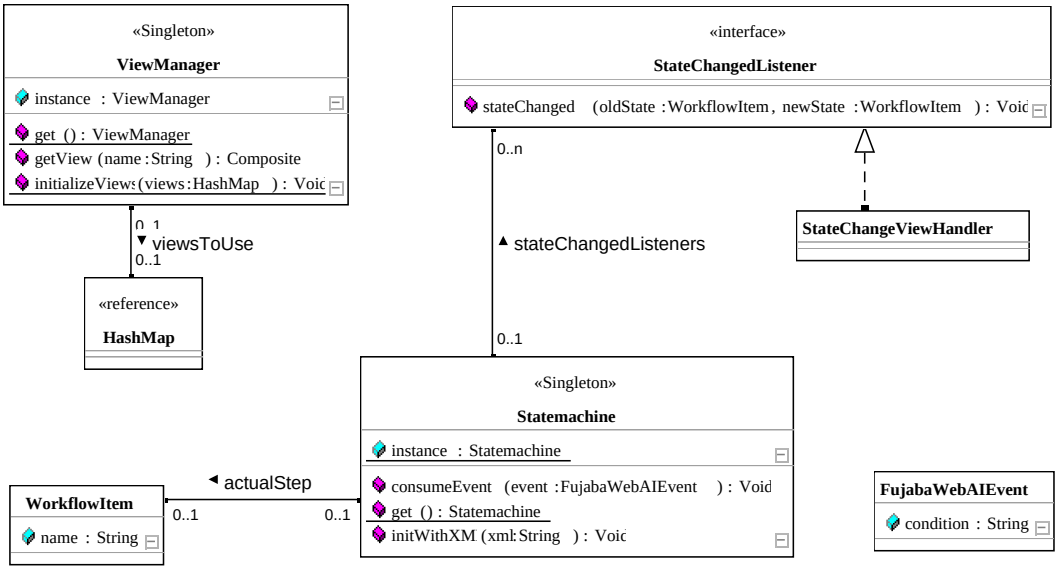

Abbildung 6.2: Klassendiagramm der FujabaWebAI Laufzeitbibliothek

Über die Assoziation `actualStep` wird hierbei der aktuell gültige Arbeitsschritt in der `Statemachine` abgespeichert. Eine Weiterschaltung der Arbeitsschritte geschieht Event-basiert. Zu diesem Zweck wurde das `FujabaWebAIEvent` eingeführt. Dieses verfügt über eine `condition`, welche die an der `Transition` des Workflows angegebene Bedingung repräsentiert. Auch hier wird über Gleichheit der angegebenen Zeichenketten eine Erfüllung der Bedingung realisiert. Zum Verarbeiten auftretender Events verfügt die `Statemachine` über die Methode `consumeEvent()`. Diese kann von beliebigen Stellen der Applikation auf der Instanz der `Statemachine` aufgerufen werden. Abhängig vom zu verarbeitenden Event wird von der `Statemachine` nun der nächste zu bearbeitende Prozessschritt berechnet und der `actualStep` auf diesen umgeschaltet.

Die Klasse `StateChangeViewHandler` implementiert das Interface `StateChanged-Handler` und ist bei der `Statemachine` als `stateChangedListener` angemeldet. Durch einen Aufruf der Methode `stateChanged()` beim Umsetzen des `actualStep` wird nun der zuständige Handler (`StateChangeViewHandler`) aufgerufen und verarbeitet die aufgetretene Änderung indem die grafische Benutzerschnittstelle der Applikation entsprechend des im Workflowdiagramm modellierten Applikationsablaufes angepasst wird.

Jeder im Workflow vorhandene Prozessschritt (`Activity`) resultiert bei Entwicklung nach dem WFuP (vgl. Kapitel 7) in der impliziten Erzeugung einer gleichnamigen UI-Komponente. Bei einer Weiterschaltung der Prozessschritte im Workflowdiagramm wird somit automatisch das UI entsprechend ausgetauscht und so der Applikationsfluss implementiert. Um eine Weiterschaltung der UI zu ermöglichen und die dedizierten UI-Komponenten entsprechend ein- bzw. auszublenden, ist die Klasse `ViewManager` imple-

mentiert. Diese Klasse ist dafür zuständig, alle im Workflow definierten Views (Prozessschritte) zu verwalten und basierend auf dem gegebenen Namen des Prozessschrittes die zugehörige View-Komponente zu ermitteln. Der `ViewManager` wird zu diesem Zweck mit allen im Workflow enthaltenen Views initialisiert. Dies geschieht durch Aufruf der Methode `initializeViews()` beim Start der Applikation.

6.2 Verwendung der FujabaWebAI Laufzeitumgebung

Die korrekte Funktionsweise der FujabaWebAI Laufzeitumgebung ist abhängig von einigen Randbedingungen. So sind grundlegende Schritte zum Initialisieren der Laufzeitumgebung ebenso nötig, wie die Sicherstellung der Einhaltung gegebener Namenskonventionen. Folgende Punkte sind bei der Verwendung der Laufzeitumgebung in Fujaba Web Applications zu beachten:

- Einbinden der Laufzeitumgebung in die zu erstellende Applikation
- Initialisierung der Laufzeitumgebung mit der erzeugten XML-Beschreibung
- Initialisierung der Laufzeitumgebung mit allen durch den Workflow modellierten zur Verfügung stehenden Views
- Erzeugen und konsumieren der `FujabaWebAIEvents` an gewünschten Stellen innerhalb der Applikationslogik
- Einhaltung von Konventionen bei der Verwendung von Zeichenketten

Einige der zur Verwendung der Laufzeitumgebung zu beachtenden Punkte werden durch das in Kapitel 7.2 beschriebene Tooling automatisch bereitgestellt. Aus Zeitgründen sind jedoch nicht alle möglichen Automatisierungen innerhalb der Fujaba Web Tools implementiert. Weitere Punkte der obigen Auflistung können nicht automatisiert werden, sondern müssen seitens des Entwicklers bei der Erstellung von Fujaba Web Applications bedacht, beziehungsweise implementiert werden.

6.2.1 Verwendung der FujabaWebAI am einfachen Beispiel - SimpleFujabaWebAIExample

Zum einfacheren Verständnis der notwendigen zu behandelnden Schritte soll im Folgenden die Verwendung der Laufzeitumgebung am einfachen Beispiel der Applikation `SimpleFujabaWebAIExample` erläutert werden. Der Ablauf der Applikation besteht aus drei einfachen Arbeitsschritten, die im Workflowdiagramm in Abbildung 6.3 dargestellt

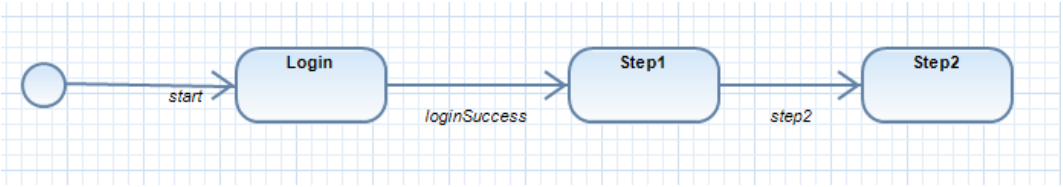

Abbildung 6.3: Workflowdiagramm einer einfachen Beispielapplikation zur Veranschaulichung der Verwendung der FujabaWebAI Laufzeitumgebung.

sind. Ausgehend vom Startzustand der Applikation führt eine Transition mit der Bedingung `start` zum ersten Ablaufschritt `Login`. Der Arbeitsschritt `Login` wird mit einer Transitionsbedingung `loginSuccess` verlassen und führt zum Ablaufschritt `Step1`. Dieser wiederum führt über die Transitionsbedingung `step2` zum letzen Ausführungsschritt.

Gemäß der Definition der Workflowdiagramme (vgl. Kapitel 5 und der entsprechenden Implementierung seitens der Fujaba Web Tools (vgl. Kapitel 7.2) resultiert jeder der definierten Workflowschritte in einer eigenen UI-Komponente. Jede Komponente wird durch eine eigene Java-Klasse realisiert, die konform zum Workflowschritt benannt ist. Jede der Klassen definiert eine eigene View, zu diesem Zweck leiten die erzeugten Klassen vom GWT-Widget `Composite` ab. Die Applikation `SimpleFujabaWebAIExample` definiert ein eigenes GWT-Module in der Datei `SimpleFujabaWebAIExample.gwt.xml`.

Einbinden der Laufzeitumgebung FujabaWebAI

Zum Verwenden der Laufzeitumgebung in der Beispielapplikation ist eine Einbindung der zugehörigen Bibliothek in das Modul der Beispielapplikation notwendig. Diese Bibliothek liegt in Form des `FujabaWebAI.jar`-Archivs vor. Wie für GWT-Bibliotheken notwendig, beinhaltet dieses Archiv alle zur Laufzeitumgebung gehörigen Klassen sowohl als Sourcecode, wie auch als Kompilat in Form einer `.class`-Datei. Neben den reinen Klasseninformationen ist die Moduldefinition der Laufzeitumgebung in Form einer `.gwt.xml`-Datei im Archiv vorhanden.

Die Bibliothek der Laufzeitumgebung wird im Ordner `war/WEB-INF/lib` der Beispielapplikation abgelegt. Um die so zur Verfügung gestellte Bibliothek innerhalb der Beispielapplikation nutzen zu können, ist zusätzlich die Einbindung des Moduls der Laufzeitumgebung in das GWT-Modul der Beispielapplikation notwendig. Dies geschieht durch Einfügen des Befehls aus Listing 6.2 in die zur Beispielapplikation gehörige Moduldefinition (`SimpleFujabaWebAIExample.gwt.xml`).

```
1  <inherits name='de.unikassel.gwtw2m.FujabaWebAI.FujabaWebAI'/>
```

Listing 6.2: Befehl zum Einbinden von FujabaWebAI in die Moduldefinition der Beispielapplikation.

Ist die Bibliothek der Laufzeitumgebung gemäß Listing 6.2 eingebunden, stehen alle dort definierten Klassen zur Nutzung bereit und können im Sourcecode oder innerhalb von Fujaba-Graphersetzungsregeln der Beispielapplikation verwendet werden.

Initialisierung der Laufzeitumgebung

Nachdem die Laufzeitumgebung in die Beispielapplikation eingebunden wurde, muss sie initialisiert werden, um den beschriebenen Applikationsablauf korrekt interpretieren zu können. Zu diesem Zweck wird im Rahmen der Codegenerierung der in den zugehörigen Workflowdiagrammen modellierte Ablauf in einer XML-Beschreibung als `workflow.xml` innerhalb des `war`-Ordners der Applikation abgespeichert. Der aus dem in Abbildung 6.3 dargestellte Worflow resultiert hierbei in der `workflow.xml` aus Listing 6.3

```
1  <?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE workflow SYSTEM "workflowdiagram.
       dtd">
2  <workflowdiagram>
3    <workflow name="TestMachine">
4      <start>
5        <name>Start</name>
6      </start>
7      <activities>
8        <activity>
9          <name>Login</name>
10         <associatedFClass>Login</associatedFClass>
11       </activity>
12       <activity>
13         <name>Step1</name>
14         <associatedFClass>Step1</associatedFClass>
15       </activity>
16       <activity>
17         <name>Step2</name>
18         <associatedFClass>Step2</associatedFClass>
19       </activity>
20     </activities>
21     <transitions>
22       <transition type="unidirectional">
23         <name>start</name>
24         <source>Start</source>
25         <target>Login</target>
26       </transition>
```

```
27        <transition type="unidirectional">
28          <name>loginSuccess</name>
29          <source>Login</source>
30          <target>Step1</target>
31        </transition>
32        <transition type="unidirectional">
33          <name>step2</name>
34          <source>Step1</source>
35          <target>Step2</target>
36        </transition>
37      </transitions>
38    </workflow>
39  </workflowdiagram>
```

Listing 6.3: Resultierende `workflow.xml` nach Codegenerierung der Beispielapplikation.

Zur Initialisierung der Laufzeitumgebung mit dem zu behandelnden Workflow muss die erzeugte XML-Beschreibung von der Laufzeitumgebung eingelesen und so ein clientseitiges Modell des abzuarbeitenden Workflows erzeugt werden. Dies sollte beim Starten der Beispielapplikation vorgenommen werden. Die `SimpleFujabaWebAIExample` Applikation definiert, wie für GWT-Applikationen üblich, ihren Startpunkt über die Implementierung des GWT-Interfaces `EntryPoint`. Die zugehörige Klasse `SimpleFujabaWebAIExample` wird von den Fujaba Web Tools automatisch erzeugt. Die Initialisierung der Laufzeitumgebung wird durch Einfügen der Quelltextzeile aus Listing 6.4 in die Methode `onModuleLoad()` der Klasse `SimpleFujabaWebAIExample` durchgeführt.

```
1  Statemachine.get().initWithXml("workflow.xml");
```

Listing 6.4: Notwendige Quelltextzeile zur Initialisierung der `Statemachine` mit dem modellierten Workflow.

Das interpretative Verhalten der Laufzeitumgebung sorgt für ein korrektes Weiterschalten der im Workflow definierten UI-Komponenten, basierend auf dem Zustand der Applikation. Um ein korrektes Weiterschalten zu ermöglichen, ist es notwendig, alle verfügbaren Komponenten beim Start der Applikation an der Laufzeitumgebung anzumelden. Da im Rahmen von GWT wie bereits in Kapitel 4.2.3 erwähnt keine reflektiven Zugriffe und auch kein reflektives Erzeugen von Klassen möglich ist, wird bereits beim Start der Applikation je eine Instanz aller verfügbarer UI-Komponenten erzeugt. Die erzeugten Komponenten werden im Anschluss von der Klasse `ViewManager` der Laufzeitumgebung verwaltet. Hierbei dient der Name des zugehörigen Workflowschrittes zur Identifizierung der UI-Komponente. Zur Ininitalisierung des ViewManager im Falle der `SimpleFujabaWebAIExample` werden die in Listing 6.5 definierten Zeilen in die Methode `onModuleLoad()` der `EntryPoint`-Klasse eingefügt.

```
1   Login login = new Login();
2   Step1 step1 = new Step1();
3   Step2 step2 = new Step2();
4
5   HashMap<String, Composite> views = new HashMap<String, Composite>();
6   views.put("Login", login);
7   views.put("Step1", step1);
8   views.put("Step2", step2);
9
10  ViewManager.initializeViews(views);
```

Listing 6.5: Notwendige Quelltextzeilen zur Initialisierung des `ViewManager`

Das eigentliche Umschalten zwischen verschiedenen UI-Komponenten wird über die Klasse `StateChangeViewHandler` der Laufzeitumgebung realisiert. Diese wird innerhalb der Methode `onModuleLoad()` der Klasse `SimpleFujabaWebAIExample` als `StateChangedHandler` bei der `Statemachine` angemeldet. Sind alle zuvor beschriebenen Vorbedingungen erfüllt, ist die Laufzeitumgebung initialisiert und kann basierend auf auftretenden `FujabaWebAIEvents` den Ablauf der Beispielapplikation steuern.

Applikationsfluss Steuerung durch Events

Die konkrete Steuerung des Applikationsflusses kann nicht automatisch aus den modellierten Workflows erzeugt werden, da hier keine Angaben dazu gemacht sind, zu welchem Zeitpunkt die zur Transition gehörige Bedingung in Form eines Events auftreten soll. Stattdessen muss der Entwickler der Applikation dafür sorgen, bei Bedarf an den gewünschten Stellen innerhalb der Applikationslogik die notwendigen Events zu erzeugen und diese über die `Statemachine` verarbeiten zu lassen. Je nach gewünschter Funktionalität der Applikation können die entsprechenden Befehle als Teil der Implementierung eines Server-Callbacks, UI-Listeners, im Rahmen eines Timers oder aber bei definierten Zuständen des Datenmodells umgesetzt werden. Die Steuerung der Transition mit der Bedingung `loginSuccess` aus Abbildung 6.3 etwa, erfolgt nach erfolgreicher Anmeldung am Server innerhalb der `onSuccess()`-Methode des Callbacks durch die Befehle aus Listing 6.6.

```
1  FujabaWebAIEvent success = new FujabaWebAIEvent("loginSuccess");
2  Statemachine.get().consumeEvent(success);
```

Listing 6.6: Befehle zum Schalten der Transition `loginSuccess` aus Abbildung 6.3.

Konventionen bei der Verwendung von Zeichenketten

Die Verwendung von Zeichenketten spielt im Rahmen der Laufzeitumgebung eine große Rolle.

- `FujabaWebAIEvents` verfügen über eine Bedingung, welche als Zeichenkette realisiert ist
- `Activities` innerhalb des Workflows werden über ihren Namen identifiziert
- `Activities` und die zugehörigen View-Klassen werden über die Gleichheit der Namen miteinander in Verbindung gebracht

An allen zuvor genannten Stellen ist es notwendig, im Rahmen der Entwicklung dafür zu sorgen, die vom Laufzeitsystem vorausgesetzten Konventionen einzuhalten. Insbesondere beim Erzeugen der notwendigen `FujabaWebAIEvents` muss sichergestellt sein, dass die am Event definierte Bedingung mit der innerhalb des Workflowdiagramms definierten Bedingung an der entsprechenden Transition übereinstimmt. Der von der Laufzeitumgebung vorgenommene Abgleich der Bedingungen, sowohl des konsumierten Events als auch des modellierten Workflows, erfolgt auf der Clientseite der Applikation durch Stringvergleich. Ein korrektes Schalten der Transition kann nur bei Gleichheit beider Zeichenketten gewährleistet werden.

Gleiches gilt für die von der Laufzeitumgebung verwendete Verwaltung der möglichen View-Komponenten und die zugehörigen Workflowschritte. Bei der Initialisierung des `ViewManager` muss sichergestellt sein, dass die zur Identifizierung der View verwendete Zeichenkette mit der Namensangabe des zugehörigen Workflowschrittes übereinstimmt. Durch die mögliche Automatisierung, indem die Ininitalisierungsbefehle ähnlich wie in Listing 6.5 an die entsprechende Stelle der `EntryPoint`-Klasse generiert werden, kann an die Fehlerquelle jedoch an dieser Stelle minimiert werden.

7 Methodisches Entwickeln mit Fujaba - Prozess und Tools

Wie bereits aus Kapitel 1 hervorgeht, teilt sich die Arbeit meiner Dissertation auf mehrere Teilbereiche auf. Unter dem Gesamtaspekt der modellgetriebenen Entwicklung von Webapplikationen entstand als Hauptteil meiner Dissertation der im folgenden Kapitel definierte Prozess zur Entwicklung komplexer Webapplikationen mit der Fujaba Toolsuite nach dem Story-Driven Modeling Ansatz.

Die genaue Art der Applikationen, die mit diesem Ansatz entstehen, ist in Kapitel 3 definiert. Die notwendigen Änderungen an den Codegenerierungsanteilen und dem Replikationsframework von Fujaba kann in Kapitel 4 nachgelesen werden. Das von mir entwickelte und über das hier beschriebene Tooling angebundene Laufzeitsystem FujabaWebAI ist in Kapitel 6 und die im Prozess verwendeten Workflowdiagramme sind in Kapitel 5 beschrieben.

Die folgenden Kapitel stellen den kompletten von mir entwickelten Web Fujaba Process, sowie die zur Unterstützung an den Pozess angegliederten Werkzeuge dar.

Mein Ziel, die Notwendigkeit für handgeschriebenen Quelltext auf ein Minimum zu reduzieren und dabei dem Entwickler ein methodisches Vorgehen inklusive Tools zur Verfügung zu stellen, stand im Rahmen meiner Forschungen stets im Vordergrund. Alle übrigen Anteile meiner Dissertation wurden entwickelt, um für den hier dargestellten Web Fujaba Process die notwendigen Grundlagen innerhalb von Fujaba zu schaffen und den intendierten Entwicklungsablauf bestmöglich zu unterstützen.

Der Web Fujaba Process beschreibt dabei die schrittweise Entwicklung einer Fujaba Web Applikation analog zu Kapitel 3 und ist dabei eng mit den zugrunde liegenden Technologien verzahnt. Toolunterstützung wird durch die Fujaba Web Tools geleistet, welche zum Großteil im Rahmen meiner Dissertation implementiert wurden. Um dem Anspruch, möglichst alle Teile der Anwendungsentwicklung auf Modellebene zu halten, gerecht zu werden, bedient sich der Entwicklungsprozess neben unterschiedlichen UML Diagrammarten der in der Fujaba Toolsuite verankerten Technologie des Story-Driven-Modeling. Die bereits vorhandenen Diagramme und Codegenerierung in Fujaba ermöglichen hier eine grafische Spezifikation der Applikationslogik.

7.1 Der Web Fujaba Process - WFuP

Der Web Fujaba Process (WFuP) als methodische Unterstützung bei der Entwicklung komplexer Webapplikationen stellt einen prinzipiell von jeglichen Tools unabhängigen Entwicklungsprozess dar. Die Einbindung von Story-Driven Modeling Mechanismen in den Prozess legt jedoch eine Bearbeitung mit Fujaba nahe. Fujaba stellt zum jetzigen Zeitpunkt das meines Wissens einzige Tool mit Story-Driven Modeling Unterstützung dar.

Der WFuP besteht grob aus zwei Teilen. Dem bereits bestehenden Fujaba Process, siehe [Gei11] und dem von mir im Rahmen der Dissertation als Ergänzung entwickelten Teil zur Entwicklung komplexer Webapplikationen. Neben dieser Zweiteilung ist zu beachten, dass der WFuP prinzipiell auch für die Entwicklung komplexer Desktopapplikationen verwendet werden kann. Hierzu müsste lediglich die im Prozess bestehende Teilung zwischen Server und Client neu interpretiert und die für den Client vorgesehenen Prozessanteile zur alleinigen Definition der grafischen Benutzerschnittstelle verwendet werden. Eine komplette Übersicht über den WFuP stellt Abbildung 7.1 dar.

Den Startpunkt des WFuP bildet, genau wie bereits im FuP definiert, die Erstellung textueller Szenarien und Usecasediagramme (1). Dieser Schritt wurde im Rahmen des WFuP nach außen gezogen und muss nicht zwingend mehrmals durchlaufen werden. Zur Aufnahme der benötigten Anforderungen können beliebige Methoden des Requirements Engineering zur Unterstützung verwendet werden. Einen Leitfaden zur Erstellung guter Szenarien bietet [NZJ13]. Die Aufnahme von Anforderungen und die daraus abgeleiteten textuellen Szenarien und Usecasediagramme dienen unter anderem zur Abstimmung mit nichttechnischen Auftraggebern. Textuelle Szenarien und Usecases bieten eine relativ einfach verständliche Möglichkeit, gemeinsam über die genaue Definition der Anforderungen an die zu erstellende Applikation zu sprechen.

Die Serverseite der Webapplikation ebenso wie Teile des Datenmodells werden aus diesen Anforderungen in bewährter Weise mittels des Fujaba Process [Gei11] erstellt (2) und der entsprechende Quelltext generiert.

Der Entwicklungsprozess für Webapplikationen sieht im weiteren Verlauf die Definition des Workflows (3) vor. Hier soll exakt jener Userworkflow abgebildet werden, der aus den Anforderungen und Szenarien ableitbar ist und somit das intendierte Nutzungsverhalten der Applikation am Besten widerspiegelt. Der komplette Ablauf der Applikation wird in einem oder mehreren Workflowdiagrammen festgehalten. Zur Definition der Workflowdiagramme bietet sich der im Tooling enthaltene Workflowdiagramm Editor an. Alle weiteren Schritte des Prozesses basieren auf den zuvor erstellten Workflowdiagrammen.

Für jeden im Workflowdiagramm definierten Schritt wird im Folgenden eine View der grafischen Benutzerschnittstelle erstellt (4). Für jede View wird eine Klasse im Klassendiagramm für die grafische Benutzerschnittstelle angelegt. Ausgehend von diesem Klassendiagramm wird für jede Klasse ein Design der Benutzerschnittstelle durchgeführt (5) und in einem anschließenden Schritt die dort definierten strukturellen Informationen über die UI, in Form der Zusammensetzung der grafischen Benutzeroberfläche, wieder zurück ins Klassendiagramm gespiegelt (6). Darauf aufbauend werden sowohl die notwendigen Server Calls oder Controller modelliert (7), als auch Quelltext aus dem bestehenden Klassendiagramm sowie allen assoziierten Diagrammen generiert (8). Die so entstandenen Java Quelltexte werden im abschließenden Schritt für die unterschiedlichen Browser und Sprachen vom Crosscompiler des *Google Web Toolkit* [GWTa] übersetzt (9). Der resultierende JavaScript Code enthält eine vollständig lauffähige Webapplikation.

Der Web Fujaba Process aus Abbildung 7.1 wurde zuerst in [EGHZ12] veröffentlicht. Die folgenden Kapitel beschreiben detailliert die Ausführungsschritte des Prozesses mit der von mir implementierten Toolunterstützung der Fujaba Web Tools.

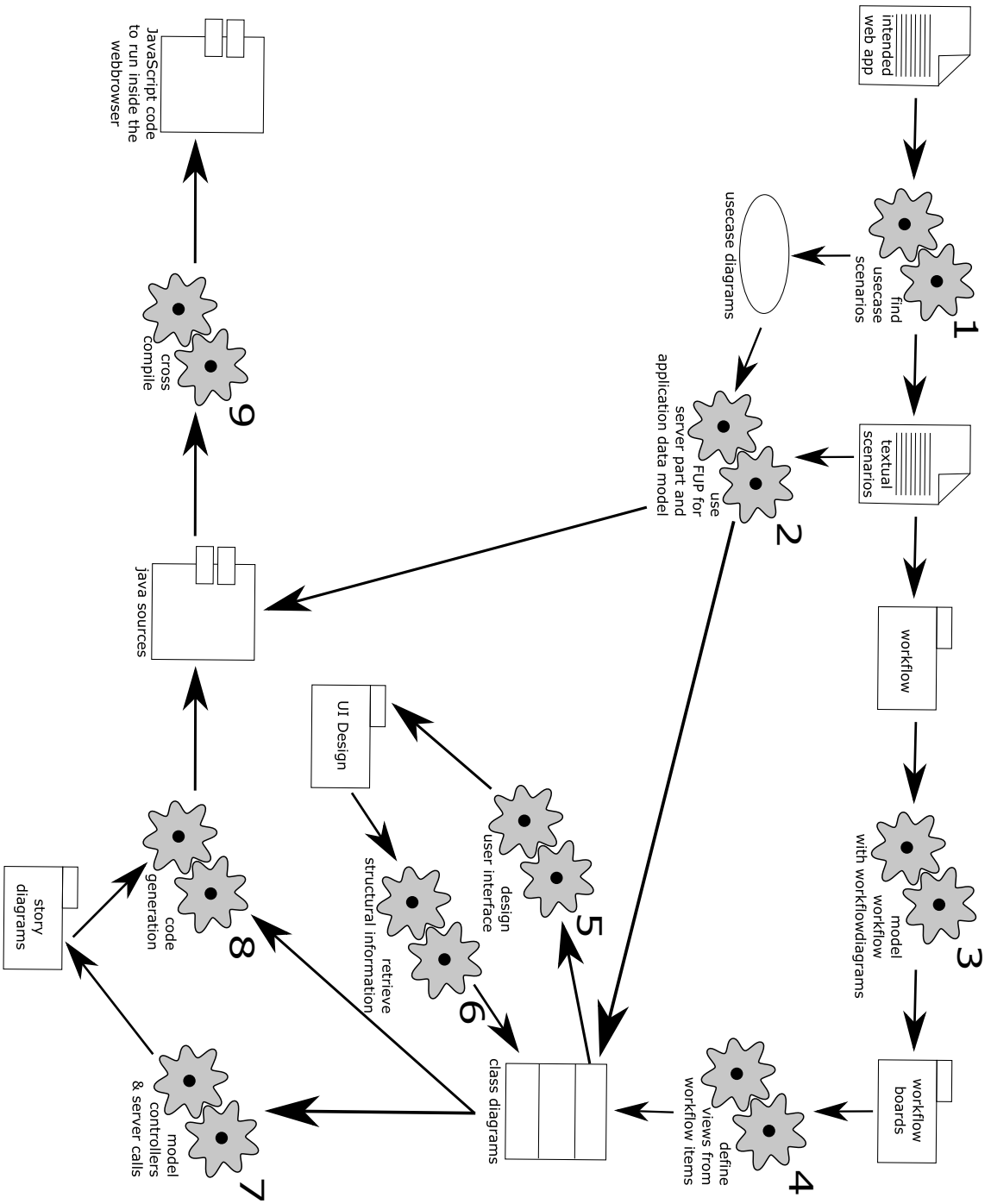

Abbildung 7.1: Schematische Darstellung des Web Fujaba Process

7.2 Die Fujaba Web Tools

Der Web Fujaba Process als methodischer Softwareentwicklungsprozess ist prinzipiell nicht an eine bestimmte Programmiersprache oder Entwicklungsumgebung gebunden. Die mit den Fujaba Web Tools angebotene und von mir entwickelte Toolunterstützung mittels der Fujaba Web Tools beschränkt sich jedoch auf Eclipse als Entwicklungsumgebung und setzt das Google Web Toolkit sowie einige weitere Eclipse Plugins voraus, um mit der Entwicklung einer Fujaba Web Applikation zu starten.

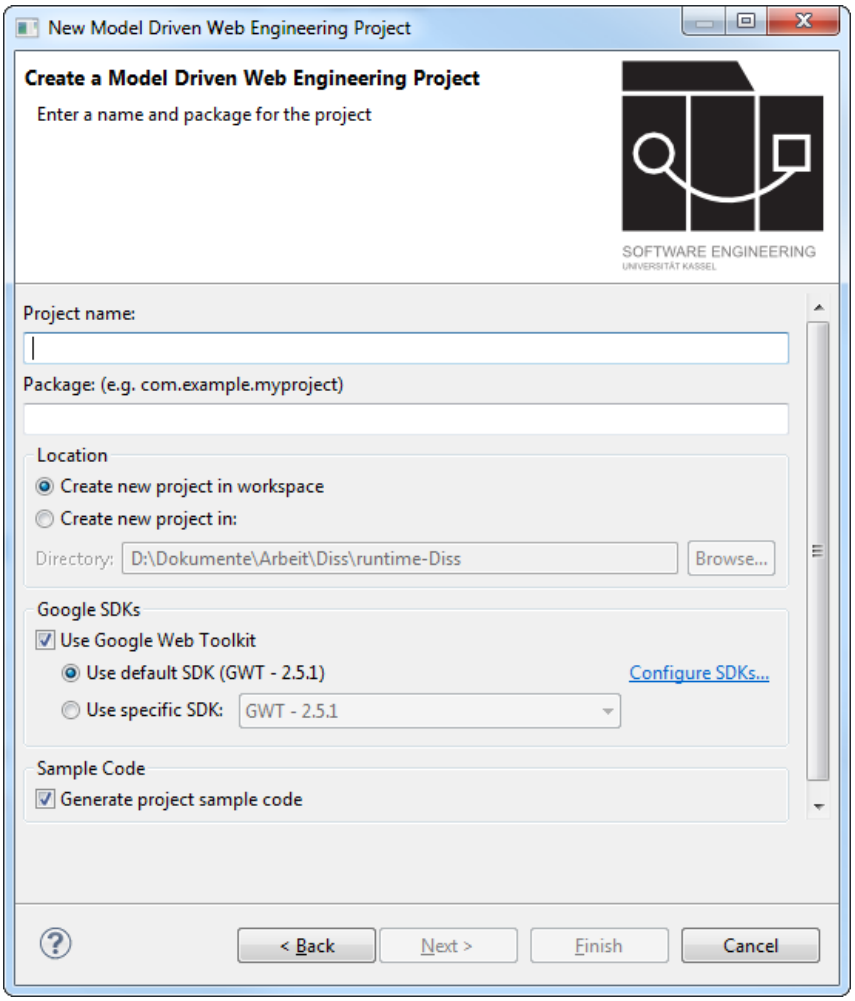

Abbildung 7.2: Fujaba Web Tools - New Project Wizard zum Erstellen von Fujaba Web Application Projekten in Eclipse

7.2.1 Erstellen von Fujaba Web Application Projekten

Die Erstellung einer komplexen Fujaba Application mit dem Web Fujaba Process kann prinzipiell ohne weitere Hilfsmittel durch das sukzessive Anlegen von textuellen Szenarien

oder Usecases über die Eclipse Umgebung gestartet werden. Um einen Einstieg in den komplexen Entwicklungsprozess zu erleichtern entschied ich mich jedoch, den Fujaba Web Tools einen Wizard mitzugeben. Dieser Wizard dient zum Anlegen eines „Model Driven Web Engineering Projectes“. Der in Abbildung 7.2 gezeigte Projekt-Wizard nimmt die notwendigen Konfigurationen automatisch vor und generiert ein Startprojekt, welches bereits alle benötigten Dateien enthält. Der Projektwizard ist in die Eclipse Entwicklungsumgebung integriert und lehnt sich an den Projektwizard des *Google Plugin for Eclipse (GPE)* [GPE] an. Die Eingabemaske aus Abbildung 7.2 erscheint nach dem Starten des Wizard und ermöglicht die Angabe eines Namens, sowie eines Hauptpakets für die gewünschte Fujaba Web Applikation. Neben diesen grundlegenden zwingend notwendigen Angaben ist es möglich anzugeben, welche Version des Google Web Toolkit für das zu erstellende Projekt verwendet werden soll. Mit einem Klick auf den *Finish* Button des Wizard wird der Aufbau der benötigten Projektstruktur gestartet. Teile dieser Struktur werden vom Google Plugin generiert.

Alle in Abbildung 7.3 unter `src` und `test` liegenden Dateien, sowie die Einbindung des GWT SDK in den Classpath des Projektes resultieren aus den Aktionen die durch das GPE angetriggert werden. Alle darüber hinaus in Abbildung 7.3 sichtbaren Dateien werden vom *Fujaba Web Application Wizard* generiert. Dies sind im Detail:

- `Projektname.ctr`: Hierbei handelt es sich um die Hauptdatei für die Entwicklung der Fujaba Web Applikation. In dieser Datei werden mit den in Fujaba zur Verfügung stehenden Editoren Klassen-, Objekt- und Storydiagramme erstellt und bearbeitet. Bereits nach der Generierung mit dem Wizard enthält diese Datei drei leere Klassendiagramme mit den Namen *Model*, *View* und *Controller*. Diese Diagramme werden im Laufe der Entwicklung in den unterschiedlichen Schritten des WFuP mit Klassen gefüllt.

- `GWTEventInterfaces.ctr` und `GWTUILibrary.ctr`: Bei diesen beiden Dateien handelt es sich um Bibliotheken, welche die vom Google Web Toolkit zur Verfügung stehenden Klassen in einer Form beinhalten, die von Fujaba verarbeitet werden kann. Damit diese Bibliotheken im Laufe der Entwicklung verwendet werden können, sind beide Dateien bereits als Abhängigkeiten in der `Projektname.ctr` eingetragen.

- `Projektname.usecasediagram` und `Projektname.usecasemodel`: In diesen Dateien werden die Informationen zu den zu erstellenden Usecases abgelegt. Die `.usecasediagram` Datei beinhaltet alle grafisch relevanten Anteile der Usecasediagramme, die Datei `.usecasemodel` speichert das dem Usecasediagramm zugrunde liegende Metamodell ab.

- `Projektname.workflowdiagram` sowie `Projektname.model` dienen in gleicher

Weise der Abspeicherung von grafischen, bzw. modellrelevanten Teilen des Workflows der Fujaba Web Applikation. Da es hierbei im Laufe der Entwicklung nötig sein kann, den Workflow weiter zu verzweigen, ist es erlaubt mehr als eine `.workflowdiagram` Datei im Projekt zu haben.

- `Projektname.szenario`: Diese Datei dient der Speicherung der textuellen Szenarien. Es ist erlaubt, meherer Szenariodateien im Projekt zu haben. Der Benutzer entscheidet nach seiner Präferenz, in welcher Weise die Szenarien über die Dateien verteilt werden.

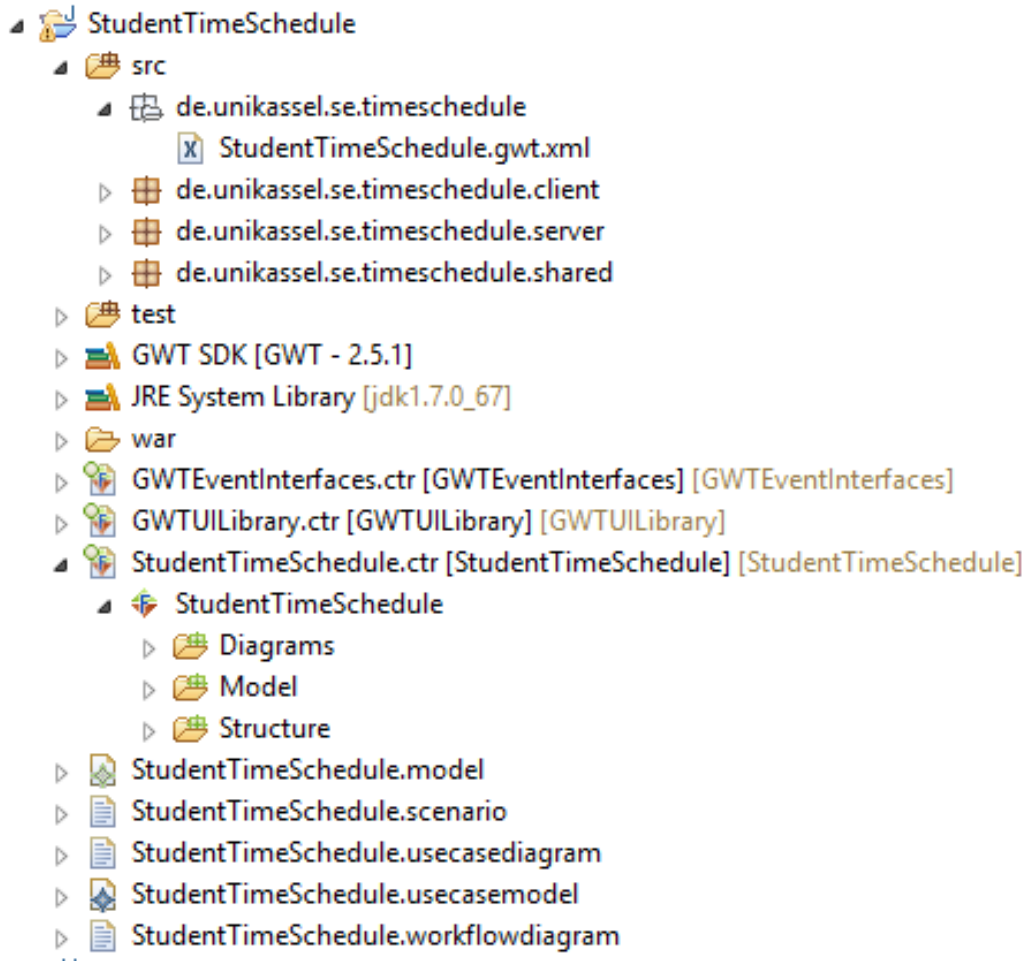

Abbildung 7.3: Erzeugte Projektstruktur nach Ausführung des New Project Wizard aus Abbildung 7.2

Die auf diese Weise automatisch erzeugten Dateien ermöglichen dem Entwickler, sofort mit der Definition von textuellen Szenarien und Usecases zu beginnen, ohne weitere Einstellungen am Projekt vornehmen zu müssen und ohne weitere Dateien anzulegen. Durch öffnen der `.szenario` respektive `.usecasediagram` Datei kann sofort mit dem ersten Schritt des WFuP begonnen werden.

7.2.2 Definieren von Usecases und textuellen Szenarien

Wie vom Web Fujaba Process definiert, werden nach Anlegen des benötigten Projektes im ersten Schritt zunächst die Anforderungen der Kunden festgehalten. Dies geschieht vorzugsweise gemeinsam mit den Kunden in einer weitgehend nicht technischen Form. Der Web Fujaba Process sieht für diese Schritte die Definition von textuellen Szenarien und Usecasediagrammen zur Festlegung der Anforderungen vor, wie aus Abbildung 7.1 entnommen werden kann.

Das Erstellen von Usecasediagrammen gemeinsam mit dem Endkunden ist keine neu entwickelte Idee. Bereits etablierte Software Entwicklungsprozesse, wie beispielsweise der Unified Process [JBR99] sehen diese Diagrammart als Entwicklungs- und Diskussionsgrundlage von Entwicklern und Endkunden vor.

Als wichtigen Teil des Gesamtprozesses unterstützen auch die Fujaba Web Tools diesen Vorgehensschritt durch geeignete Editoren. Im Rahmen der vorliegenden Arbeit wurde von mir mit dem *Graphiti Framework für Eclipse* [Gra] ein Editor erstellt, der es ermöglicht, Usecasediagramme zu zeichnen und abzuspeichern.

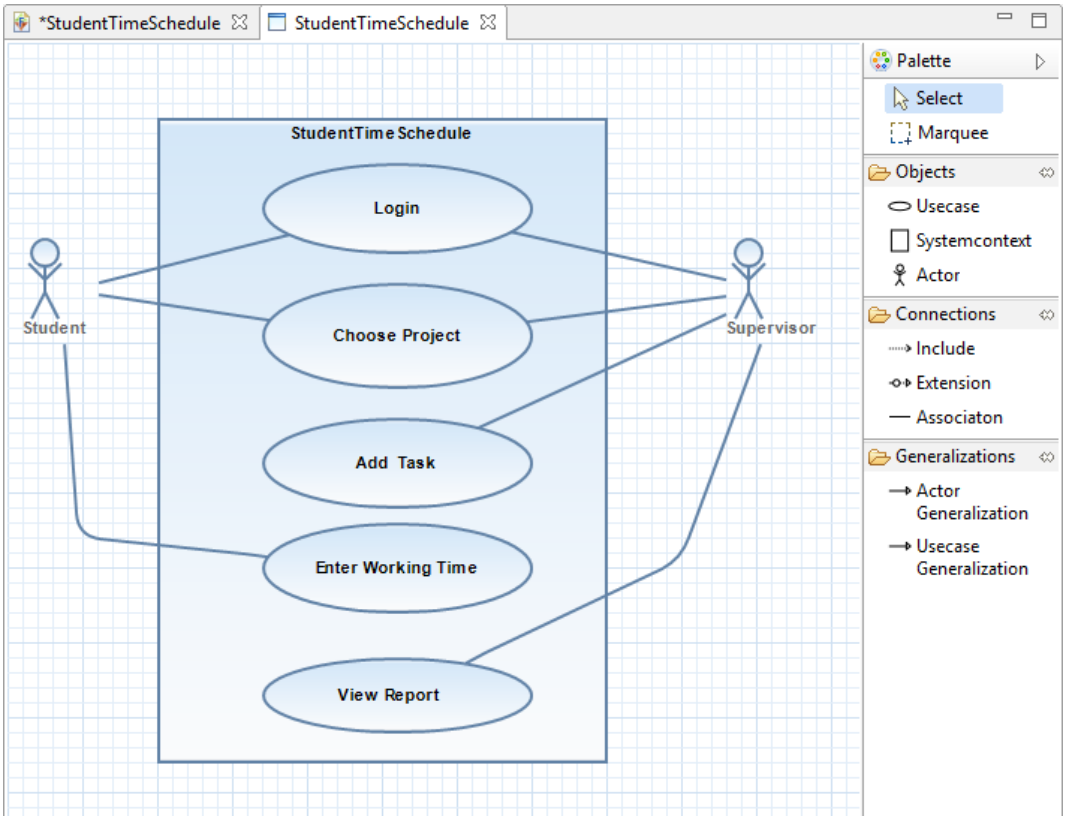

Abbildung 7.4: Usecasediagramm Editor mit Usecasediagramm einer studentischen Zeiterfassung

Abbildung 7.4 zeigt den von mir zur Verfügung gestellten Usecasediagramm Editor. Der Editor verfügt am rechten Rand über eine Palette, welche Werkzeuge zum Zeichnen von Elementen eines Usecasediagramms enthält. Die Palette spezifiziert vier verschiedene Bereiche, aus denen Werkzeuge ausgewählt werden können. Zuoberst gibt es Standardwerkzeuge zum Auswählen von Elementen. Diese Werkzeuge sind auf alle Diagrammelemente anwendbar. Die zweite Kategorie der Palette mit dem Namen *Objects* bietet Werkzeuge zum Erstellen von objektwertigen Diagrammelementen an. Objektwertige Elemente sind all jene, die anschließend mit Hilfe von Kanten verbunden werden können. Sie stellen gewissermaßen die Knoten im Diagrammmodell dar.

Die Fujaba Web Tools definieren für die mitgelieferten Usecasediagramme die Typen `Usecase`, `Systemcontext` und `Actor`. Die dritte Kategorie bietet Werkzeuge zum Erstellen von Verbindungen an. `Include` und `Extension` können zwischen `Usecases` angelegt werden, `Association` Verbindungen werden zwischen `Actor` Objekten und `Usecases` gezogen. Die letzte Kategorie bietet Generalisierungen an. Mit deren Hilfe können sowohl `Actor` Objekte, als auch `Usecases` in eine Art Typhierarchie gebracht werden. Die Usecasediagramme, wie sie von den Fujaba Web Tools unterstützt werden, richten sich in ihrer Syntax nach den Spezifikationen für Usecasediagramme aus [Obj10].

Die Arbeit an einem Fujaba Web Application Usecasediagramm sollte mit dem Anlegen eines `Systemcontext` Objektes begonnen werden. Dieses Objekt repräsentiert das zu beschreibende System. Mit einem Klick auf das entsprechende Tool in der Palette und anschließendem Klick auf die Zeichenfläche wird ein `Systemcontext` als Rechteck mit einer vordefinierten Größe angelegt. Am oberen Rand des Objektes erscheint ein Textfeld für die Eingabe des gewünschten Namens. Dieser Inplace Editor kann zu einem beliebigen späteren Zeitpunkt durch Doppelklick auf den Namen des `Systemcontextes` abermals geöffnet werden, um das Objekt umzubenennen. Diese Funktionalität ist auch für die anderen Objekte des Diagrammes vorhanden. Abbildung 7.5 zeigt das Inplace Editing am Beispiel eines Usecases.

Auf analoge Weise können `Actor` und `Usecase` Objekte auf der Zeichenfläche angelegt werden. Verbindungen zwischen Objekten auf der Zeichenfläche können entweder über die Palette oder über ein spezifisches Mouseover Menü direkt an jedem Objekt in ein Usecasediagramm eingefügt werden. Die `Include` und `Extension` Verbindungen sind editorseitig so eingeschränkt, dass sie nur zwischen zwei `Usecase` Objekten gezogen werden können, die `Association` Verbindungen sind nur zwischen `Actor` und `Usecase` Objekten erlaubt. Die Generalisierungen sind auf gleiche Weise anzulegen, wie die zuvor beschriebenen Verbindungstypen. Hierbei wird vom Editor sichergestellt, dass eine *Actor Generalization* lediglich zwei `Actor` Objekte und eine `Usecase Generalization` zwei `Usecase` Objekte verbinden kann. Abbildung 7.6 zeigt das Mouseover Menü für einen Usecase. Hier sind die unterschiedlichen Verbindungstypen, die Generalisierung, sowie ein

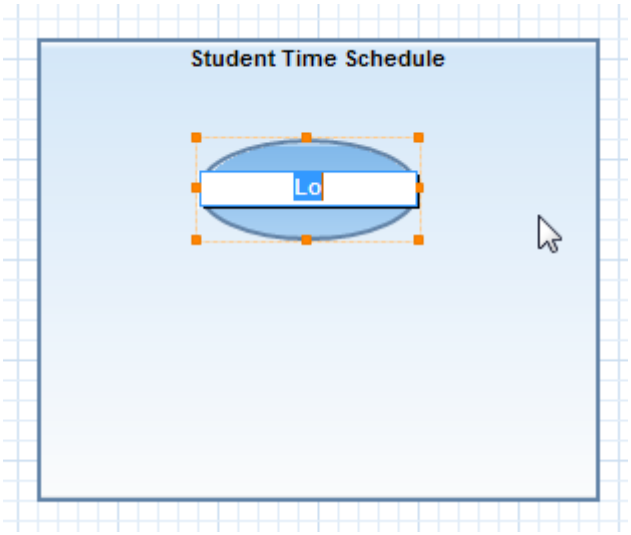

Abbildung 7.5: Inplace Editing der *Usecase* Namen

Tool zum Löschen des *Usecase* auswählbar. Das Graphiti Framework stellt komfortable Bedienoptionen automatisch bereit. So kann definiert werden, welche Tools bei einem Mouseover auf einem Element angezeigt werden sollen.

Alle relevanten Informationen über die gezeichneten Diagramme werden in der Datei `Projektname.usecasediagram` abgespeichert. Im Gegensatz zu den Workflowdiagrammen, vergleiche Abschnitt 7.2.4, wird beim Usecasediagrammeditor keine Trennung von grafischen Elementen und Modellelementen beim Abspeichern vorgenommen. Der Grund dafür liegt darin, dass im Rahmen des WFuP keine weiteren Berechnungen auf dem Metamodell der Usecasediagramme durchgeführt werden müssen. Es ist somit nicht notwendig, die Modellinformationen separat zugreifbar zu haben. Der in Abschnitt 7.2.1 beschriebene Projektwizard legt standardmäßig bereits ein Usecasediagramm mit dem Projektnamen an. Sollte mehr als ein Usecasediagramm zur Spezifikation aller notwendi-

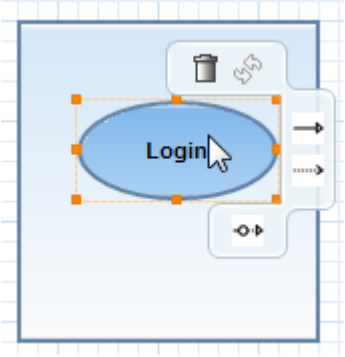

Abbildung 7.6: Tools zum Erstellen von Verbindungen, sowie Löschen des *Usecase* werden als Mouseover Menü angezeigt und sind von dort auswählbar.

ger Anforderungen notwendig sein, so ist es über das Eclipse Menü jederzeit problemlos möglich, weitere Diagramme anzulegen.

Während mit den gezeichneten Usecasediagrammen ein Überblick über das gesamte System geschaffen werden soll, den sowohl Endkunden, als auch Entwickler nachvollziehen können, geht es bei der Erstellung von textuellen Szenarien um die genauere Beschreibung der Abläufe des Systems. Mindestens alle im Usecasediagramm festgehaltenen Usecases sollten als Szenario genauer beschrieben werden. Um dies in einer für beide Seiten verständlichen Form zu tun, empfiehlt sich die Verwendung von konkreten Beispielen.

Textuelle Szenarien dienen wie die Usecasediagramme dem besseren Verständnis der Kunden vom zu planenden System. Viele Endkunden sind nicht in der Lage formalisierte Anforderungsdefinitionen zu lesen und zu verstehen, ebenso ist es nicht möglich, Quelltextbeispiele zur Verdeutlichung und Definition von Anforderungen heranzuziehen. Im Gegensatz dazu können textuelle Szenarien von allen beteiligten gelesen und verstanden werden. Mehr noch, es ist Kunden selbst möglich, textuelle Szenarien anzufertigen. Eine sorgfältige Definition dieser Szenarien hat hierbei hohe Priorität, da diese das allgemeine Verständnis des zu erstellenden Softwaresystems erhöhen können und somit Probleme im späteren Verlauf der Entwicklung abschwächen. In [NZJ13] beschreiben die Autoren die große Rolle guter Beispiele und Szenarien für eine gelungene Softwareentwicklung. Sie betonen dabei vor allem die Wichtigkeit **konkreter** Beispiele.

Die textuellen Szenarien, wie sie in [NZJ13] eingeführt werden, liegen auch denen der Fujaba Web Applications zugrunde. Sie bestehen aus der Beschreibung einer Ausgangssituation, eines oder mehrerer Tätigkeitsschritte, sowie einer Endsituation und verfügen über einen Szenario bezogenen Titel. Szenarien dieser Art wurden bereits seit einigen Jahren am Fachgebiet Software Engineering auch in der Vorlesung Programmiermethodik eingesetzt. Hierbei fiel auf, dass es vor allem unerfahrenen Benutzern, wie den Studenten der Vorlesung, oftmals schwer fällt, sich an die gegebene Syntax für textuelle Szenarien zu halten. Um diesem Problem entgegen zu wirken, wurde von mir im Rahmen dieser Arbeit, als Teil der Fujaba Web Tools, ein textueller Editor für Szenarien entwickelt, der über eine rudimentäre Syntaxüberprüfung und -vervollständigung verfügt.

Die Implementierung des Editors erfolgte mit Hilfe des Xtext Frameworks für Eclipse, vergleiche [xte]. Xtext bietet Werkzeuge zur Entwicklung von Programmiersprachen und Domänenspezifischen Sprachen an. Hierfür ist es notwendig, die zu unterstützende Sprache innerhalb einer Grammatik zu definieren. Ein kompletter Editor inklusive Syntaxhighlighting und Vervollständigung kann im Anschluss automatisch aus dieser Grammatik erzeugt werden. Im Falle der textuellen Szenarien handelt es sich um eine vergleichsweise einfache Grammatik, welche in Listing 7.1 dargestellt ist.

```
1  grammar de.unikassel.se.gwtw2m.Textscenario with org.eclipse.xtext.common.Terminals
2
3  generate textscenario "http://www.unikassel.de/se/gwtw2m/Textscenario"
4
5  Scenarios:
6    scenarios+=Scenario*;
7
8  Scenario:
9    headline=STRING
10   startsituation = Startsituation
11   steps += Step+
12   resultsituation = Resultsituation
13   ;
14
15 Resultsituation:
16   'Result situation: ' description=STRING
17 ;
18
19 Step:
20   'Step: ' description=STRING
21 ;
22
23 Startsituation:
24   'Start situation: ' description=STRING
25 ;
```

Listing 7.1: Xtext Grammatik zur Definition der Syntax textueller Szenarien

Eine Szenariodatei kann gemäß dieser Grammatik eines oder auch mehrere Szenarien enthalten. Jedes Szenario besteht aus einer Überschrift, der Beschreibung einer Startsituation, einem oder mehreren Szenarioschritten, sowie der Beschreibung einer Endsituation. Der mittels der vorliegenden Grammatik mit Hilfe von XText erzeugte Texteditor bietet dem Benutzer einfache Vervollständigung an und gibt Warnungen aus, wenn eines der Elemente des Szenarios vergessen wird. Durch diese Maßnahmen soll zukünftig die syntaktische Korrektheit einzelner Szenarien garantiert sein.

Die textuellen Szenarien werden in einer Datei mit der Endung `.szenario` abgespeichert. Eine initiale Datei mit dem Projektnamen wird bereits von dem unter Kapitel 7.2.1 beschriebenen Projektwizard erzeugt. Die Arbeit an den textuellen Szenarien kann sofort mit dieser Datei begonnen werden. Der generierte Editor ist als Standardeditor für diesen Dateityp gesetzt und wird von der Eclipse Umgebung automatisch geöffnet. Das Anlegen weiterer Szenariodateien kann über das Eclipse Menü erfolgen. Abbildung 7.7 zeigt den generierten Editor für textuelle Szenarien bei der Bearbeitung eines Szenarios für die Beispielanwendung *Student Time Schedule*.

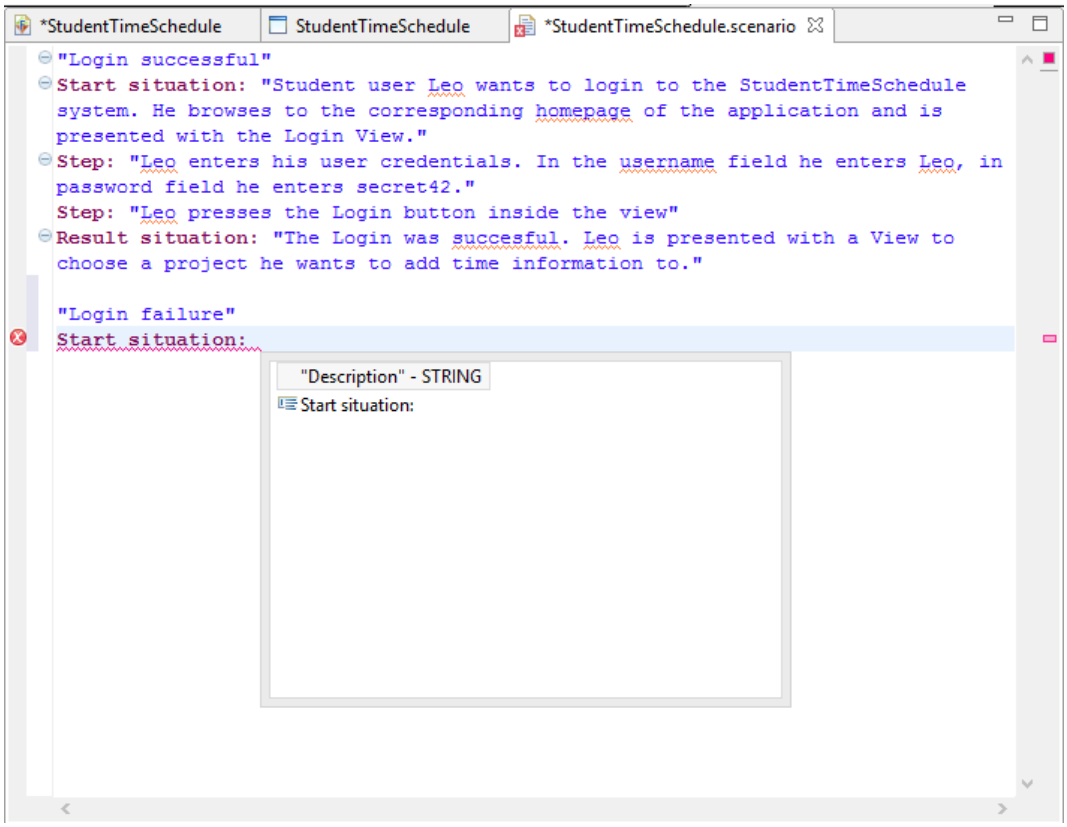

Abbildung 7.7: Generierter Xtext Editor für textuelle Szenarien. Geöffnet sind Szenarien der Beispielanwendung *Student Time Schedule*.

Die textuellen Szenarien sollten mit Bedacht gewählt und unter allen Projektbeteiligten gut abgesprochen werden. [NZJ13] gibt weitere Hinweise zur Verwendung von Beispielen und zur Kunst, gute Szenarien zu schreiben.

Die in diesem Kapitel beschriebenen Usecasediagramme und textuellen Szenarien dienen sowohl als Grundlage für die weiterführenden Schritte des Web Fujaba Process, sowie auch als Dokumentationsgrundlage des zu erstellenden Systems. Der WFuP teilt sich an dieser Stelle in zwei Stränge auf. In einem Strang werden, gemäß des im FuP beschriebenen Vorgehens, Datenmodell und Businesslogik basierend auf den erstellten Usecases und Szenarien entwickelt. Dieser Teil betrifft sowohl den Client als auch den Server der Webapplikation. Um geeignete Datenmodelle und Logik zu modellieren, sollten in diesem Bereich die im Kapitel 4 beschriebenen Codestile und Stereotypen, sowie die korrekte Anbindung an WebCoObRA beachtet werden. Dieser Teil ist genauer in 7.2.3 beschrieben. Der zweite Strang des Prozesses betrifft hauptsächlich die Clientseite der Applikation und beschäftigt sich mit der Definition grafischer Benutzerschnittstellen und des Applikationsablaufs mit gleichzeitigem Wechsel der grafischen Benutzerschnittstellen.

7.2.3 Erstellung des Server Parts und des Datenmodells basierend auf dem Fujaba Process

Der Fujaba Process [GZ05b] wurde ab dem Jahr 2001 von Professor Zündorf und Leif Geiger nach den Ideen von [KNNZ00] entworfen und in den Folgejahren von Leif Geiger, Ira Diethelm und Professor Zündorf weiterentwickelt [GZ06]. Im Rahmen seiner Promotion im Jahr 2011 wurde der Prozess von Leif Geiger in [Gei11] genau beschrieben und die Toolunterstützung seitens der Fujaba Toolsuite erläutert. Der in der vorliegenden Arbeit vorgestellte Web Fujaba Process erweitert den Fujaba Process hinsichtlich der Entwicklung komplexer Fujaba Web Applikationen (vgl. Kapitel 3). Diese Erweiterung bezieht sich im Besonderen auf die Entwicklung von User Interface Komponenten sowie die Integration des Userworkflows in den Entwicklungsprozess.

Die im Rahmen des Fujaba Process vorgestellten Tools und Methoden finden jedoch auch bei der Entwicklung von Web Applikationen Anwendung. So beruhen etwa die Entwicklung des Server Parts einer Applikation, ebenso wie die Definition des Datenmodells, nach wie vor auf dem Vorgehen, wie es in [Gei11] für traditionelle Applikationen beschrieben ist. Die im Fujaba Process vorgesehenen Usecases und textuellen Szenarien können jedoch durch die im WFuP vorgeschriebenen Schritte abgedeckt und anschließend in den Fujaba Process eingeleitet werden. Auf diese Weise wird durch die zuvor beschriebenen neuen Editoren auch für diese Schritte Toolunterstützung innerhalb der Fujaba Toolsuite geboten.

Um die besonderen Anforderungen hinsichtlich Datenmodell und Server bei der Entwicklung von Web Applikationen abzudecken, wurden an der Codegenerierung und der Integration des CoObRA Frameworks in der Fujaba Toolsuite Änderungen vorgenommen. Diese sind detailliert in Kapitel 4 beschrieben. Fujaba Web Applikationen beinhalten einen großen Teil des Datenmodells auf dem Webclient. Hierfür ist es notwendig, Browser kompatible Datenmodelle zu erstellen, also Modelle zu erzeugen, die im Anschluss vom GWT-Crosscompiler übersetzt werden können. Das prinzipielle Vorgehen bei der Erstellung eines Datenmodells hat sich im Vergleich zum Fujaba Process nicht grundlegend verändert und kann in [Gei11] nachgelesen werden.

Die Erzeugung Browser kompatibler Datenmodelle basiert hauptsächlich auf den in Kapitel 4 erläuterten Anpassungen an der Fujaba Toolsuite und am CoObRA Framework. Diese Anpassungen stellen sicher, dass alle erzeugten Quelltexte vom Crosscompiler zu JavaScript verarbeitet werden können und somit die Datenmodell auch im Webbrowser lauffähig sind. Um die gewünschte Kompatibilität zu erreichen ist es notwendig, an den entsprechenden Diagrammen in Fujaba den Codestyle „GWT “zu verwenden. Dieser triggert beim Start der Codegenerierung die Verwendung der korrekten Templates und sorgt so für die notwendige Kompatibilität. Sollen Datenmodellanteile auf dem Client

verwendet werden, ist im Zusammenhang mit GWT weiterhin darauf zu achten, die entsprechenden Klassen im korrekten Java Paket der Applikation abzulegen. Lediglich Klassen innerhalb der Pakete `.shared` sowie `client` werden vom GWT Crosscompiler behandelt (vgl. Kapitel 4.2.3). Bei der Implementierung von rein serverseitigen Datenmodellanteilen ist keine weitere Sonderbehandlung notwendig. Es ist mit der Fujaba Toolsuite möglich, serverseitig Services zu modellieren. Für diese ist wiederum einige Sonderbehandlung notwendig. GWT-RPC Services schreiben eine gewissen Namensgebung und eine einzuhaltende Klassenhierarchie vor. Diese muss vom Entwickler im Falle einer Implementierung mittels WFuP selbstständig erzeugt und eingehalten werden. Für jeden verwendeten Service sind verschiedene Klassen und Interfaces notwendig, die sich wie folgt aufteilen:

- Im Pakte `.client` ist ein Interface mit dem gewünschten Servicenamen anzulegen. Ein Beispiel wäre hier etwa `LoginService`. Dieses Interface leitet vom Interface `RemoteService` ab.
- Im Paket `client` ist außerdem eine sogenanntes Async-Interface anzulegen. Zum `LoginService` gehört das Interface `LoginServiceAsync`. Beide Interfaces werden über die Namensgebung automatisch miteinander in Verbindung gebracht.
- Im Paket `.server` wird eine Klasse mit dem Namen ServicenameImpl angelegt. Im Beispiel wäre dies `LoginServiceImpl`. Diese Klasse leitet von der Klasse `RemoteServiceServlet` ab und implementiert gleichzeitig das im `.client` Paket angelegte Serviceinterface `LoginService`.

Eine genauere Beschreibung des GWT und des dort verwendeten RPC-Mechanismus ist Kapitel 4.2.3 zu entnehmen. Mit Fujaba auf diese Weise modellierte und generierte Services können im Anschluss sowohl in handgeschriebenem Quelltext, wie auch bei der Definition von Businesslogik mittels Story-Driven-Modelling verwendet werden. Eine speziell zur Verwendung mit GWT-RPC eingesetzte Erweiterung der Fujaba-Transitionen wurde bereits in Kapitel 4.3.4 dargestellt.

7.2.4 Modellierung des der Applikation zu Grunde liegenden Workflows

Kapitel 3 definiert Fujaba Web Applikationen. All diesen Applikationen ist gemeinsam, dass sie einen bestimmten Workflow abbilden. Der Web Fujaba Process sieht vor, die Applikationen auch anhand dieses Workflows zu designen.

Nachdem die Anforderungen gemeinsam mit dem Endkunden ausgelotet und anhand von Usecasediagrammen und textuellen Szenarien festgehalten worden sind, kann nun der Workflow, der mit Hilfe der Anwendung abgebildet werden soll, definiert werden. Hier

können die bereits erstellten Szenarien und Usecasediagramme als Diskussionsgrundlage dienen.

Der Workflow wird durch Workflowdiagramme spezifiziert. Diese Diagramme wurden von mir neu eingeführt und werden in Kapitel 5 genauer beschrieben. Der Workflow stellt an dieser Stelle sowohl die mittels der Applikation zu verrichtende Tätigkeit in ihren einzelnen Schritten, als auch die unterschiedlichen Benutzerschnittstellen der Applikation dar. In [EGHZ12] wird bereits von mir definiert, dass für jeden Schritt des Workflows eine grafische Benutzerschnittstelle implementiert wird. Der Wechsel zwischen den einzelnen Schritten des Workflows hat immer auch eine Änderung der grafischen Benutzerschnittstelle zur Folge. Hierbei ist zu beachten, dass es möglich ist, einen einzelnen Schritt des Workflows genauer zu definieren, indem ein sogenannter **`Subworkflow`** an dieser Stelle eingehängt wird. Dieser wird anschließend durch ein eigenes Workflowdiagramm mit Start und Endpunkt definiert. Im Falle eines **`Subworkflows`** ist es sinnvoll, mindestens einen Endpunkt zu verlangen, da an dieser Stelle zurück zum aufrufenden Workflowschritt gesprungen und die Arbeit an der dort definierten Stelle fortgesetzt werden muss.

Die ein- und ausgehenden Kanten eines Workflowschrittes können textuell beschriftet werden. Die Beschriftung stellt einen Trigger dar, der das Umschalten in den nächsten Schritt auslöst. Hierbei können einfache Bezeichner ebenso verwendet werden, wie komplexe boolesche Ausdrücke. Die komplette Ablauflogik der Applikation wird aus den Workflowdiagrammen und den vorhandenen Triggern automatisch generiert und in die fertige Anwendung eingebaut.

Die Fujaba Web Tools stellen zur Unterstützung des beschriebenen Schrittes der Workflowdefinition einen Editor bereit. Der Workflowdiagrammeditor wurde mit dem Graphiti Framework [1] als Plugin für die Eclipse IDE entwickelt.

Der in Kapitel 7.2.1 vorgestellte Projektwizard erzeugt bereits eine initiale Workflowdiagramm-Datei. Diese stellt den zentralen Workflow der Applikation dar und kann im Folgenden mit dem mitgelieferten Editor bearbeitet werden. Abbildung 7.8 zeigt den bereitgestellten Editor nach dem Öffnen der zentralen Workflowdatei.

Auf der linken Seite des Editorfensters befindet sich die Zeichenfläche, auf welcher die einzelnen Elemente des Workflows platziert und bearbeitet werden können. Zum Erzeugen der einzelnen Elemente bietet die am rechten Rand des Editors angezeigte Palette die notwendigen Werkzeuge an. Ein Element wird in der Palette ausgewählt und anschließend an beliebiger Stelle der Zeichenfläche erstellt. Beim Erstellen von Elementen werden im Hintergrund die zugehörigen Modellinformationen des Workflows angelegt. Diese Modellinformationen werden in der Datei `Projektname.model` abgespeichert.

[1] http://www.eclipse.org/graphiti/

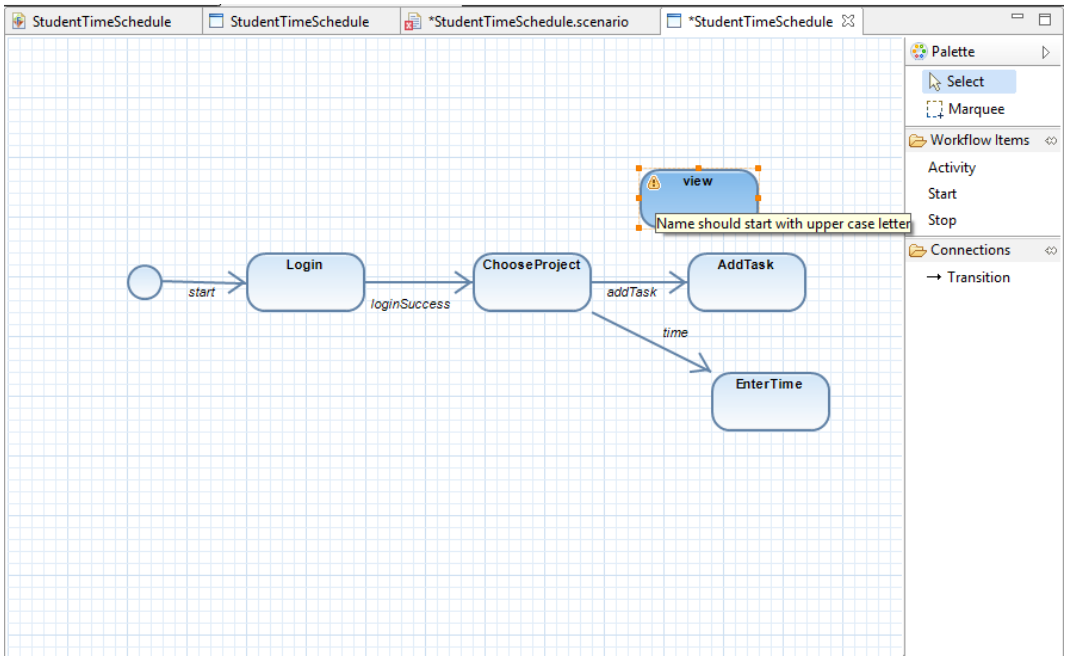

Abbildung 7.8: Workflowdiagramm Editor beim Erstellen des Workflows zur Beispielanwendung StudentTimeSchedule. Die angelegte Activity `view` zeigt über einen Marker einen nicht korrekt spezifizierten Namen an.

Während die Modellinformationen des gesamten Workflows inklusive aller Kinder innerhalb einer Modelldatei abgespeichert wird, werden die Diagramminformationen für jeden Workflow separat in den Dateien mit der Endung `.workflowdiagram` abgelegt.

Workflows können aus den Elementen **`Start`**, **`Stop`**, **`Activity`** und **`Transition`** zusammengesetzt werden. **`Activity`** steht hierbei für einen beliebigen Schritt innerhalb des Workflows. Jede **`Activity`** besitzt einen Namen, welcher die Aktivität dieses Schrittes beschreibt und gleichzeitig als Bezeichner für die zugehörige grafische Benutzerschnittstelle dient. In eine **`Activity`** können weitere Workflowelemente eingeschachtelt werden. Auf diese Weise entsteht ein **`Subworkflow`**, welcher in einer separaten Datei mit der Endung `.workflowdiagram` im Arbeitsverzeichnis der Applikation abgespeichert wird. Der Name dieser neuen Datei hängt von dem Namen des Workflowschrittes ab, der als Vater des neuen Workflows dient. Das Benamen einer **`Activity`** innerhalb eines Workflows geschieht, analog zu Kapitel 7.2.2, per Inplace-Editing nach dem Erzeugen des Elements. Das erneute Öffnen des Inplace Editors zum Umbenennen einer *Activity* geschieht, wie beim Usecasediagrammeditor, über einen Doppelklick auf den zu ändernden Namen. Wie auch beim Usecasediagrammeditor wurde für den Workfloweditor Mousover Tooling implementiert. Die Auswahlmöglichkeiten sind in diesem Fall allerdings geringer, da bei einem Workflow weniger unterschiedliche Verbindungstypen zur Verfügung stehen. Für

`Activity`, `Start` und `Stop` Objekte steht im Mouseover jeweils die Möglichkeit zum Löschen des Objektes, sowie zum Erzeugen einer *Transition* zur Verfügung.

Sollte ein zu vollziehender Arbeitsschritt eines Workflows zu komplex sein, kann es notwendig werden, diesen wiederum in einzelne Schritte einzuteilen. Aus diesem Grund ist es möglich, einer `Activity` Kinder hinzuzufügen. Diese Kinder bilden einen sogenannten Subworkflow. Aus Gründen der Ausführungslogik sollte darauf geachtet werden, dass der Subworkflow mindestens einen Endpunkt besitzt, damit zum einbettenden Workflow zurück gesprungen und die Ausführung dort fortgesetzt werden kann. Das Anlegen eines Subworkflows geschieht im Fujaba Web Tooling automatisch. Wird ein Element als Kind in einer `Activity` angelegt, wird automatisch ein neuer Workflow im Modell angelegt und eine neue Diagrammdatei mit dem Namen `.workflowdiagram` angelegt. Im Vaterworkflow wird das Vorhandensein eines Subworkflows dadurch angezeigt, dass ein Quadrat mit einem Pluszeichen in der oberen rechten Ecke in der Vateractivity dargestellt wird. Wie in Abbildung 7.9 dargestellt ist es möglich den neuen Workflow über ein Kontextmenü zu öffnen.

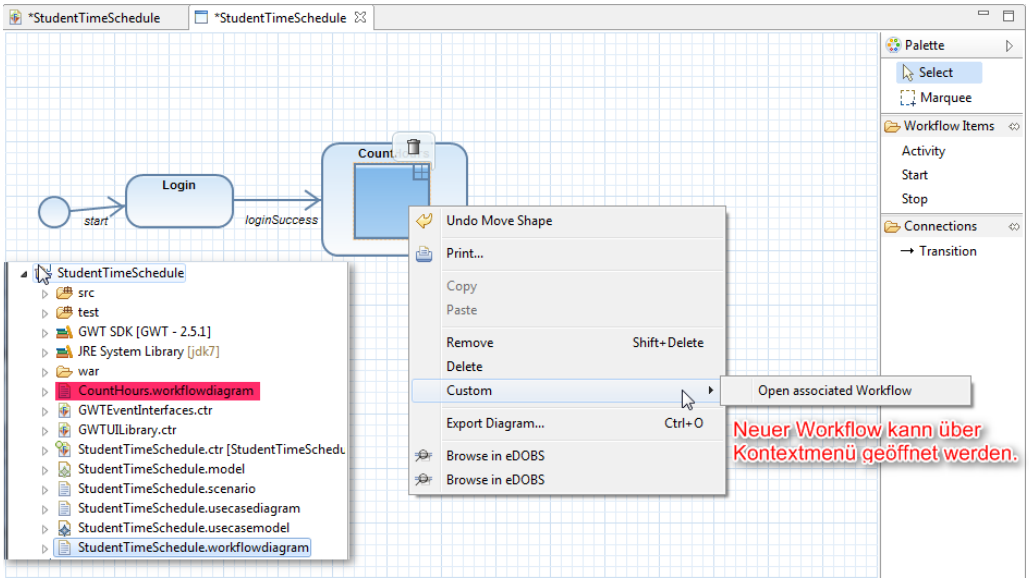

Abbildung 7.9: Subworkflow mit Kontextmenü und neu angelegter Workflowdatei im Dateibaum des Projektes.

Per Konvention wird ein `Subworkflow` immer automatisch nach der `Activity` benannt, in welche er eingebettet ist. Dies bringt die Herausforderung mit sich, beim Umbenennen einer `Activity` sicherzustellen, dass ein vorhandener `Subworkflow` sowie die zugehörige Diagrammdatei ebenfalls umbenannt werden. Hierfür sorgt das Fujaba Web Tooling automatisch, um so etwaige Fehler beim Benutzer zu vermeiden.

Das Anlegen von `Start` und `Stop` Elementen erfolgt analog zum Anlegen einer `Activity` über die Auswahl in der Palette. Hierbei können `Start` und `Stop` Elemente nicht benannt werden, weshalb das Anzeigen eines Inplace Editors an dieser Stelle nicht erfolgt.

Die Erstellung von Transitionen kann auf unterschiedlichen Wegen erfolgen. Zum einen existiert die Möglichkeit, über die Palette das Tool für Transitionen auszuwählen und anschließend per Klick-Drag-Klick eine Transition zu erzeugen. Wie bereits in 7.2.2 beschrieben ist es möglich, das Anlegen von Verbindungen auf die Stellen zu Beschränken, an welchen sie syntaktisch erlaubt sind. Im Falle der vorliegenden Workflowdiagramme wurde hier jedoch keine Einschränkung getroffen, um auch das Erstellen von zyklischen Workflows zu erlauben. Eine zweite Möglichkeit zur Erstellung von Transitionen bietet das bereits beschriebene Mouseover Tooling an. Hier kann der Button für die Transition per Klick ausgewählt werden und anschließend per Drag-and-Klick die Verbindung zu einem weiteren Element gezogen werden. Der Text der Transition kann im Anschluss ans Erzeugen mittels eines Doppelklicks auf den bereits angezeigten Text geändert werden. Wie bei dem Namen der `Activity` öffnet sich auch hier ein Eingabefeld, in das der neue Text eingegeben werden kann. Anders als bei einer `Activity` stellt der eingegebene Text jedoch nicht zwingend den Namen einer Transition dar. Vielmehr handelt es sich um eine Bedingung, ein Event oder einen anderen Trigger, welcher das Weiterschalten in den nächsten Zustand des Workflows anstößt. Dies ist notwendig, da der Workflow den Ausführungsfluss der Anwendung widerspiegelt und somit an Wechseln zwischen unterschiedlichen `Activities` auch ein Wechsel der grafischen Benutzeroberfläche vollzogen werden muss.

Die Überwachung und Ausführung dieses Zustandsautomaten übernimmt eine mit ausgelieferte Laufzeitumgebung für Fujaba Web Applikationen, welche die automatisch aus der Workflow Beschreibung generierten Ablaufinformationen in konkrete Zustandswechsel übersetzt. Der Benutzer muss jedoch sicherstellen, dass an den notwendigen Stellen die entsprechenden Informationen an diese Laufzeitumgebung übergeben werden. So ist es beispielsweise denkbar, dass der Controller eines Buttons nach erfolgreicher Abarbeitung ein Event feuert, welches die Laufzeitumgebung dazu veranlasst, die Anwendung in einen neuen Zustand zu versetzen. Eine genauere Definition der Laufzeitumgebung und ihrer Funktionsweise ist in Kapitel 6 nachzulesen.

7.2.5 Views passend zum Workflow definieren

Das Durchführen der in Kapitel 7.2.4 beschriebenen Schritte zur Definition des Workflows der Fujaba Web Applikation führt zu einem Grundgerüst der ausführbaren Anwendung. Die einzelnen logischen Schritte und somit auch die Ablauflogik der Applikation wurden mittels der Workflowdiagramme spezifiziert. Um eine lauffähige Webanwendung

zu erzeugen, wird in diesem Schritt eine grafische Benutzerschnittstelle benötigt, die mit den logischen Schritten des Workflows verbunden ist. Um die Arbeit beim Entwickler der Applikation möglichst gering zu halten und die Fehlerquellen an dieser wichtigen Verbindungsstelle zu minimieren, wurde das Erstellen von Views passend zum definierten Workflow von mir automatisiert. Per Konvention wird, wie bereits in [EGHZ12] veröffentlicht, jedem Schritt im Workflow genau eine View zugeordnet. Die Zuordnung geschieht in diesem Fall über den Namen des Workflowschrittes. Die Fujaba Web Tools generieren nach dem Anlegen eines Workflowschrittes automatisch eine View Klasse mit demselben Namen, der für den Workflowschritt ausgewählt wurde. Um etwaige Fehler bei der

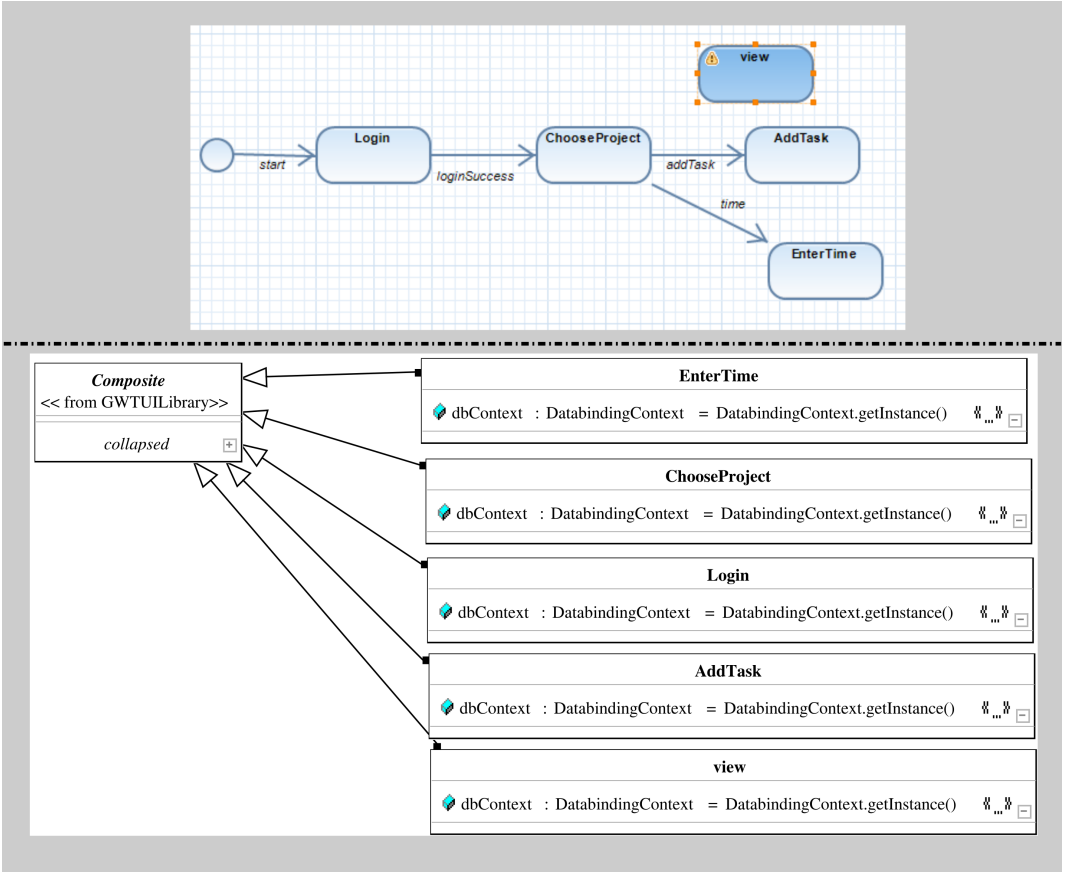

Abbildung 7.10: Workflow und assoziierte Klassen des Fujaba Klassendiagramms `View`

Codegenerierung zu vermeiden, wird über eine Validierung im Workflowdiagrammeditor sichergestellt, dass es sich um einen validen Java Klassennamen handelt. Fehler bei der Namensgebung werden dem Benutzer als Markierung am jeweiligen Workflowschritt angezeigt. Die entsprechende View Klasse wird im Fujaba View Klassendiagramm des geöffneten Fujaba Web Application Projektes angelegt. Die Übersetzung in Java Quelltext wird anschließend vom Codegenerator der Fujaba Toolsuite übernommen. Abbildung 7.10 zeigt exemplarisch einen Workflow, sowie das zugehörige View-Klassendiagramm.

Um die Weiterverarbeitung der eingefügten Klassen im Web Fujaba Process zu ermöglichen ist es notwendig, dass die Views von der Klasse `com.google.gwt.user.client.ui.Composite` erben. Dies garantiert die benötigten Verhaltensweisen und ermöglicht ein Design der Benutzerschnittstellen mit dem *GWT Designer* [GWTb], wie in Kapitel 7.2.6 beschrieben.

Das Automatisierte Anlegen und Verwalten der View Klassen über die Fujaba Web Tools ermöglicht dem Benutzer eine einfache Handhabung seiner Entwicklungsaufgaben. Alle Änderungen am Workflow werden auf diese Weise automatisch in den View Klassen reflektiert. Umbenennungen einer *Activity* im Workflowdiagramm führen automatisch zu der entsprechenden Änderung des Namens der assoziierten View Klasse. Verschachtelungen von View Klassen werden hingegen nicht automatisch von den Fujaba Web Tools aufgelöst. Diese Verschachtelungen kommen im Fall einer Vater-Kind-Beziehung verschiedener Workflows zustande. In diesem Fall wird lediglich für jede der *Activities* eine View Klasse angelegt. Der Subworkflow ist jedoch für die Ablaufsteuerung im Rahmen des Laufzeitsystems wichtig. Die Elemente des Subworkflows müssen jedoch manuell in die UI der Activity eingebunden werden, da der Workflow nicht dazu in der Lage ist, die genaue Verschachtelung der Elemente zu beschreiben. Zu diesem Zweck kann beispielsweise im GWT-Designer die erstellte UI-Komponente dynamisch in die Palette eingefügt und anschließend in die UI der Activity eingebunden werden. Die Erstellung der grafischen Benutzerschnittstelle inklusive Controllern und Databinding an Modellelemente, wie sie in den folgenden Kapiteln beschrieben wird, bildet den Abschluss der Entwicklung einer Fujaba Web Applikation.

7.2.6 Design der einzelnen User Interface Komponenten

Das Design einer grafischen Benutzerschnittstelle ist eine anspruchsvolle Aufgabe. Viele Elemente aus den Bereichen Design und Ergonomie spielen in diesen Bereich mit hinein. Von Seiten der Softwareentwickler gehört das Erstellen grafischer Benutzerschnittstellen meist zu den weniger geschätzten Aufgaben. Für die Erstellung einer komplexen Fujaba Web Anwendung ist das Definieren und Programmieren einer grafischen Benutzerschnittstelle jedoch unerlässlich. Aus diesem Grund wird, im Gegensatz zum bestehenden Fujaba Process [Gei11], in der Erweiterung zum Web Fujaba Process explizit auf diesen Aspekt der Anwendung eingegangen und Vorgehensweisen zur Erstellung der benötigten grafischen Benutzeroberfläche vorgeschlagen.

Für die reine Erstellung einer GUI gibt es unterschiedliche Möglichkeiten, die im Rahmen der Überlegungen zum Web Fujaba Process in Betracht gezogen werden mussten. Zum Einen bietet sich selbstverständlich die Möglichkeit, die Benutzerschnittstelle anhand von Designvorgaben per Hand zu programmieren. Zahlreiche Oberflächenbibliotheken, auch

für den Web Bereich, stehen für diese Tätigkeit zu Verfügung und bieten dem Programmierer die notwendigen Klassen für Fenster, Buttons, Textfelder etc. Beim händischen Programmieren einer Benutzeroberfläche handelt es sich im Gegensatz zum Entwurf von Logikanteilen einer Anwendung jedoch um eine stupide Tätigkeit, die zudem fehleranfällig ist. Neben diesem Nachteil widerspricht ein händisches Programmieren der Benutzeroberfläche auch den Zielsetzungen des Web Fujaba Process, die komplette Anwendung zu modellieren und den benötigten Quelltext generieren zu lassen. Hier böte sich selbstverständlich die Möglichkeit, die benötigten Oberflächenklassen mittels der Fujaba Toolsuite mit Storydiagrammen zu modellieren und anschließend den entsprechenden Quelltext generieren zu lassen. Auch diese Möglichkeit bietet jedoch eine hohe Fehleranfälligkeit und resultiert zudem in unübersichtlichen Storydiagrammen. Die Wartbarkeit der Oberfläche wird somit durch die Modellierung keinesfalls erhöht, sondern im Gegenteil, im Vergleich zur Programmierung mit Hand, sogar noch reduziert.

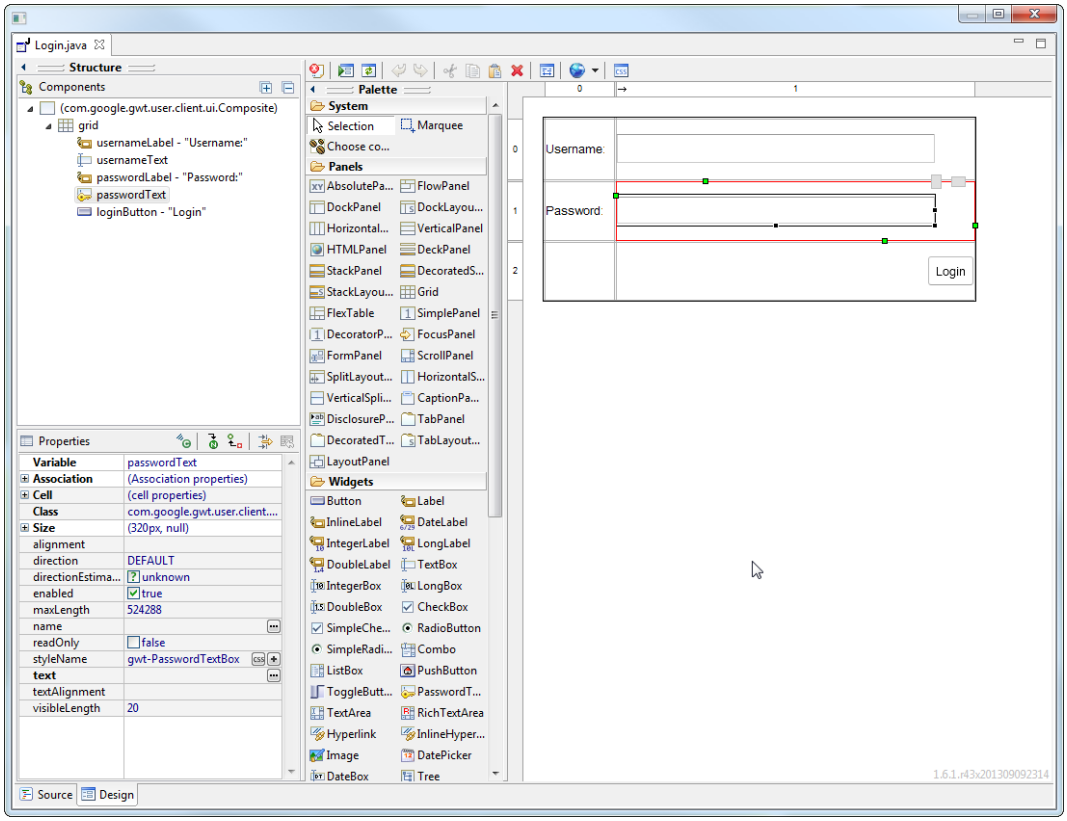

Abbildung 7.11: GWT Designer der Firma Google

Da die Problematik der Oberflächenerstellung im Bereich der Softwareentwicklung kein neues Problem ist, existieren bereits heute für viele Oberflächenframeworks grafische Editoren, die es ermöglichen, die gewünschte Oberfläche nach dem What-You-See-Is-What-You-Get Prinzip (WYSIWYG) zu erstellen. Dies erschien auch mir die Vorgehensweise

der Wahl zu sein, um eine quelltextferne Entwicklung nach dem Web Fujaba Process zu unterstützen. Glücklicherweise existieren auch für das Google Web Toolkit und seine Oberflächenbibliothek Editoren, die ein solches Vorgehen erlauben. In der Eclipse Umgebung kann hier der GWT-Designer [GWTb] verwendet werden. Beim GWT Designer handelt es sich seit der Übernahme der Software von der Entwicklerfirma Instantiations durch Google im Jahr 2010 um ein frei Verfügbares Werkzeug, welches unter der Eclipse Public License veröffentlicht ist. Der GWT Designer, wie in Abbildung 7.11 gezeigt, bietet dem Benutzter die Möglichkeit, beliebige Benutzerschnittstellen mit GWT Oberflächenelementen zu gestalten und abzuspeichern. Die Oberflächenelemente sind in einer *Palette* angeordnet und können von dort per Drag and Drop auf die Zeichenfläche gezogen und angeordnet werden. Die komplette Benutzerschnittstelle ist zum einen in der Zeichenfläche sichtbar, zum anderen kann in einer *Components* genannten Baumansicht die Struktur der Oberfläche betrachtet und Elemente neu angeordnet werden. Einstellungen an einzelnen Elementen der Oberfläche können über die *Properties* vorgenommen werden. Eine Besonderheit des GWT Designers besteht darin, dass der Quelltext der Oberfläche, sowie die WYSIWYG Ansicht parallel verfügbar sind. Ein Wechsel in den Quelltext bleibt dadurch prinzipiell zu jedem Zeitpunkt in der Entwicklung möglich. Der Parsingmechanismus des GWT Designers erlaubt neben dem parallelen Entwickeln in Quelltext und Grafik darüber hinaus das Einbinden eigener Komponenten in die Palette des GWT Designers. Auf diese Weise kann eine komplexe Oberfläche modular aus mehreren Teilen zusammengesetzt, sowie definierte Oberflächen an anderer Stelle wiederverwendet werden. Der GWT Designer nimmt die Zusammensetzung der Oberfläche aus den gewählten Komponenten auf Quelltext Seite im Konstruktor der Oberflächenklasse vor. Listing 7.2 zeigt den Quelltext zur Oberfläche aus Abbildung 7.11.

```
1  public Login() {
2    Grid grid = new Grid(3, 2);
3    initWidget(grid);
4    grid.setSize("450px", "192px");
5
6    Label usernameLabel = new Label("Username:");
7    grid.setWidget(0, 0, usernameLabel);
8
9    TextBox usernameText = new TextBox();
10   grid.setWidget(0, 1, usernameText);
11   usernameText.setWidth("320px");
12
13   Label passwordLabel = new Label("Password:");
14   grid.setWidget(1, 0, passwordLabel);
15
16   PasswordTextBox passwordText = new PasswordTextBox();
17   grid.setWidget(1, 1, passwordText);
18   passwordText.setWidth("320px");
19
```

```
20      Button loginButton = new Button("Login");
21      grid.setWidget(2, 1, loginButton);
22      grid.getCellFormatter().setHorizontalAlignment(2, 1, HasHorizontalAlignment.
            ALIGN_RIGHT);
23    }
```

Listing 7.2: Zusammensetzung der Oberflächenklasse im Konstruktor mit lokalen Variablen.

Wie aus Listing 7.2 ersichtlich wird, werden alle Elemente der Oberfläche im Konstruktor als lokale Variablen angelegt. Dies würde ein späteres Zugreifen auf die Elemente der Oberfläche erschweren. Die Funktion *Expose Widget* aus dem Kontextmenü des GWT Designers erlaubt es jedoch, einzelne Elemente öffentlich zugänglich zu machen, indem sie in Felder der Oberflächenklasse umgewandelt, sowie Zugriffsmethoden generiert werden. Listing 7.3 zeigt den Quelltext für den `loginButton` nach Anwendung von *Expose Widget*.

```
1   private Button loginButton;
2    public Login() {
3          ...
4      loginButton = new Button("Login");
5      grid.setWidget(2, 1, loginButton);
6      grid.getCellFormatter().setHorizontalAlignment(2, 1, HasHorizontalAlignment.
           ALIGN_RIGHT);
7    }
8    public Button getLoginButton() {
9       return loginButton;
10   }
```

Listing 7.3: Quelltext des Login Buttons nach Anwendung der Funktion *Expose Widget*.

Der Web Fujaba Process sieht vor, die einzelnen Teile der Benutzerschnittstelle, wie oben beschrieben, mit Hilfe des GWT Designers grafisch zu definieren. Hierfür können die von Fujaba generierten View Klassen mit dem GWT-Designer geöffnet und anschließend gestaltet werden. Der Anspruch des quelltextlosen Entwickelns einer Fujaba Web Anwendung bleibt somit gewahrt und alle Vorteile generierten Quelltextes können übernommen werden. Da es sich beim GWT Designer darüber hinaus um ein quelloffenes Werkzeug handelt, konnten weitere Anpassungen für die reibungslose Einbindung des Designers in den Web Fujaba Process vorgenommen werden. Details hierzu sind in Kapitel 7.2.8 nachzulesen.

7.2.7 Strukturelle Informationen aus den User Interfaces zurück ins Klassendiagramm überführen

In Kapitel 7.2.6 wurde beschrieben, wie die User Interfaces mit Hilfe des GWT Designers grafisch erstellt und der entsprechende Quelltext automatisch generiert werden kann. Die Informationen über die Benutzerschnittstellen liegen zu diesem Zeitpunkt sowohl als Grafik im GWT Designer, als auch als Java Quelltext vor. Für den Entwickler stellt diese Tatsache zunächst kein Problem dar, die Weiterverarbeitung der Informationen im fortschreitenden Entwicklungsprozess stellt sich jedoch als Hindernis heraus. Dem Web Fujaba Process folgend, werden die übrigen Teile der Applikation mit Hilfe der Storydiagramme und Klassendiagramme der Fujaba Toolsuite entwickelt. Dies gilt insbesondere auch für die Kontrollelemente der grafischen Benutzerschnittstelle. Die grafische Repräsentation der Elemente wurde bereits definiert, eine Funktion können diese jedoch erst erfüllen, wenn die entsprechende Anwendungslogik über `Controller` und `Event Handler` an die grafischen Elemente angefügt wurden. Die Implementierung dieser Funktionen soll jedoch modellbasiert erfolgen. Fujaba bietet über die Storydiagramme und Action Charts ausreichende Möglichkeiten, beliebige Funktionalität zu implementieren. Eine Voraussetzung hierfür stellt jedoch die Verfügbarkeit der benötigten Informationen innerhalb der Fujaba Toolsuite dar. Zu diesem Zeitpunkt im Entwicklungsprozess wurde von Fujaba für jede **`Activity`** jedes Workflows aus den definierten Workflowdiagrammen eine User Interface Klasse erstellt und der Java Quelltext für die erstellten Klassen generiert, siehe Kapitel 7.2.4 und 7.2.5. Die weitere Definition der User Interfaces lief außerhalb der Fujaba Toolsuite ab und hat somit keine Repräsentation in den Fujaba Diagrammen. Für die weitere Verarbeitung der User Interface Klassen mit Fujaba, beispielsweise beim Erstellen von Controller-Klassen, ist es jedoch notwendig, dass die interne Struktur der User Interfaces mit allen notwendigen Elementen innerhalb der Fujaba Toolsuite bekannt ist. Es ist somit notwendig, die strukturellen Informationen aus dem Quelltext der User Interface Klassen zurück in die Fujaba Klassendiagramme zu überführen.

Für die Rückführung der Informationen aus dem Quelltext in die Fujaba Klassendiagramme wurde mit Hilfe von Alexander Jahl eine neue Funktion in die Eclipse Umgebung eingefügt. Über die in Abbildung 7.12 markierte Schaltfläche wird auf allen selektierten Java Klassen ein Parsing Vorgang gestartet und die gewonnenen Informationen anschließend verarbeitet und ins Fujaba Klassendiagramm eingepflegt. Der für die Rückgewinnung der Informationen verwendete Parser wurde eigens für diesen Zweck von Hand geschrieben und auf die Verwendung im gegebenen Anwendungsumfeld optimiert. Für die einfachere Handhabung der eingelesenen Informationen wurde auf die SDMLib Bibliothek [SDM] zurückgegriffen. Bei SDMLib handelt es sich um eine Graph- und Storydriven-Modeling Bibliothek, welche am Fachgebiet für Software Engineering der Universität Kassel ent-

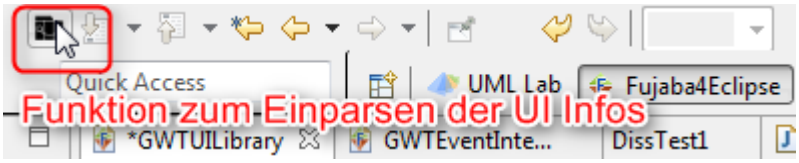

Abbildung 7.12: Bedienelement zum Start des Einparsens von User Interface Informationen.

wickelt wird. Sie erlaubt das einfache Erstellen und Handhaben von Klassenmodellen im Arbeitsspeicher, sowie das Generieren von Diagrammen und Quelltext aus den verwendeten Informationen. Mithilfe von SDMLib werden die eingelesenen Informationen im Arbeitsspeicher zu einem Klassendiagramm zusammengesetzt. Ist der Einlesevorgang beendet, werden die gewonnenen Informationen weiterverarbeitet. Für jede Klasse wird die entsprechende **`FClass`** aus den Fujaba Klassendiagrammen herausgesucht und alle gefundenen Attribute und Assoziationen entsprechend der in SDMLib vorliegenden Informationen ergänzt. Hierbei ist es wichtig zu beachten, dass die benötigten Typinformationen aus den im Fujaba Projekt als Referenz vorliegenden `GWTUILibrary.ctr` gezogen werden. Dies ist notwendig, um ein späteres Navigieren innerhalb von Storydiagrammen zu ermöglichen. Sind alle Informationen aus dem Quelltext ins Fujaba Klassendiagramm übertragen ist es zwingend notwendig, die User Interface Klassen in Fujaba mit dem Stereotyp ≪`Reference`≫ zu versehen. Auf diese Weise wird verhindert, dass die gerade gewonnenen strukturellen Informationen beim erneuten Generieren überschrieben werden. Änderungen am User Interface sollten, wie in Kapitel 7.2.6 beschrieben, über den GWT Designer vorgenommen werden. Anschließend ist ein erneutes Einparsen der Klassen mit den hier beschriebenen Hilfsmitteln notwendig. Abbildung 7.13 zeigt die **`Login`** Klasse wie sie ursprünglich im Klassendiagramm angelegt wurde. Die Vererbungsinformationen sind bereits vorhanden, jedoch existieren keine weitergehenden Informationen

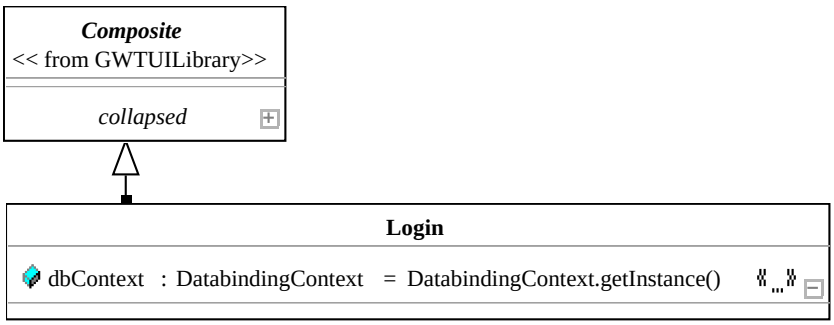

Abbildung 7.13: User Interface Klasse vor der Rückführung der Informationen aus dem User Interface ins Klassendiagramm

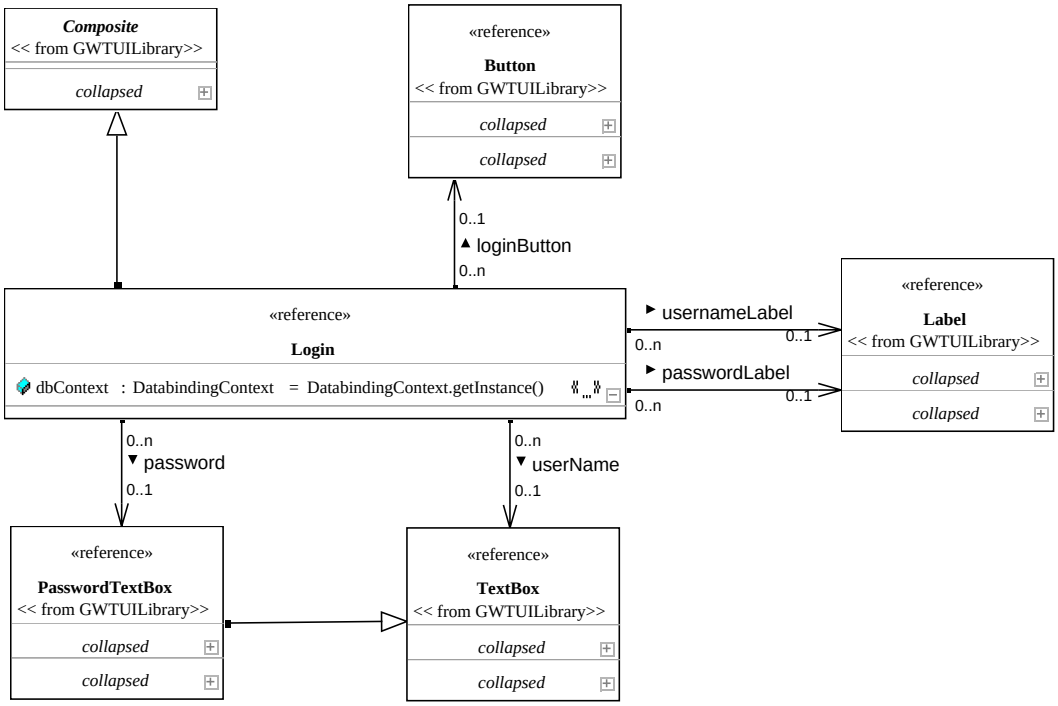

Abbildung 7.14: View Klasse nach der Rückführung der strukturellen Informationen aus der User Interface Klasse ins Klassendiagramm

über die Zusammensetzung des konkreten User Interface. Abbildung 7.14 hingegen zeigt das Klassendiagramm der `Login` Klasse nach dem oben beschriebenen Rückführungsvorgang. Die Zusammensetzung der User Interface Klasse wie sie im GWT-Designer definiert wurde ist über Assoziationen zu den entsprechenden GWT Klassen am Klassendiagramm ablesbar. Zusätzlich ist, wie bereits beschrieben, der Stereotyp ≪`Reference`≫ an der `Login` Klasse eingefügt worden, um ein etwaiges Überschreiben des Quelltextes beim erneuten Generieren zu vermeiden. Die Abbildung der strukturellen Informationen des GWT-Designers mittels Assoziationen im Klassendiagramm ermöglicht das einfache navigieren durch die View Klasse mittels Graphersetzungsregeln in den Storydiagrammen der Controller Klassen. Das genaue Vorgehen hierzu ist in Kapitel 7.2.8 nachzulesen.

7.2.8 Modellieren von Controllern für die User Interfaces

In Kapitel 7.2.6 wurde das Vorgehen zum Erstellen der grafischen Benutzerschnittstelle beschrieben. Ein gut designtes User Interface reicht jedoch für eine funktionale Anwendung nicht aus. Um die Benutzerschnittstelle zum Leben zu erwecken, muss definiert werden, welche Aktionen beim Betätigen der unterschiedlichen Bedienelemente ausgeführt werden sollen. Dafür bietet der GWT-Designer die Möglichkeit, Controller

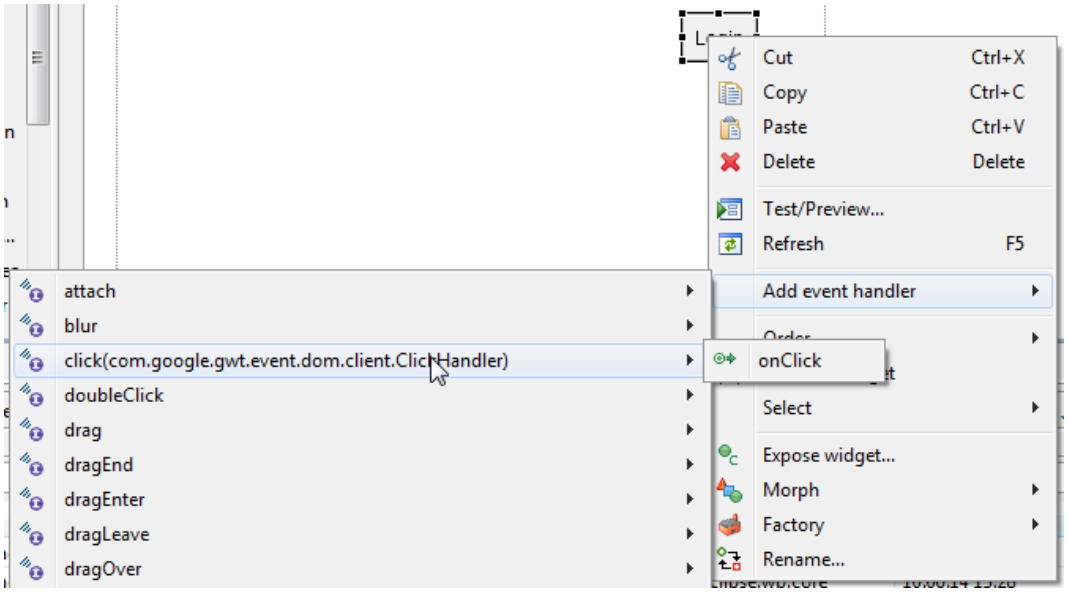

Abbildung 7.15: Menüeintrag zum Einfügen von Standard Controllern im GWT-Designer.

an Bedienelementen anzumelden. Dies geschieht im Kontextmenü mit der Aktion *Add event handler*. Die Einträge im Kontextmenü sowie den erscheinenden Submenüs sind vom ausgewählten Bedienelement abhängig. Für jedes gewählte Element werden vom GWT-Designer die verfügbaren möglichen Event Handler berechnet und diese mit ihren zu implementierenden Methoden in die Menüstruktur eingefügt. Abbildung 7.15 zeigt das berechnete Standardmenü des GWT-Designers bei der Auswahl auf einem Button. Über die Selektion des Menüeintrages *onClick* wird automatisch ein `ClickHandler` am entsprechenden Button angemeldet und der in Listing 7.4 gezeigte Quelltext in die Java-Datei der Userinterface Klasse generiert.

```
1 Button btnLogin = new Button("Login");
2     btnLogin.addClickHandler(new ClickHandler() {
3       public void onClick(ClickEvent event) {
4       }
5     });
```

Listing 7.4: Vom GWT-Designer generierter Standard Quelltext für Event Handler

Wie aus Listing 7.4 ersichtlich, wird eine anonyme innere Klasse in die Userinterface Datei generiert und als `ClickHandler` am Login Button angemeldet. Das genaue Verhalten dieses `Click Handlers` kann innerhalb der entsprechenden Methode, im Beispiel die Methode `public void onClick(ClickEvent event)` aus Zeile 3 des Listings, per Hand implementiert werden. Auf diese Weise werden für jedes UI Element und jeden möglichen Event Handler Typ separate anonyme innere Klassen generiert.

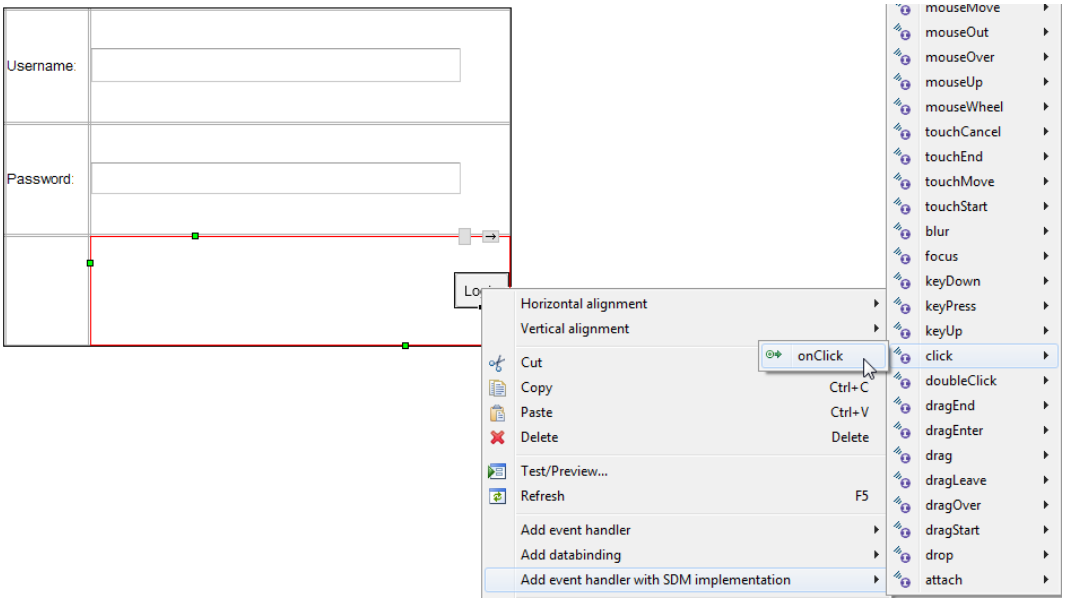

Abbildung 7.16: Eintrag im Rechtsklick-Menü zum Eintragen von Controllern, deren Logik über Story-Driven-Modeling Mechanismen implementiert wird.

Das zuvor beschriebene Standardverhalten des GWT-Designers für die Anmeldung von Controllern, sowie die notwendige Implementierung des Verhaltens von Hand sind nicht passend für die Verwendung im Web Fujaba Process. Hier ist es vorgesehen, möglichst die komplette Anwendung mit Modellen und Techniken des Story-Driven-Modeling zu definieren und anschließend zu generieren. Dies schließt auch die Definition und Implementierung der UI Controller ein. Aus diesem Grund wurde von mir im Rahmen der Fujaba Web Tools eine Erweiterung für den GWT-Designer erstellt, die es ermöglicht, UI Controller an Oberflächenelementen anzumelden und das Verhalten anschließend mit Story-Diagrammen zu modellieren.

Die Erweiterung wurde über den Eclipse Plugin Mechanismus realisiert und benutzt den *Extension Point* `org.eclipse.wb.core.java.javaInfoInitializationParticipators`. Die Implementierung des Extension Points auf Seiten der Fujaba Web Tools ist in der Klasse `de.unikassel.se.gwtw2m.sdm.eventhandling.SDMEventHandlingJavaInfoInitializationParticipator` zu finden. Der verwendete Extension Point wird per Broadcast jedesmal benachrichtigt, wenn vom GWT-Designer ein Objekt des Typs `org.eclipse.wb.core.model.JavaInfo` angelegt wird.

Beim Typ **`JavaInfo`** handelt es sich um ein internes Modellobjekt des GWT-Designers. Alle Objekte dieses Typs haben eine Zuordnung im Eclipse Java AST. Für jedes **`JavaInfo`** Objekt des GWT-Designers wird von den Fujaba Web Tools bestimmt, wel-

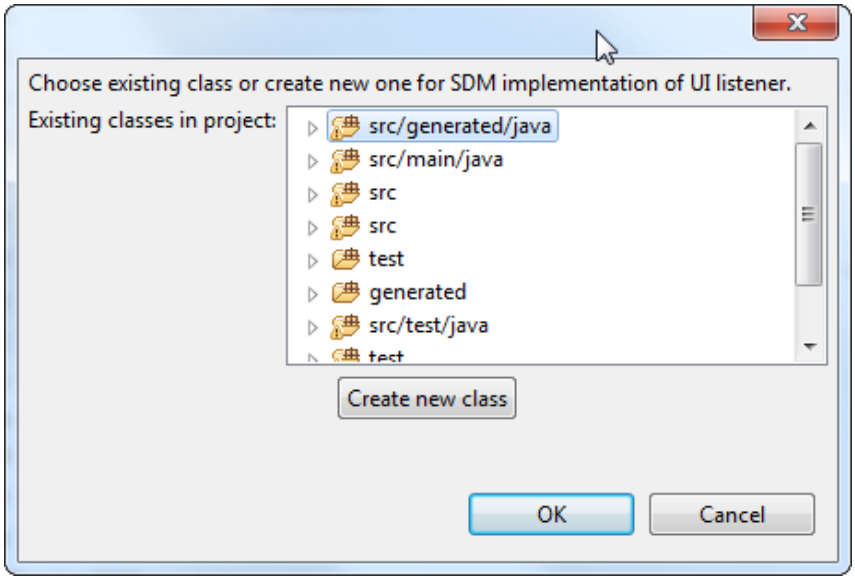

Abbildung 7.17: Dialog zum Auswählen einer Bestehenden oder zum Anlegen einer neuen Controller Klasse mit Story-Driven-Modeling Implementierung.

che `EventHandler` und Methoden an diesem Typ angemeldet werden können und die entsprechenden Menüeinträge im Rechtsklick-Menü generiert. Die Bestimmung der möglichen Elemente verläuft analog zu den Berechnungen des GWT-Designers.

Für jedes **`JavaInfo`** Objekt wird zunächst eine **`SDMEventProperty`** erzeugt. Diese sorgt über einen `BroadcastListener` am zugeordneten **`JavaInfo`** Objekt für die Erzeugung des Context Menüs und dessen Einträgen. Die Methode `public void addContextMenu(...)` der abstrakten Klasse `org.eclipse.wb.core.model.broadcast.ObjectEventListener` wird zu diesem Zweck implementiert. Die Einträge im Menü werden über sogenannte **`SDMListenerProperties`** und **`SDMListenerMethodProperties`** bestimmt und entsprechend eingetragen. Die Erzeugung und Berechnung der Properties erfolgt, wie bereits erwähnt, analog zu den Berechnungen des GWT-Designers, es werden lediglich eigene Property Objekte für resultierende Struktur verwendet. Die entstehenden Menüeinträge beider Kontextmenüs unterscheiden sich somit nicht. Alle Event Handler, die im Standardverhalten appliziert werden können, können dies auch in der SDM Variante. Abbildung 7.16 zeigt das Context Menü für einen Controller mit SDM Implementierung zum Vergleich.

Nach Auswahl des gewünschten Controllers und seiner Methode aus dem Kontext Menü *Add event handler with SDM implementation* werden nun, entgegen des Vorgehens im GWT-Designer, keine anonymen inneren Klassen in den Quelltext des User Interfaces generiert, die anschließend per Hand implementiert werden müssen. Stattdessen öffnet sich der in Abbildung 7.17 gezeigte Dialog, der es dem Benutzer erlaubt, eine bestehende Klasse als Controller wiederzuverwenden oder eine neue Klasse anzulegen. Dieser Weg

erlaubt nun auch, bestehende Klassen um neue Controller-Methoden zu erweitern, oder an verschiedenen Stellen mit gleichem Verhalten denselben Controller zu verwenden.

Soll eine neue Klasse zur Definition des gewünschten Controllers angelegt werden, erfolgt im Dialog aus Abbildung 7.17 ein Klick auf die Schaltfläche *Create new class*. Wie im Eclipse Umfeld üblich öffnet sich anschließend der in Abbildung 7.18 gezeigte Dialog zum Anlegen einer neuen Klasse. Der *New SDM Handler Class* Dialog ermöglicht die Angabe von Paket und Klassennamen. Um Typsicherheit zu gewährleisten, ist der über die Auswahl im Kontextmenü zuvor bestimmte Event Handler bereits im Dialog angegeben.

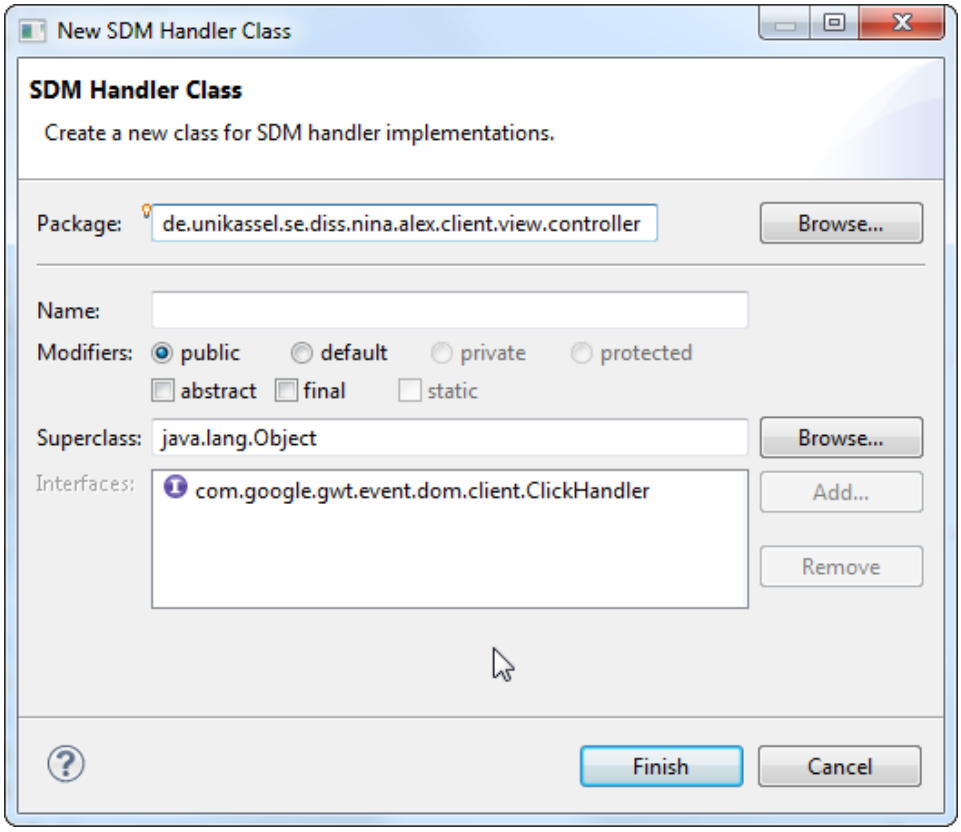

Abbildung 7.18: Dialog zum Erstellen einer neuen Controller Klasse mit Story-Driven-Modeling Implementierung.

Nach Auswahl einer bestehenden oder Anlegen einer neuen Klasse für den benötigten Controller wird nun der in Listing 7.5 angegebene Quelltext in die UI Klasse generiert.

```
1 Button loginButton = new Button("Login");
2 loginButton.addClickHandler(new LoginClickController());
```

Listing 7.5: Abgewandelter Quelltext für Event Handler von den Fujaba Web Tools generiert.

Hierbei wird lediglich eine Instanz der gewählten Klasse erzeugt und als Event Handler an der entsprechenden Oberflächenkomponente angemeldet. Die genaue Implementierung des Verhaltens soll mit den Fujaba Story Diagrammen vorgenommen werden. Zu diesem Zweck muss das *Controller* Klassendiagramm des zugehörigen Fujaba Projektes stets erweitert und angepasst werden. Beim Wiederverwenden einer bestehenden Klasse wird geprüft, ob die zu implementierende Methode und das zu implementierende Interface bereits korrekt im Klassendiagramm vorhanden sind. Ist dies nicht der Fall, wird das Interface im Klassendiagramm hinzugefügt und die Generalisierung zwischen Interface und Controller-Klasse entsprechend gezogen. Die notwendige zu implementierende Methode wird über die Fujaba Funktion *Implement Method* der Controller-Klasse hinzugefügt und so im Klassendiagramm eingetragen. Beim Anlegen einer neuen Klasse wird diese ebenfalls mit allen benötigten Interfaces und Methoden im zugehörigen Fujaba Projekt eingetragen. Abbildung 7.19 zeigt das entsprechende Klassendiagramm nach Einfügen des `LoginClickController` am *Login Button*. Über das Fujaba Kon-

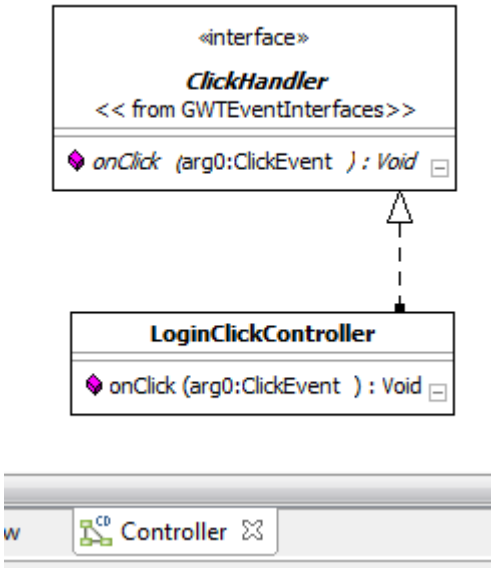

Abbildung 7.19: Fujaba Controller Klassendiagramm mit eingefügter Controllerklasse.

textmenü kann anschließend auf der zu implementierenden Methode `onClick(...)` die Funktion *Create / Goto Method Body* ausgewählt und somit in das zu erstellende Storydiagramm gewechselt werden. Die Implementierung des gewünschten Verhaltens mittels Storydiagrammen erfolgt gemäß der in [NZJ13] beschriebenen Art und Weise. Die strukturellen Informationen aus der grafischen Benutzerschnittstelle, die, wie in Kapitel 7.2.7 beschrieben, in die Fujaba Toolsuite zurückgeführt wurde, ermöglichen an dieser Stelle ein Navigieren mittels Graphersetzungsregeln durch die Benutzeroberfläche. Abbildung 7.20 zeigt eine beispielhafte Implementierung des Login Vorganges mit der zuvor erzeugten grafischen Benutzerschnittstelle. Zunächst wird über einen Methodenaufruf von

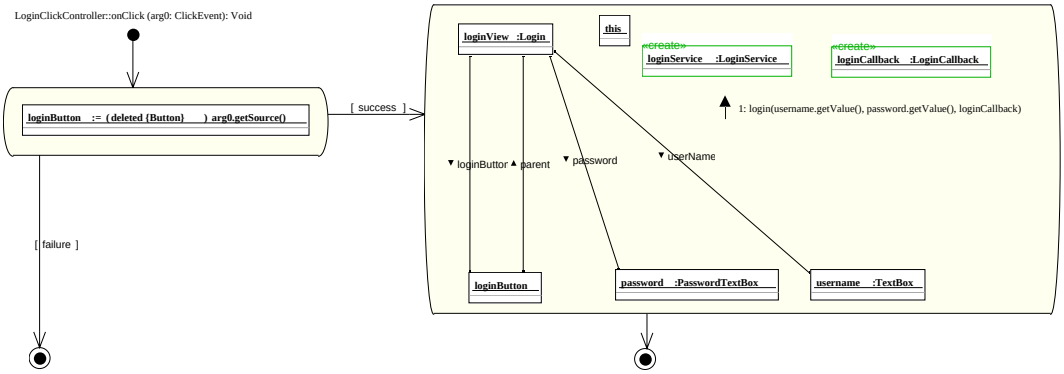

Abbildung 7.20: Fujaba Storydiagramm mit beispielhafter Implementierung für einen Klick auf den Login Button.

`ClickEvent.getSource()` der *Login Button* als Quelle des Events festgestellt und verifiziert. Ist dieser Prozess erfolgreich, kann anschließend die **`Login`** View Klasse über die Kanten `parent` und `loginButton` ermittelt werden. Von dieser Klasse ausgehend werden über die Kanten `username` und `password` die Felder für Benutzername und Passwort zugewiesen. Diese Informationen sind notwendig, um den Login Vorgang über die Klasse **`LoginService`** zu starten. Analog zum beschriebenen einfachen Beispiel sind auch komplexere Berechnungen auf dem zugrunde liegenden Datenmodell mittels Storydiagrammen möglich. Einen einfachen Weg der Navigation vom Userinterface zum Datenmodell kann hier das angebotene und in die Fujaba Web Tools integrierte Databinding, wie in Kapitel 7.2.8 beschrieben, ermöglichen. Die Verarbeitung von Serveraufrufen, wie hier am Beispiel des Login, werden dem Benutzer mit den Fujaba Web Tools stark vereinfacht. Genauere Informationen zu Serveraufrufen und allgemeinem Ablauf der Fujaba Web Applications sind den Kapiteln 4, 7.2.3 und 6 zu entnehmen.

Databinding für GWT User Interface Komponenten mit dem GWT-Designer

Die Literatur kennt, abhängig vom verwendeten Kontext oder Framework, zahlreiche Definitionen für den Begriff Databinding. Im Kontext dieser Arbeit definiert sich Databinding wie folgt:

Definition 7.1 (Databinding) *Databinding bezeichnet den Prozess, Elemente der grafischen Benutzerschnittstelle mit Teilen der Business Logik zu verbinden, um wechselseitig auf Änderungen zu reagieren.*

Dieser Prozess kann beispielsweise hilfreich sein, um Elemente der grafischen Benutzeroberfläche automatisch mit Daten aus dem zugrunde liegenden Datenmodell zu befüllen.

Genauso kann das Datenmodell bei Änderungen über die grafische Benutzerschnittstelle direkt aktualisiert werden. Vorstellbar ist dieses Vorgehen beispielsweise bei der Anzeige von Werten aus dem Datenmodell innerhalb von Textfeldern oder Listen. Der GWT-Designer bietet für die Erstellung von Databindings zunächst keine Funktionen an. Dies ist insbesondere darin begründet, dass auf Seite von GWT die reflektiven Zugriffsmöglichkeiten auf Modellelemente nicht vorhanden sind. Durch die Verwendung der Fujaba Codegenerierung für die Fujaba Web Applications wird diese reflektive Schicht jedoch für das Datenmodell mit generiert und kann somit für die Erstellung und Aktualisierung innerhalb der Databindings verwendet werden. Der SWT-Designer, welcher auf der selben Codebasis entwickelt wurde, wie der GWT-Designer bietet die Möglichkeit zum Databinding von Bean Objekten und SWT Oberflächenelementen an. Die Vorgehensweise des SWT Designers wurde für die Verwendung mit GWT analysiert und anschließend eine eigene Databinding Erweiterung für den GWT-Designer entwickelt.

In ähnlicher Weise, wie für die in 7.2.8 beschriebenen EventHandler wird für das Databinding über den Eclipse Extension Point `org.eclipse.wb.core.java.javaInfoInitializationParticipators` eine Erweiterung eingefügt, welche auf die Erzeugung von `JavaInfo` Objekten reagiert. Für jedes erzeugte Objekt dieses Typs wird der Menüeintrag *Add databinding* ins Kontextmenü eingetragen. Dieser Eintrag beinhaltet die beiden im Bereich Databinding möglichen Aktionen *Attach databinding for selected visual component* und *Attach bean to view class*. Abbildung 7.21 zeigt das Kontextmenü bei Selektion eines Textfeldes aus der Login View. Zum Erzeugen eines Databindings ist es zunächst notwendig, ein Objekt aus der Modellstruktur bei der grafischen Benutzerschnittstelle anzumelden. Dies geschieht über die Aktion *Attach bean to view class*. Bei Auswahl des zugehörigen Menüeintrages aus dem Kontextmenü aus Abbildung 7.21 öffnet sich der in Abbildung 7.22 dargestellte Dialog. Dieser Dialog ermöglicht die Angabe einer Klasse aus dem Datenmodell im Feld *Datatype*. Alle möglichen Typen werden hier in einer Drop-Down Liste angezeigt und können von dort ausgewählt werden. Über das Feld *Name* kann des gewünschte Modellelement mit einem Variablennamen versehen werden, über welchen es im Databinding zugreifbar sein soll. Optional kann über das Feld *Constructor Expression* angegeben werden, in welcher Weise das gewünschte Objekt erzeugt werden soll. Ein Klick auf den Button *Add Bean* fügt das gewählte Modellelement mit den angegebenen Eigenschaften in die View Klasse ein. Alle Attribute des Modellelements können anschließend an grafische Elemente gebunden werden. Das Einfügen des Modellelementes als Bean in die View Klasse führt dazu, dass der GWT Designer im Hintergrund den entsprechenden Quelltext für diese Aktion in die View Klasse generiert. Für das Einfügen wird der in Listing 7.6 gezeigte Quelltext erzeugt. Es genügt an dieser Stelle, ein Feld mit dem Typ des gewünschten Businessobjektes und dem angegebenen Namen in der View Klasse anzulegen und das Feld entsprechend zu initialisieren. Wurde keine *Constructor Expression* im Dialog aus

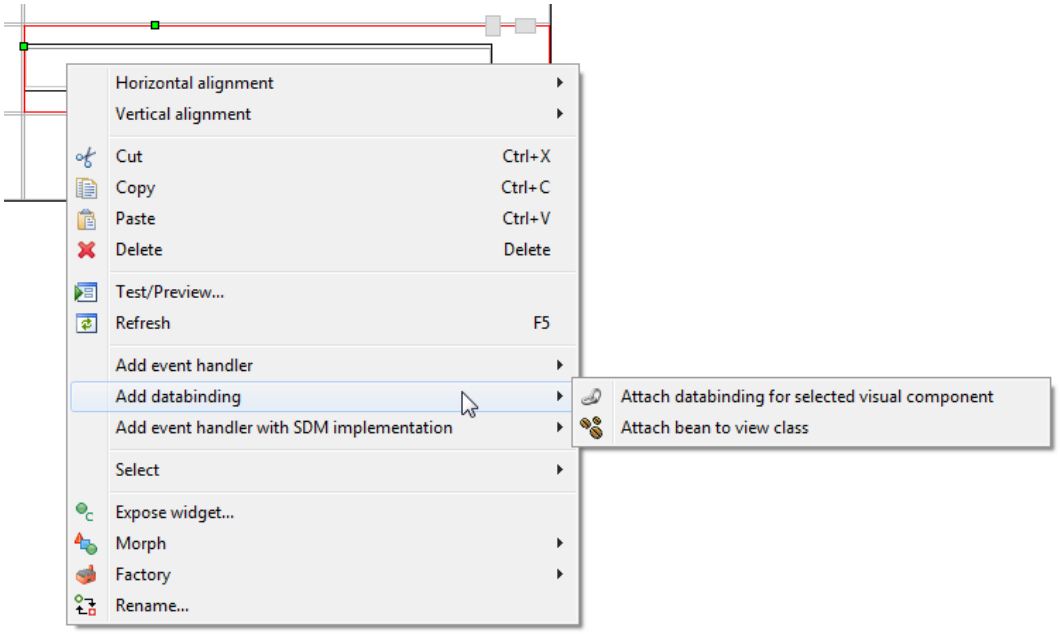

Abbildung 7.21: Kontextmenüeintrag zum Einfügen von Databindings in Fujaba Web Applications mit dem GWT-Designer.

Abbildung 7.22 angegeben wird der Default Konstruktor der Modellklasse aufgerufen.

```
1  private User user = new User();
```

Listing 7.6: Quelltext zum Anmelden des Business Objektes innerhalb der View Klasse.

Für das Databinding ist der Aufruf des Default Konstruktors ausreichend, um den Typ das Objektes und die darin enthaltenen Attribute prinzipiell beim Tooling bekannt zu machen. Das Objekt `user` kann im Anschluss , etwa im Rahmen der Initialisierung der Applikation mit modellierten Action Charts, mit anderen Werten, beispielsweise aus der bestehenden Session, überschrieben werden.

Für das Databinding zwischen Modellelementen und View Elementen im GWT-Designer, wie es von den Fujaba Web Tools vorgesehen wird, ist es notwendig, dass die Modellelemente einige Kriterien erfüllen. Dies deckt sich jedoch mit den Eigenschaften, die den Modellelementen während ihrer Erzeugung nach dem Web Fujaba Process zugeschrieben werden sollten. Es ist notwendig, dass der Codestyle für die Modellelemente auf GWT eingestellt ist. Dies führt, wie in Kapitel 4 beschrieben dazu, dass eine andere Routine und andere Templates für die Codegenerierung der Fujaba Toolsuite verwendet werden. Desweiteren muss das Modellelement von `CObject` erben, was ebenfalls bereits im Bereich der Datenreplikation beschrieben wurde. Abschließend ist es für das Databinding notwendig, das Interface `de.unikassel.se.gwtw2m.databinding.model.`

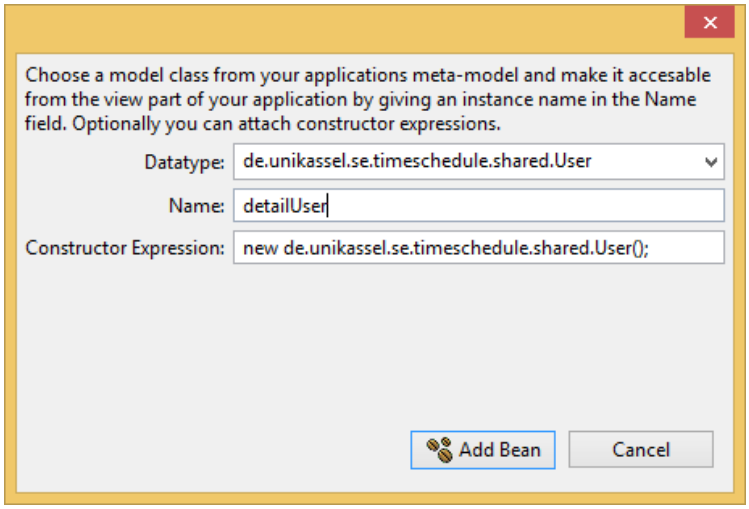

Abbildung 7.22: Dialog zum Einfügen einer JavaBean zum Databinding innerhalb von View Klassen einer Fujaba Web Application.

`GenericAccessibleBean` zu implementieren. Dieser Datentyp wird von der entwickelten Databinding Bibliothek im Modellanteil benötigt. Über die Funktionen `get(String fieldName)` und `getType(String fieldName)` kann auf die Business Objekte der Applikation mittels eines `Strings` zugegriffen werden. Auf diese Weise wird die in GWT fehlende reflektive Schicht ersetzt und die notwendigen Grundlagen zum Binden von UI-Elementen und Business Objekten geschaffen. Die notwendigen Funktionen werden von Fujaba automatisch generiert, wenn an die Modellelemente im Fujaba Klassendiagramm der Stereotyp ≪`GenericAccessibleBean`≫ angefügt wird. Die notwendigen Änderungen an der Fujaba Codegenerierung und der neue Stereotyp wurden im Rahmen dieser Arbeit von mir angelegt.

Sobald das Business Objekt als Bean in die View Klasse eingebunden wurde, stehen alle Felder des Objektes zum Binden an Oberflächenelemente bereit. Über die Auswahl des Menüeintrages *Attach databinding for selected visual component* aus dem in Abbildung 7.21 gezeigten Menü kann ein Databinding mit dem zu diesem Zeitpunkt selektierten Oberflächenelement etabliert werden. Nach einem Klick auf den Menüeintrag öffnet sich der in Abbildung 7.23 gezeigte Dialog. Auf der linken Seite des Dialogs kann nun das zuvor als Bean eingebundene Business Objekt in der Drop-Down-Box ausgewählt werden. Alle zum Databinding verfügbaren Attribute der Bean werden anschließend im Feld *Attribute* angezeigt. Auf der rechten Seite des Dialogs wird die ausgewählte Oberflächenkomponente mir ihren zum Binden verfügbaren Attributen angezeigt. In beiden *Attribute* Feldern muss nun das gewünschte Element ausgewählt und anschließend auf den Button *Bind* geklickt werden. Die Databinding Bibliothek fügt nun den benötigten Quelltext in die Datei der View Klasse ein. Für das Binding eines `text` Attributes eines Labels an den Namen eines `Users` sieht dieser Quelltext wie in Listing 7.7 aufgeführt aus.

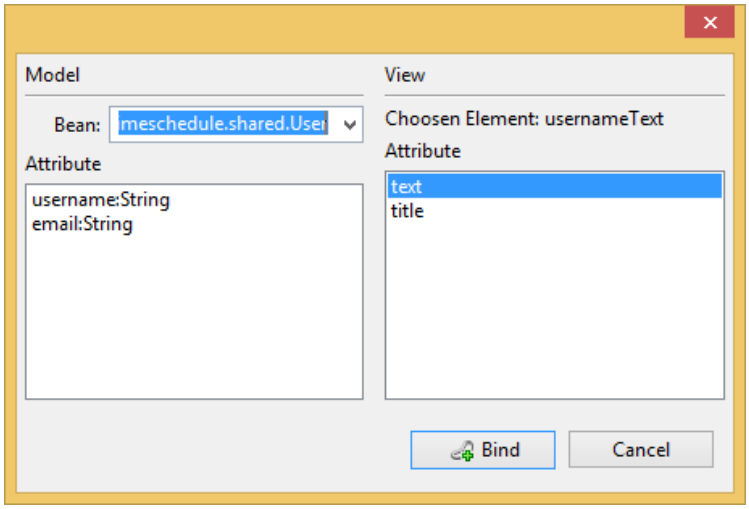

Abbildung 7.23: Dialog zum etablieren eines Databindings zwischen einer visuellen Komponente und einem Modellelement für Fujaba Web Applications.

```
1  final TextBox nameField = new TextBox();
2  final User user = new User();
3
4  UIObservable nameLabelObservable = new UIObservable();
5  nameLabelObservable.observe(nameField, "text");
6  ModelObservable userObservalble = new ModelObservable();
7  userObservalble.observeValue(user, "username");
8  dbContext.bind(nameLabelObservable, userObservalble);
```

Listing 7.7: Quelltext zum Binden des name Attributes eines User Objektes an das text Attribut eines Labels.

An den Zeilen des Listings 7.7 wird bereits die grundlegende Funktionalität des von uns implementierten Databinding-Mechanismus deutlich. Ähnlich wie beim SWT-Databinding des WindowBuilders, [Win], existieren spezielle `Observable` Objekte für User Interface und Model. Diese `Observables` speichern die zu verbindenden Attribute ab. Ein allgemeingültiger `DatabindingContext` verbindet die beiden `Observables` miteinander und benachrichtigt über `PropertyChange` Mechanismen die jeweiligen Objekte über Änderungen an ihrem Verbindungspartner. Der `DatabindingContext` wird bei jeder View Klasse automatisch als Attribut im Klassendiagramm eingetragen und initialisiert. Es handelt sich hierbei um ein Singleton, sodass nur eine Instanz für die komplette Applikation vorhanden ist. Das Anlegen des `DatabindingContext` Attributes erfolgt beim Erzeugen der View Klasse aus dem Workflow Diagramm, wie in Kapitel 7.2.5 beschrieben. Beim anschließenden Generieren des Quelltextes aus dem Klassendiagramm wird der entsprechende Quelltext zum Erzeugen und Initialisieren des Attributes mit in die Klasse übernommen. Die Databinding Bibliothek sowie die Toolunterstüt-

zung im GWT-Designer setzen das Vorhandensein des `DatabindingContext` Attributes voraus. Sollte eine Klasse ohne vorheriges Durchlaufen des Web Fujaba Process und dessen Tools erzeugt worden sein, ist es am Nutzer sicherzustellen, dass das Attribut vorhanden ist, um das Databinding zu nutzen, gleiches gilt für alle durch den Stereotyp ≪`GenericAccessibleBean`≫ erzeugten Methoden im Quelltext der Business Objekte. Die Databinding Bibliothek für die Benutzung im Web Fujaba Process ist zum gegenwärtigen Zeitpunkt noch nicht komplett implementiert. Bislang kann nicht für alle UI Komponenten ein Databinding angeboten werden. Die Entwicklung in diesem Bereich kann jedoch weitergeführt werden. Alle unterstützten Oberflächenkomponenten und ihre Typen sind in der Datei `GWT_UI_Available_Classes_and_Properties.properties` hinterlegt. Diese Datei liegt im *properties* Verzeichnis des *de.unikassel.se.gwtw2m.databinding* Plugins.

7.2.9 Codegenerierung und Erzeugung der lauffähigen Fujaba Web Applikation

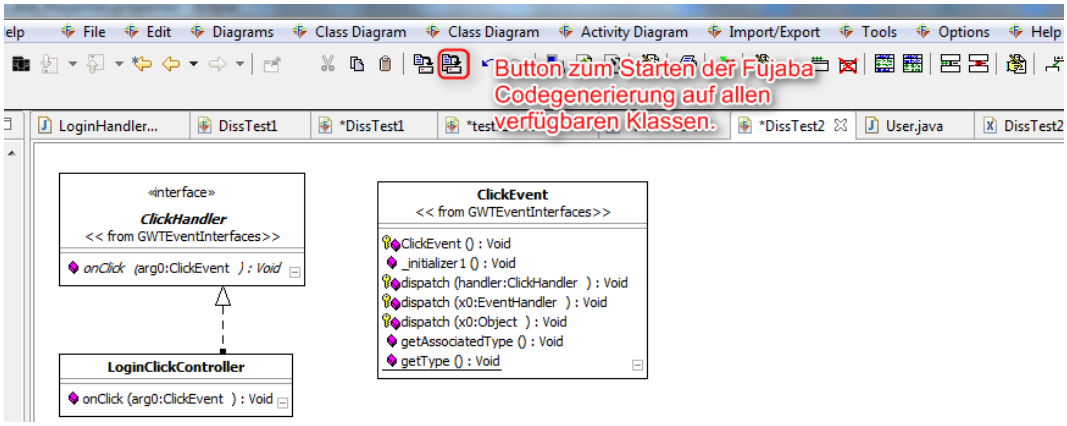

Abbildung 7.24: Fujaba Toolsuite mit Controller Klassendiagramm, ein Klick auf den in der Toolbar markierten Button startet die Codegenerierung für alle verfügbaren Klassen.

Der letzte Schritt des Web Fujaba Process ist die Erzeugung einer lauffähigen Fujaba Web Application. Der wichtigste, aber auch selbstverständlichste Punkt auf diesem Weg ist das Generieren von Quelltext aus allen im Verlauf des Prozesses entstandenen Klassen- und Storydiagrammen. Schon einige Male während der Entwicklung der Applikation waren Codegenerierungsschritte vonnöten, beispielsweise beim Erzeugen des Quelltextes aus den automatisch angelegten View Klassen oder beim Erzeugen des für Controller und Databinding benötigten Quelltextes innerhalb der View Klassen. Die bisherigen Codegenerierungsschritte liefen jedoch automatisiert ab. Die Fujaba Web Tools

lieferten alle benötigten Quelltextfragmente automatisch an die richtige Stelle. Zum Abschluss der Anwendungsentwicklung wird die finale Codegenerierung vom Entwickler angestoßen. Dies geschieht, wie in der Fujaba Toolsuite üblich, über einen Klick auf den entsprechenden Button in der Toolbar. Abbildung 7.24 zeigt die Fujaba Toolsuite mit entsprechendem Button.

Das Ausführen der Codegenerierung durch den Benutzer führt dazu, dass nun alle Klassendiagramme, Storydiagramme und sonstige mit Codegenerierungsfunktionalität versehenen Diagrammtypen innerhalb der Fujaba Toolsuite durchlaufen und für alle darin enthaltenen Informationen der entsprechende Quelltext in Java Dateien geschrieben wird. Einige der Klassen werden hierbei nicht überschrieben, dies trifft vor allem für die View Klassen zu, die nach dem Einlesen der strukturellen Informationen in Schritt 7.2.7 explizit von der Codegenerierung ausgeschlossen wurden. Ebenso werden alle aus externen Bibliotheken stammenden Klassen nicht angetastet, auch diese sind innerhalb der Klassendiagramme mit dem Stereotyp ≪`Reference`≫ versehen.

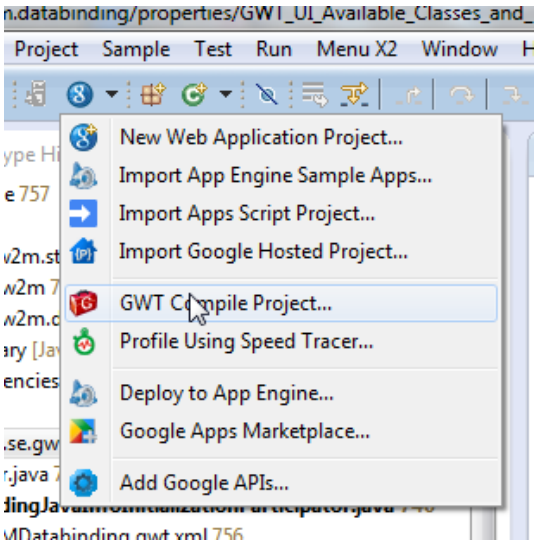

Abbildung 7.25: Menüeintrag zum Crosscompilen eines Web Application Projektes mit dem Google Plugin for Eclipse.

Für alle serverseitigen Klassen, sowie für die Controllerklassen und die dort mit Storydiagrammen entwickelten Methoden, wird nun zum ersten Mal Quelltext generiert. Ist die Fujaba Codegenerierung durchlaufen, sind alle für die Ausführung der Fujaba Web Applikation benötigten Informationen vorhanden. Die Business Logik wurde mittels Klassendiagrammen und Storydiagrammen modelliert, die serverseitigen Klassen, Services und deren Logik wurden modelliert, die Ausführungslogik der kompletten Applikation wurde über Workflowdiagramme definiert, View Klassen wurden mittels des GWT-Designers definiert und die entsprechenden Controller, sowie das Databinding über die von den Fujaba Web Tools zur Verfügung gestellten Werkzeuge implementiert. Die

weiteren Schritte, welche zur Ausführung der Fujaba Web Applikation innerhalb eines Webbrowsers notwendig sind, sind unabhängig von den Fujaba Web Tools oder der zugrunde liegenden Entwicklungsprozesse. Der Java Quelltext der Fujaba Web Applikation muss nun vom GWT-Crosscompiler in browserfähigen JavaScript Code übersetzt werden. Dies übernimmt das Google Plugin for Eclipse [GPE]. Abbildung 7.25 zeigt das entsprechende Menü in der Eclipse Plattform.

Nach abgeschlossenem Durchlauf des GWT-Crosscompilers liegen im `war`-Ordner des erstellten Eclipse Projektes alle für die Applikation notwendigen Daten vor. Diese können nun mit beliebigen Build-Mechanismen, wie beispielsweise Maven [Mav] oder Ant [Ant], jedoch auch durch simplen Export mit Eclipse-Bordmitteln zu einem `war`-Archiv gepackt werden. Hierbei müssen alle verwendeten Bibliotheken mit ins Archiv gepackt werden, um eine korrekte Ausführung der Applikation zu garantieren. Das so erstellte `war`-Archiv kann nun auf einem Webserver deployed werden. Anschließend ist die Applikation unter dem auf dem Webserver angegebenen Pfad mit jedem Browser aufrufbar.

8 Verwandte Arbeiten

Die im Rahmen der vorliegenden Dissertation vorgestellten Ansätze bilden einen Brückenschlag zwischen Graphtransformationen und der Entwicklung von Applikationen mit dem Story-Driven-Modeling Ansatz auf der einen und der Disziplin des Model-Driven Web Engineering auf der anderen Seite. Vorstellungen der clientseitigen Graphtransformationen innerhalb der Web Engineering Community führten oftmals zu Erstaunen hinsichtlich der Mächtigkeit dieses Ansatzes, der in ähnlicher Form in diesem Bereich bislang nicht zu finden war. Die Entwicklungen im Rahmen der vorliegenden Arbeit und die damit einhergehende Einsicht in die Web Engineering Community brachte eine Anzahl an Methoden und Vorgehensweisen mit sich, die sich bislang vor allem im Rahmen der Entwicklung traditioneller (Page-based) Webanwendungen bewährt hatten. Die Arbeiten zu komplexen Rich Internet Applications, wie sie mit den Fujaba Web Applications vorgestellt wurden, waren indes bei den meisten Ansätzen bislang nur rudimentär vorhanden. Eine komplette Verlagerung der Businesslogik auf die Clientseite war in keinem der existierenden Ansätze vorgesehen.

8.1 Web Engineering Methoden

8.1.1 OOH4RIA

Der Ansatz OOH4RIA [PDMG08b], [PDMG08a] stellt eine modellgetriebene Software-Entwicklungsmethodik für Rich Internet Applications (RIAs) dar, welche auf der Object-Oriented Hypermedia Design Method [SR95] basiert.

Die Methodik wird hierbei durch das OOH4RIA Tool [OOH] gestützt. Das OOHH4RIA Tooling setzt, wie auch der in dieser Arbeit dargestellte Ansatz zur Entwicklung von Fujaba Web Applications, auf das Google Web Toolkit auf.

Im Rahmen der Methodik werden unterschiedliche Modelle zur Definition der Gesamtapplikation genannt. Basierend auf dem in [PDMG08b] angegebenen Entwicklungsprozess startet die Methodik mit der Definition eines sogenannten *Domain Model* der Applikation. Das *Domain Model* kann mit Hilfe des Tools grafisch beschrieben werden und repräsentiert gewissermaßen das Datenmodell der Applikation. Zur Beschreibung

kommt hier eine eigene Domain Specific Language (DSL) zum Einsatz, die in ihrer Ausprägung einer Darstellung des Modells als UML-Klassendiagramm ähnelt, dieses jedoch um einige Elemente ergänzt. Im *Domain Model* werden alle modellseitigen Elemente der Applikation inklusive ihrer Beziehungen zueinander beschrieben.

Die Modellierung der User Interfaces und der Abläufe zwischen den User Interfaces werden im sogenannten *Navigation Model* und dem zugehörigen `Presentation Model` angegeben. Das *Navigation Model* definiert hierbei die einzelnen sogenannten *Domain Concepts* und die Flussbeziehungen zwischen diesen. Basierend auf dem erstellten *Navigation Model* wird das *Presentation Model* erzeugt. Hierbei handelt es sich um eine Repräsentation der einzelnen innerhalb der Applikation vorkommenden Userinterfaces. Die Methodik spricht hier von *Screens*. Jeder im *Presentation Model* definierte *Screen* kann durch einen UI Designer statisch entwickelt werden. Ein *Orchestration Model* bildet anschließend die Beziehungen der einzelnen in den *Screens* beschriebenen UI-Komponenten untereinander ab und fügt der Applikation so dynamisches Verhalten hinzu. Das *Orchestration Model* wird als Statechart repräsentiert.

Basierend auf den zuvor dargestellten Modellen der OOH4RIA-Methodik wird in einem abschließenden Schritt Quelltext generiert. Hierbei fließen die Elemente des *Domain Model* und des *Navigation Model* in den serverseitigen Quelltext ein, die Elemente des *Orchestration Model* und des *Presentation Model* werden zur Erzeugung des clientseitigen Quelltextes verwendet.

Die OOH4RIA Entwicklungsmethodik stellt sich als der dem hier vorgestellten Vorgehen ähnlichste Ansatz dar. Während OOH4RIA ein Entwicklungsvorgehen im Model-Driven-Development darstellt, in dem Modelle schrittweise durch Verfeinerung aus anderen Modellen erzeugt werden, stellt der WFuP einen Story-Driven-Modeling Prozess dar. Ein ganzheitliches Vorgehen von der Defintion der Anforderungen in Form von Usecases und textuellen Szenarien bis zur abschließenden Erzeugung der Applikation wird durch diesen Prozess beschrieben. Die von mir eingeführten Workflowdiagramme stellen zwar in ihrer interpretierten Ausprägung innerhalb der Laufzeitumgebung ebenfalls Statecharts dar, unterscheiden sich jedoch von den *Orchestration Models* des OOH4RIA dahingehend, dass jeder Schritt innerhalb des Workflows einen einzigen Ablaufschritt der Applikation beinhaltet. Jeder Schritt ist exakt einem User Interface (`View`) zugeordnet. Logische Elemente, sei es zur kleinteiligen Definition der Abläufe innerhalb der einzelnen `View`, oder aber in der Applikationslogik werden durch Graphtransformationen in Form von *Aktivitätsdiagrammen* beschrieben.

Fujaba Web Applications beinhalten darüber hinaus keine derart strikte Trennung zwischen Client und Server. Anders als in OOH4RIA werden im Rahmen von Fujaba Web Applications Datenmodelle ebenfalls in clientseitigen Quelltext übersetzt und auf den

Client ausgelagert. Gleiches gilt für Anteile clientseitiger Businesslogik, die in Form von *Aktivitätsdiagrammen* modelliert werden. Eine Modellierung der notwendigen Businesslogik kann nach meinem Wissen mit OOH4RIA nicht vorgenommen werden, alle Datenmodellanteile werden gemäß der Dokumentation in serverseitigen Quelltext übersetzt.

Zur Sicherung der Daten wird im Zusammenhang mit OOH4RIA von einer Hibernate basierten Lösung gesprochen (vgl. [OOH]). Im Rahmen der von mir entwickelten Fujaba Web Applications wird die Persistierung der Daten, wie auch die Replikation clientseitiger Daten, von WebCoObRA übernommen.

Die Ablauflogik der Applikation und die Navigation zwischen einzelnen `Views` wird, anders als bei OOH4RIA, im Rahmen der von mir entwickelten Applikationen nicht generiert, sondern stattdessen innerhalb des Clients von der Laufzeitumgebung sichergestellt. Zu diesem Zweck beinhaltet die Laufzeitumgebung einen Interpreter, der die zugrunde liegende Workflowbeschreibung einliest und entsprechend interpretiert.

Interaktive Aspekte der Applikation werden im Rahmen der Fujaba Web Applications in Form von Action Charts oder speziellen Implementierungen der UI-Handler mit Graphtransformationen als *Aktivitätsdiagramme* modelliert.

8.1.2 UML-based Web Engineering - UWE

Bei UWE [NK02] handelt es sich um eine Kombination aus Notation, Entwicklungsprozess und Tooling für die Entwicklung von Webapplikationen. Hierbei basieren alle Teile des Ansatzes auf Standards, wie beispielsweise UML. Der Ansatz ist jedoch nicht allein auf die Nutzung von UML beschräkt. Wo immer es möglich ist, wird auf standardkonforme UML Notation und UML Diagrammtypen zurückgegriffen. Zur Modellierung spezifischer Eigenschaften von Webapplikationen wurde ein eigenes UML-Profil definiert, in welchem notwendige Stereotypen etc. definiert sind.

UWE deckt als Gesamtansatz den kompletten Entwicklungsprozess einer Webapplikation ab und ist zu diesem Zweck in die folgenden Teilbereiche gegliedert, die jeweils mit eigenen Modellen beschrieben und über Methoden des Model-Driven-Development (MDD) zusammengefügt werden.

- Requirements: Requirements werden in UWE durch zwei unterschiedliche Diagrammarten festgehalten. Zum einen werden Usecasediagramme erstellt, die die Anforderungen der gesamten Applikation beschreiben. Zusätzlich wird für jeden identifizierten Usecase eine Aktivität angelegt, welche die Anforderungen des Usecase detaillierter beschreibt. Hier kommen Ablaufdiagramme zur Modellierung der Aktivitäten zum Einsatz.

- Content: Hierbei handelt es sich um ein traditionelles Klassendiagramm, welches alle zur Laufzeit der Applikation notwendigen Klassen modelliert. Das in diesem Schritt erzeugte Klassendiagramm spiegelt somit das Datenmodell der Applikation wieder.

- Navigation: Das Navigation Model stellt die Verbindungen zwischen unterschiedlichen Teilen (Webseiten) der Applikation dar. Seiten werden hierbei als Knoten, Navigationsmöglichkeiten als Verbindung zwischen zwei Knoten dargestellt. Ist eine Verbindung zwischen zwei Knoten im Modell vorhanden, so kann zur Laufzeit zwischen den zugehörigen Seiten der Applikation navigiert werden.

- Presentation: Die genaue Zusammensetzung einer einzelnen Seite der Applikation wird durch das sogenannte Presentation Model dargestellt. Hier kann für jede im Navigation Model vorkommende Seite genauer definiert werden, aus welchen Elementen diese zusammengesetzt ist und wie diese miteinander verbunden sind.

- Process: Ähnlich wie die Requirements werden auch alle das Process Model betreffenden Informationen in zwei unterschiedlichen Diagrammarten festgehalten. Zunächst wird im Process Structure Model die allgemeine Struktur aller in der Applikation vorkommenden Prozesse, sowie deren Verbindung untereinander festgehalten. Die Diagramme haben Klassendiagrammcharakter. Anschließend werden die einzelnen in der Struktur festgehaltenen Prozesse durch Process Flow Diagramms genauer beschrieben. Hierbei handelt es sich um Flussdiagramme, mit denen der Arbeitsfluss der Prozesse detailliert dargestellt werden kann.

Für einige der oben beschriebenen Modelle existieren zudem Modell-zu-Modell-Transformationen, mit welchen, gemäß MDD, das Grundgerüst der Modells aus vorigen Schritten erzeugt werden kann und im Anschluss weiter verfeinert wird.

Der UWE Ansatz stellt, wie der im Rahmen dieser Arbeit dargestellte Ansatz zum Story-Driven-Modeling von Webapplikationen, einen ganzheitlichen Ansatz dar. Es wird ein Prozess definiert, der dem Nutzer die Modellierung einer Webapplikation ermöglicht. Anders als die im Rahmen dieser Arbeit zu entwickelten Applikationen handelt es sich bei den mit UWE erzeugten Webapplikationen um traditionelle Page-based Applikationen, nicht um RIAs. In der Zwischenzeit existieren einige Ansätze, UWE für die Entwicklung von RIAs zu erweitern [KPZM09], [PLMC+08].

In [KPZM09] wird zur Entwicklung von RIAs ebenfalls ein Ansatz vorgestellt, in welchem durch die Modellierung von sogenannten Patterns für Events und Timer wiederverwendbare Komponenten für den Modellierungsprozess von RIAs erzeugt werden. Die Patterns werden in Form von Statecharts modelliert und sollen anschließend im Modellierungsprozess von UWE eingesetzt werden können, beispielsweise beim Presentation Model.

Die im Rahmen des Story-Driven-Modeling von mir verwendeten Statecharts haben gegenüber den oben vorgestellten Patterns den Vorteil, komplexe, spezifisch auf die Applikation abgestimmte Verhaltensweisen in Form von Action Charts modellieren zu können. Hierbei sind die Action Charts nicht auf vordefinierte Pattern eingeschränkt. Zudem werden innerhalb der Action Charts Graphersetzungsregeln zur Modellierung des reaktiven Verhaltens eingesetzt, die es erlauben, komplexe Operationen auf dem im Client vorliegenden Objektgraphen vorzunehmen. Durch den adaptierten Codegenerierungsmechanismus liegen bei Fujaba Web Applications alle modellierten Graphersetzungsregeln der Action Charts auf dem Client vor und werden dort ausgeführt. So kann auf clientseitig auftretende Events, beispielsweise PropertyChange Events, direkt reagiert werden.

8.1.3 Web Modeling Language - WebML

Die Web Modeling Language - WebML [Webb] stellt eine standardgemäße Notation zur Spezifizierung komplexer Webseiten auf konzeptuellem Level bereit. Toolunterstützung für WebML bietet die Entwicklungsumgebung WebRatio [Weba]. Innerhalb der Entwicklungsumgebung können Webapplikationen modelliert und anschließend automatisch generiert werden.

Auch WebML definiert hierbei einen kompletten Entwicklungsprozess für die Entwicklung von Webapplikationen, der inkrementell durchlaufen und zyklisch wiederholt werden sollte. Bei jedem Durchlauf entsteht ein Prototyp der Applikation, der getestet und evaluiert werden kann. Auf diese Weise wird eine sukzessive Weiterentwicklung der Applikation ermöglicht. Hauptunterstützung von WebML wird bei Durchlaufen des Prozessen in den Phasen *Data-Design* und*Hypertext-Design* geboten.

Das von WebML definierte Data Model wird zur Modellierung der Anwendungsdaten benutzt und hat Ähnlichkeiten mit klassischen Klassendiagrammen oder Entity-Relationship-Modellen zur Modellierung von Datenbanken. Hier werden alle Daten in Form von Entities und Verbindungen zwischen Entities beschrieben.

Im Hypertext-Model der WebML wird sowohl die Zusammensetzung der Webapplikation (Composition), als auch die Navigation der Elemente (Navigation) beschrieben. Das Hypertext-Model kann aus 5 grundlegenden Bausteinen aufgebaut werden:

- Areas und Site views: Diese Elemente dienen zur hierarchischen Gliederung einer Webapplikation. Durch Site views können Applikationseinheiten definiert werden, die auf die Anforderungen vorgesehener Nutzergruppen abgestimmt sind. Areas hingegen bilden Gruppen von Pages innerhalb einer Site view. Innerhalb klassischer Webseiten können Areas etwa die Hauptkategorien der Navigation darstellen.

- Pages: Pages stellen die tatsächlichen Seiten einer Webapplikation dar, die vom Server an den Benutzer übertragen werden. Jede Page besteht dabei gewöhnlich aus mehreren Units, die den Inhalt der Seite abbilden. Pages können innerhalb eines Site view als *home* definiert werden. Dies hat zur Folge, dass die entsprechende Page automatisch angezeigt wird, sobald die Site view aufgerufen wird.

- Units: Hierbei handelt es sich um die elementaren Elemente einer Seite. Über Units können Daten aus dem Datenmodell der Webapplikation angezeigt werden. Es wird weiterhin unterschieden zwischen Data units zur Darstellung eines einzelnen Datenobjektes, und Units die zur Repräsentation mengenwertiger Daten verwendet werden. Optional können Units mit einem Selector verbunden werden, über den beispielsweise Einschränkungen der darzustellenden Daten etc. realisiert werden können.

- Operations: Durch die Verwendung von Operations können Aktionen modelliert werden, die bei der Navigation einer Webapplikation auftreten können. Da Operations nicht zur Darstellung von Daten verwendet werden, werden sie außerhalb von Pages im Modell platziert.

- Links: Durch Links wird die Navigation einer Seite angegeben. Links können zwischen Units einer einzelnen Page, zwischen Units unterschiedlicher Pages und zwischen unterschiedlichen Pages spezifiziert werden. Jeder Link kann mit einem sogenannten Context ergänzt werden, mit welchem die Kompatibilität der verbundenen Units sichergestellt werden kann.

In ihrer ursprünglichen, oben beschriebenen, Form ist WebML zur Modellierung traditioneller Page-based Applikationen geeignet. Im Rahmen von [BCFC06] und [CC07] wurde der Ansatz jedoch sukzessive auf die Anforderungen von RIAs erweitert. Die Einführung eines Dynamic Model inklusive sogenannter Computation Sequences bieten die Möglichkeit, Applikationsverhalten mit WebML zu modellieren. Diese können nach meinem Verständnis als Zwischenlösung zwischen der in Fujba Web Applications vorgenommenen Modellierung von UI-Listenern, in Form von Graphersetzungsregeln, und den innerhalb der Workflowdiagramme modellierten Übergängen unterschiedlicher Views angesehen werden. Während die vorgestellten Computing sequences stets die zugehörige Page und alle dort dargestellten Widgets kennt und diese verändert, können die in Fujaba modellierten UI-Listener davon teilweise unabhängig sein. Die modellierten Graphersetzungsregeln können ebenso zum Absetzen vonC RPC-Calls an den Server genutzt werden, um so neue Daten abzurufen oder aber lediglich Operationen auf der clientseitigen Datenstruktur durchzuführen. Die Transitionen der Workflowdiagramme hingegen resultieren immer in einem Wechsel der View.

Die WebML unterzog sich zuletzt einem Wandel, von einer Modellierungssprache für

komplette Webapplikationen hin zu einer Modellierungssprache für UI Modellierung. WebML wird in diesem Zuge zur Interaction Flow Modeling Language (IFML) [IFM]. Diese stellt einen Standard der Object Management Group (OMG) zur Beschreibung von Inhalt, Kontrollfluss und Verhalten von Applikationsfrontends dar. Hierbei werden als Ziel nicht nur RIAs, sondern auch traditionelle Webseiten und mobile Applikationen, Client-Server Applikationen und Desktopapplikationen genannt. Ein ganzheitlicher Ansatz zur Modellierung der kompletten Applikation (Client und Server), wie dies der in dieser Arbeit vorgeschlagene WFuP vorsieht, ist hier nach meinem Wissen nicht vorgesehen.

8.1.4 Object-Oriented Web-Solutions - OOWS

Der Modellierungsansatz OOWS [PGIP01], [PFP03], [OP01] definiert, ähnlich wie die vorgenannten Ansätze mehrere Modelle, die im Rahmen des Entwicklungsprozesses in den Prozessschritten

- fuctional requirements elicitation
- conceptual modeling
- navigation and presentation modeling

erstellt werden. Die Sammlung der Anforderungen im ersten Prozessschritt kann hierbei beliebig mit Usecases und textuellen Szenarien vorgenommen werden. Ziel des Schrittes ist es, ein erstes konzeptuelles Schema der zu entwickelnden Applikation zu erzeugen.

Das durch die Anforderungen definierte konzeptuelle Schema der Applikation wird im Folgenden durch geeignete Modellierungsmethoden weiter verfeinert und auf diese Weise die Struktur sowie das Verhalten der Applikation festgehalten.

Die Navigationseigenschaften der Applikation werden im abschließenden Prozessschritt in Form einer sogenannten *Navigational Map* modelliert. Für alle unterschiedlichen Arten von Applikationsnutzern werden jeweils separate Modelle angelegt. Auf diese Weise wird die Navigation zwischen einzelnen Elementen der Applikation basierend auf dem Nutzertyp in Form einer Art von Erreichbarkeitsgraph abgebildet.

Basierend auf den in den *Navigational Maps* festgehaltenen Elementen (*Navigational contexts*) wird ein Presentation Model erzeugt. Dieses enthält die *Navigational contexts* als Basiselemente, die im Folgenden durch die Angabe von *Presentation Patterns* jeweils weiter verfeinert werden.

Zur Modellierung von RIAs wurde auch für den OOWS Ansatz eine Erweiterung erstellt [OOW]. Hier werden im Rahmen des *conceptual modeling* Modellierungsschritte für

Web 2.0 Patterns, sowie *RIA Interfaces* eingeführt. Das ehemalige *Navigational Model* wurde durch ein *Abstract Interaction Model* ersetzt, das *Presentation Model* vom zuvor erwähnten *RIA Interface Model*.

Im *Abstract Interaction Model* werden Interaktionen des Nutzers mit der Webapplikation modelliert, die durch Bedienung eines User Interface vorgenommen werden und der Erfüllung einer bestimmten Aufgabe dienen. Die Interaktionen werden hierbei abstrakt modelliert, ohne etwaige technische Details zu berücksichtigen. Zur Modellierung dienen *User Diagrams*. Für alle unterschiedlichen Nutzertypen eines Systems werden jeweils eigene Diagramme angelegt, welche ihrerseits unterschiedliche *Interaction Contexts* beinhalten. Hierbei wird im Rahmen der *Interaction Contexts* zwischen unterschiedlichen Leveln unterschieden, die die Erreichbarkeit eines *Interaction Context* beschreiben. Nur *Interaction Contexts* des ersten Levels sind jederzeit erreichbar. Einzelne *Interaction Contexts* können Ketten bilden. Nachfolgende Elemente der Kette sind in diesem Fall nur dann erreichbar, wenn die vorstehenden Elemente erfolgreich abgeschlossen sind. Auf diese Weise können Arbeitsabläufe, ähnlich der Workflowdiagramme der vorliegenden Arbeit, modelliert werden. *Interaction Contexts* können durch die Angabe von *Abstract Interaction Units* weiter verfeinert werden. Diese stellen zumeist Severanfragen dar. *Web 2.0 Patterns* dienen der zusätzlichen vereinfachten Modellierung wiederkehrender Interaktionsmuster innerhalb von RIAs. Die vordefinierten Pattern können bei der Definition einzelner *Interaction Contexts* beziehungsweise *Abstract Interaction Units* verwendet werden.

Die Beschreibung der für RIAs notwendigen User Interfaces wird durch das *RIA Interface Model* modelliert. Hier werden unterschiedliche *Widgets* und abstrakte Beschreibungen interaktiver Funktionen innerhalb des Metamodells definiert. Außerdem wird ein zweites Metamodell für die Erzeugung von Adobe Flex Applikationen erzeugt. Neben reinen visuellen Komponenten werden im *RIA Interface Model* auch mögliche Events berücksichtigt. Zur Behandlung der auftretenden UI-Events werden Regeln definiert, welche im Flex-Metamodell eingehängt werden.

OOWS2.0 stellt Modellierungsmöglichkeiten für die Erzeugung von RIA Applikationen mit Adobe Flex zur Verfügung. Neben der unterschiedlichen technischen Realisierung zeichnen sich die im Rahmen dieser Arbeit definierten Fujaba Web Applications mit ihrem WFuP Entwicklungsprozess auch durch die Implementierung und Ausführung von Datenmodellen und Graphersetzungsregeln auf dem Webclient aus. Die im Rahmen des WFuP vorgestellten Modellierungsansätze stellen darüber hinaus, gerade in Zusammenhang mit dem verfügbaren Tooling, einen weniger abstrakten Ansatz der Modellierung von Webapplikationen dar.

8.2 Google Wave Operational Transformations

Google Wave Operational Transformations [DWL10] ist ein Replikationsframework von Google, das für die von Google entwickelten Multi-User Applikationen eingesetzt wird. Die Wave Operational Transformations (OT) erlauben unterschiedlichen Nutzern simultan an einem geteilten Dokument zu arbeiten. Hierbei werden die Änderungen live zwischen den unterschiedlichen Instanzen repliziert. Auf diese Weise wird es möglich, Änderungen anderer Nutzer sofort, Zeichen für Zeichen, in der eigenen Instanz des Dokumentes mitzuverfolgen. Ähnlich wie beim in Kapitel 4.3.2 beschriebenen WebCoObRA Framework, werden die Dokumente zwischen den Clients repliziert. Jeder Client kann auf seiner lokalen Kopie des Dokumentes jederzeit Änderungen durchführen.

Im Rahmen von OT basieren die Datenstrukturen auf sogenannten *Waves*. Jede *Wave* ist ihrerseits aus meheren *Wavelets* aufgebaut. *Wavelets* wiederum beinhalten Dokumente. Jedes Dokument hat eine XML-Struktur, ergänzt um spezielle Annotationen. *Wavelets* bilden die Basis für die Anwendung der OT. Zur Replikation und Anwendung der Änderungen werden von Client und Server sogenannte *state spaces* aufgebaut. Jeweils ein Client und der Server sind über einen solchen *state space* miteinander verbunden und synchronisieren auf diese Weise ihre Änderungen. Auf dem Server wird für jeden verbundenen Client ein eigener *state space* aufgebaut. Der Server ist für die Synchronisation und die Übertragung von Änderungen zwischen den verschiedenen *state spaces* zuständig. Innerhalb eines *state spaces* werden sowohl vom Client als auch vom Server sequentiell Änderungen gesendet. Die notwendigen OT die vom Client, bzw. Server durchgeführt werden, um zu einem bestimmten Zustand innerhalb des *state space* zu traversieren, werden unabhängig voneinander berechnet und sind nicht notwendigerweie gleich. Es können also auf dem Client und dem Server Änderungen in unterschiedlicher Reihenfolge durchgeführt werden, solange sie zum selben Endergebnis führen. Dies ist vor allem auch der Tatsache zuzuschreiben, dass lokal auf dem Client vorgenommene Änderungen sofort angewendet werden. Zur einfacheren Synchronisation des Servers wurden im Rahmen einer Erweiterung der OT *Acknowledgements* für die verbundenen Clients eingeführt. Dies ermöglicht es dem Client, den vom Server verwendeten Pfad zu berechnen und auf dieser Basis zum Serverpfad passende Operationen zu übermitteln. Das Berechnen und Pflegen einzelner *state spaces* für alle Clients wird auf diese Weise überflüssig und die serverseitige Synchronisation vereinfacht.

Die tatsächliche Transformation wird in Form von Dokumentoperationen auf einem XML-Stream durchgeführt. Im Gegensatz zu dem in WebCoObRA verwendeten Ansatz werden die Wave Operational Transformations mit diesen Dokumentoperationen auf Textbasis durchgeführt. WebCoObRA verwendet zum Übertragen der auftretenden Änderungen ebenfalls einen Changestream im textuellen Format. Die mittels Change-

stream übertragenen Änderungen werden anschließend jedoch auf Basis der Graphstruktur, die das zugrunde liegende Modell repräsentiert, durchgeführt und stellen somit keine Dokumentenoperationen sondern Graphtransformationen dar.

9 Fazit und Ausblick

Im Rahmen der vorliegenden Arbeit wird ein Konzept zum Modellieren komplexer Webapplikationen mit dem Story-Driven-Modeling (SDM) Ansatz vorgestellt und dessen Entstehung beschrieben. Das Gesamtkonzept wird hierbei durch den von mir entwickelten modellbasierten Web Fujaba Process gestützt. Dieser Prozess erweitert den bereits vorhandenen modellbasierten Fujaba Process. Über die dort definierten Vorgehensweisen hinaus stellt der Web Fujaba Process Methoden zur Ablaufsteuerung der entstehenden Webapplikation wie auch zur Modellierung der notwendigen grafischen Benutzerschnittstellen bereit. Die zentrale Rolle bei diesen Schritten spielen die sogenannten Workflowdiagramme. Diese Diagramme bilden den von der Applikation zu implementierenden Userworkflow ab und fungieren als dediziertes Entwicklungsartefakt. Basierend auf den im Workflowdiagramm festgehaltenen Arbeitsschritten werden die einzelnen grafischen Benutzerschnittstellen der Applikation (View) erstellt. Zudem wird von einem interpretativen Laufzeitsystem der Ablauf der Applikation, basierend auf den im Workflowdiagramm festgehaltenen Zuständen und Transitionen, interpretiert. Auf diese Weise soll eine Diskrepanz zwischen abzuarbeitendem Userworkflow und in der Applikation implementiertem tatsächlichen Applikationsablauf vermieden werden.

Das Ziel der Arbeit, die Mechanismen des Story-Driven-Modeling auf die Entwicklung komplexer Webanwendungen zu übertragen, konnte mit dem vorgestellten Prozess und dem entwickelten Tooling erreicht werden. Die bei Durchführung des entworfenen Entwicklungsansatzes entstehenden Webapplikationen wurden im Rahmen der vorliegenden Arbeit definiert. Neben den genauen Eigenschaften der Webapplikationen wurden alle zur Realisierung der Eigenschaften notwendigen technischen Umsetzungen in Form von Anpassungen der Fujaba Codegenerierung und des Fujaba Diagrammkanons beschrieben. Im einzelnen wurden hierbei:

- die vorhandenen Fujaba Klassendiagramme um den Codestyle *GWT* erweitert, um die Erzeugung von GWT-kompatiblem Quelltext zu ermöglichen.
- der Stereotyp *«GWTClassTable»* für Klassendiagramme eingeführt.
- die vorhandene Fujaba Laufzeitbibliothek zur Ausführung von Graphersetzungsregeln auf die von GWT gestellten Anforderungen an clientseitigen Quelltext angepasst.

- die Templates für die Codegenerierung so angepasst, dass automatisch generische Zugriffsmethoden für Attribute erzeugt werden. Auf diese Weise, und durch die Anpassung der Laufzeitbibliothek, kann das Fehlen reflektiver Zugriffsmöglichkeiten auf dem Webclient kompensiert werden.

- das vorhandene Replikationsframework CoObRA zu WebCoObRA erweitert, um so replizierte Anwendungsdaten innerhalb der Webclients zu ermöglichen. Dies erforderte neben Anpassungen am Laufzeitsystem vor allem die Etablierung der Client-Server Kommunikation des Replikationsframeworks über GWT-RPC.

- die in Fujaba vorhandenen Statecharts soweit semantisch zu Action Charts erweitert, dass das komplette reaktive Verhalten einer Webapplikation modelliert werden kann und so komplexe Implementierungsdetails, wie beispielsweise asynchrone Serveraufrufe, komplett vom Entwickler versteckt werden können.

- die zur Modellierung des Anwendungsablaufs notwendigen Workflowdiagramme eingeführt und beschrieben

Die Anpassungen an Fujaba ermöglichten in der Folge erstmals, komplexe Graphtransformationen, wie sie mit Fujaba spezifiziert werden können, innerhalb des Webclients auszuführen. Webbrowser wurden auf diese Weise zu einer weiteren möglichen Graphtransformations-Engine erweitert.

Neben den notwendigen Anpassungen an Fujaba wurden alle durch den neu definierten Entwicklungsprozess hinzukommenden Entwicklungsschritte mit Tooling gestützt. Hierzu wurde insbesondere:

- Ein grafischer Editor zum Erstellen und Editieren der Workflowdiagramme erzeugt.

- Eine Anbindung der Workflowdiagramme an die Klassendiagramme von Fujaba realisiert.

- Der GWT-Designer zur Spezifikation der aus dem Workflowdiagramm abgeleiteten GUI-Komponenten (Views) in den Entwicklungsprozess eingebunden

- Eine Rückgewinnung der mittels GWT-Designer spezifizierten grafischen Benutzeroberflächen in die Klassendiagrammstruktur von Fujaba gewährleistet

- Der GWT-Designer um Möglichkeiten zur Anbindung von Fujaba UI-Handlern erweitert. Neben der Möglichkeit einen UI-Handler klassisch per Hand zu programmieren wurde hier die Möglichkeit geschaffen, die Funktionalität des UI-Handlers mit SDM Methoden in Fujaba vorzunehmen.

- Der GWT-Designer um Möglichkeiten zum Databinding erweitert. Auch hier

kommt die durch die Adaption der Fujaba Codegenerierung erzeugte generische reflektive Zugriffsschicht zum Einsatz, ohne die eine generische Anbindung von Datenmodell-Werten an UI-Komponenten nicht möglich wäre.

Die bereits im klassichen Fujaba Process als Entwicklungsartefakte enthaltenen, aber bis dahin nicht durch Fujaba-Tooling unterstützten *Usecases* und *textuellen Szenarien* wurden zudem durch die Implementierung neuer Editoren in den Entwicklungsprozess integriert. Die zur Entwicklung notwendigen Bibliotheken und Entwicklungsartefakte können automatisch über einen von mir bereitgestellten Project-Wizard in Eclipse erzeugt werden.

Die Entwicklung komplexer Webapplikationen mit dem Story-Driven-Modeling Ansatz ist mit den im Rahmen der vorliegenden Arbeit vorgestellten Vorgehensweisen und Tools durchführbar. In einigen Punkten lassen sich jedoch Erweiterungen sowie weitergehende Fragestellungen ableiten.

Die Nutzung von WebCoObRA in der hier vorgestellten Form hat einige Nachteile. Zum Einen ist die Implementierung eines Offline-Modus für Webapplikationen mit dem hier vorgestellten Ansatz nur bedingt möglich. Eine Speicherung der offline vorgenommenen Änderungen am Modell ist nur bedingt möglich. Sobald die Anwendung geschlossen wird, gehen alle bis dahin nicht auf den Server übertragenen Änderungen verloren. Hier wäre eine Anbindung an den clientseitigen Speicher, wie er im Zuge von HTML5 eingeführt wurde, wünschenswert. Ebenso ist die Serverseite des WebCoObRa Frameworks in seiner beschriebenen Form darauf angewiesen, das komplette Datenmodell im Speicher vorzuhalten. Eine Speicherung erfolgt in Textform, als CTR-File. Gerade für große Datenmengen oder sicherheitskritische Systeme kann dies zum Problem werden. Im Rahmen von [Hah13] wurden weitergehende Ansätze in Form verteilter Serverknoten mit Anbindung einer NoSQL-Datenbank prinzipiell evaluiert. Das dort entstandene DRUGSTORE-Framework könnte spannende Ansätze zur Erweiterung des hier beschriebenen WebCoObRA bieten.

Das im Rahmen der vorliegenden Arbeit entwickelte Tooling zur Modellierung von Webapplikationen bietet alle notwendigen Artefakte und deren Bearbeitungsmöglichkeiten an. Eine weitergehende Integration der einzelnen Schritte miteinander sowie die Erweiterung von prototypisch umgesetzten Anteilen, wie etwa dem vorhandenen Databinding-Mechanismus, um zusätzliche Funktionalitäten wären notwendig, um den vorgestellten Ansatz alltagstauglich zu machen.

Die vorgestellte Arbeit basiert zu großen Teilen auf dem Google Web Toolkit und dessen Crosscompiler-Funktionalitäten. Dies brachte im Rahmen der Entwicklung eine Vereinfachung der Codegenerierungs-Mechanismen mit sich. Gerade die komplexe Codegenerierung für Graphtransformationen musste so nur marginal angepasst werden. Um mit

modernen JavaScript-Bibliotheken, wie etwa AngularJS, besser integriert werden zu können, wäre eine komplette Portierung der Codegenerierung auf JavaScript denkbar.

Große Teile des hier vorgestellten Toolings, insbesondere alle Aspekte die Entwicklung der UI betreffend, basieren auf dem GWT-Designer der Firma Google. Technisch bildet der als Eclipse Projekt verfügbare *WindowBuilder* [Win] die Grundlage dieses Editors. Mit dem *WindowBuilder* ist es technisch möglich, weitere grafische Benutzerschnittstellen für Java zu entwickeln. Hier können vor allem Swing und SWT genannt werden. Aufgrund der gleichen technischen Basis ließe sich die im GWT-Designer implementierte Toolunterstützung für den Web Fujaba Process somit auf weitere Java-Benutzerschnittstellen übertragen. Interessant wäre in diesem Zusammenhang, die Anwendbarkeit des Web Fujaba Process - oder einer angepassten Form - auf traditionelle Java-Anwendungen zu untersuchen. Außerdem wäre eine Antwort auf die Frage, ob sich auch Desktopanwendungen anhand eines Workflows spezifizieren lassen, oder ob hier grundlegend andere Voraussetzungen gegeben sind, sehr aufschlussreich.

Literaturverzeichnis

[ADH+09] Aschenbrenner, Nina ; Dreyer, Jörn ; Hahn, Marcel ; Jubeh, Ruben ; Schneider, Christian ; Zündorf, Albert: Building Distributed Web Applications based on Model Versioning with CoObRA: an Experience Report. In: *Proc. 2009 Intl. Workshop on Comparison and Versioning of Software Models* IEEE, ACM, Mai 2009, 19-24

[ADJZ08] Aschenbrenner, Nina ; Dreyer, Jörn ; Jubeh, Ruben ; Zündorf, Albert: Fujaba goes Web 2.0. In: Assman, Uwe (Hrsg.) ; Johannes, Jendrik (Hrsg.) ; Zündorf, Albert (Hrsg.): *6th International Fujaba Days*. Dresden, Germany, 2008, 10-14

[Ant] *Apache Ant - Welcome*. `http://ant.apache.org/`. – zuletzt besucht: 06.03.2016

[Bay] *Prof. Dr. Bernhard Westfechtel*. `http://www.ai1.uni-bayreuth.de/de/team/Westfechtel_Bernhard/`. – zuletzt besucht: 23.01.2016

[BBG+05] Beydeda, Sami ; Book, Matthias ; Gruhn, Volker u. a.: *Model-driven software development*. Bd. 15. Springer, 2005

[BCFC06] Bozzon, Alessandro ; Comai, Sara ; Fraternali, Piero ; Carughi, Giovanni T.: Conceptual modeling and code generation for rich internet applications. In: *Proceedings of the 6th international conference on Web engineering* ACM, 2006, S. 353–360

[Bea] *BeanShell - Lightweight Scripting for Java*. `http://www.beanshell.org/`. – zuletzt besucht: 23.01.2016

[CC07] Comai, Sara ; Carughi, Giovanni T.: A Behavioral Model for Rich Internet Applications. In: Baresi, Luciano (Hrsg.) ; Fraternali, Piero (Hrsg.) ; Houben, Geert-Jan (Hrsg.): *ICWE* Bd. 4607, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–73596–0, 364-369

[CMCM11] Cocco, Luisanna ; Mannaro, Katiuscia ; Concas, Giulio ; Marchesi, Michele: Simulating Kanban and Scrum vs. Waterfall with System Dy-

namics. In: SILLITTI, Alberto (Hrsg.) ; HAZZAN, Orit (Hrsg.) ; BACHE, Emily (Hrsg.) ; ALBALADEJO, Xavier (Hrsg.): *XP* Bd. 77, Springer, 2011 (Lecture Notes in Business Information Processing). – ISBN 978–3–642–20676–4, 117-131

[Cor] *Welcome to CORBA Website*. `http://www.corba.org/`. – zuletzt besucht: 15.02.2016

[Dre] *Uwe Aßmann's Homepage*. `http://www1.inf.tu-dresden.de/~ua1/`. – zuletzt besucht: 23.01.2016

[DWL10] DAVID WANG, Alex M. ; LASSEN, Soren: *Google Wave Operational Transformation*. http://wave-protocol.googlecode.com/hg/whitepapers/operational-transform/operational-transform.html, 2010. – zuletzt besucht: 17.02.2016

[EGHZ12] EICKHOFF, Christoph ; GEIGER, Nina ; HAHN, Marcel ; ZÜNDORF, Albert: Developing enterprise web applications using the story driven modeling approach. In: *Current Trends in Web Engineering*. Springer, 2012, S. 196–210

[Erl08] ERL, Thomas: *Soa: principles of service design*. Bd. 1. Prentice Hall Upper Saddle River, 2008

[FD1] *Fujaba Days 2003 - Proceedings*. `http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/tr-ri-04-247.pdf`

[FD2] *Fujaba Days 2004 - Proceedings*. `http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/tr-ri-04-253.pdf`

[FD3] *Fujaba Days 2005 - Proceedings*. `http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/tr-ri-05-259.pdf`

[FD4] *Fujaba Days 2006 - Proceedings*. `http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/FujabaDays2006_Conference-Proceedings.pdf`

[FD5] *Fujaba Days 2007 - Proceedings*. `http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/FD07Proceedings.pdf`

[FD6] *Fujaba Days 2008 - Proceedings*. http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/tud-fi08-09.pdf

[FD7] *Fujaba Days 2009 - Proceedings*. http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/FDays2009-Proceedings.pdf

[FD8] *Fujaba Days 2011 - Proceedings*. http://www.fujaba.de/fileadmin/Informatik/Fujaba/Resources/Publications/Fujaba_Days/FujabaDays2011.pdf

[FRSF10] Fraternali, Piero ; Rossi, Gustavo ; Sánchez-Figueroa, Fernando: Rich internet applications. In: *Internet Computing, IEEE* 14 (2010), Nr. 3, S. 9–12

[G+05] Garrett, Jesse J. u. a.: Ajax: A new approach to web applications. (2005)

[GEH+09] Geiger, Nina ; Eickhoff, Christoph ; Hahn, Marcel ; Witzky, Ingo ; Zündorf, Albert: Future Web Application Development with Fujaba. In: Gorp, Pieter V. (Hrsg.): *Proceedings of the 7th International Fujaba Days*, 2009, S. 51–55

[Gei11] Geiger, Leif: *Fehlersuche im Modell - Modellbasiertes Testen und Debuggen*, University of Kassel, Diss., 2011. http://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:34-2011071338286/3/DissertationLeifGeiger.pdf

[Geo10] George, Tobias ; Kassel, Software Engineering Research G. (Hrsg.): *Implementierung des reaktiven Verhaltens von Web-Anwendungen mit Statecharts*. 2010

[GGH+10] Geiger, Nina ; George, Tobias ; Hahn, Marcel ; Jubeh, Ruben ; Zündorf, Albert: Using Actions Charts for Reactive Web Application Modelling. In: *Current Trends in Web Engineering*, Springer, 2010, 49–60

[Goo] *Google Docs - Online-Textverarbeitung und -Dokumentenüberarbeitungen für Unternehmen*. https://apps.google.com/products/docs/. – zuletzt besucht: 19.01.2016

[GPE] *Google Plugin for Eclipse | Google Developers*. https://developers.google.com/eclipse/. – zuletzt besucht: 06.03.2016

[Gra] *Graphiti - a Graphical Tooling Infrastructure*. http://www.eclipse.org/

graphiti/. – zuletzt besucht: 06.03.2016

[GSR05] Geiger, Leif ; Schneider, Christian ; Reckord, Carsten: Template- and modelbased code generation for MDA-Tools. In: *Proc. of the 3rd International Fujaba Days* Citeseer, 2005, S. 57–62

[GW06] Giese, Holger ; Wagner, Robert: Incremental Model Synchronization with Triple Graph Grammars. In: Nierstrasz, Oscar (Hrsg.) ; Whittle, John (Hrsg.) ; Harel, David (Hrsg.) ; Reggio, Gianna (Hrsg.): *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genova, Italy* Bd. 4199, Springer Verlag, 10 2006 (Lecture Notes in Computer Science (LNCS)), S. 543–557

[GWTa] *[GWT]*. http://www.gwtproject.org/. – zuletzt besucht: 06.03.2016

[GWTb] *GWT Designer GUI Builder | Eclipse Plugins, Bundles and Products - Eclipse Marketplace*. https://marketplace.eclipse.org/content/gwt-designer-gui-builder. – zuletzt besucht: 06.03.2016

[GWTc] *[GWT] Documentation - Widget Gallery*. http://www.gwtproject.org/doc/latest/RefWidgetGallery.html. – zuletzt besucht: 12.01.2016

[GZ05a] Geiger, Leif ; Zündorf, Albert: Statechart modeling with fujaba. In: *Electronic Notes in Theoretical Computer Science* 127 (2005), Nr. 1, S. 37–49

[GZ05b] Geiger, Leif ; Zündorf, Albert: Story driven testing - SDT. In: *SCESM '05: Proceedings of the fourth international workshop on Scenarios and state machines: models, algorithms and tools*. New York, NY, USA : ACM, May 2005. – ISBN 1–58113–963–2, 1–6

[GZ06] Geiger, Leif ; Zündorf, Albert: Developing Tools with Fujaba XProM. In: Lämmel, Ralf (Hrsg.) ; Saraiva, João (Hrsg.) ; Visser, Joost (Hrsg.): *GTTSE* Bd. 4143, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 3–540–45778–X, 344-356

[Hah13] Hahn, Marcel: *Entwicklung des DRUGSTORE Framework*, Uni Kassel, Diplomarbeit, 2013

[HPI] *Prof. Dr. Holger Giese - Hasso-Plattner-Institut*. http://hpi.de/giese/personen/prof-dr-holger-giese.html. – zuletzt besucht: 23.01.2016

[HSSJS08] Hoyer, Volker ; Stanoesvka-Slabeva, Katarina ; Janner, Till ; Schroth, Christoph: Enterprise mashups: Design principles towards the

long tail of user needs. In: *Services Computing, 2008. SCC'08. IEEE International Conference on* Bd. 2 IEEE, 2008, S. 601–602

[htm] *HTML5*. https://www.w3.org/TR/html5/. – zuletzt besucht: 10.02.2016

[IFM] *IFML: The Interaction Flow Modeling Language | The OMG standard for front-end design*. http://www.ifml.org/. – zuletzt besucht: 19.02.2016

[JBR99] Jacobsen, Ivar ; Booch, Grady ; Rumbaugh, James: *The Unified Software Development Process: The complete guide to the Unified Process from the origial designers*. 1. Addison Wesley, 1999

[JRE] *[GWT] Documentation - JRE Emulation*. http://www.gwtproject.org/doc/latest/RefJreEmulation.html. – zuletzt besucht: 09.02.2016

[KNNZ00] Köhler, Hans J. ; Nickel, Ulrich ; Niere, Jörg ; Zündorf, Albert: Integrating UML diagrams for production control systems. In: *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. New York, NY, USA : ACM, 2000. – ISBN 1–58113–206–9, S. 241–251

[KPZM09] Koch, Nora ; Pigerl, Matthias ; Zhang, Gefei ; Morozova, Tatiana: Patterns for the Model-Based Development of RIAs. In: Gaedke, Martin (Hrsg.) ; Grossniklaus, Michael (Hrsg.) ; Díaz, Oscar (Hrsg.): *ICWE* Bd. 5648, Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–02817–5, 283-291

[Mav] *Maven - Welcome to Apache Maven*. https://maven.apache.org/. – zuletzt besucht: 06.03.2016

[Min] *Lego Mindstorms Homepage*. http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com. – zuletzt besucht: 06.01.2016

[Mod15] *QVT - Query View Transformation - Spezifikation 1.2*. http://www.omg.org/spec/QVT/1.2/PDF/. Version: 2015

[NK02] Nora Koch, Andreas K.: The Expressive Power of UML-based Web Engineering, 2002, S. 105–119

[NL04] Newcomer, Eric ; Lomow, Greg: *Understanding SOA with web services (independent technology guides)*. Addison-Wesley Professional, 2004

[NZJ13] Norbisrath, Ulrich ; Zündorf, Albert ; Jubeh, Ruben: *Story Dri-*

ven Modeling. 2013. – ISBN 9781483949253

[Obj10] Object Management Group: *OMG Unified Modeling LanguageTM (OMG UML), Superstructure*. http://www.omg.org/spec/UML/2.3/. Version: 2.3, Mai 2010

[OOH] *OOH4RIA*. http://suma2.dlsi.ua.es/ooh4ria/. – zuletzt besucht: 18.02.2016

[OOW] *OOWS 2.0: A Model-driven Web Engineering Method for the Development of Web 2.0 Applications*. http://de.slideshare.net/fravalgi/oows-20-a-model-driven-web-engineering-method-for-the-production-of-web-20-app – zuletzt besucht: 19.02.2016

[OP01] Oscar Pastor, Joan F. Silvia Abrahao A. Silvia Abrahao: An Object-Oriented Approach to Automate Web Applications Development. In: *Electronic Commerce and Web Technologies*, 2001. – ISBN 978–3–540–44700–9, 16–28

[OPR95] Otte, Randy ; Patrick, Paul ; Roy, Mark: *Understanding CORBA (Common Object Request Broker Architecture)*. Prentice-Hall, Inc., 1995

[PDMG08a] Pérez, Sandy ; Díaz, Oscar ; Meliá, Santiago ; Gómez, Jaime: Facing Interaction-Rich RIAs: The Orchestration Model. In: Schwabe, Daniel (Hrsg.) ; Curbera, Francisco (Hrsg.) ; Dantzig, Paul (Hrsg.): *ICWE*, IEEE, 2008. – ISBN 978–0–7695–3261–5, 24-37

[PDMG08b] Pérez, Sandy ; Díaz, Oscar ; Meliá, Santiago ; Gómez, Jaime: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: Schwabe, Daniel (Hrsg.) ; Curbera, Francisco (Hrsg.) ; Dantzig, Paul (Hrsg.): *ICWE*, IEEE, 2008. – ISBN 978–0–7695–3261–5, 13-23

[PFP03] Pastor, Oscar ; Fons, Joan ; Pelechano, Vicente: Oows: A method to develop web applications from web-oriented conceptual models. In: *International Workshop on Web Oriented Software Technology (IWWOST)*, 2003, S. 65–70

[PGIP01] Pastor, Oscar ; Gómez, Jaime ; Insfrán, Emilio ; Pelechano, Vicente: The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. In: *Information Systems* 26 (2001), Nr. 7, S. 507–534

[PLMC+08] Preciado, Juan C. ; Linaje, Marino ; Morales-Chaparro, Rober

; SANCHEZ-FIGUEROA, Fernando ; ZHANG, Gefei ; KROISS, Christian ; KOCH, Nora: Designing rich internet applications combining uwe and rux-method. In: *Web Engineering, 2008. ICWE'08. Eighth International Conference on* IEEE, 2008, S. 148–154

[Rum13] RUMPE, Bernhard: *Modellierung mit UML: Sprache, Konzepte und Methodik*. Springer-Verlag, 2013

[Sch] *Heinz Nixdorf Institut: Modelltransformationen*. `https://www.hni.uni-paderborn.de/swt/forschung/modelltransformationen/`. – zuletzt besucht: 23.01.2016

[Sch07] SCHNEIDER, Christian: *CoObRA: Eine Plattform zur Verteilung und Replikation komplexer Objektstrukturen mit optimistischen Sperrkonzepten*, Diss., 2007. `http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007121319874`

[SDM] *GitHub - fujaba/SDMLib*. `http://www.sdmlib.org`. – zuletzt besucht: 06.03.2016

[SE] *Software Engineering Research Group Kassel | Department of Computer Science and Electrical Engineering*. `http://seblog.cs.uni-kassel.de/`. – zuletzt besucht: 23.01.2016

[SR95] SCHWABE, Daniel ; ROSSI, Gustavo: The object-oriented hypermedia design model. In: *Commun. ACM* 38 (1995), Nr. 8, S. 45–46. `http://dx.doi.org/http://doi.acm.org/10.1145/208344.208354`. – DOI http://doi.acm.org/10.1145/208344.208354. – ISSN 0001–0782

[SZN04] SCHNEIDER, Christian ; ZÜNDORF, Albert ; NIERE, Jörg: CoObRA - a small step for development tools to collaborative environments. In: *Workshop on Directions in Software Engineering Environments in 26th international conference on software engineering*. Edinburgh, Scotland, UK, May 2004

[TUD] *Fachgebiet Echtzeitsysteme: ES*. `http://www.es.tu-darmstadt.de/`. – zuletzt besucht: 23.01.2016

[Vel] *Apache Velocity Site - The Apache Velocity Project*. `http://velocity.apache.org`. – zuletzt besucht: 12.01.2016

[Vic] *University of Victoria - Health Information Science - Adjunct Faculty - Jens Weber-Jahnke - University of Victoria*. `https://www.uvic.ca/hsd/hinf/people/adjunct/weber/index.php`. – zuletzt besucht: 23.01.2016

[Weba] *Rapid mobile app and web application development plattform*. http://www.webratio.com/site/content/en/home. – zuletzt besucht: 19.02.2016

[Webb] *Welcome to webml.org*. http://www.webml.org/webml/page1.do. – zuletzt besucht: 19.02.2016

[Win] *WindowBuilder*. https://eclipse.org/windowbuilder/. – zuletzt besucht: 22.02.2016

[wwwa] *Microsoft popfly in der Wikipedia*. http://en.wikipedia.org/wiki/Microsoft_Popfly. – zuletzt besucht: 06.03.2016

[wwwb] *Was wurde aus iGoogle?* https://support.google.com/websearch/answer/2664197?hl=de. – zuletzt besucht: 31.07.2014

[wwwc] *Yahoo Pipes*. http://pipes.yahoo.com/pipes/. – zuletzt besucht: 31.07.2014

[wwwd] *FAST EU Project*. http://cordis.europa.eu/project/rcn/85546_en.html. – zuletzt besucht: 06.01.2016

[wwwe] *Fujaba Tool Suite*. http://www.fujaba.de/. – zuletzt besucht: 06.01.2016

[wwwf] *Fujaba Project update site at Kassel University*. https://seblog.cs.uni-kassel.de/projects/fujaba/. – zuletzt besucht: 06.01.2016

[wwwg] *GitHub - isomorphic-software/smartgwt: Smart GWT is a GWT-based framework that allows you to not only utilize its comprehensive widget library for your application UI, but also tie these widgets in with your server-side for data management.* https://github.com/isomorphic-software/smartgwt. – zuletzt besucht: 06.03.2016

[xte] *XText - Language Development Made Easy!* https://www.eclipse.org/Xtext/. – zuletzt besucht: 06.03.2016

[Zün01] Zündorf, A.: *Rigorous Object Oriented Software Development*. Habilitation Thesis, University of Paderborn, 2001