ulm university universität
uulm

Institut für Datenbanken und Informationssysteme
Universität Ulm

James-Franck-Ring
89069 Ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Implementierung einer generischen und web-basierten Entwicklungsumgebung von Prozessmodellen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Raphael Herfort
raphael.herfort@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
David Knuplesch

15. Februar 2013

Fassung 15. Februar 2013

Satz: PDF-LaTeX 2$_\varepsilon$

Zusammenfassung

Unternehmen müssen sich heutzutage verstärkt der Problematik stellen, dass sie schnell auf neue geänderte inner- oder außerbetriebliche Begebenheiten reagieren müssen. Die Komplexität der Vernetzung zwischen den einzelnen Unternehmen und ihren Kunden lassen stark zusammenhängenden Geschäftsprozesse entstehen. Die Prozesse sind von schnellen unkontrollierten Änderungen der beteiligten Parteien in ihrem Ablauf durch Inkonsistenzen und fehlerhaftes Verhalten bedroht. Neuartige Prozessmodellierungssprachen -techniken und Analyseverfahren könnten helfen diese komplexen geschäftlichen Strukturen abzubilden. Dies ist unabdingbar um Inkonsistenzen und fehlerhaftes Verhalten von Prozessen frühzeitig erkennen und korrigieren zu können. Zur Validation solcher neuartiger Ansätze sind prototypische Implementierungen jedoch unentbehrlich. Für spätere Forschungsprototypen wird eine Grundlage benötigt, auf der weiter aufbauend die neuartigen Prototypen entwickelt und erweitert werden können.

Die Arbeit beschreibt die Entwicklung der generischen und web-basierten Modellierungsumgebung Chameleon. Das Hauptaugenmerk der Entwicklungsumgebung liegt dabei auf der Definition von Geschäftsprozessen und verwandten Kernpunkten. Chameleon unterstützt die Generierung und Partizipation unterschiedlicher Meta-Modelle, und ist somit nicht auf eine bestimmte Modellierungssprache festgelegt

Im Gegensatz zu den reinen Zeichentools bietet diese Modellierungsumgebung jedoch auch generische Schnittstellen, die auf Basis des jeweiligen Meta-Modells den einfachen Zugriff auf Elemente eines Modells ermöglicht. Auf Basis dieser Schnittstellen können leicht Plug-Ins zur Korrektheitsprüfung und Modellverifikation für die einzelnen Meta-Modelle erstellt werden. Ebenso leicht können Plug-Ins zur Modelltransformation von einem Meta-Modell in ein anderes realisiert werden.

Ein Anwendungsgebiet der Modellierungsumgebung Chameleon wird daher die Realisierung von Forschungsprototypen für die oben genannten neuartigen Prozessmodellierungssprachen und von passenden Verifikationswerkzeugen sein.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In der heutigen Geschäftswelt hängt der ökonomische Erfolg eines Unternehmens von der Möglichkeit ab auf Änderungen in seinem Umfeld schnell und flexibel zu reagieren [1]. Gründe für diese Veränderungen können vielfältig sein; beispielsweise die Einführung eines neuen Gesetzes oder gewisse Abweichungen im Umgang mit den Kunden [2]. Kollaborative Geschäftsprozesse stellen Unternehmen vermehrt vor das Problem von Inkonsistenzen in ihrem täglichen Geschäftsablauf. Es werden Prototypen benötigt, die neue innovative Prozessmodellierungssprachen implementieren, um die rasant wachsende Problematik zu bewältigen.

Es gibt eine Vielzahl von Programmen zur Modellierung verschiedenartiger Problemstellungen. Doch viele dieser Systeme sind konkrete Umsetzungen bestimmter Aufgabestellungen, wie die reine Prozessmodellierung oder die Erstellung eines Organigramm. Auch stellen die meisten nur eine begrenzte Anzahl von Modellierungssprachen zur Verfügung, die meist noch fest in das jeweilige System eingebettet sind, wie es beim Aristaflow Template Editor der Fall ist.

Umsetzungen, die eine generische Grundstruktur aufweisen, sind hingegen meist nur reine Zeichentools, wie das Modellierungswerkzeug Visio. Sie bieten oft eine große Palette an Modellierungssprachen, auf die zurückgegriffen werden kann, und sind weiterhin größtenteils leicht um neue Sprachen erweiterbar.

Ein großer Nachteil dieser Tools besteht aber darin, dass sie keinerlei Methoden bereitstellen, um auf den angefertigten Modellen weiterführende Aufgaben realisieren zu können. So sind Tests zur Verifikation der jeweiligen Modelle nur schwer realisierbar; beispielsweise ein Test zu Vermeidung fehlerhafter Datenflüsse oder Deadlocks. Die bekannten Programme bieten oft nur einen einfachen XML-Export an. Dabei wird so nur eine grafische Repräsentation der modellierten Modelle für den weiteren Gebrauch bereitgestellt. Die Beschreibung der grafischen Elemente ist dabei allerdings nicht ausreichend um die Semantik des zugrunde liegenden Modells ab zu leiten. Diese eindeutige Interpretation ist allerdings Voraussetzung für weiterführende Tests, wie sie oben benannt sind.
Darüber hinaus sind viele dieser Programme reine Desktop-Anwendungen. Die Struktur von heutigen Organisationen, wie Universitäten oder Unternehmen, macht es gleichwohl notwendig, dass Programme zur Modellierung verteilt auf verschiedenen Endgeräten und von verschiedenen Standorten aus abrufbar sind.
Für die neuartigen Forschungsprototypen ist es wünschenswert die angesprochenen Aspekte, der Flexibilität, der Erweiterbarkeit, der Möglichkeit zu weiterführenden Tests und der ständigen und verteilten Erreichbarkeit einer Webplattform in einer Umgebung zu vereinen.

1.2 Zielsetzung

Diese Arbeit wird im Rahmen des Forschungsprojektes C^{3}Pro verfasst. Sie soll dazu dienen ein Werkzeug an die Hand zu geben, welches bezüglich der Analyse von Flexibilität und Adaptivität von gemeinschaftlichen Geschäftsprozessen als Grundlage für weitere Forschungsprototypen dienen soll [3, 4]. Die Aufgabe in dieser Arbeit ist eine Webplattform zu realisieren, die eine Modellierung von Prozessen in verschiedenen Modellierungssprachen, beziehungsweise auf Grundlage verschiedener Meta-Modellen, ermöglicht. Diese Plattform soll dabei sowohl das Erstellen und Verwalten von Prozess-Meta-Modellen als auch das von den zugehörigen Prozessmodellen unterstützen. Zusätzlich soll die Plattform Schnittstellen bereitstellen, welche das einfache Erstellen von Plugins für Korrektheitsprüfung sowie Modellverifikation und -transformation erlauben.
Ein weiteres Augenmerk liegt darin, dass ein Mehrbenutzersystem geschaffen wird, in dem durch die Verteilung von Kompetenzen eine leichtere Handhabung des Systems ermöglicht wird. So soll dem Modellierer die Möglichkeit geboten werden, frei mit den, vom Meta-Model angebotenen graphischen Elementen, zu modellieren. Dem Verwalter von Meta-Modellen

wird eine Reihe von Schnittstellen zur Verfügung gestellt, über die er seine Meta-Modelle leicht in das System importieren und später verwalten kann. Der Administrator zuletzt ist mit der Verwaltung der einzelnen Benutzergruppen betraut.
Bei der Umsetzung sollen gängige Web-Technologien zum Einsatz kommen, welche einen vielversprechenden Ansatz zur Realisierung organisationsweiter und -übergreifender wissenschaftlicher Anwendungen darstellen [5].

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich wie folgt: In Kapitel 2 werden die wichtigsten Grundlagen beschrieben und definiert. Dies dient einem besserem Verständnis über das fachspezifische Vokabular als auch einer einheitlichen Bezeichnung für verschiedene Teilaspekte.
In Kapitel 3 wird das Plugin-System für die Meta-Meta-Modelle vorgestellt. Hierfür wird zuerst eine Analyse der Anforderungen an das spätere System gemacht um anschließend die Ergebnisse in die Integration von Meta-Modelle in der Modellierungsumgebung einfließen zu lassen.
In Kapitel 4 wird die genaue Systemarchitektur der zu implementierenden Anwendung vorgestellt. Dies geht von dem grundlegenden Aufbau über das Verhalten in konkreten Situationen bis zu ersten Beschreibungen der GUI des zu implementierenden Systems.
Im Kapitel 5 werden vereinzelte Implementationsmethoden herausgegriffen und an der konkreten Implementierung der entsprechenden Komponente erläutert.
Zu guter Letzt wird in Kapitel 6 eine Zusammenfassung dieser Arbeit und der in ihr angesprochenen Themen gegeben und ein Ausblick auf mögliche Folgeprojekte gewagt.

2 Grundlagen

In diesem Kapitel wird zunächst auf das C^{3}Pro Projekt näher eingegangen, in dessen Rahmen diese Arbeit verfasst wird. Weiter führen wir wichtige und grundlegende Begriffe und Definitionen bezüglich Modelle und Prozesse ein. Alle Beschreibungen beziehungsweise Illustrationen werden dabei beispielhaft in BPMN (Business Process Model and Notation) dargestellt.

2.1 Das C^{3}Pro Projekt

Zuerst betrachten wir das C^{3}Pro Projekt ein wenig genauer, in dessen Rahmen diese Arbeit verfasst wird. C^{3}Pro steht für *„Change and Complaince for Collaborative Processes“*. Es ist ein kooperatives Forschungsprojekt der Universitäten Ulm und Wien. Der Schwerpunkt dieser Forschungsinitiative bildet die Unterstützung von Flexibilität und Korrektheit von kollaborativen Geschäftsprozessen. Ein Beispiel für einen kollaborativen Geschäftsprozess ist in Abbildung 2.1 dargestellt. Während Änderungen an Geschäftsprozessen und die Regelsteuerung dieser in einem innerbetrieblichen Rahmen gründlich diskutiert wurde, bleibt die offene Frage wie sich die beiden Punkte auf betriebsübergreifende Prozesse auswirken [6]. In vielen Anwendungsdomänen von Informationssystemen nimmt die Konformität von Geschäftsprozessen eine zunehmend wichtigere Rolle ein [7]. Im Rahmen von C^{3}Pro klassifiziert die Forschungsgruppe Geschäftsprozessmodelle in drei Ebenen der Korrektheit, da die Definition und Änderung von Geschäftsprozessmodellen nicht in einer beliebigen Weise vorgenommen werden kann [6].

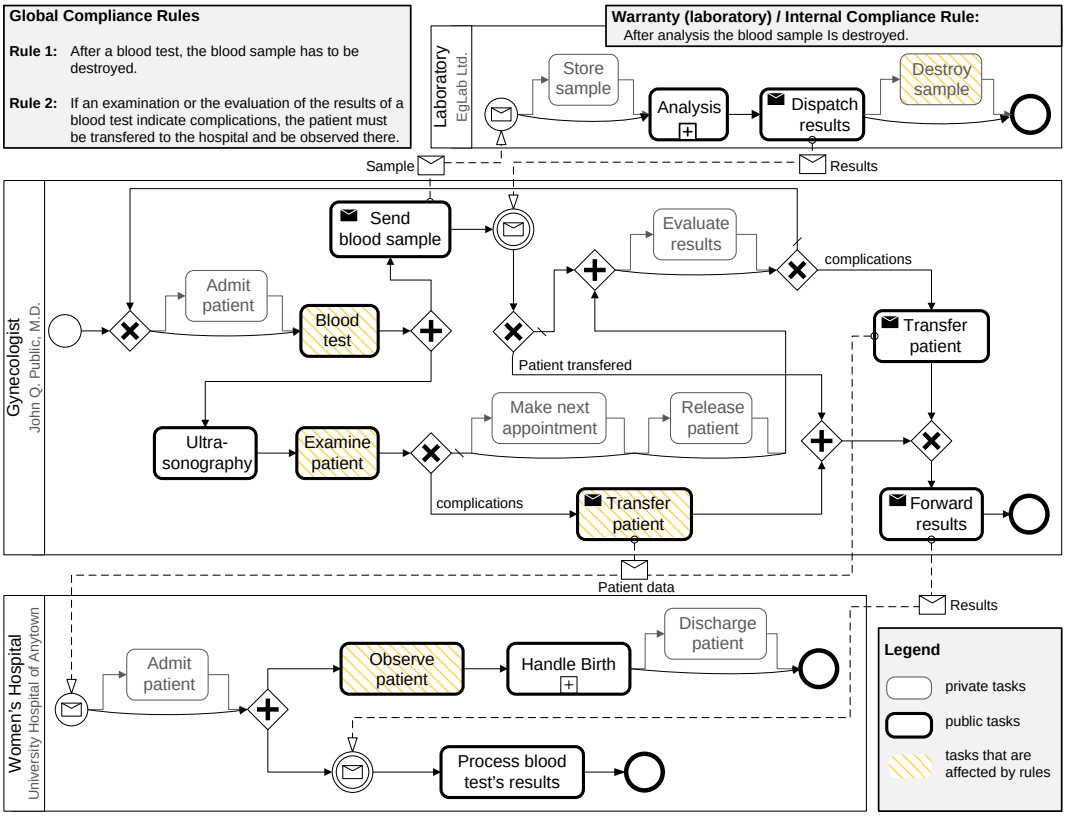

Abbildung 2.1: Beispiel für ein kollaborativen Geschäftsprozess aus [6]

In Abbildung 2.2 werden die verschiedenen Ebenen der Korrektheit eines Geschäftsprozessmodells veranschaulicht.

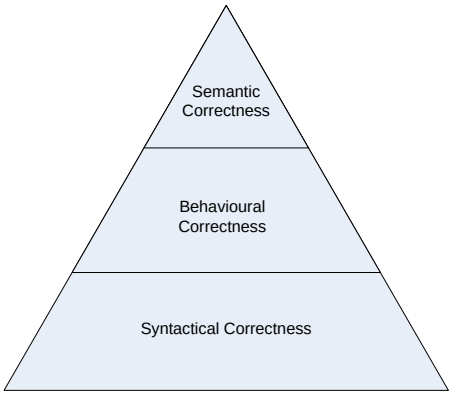

Abbildung 2.2: Pyramide der Korrektheitsebenen eines Geschäftsprozessmodells aus [6]

Die ***syntaktische Korrektheit*** bezieht sich auf die richtige Verwendung und Anordnung der einzelnen Elemente des zugrunde liegenden Meta-Modells. Die ***Verhaltenskorrektheit*** hingegen verlangt von einem Prozessmodell, dass es ausführbar ist, was die Eigenschaften wie das Nichtvorhandensein von Deadlocks oder Lifelocks beinhaltet. Weiter wird ein korrekter Datenfluss vorausgesetzt; beispielsweise müssen Datenelemente erst gefüllt werden, bevor sie das erste Mal gelesen werden. Die ***semantische Korrektheit*** setzt letztlich voraus, dass auf dem Prozessmodell basierende Prozessausführungen die auferlegten Complianceregeln, wie sie zum Beispiel von Gesetzen oder Standards und Richtlinien kommen, erfüllen [6]. In diesem Zusammenhang wird auch von Business Process Compliance gesprochen [6, 8].

2.2 Das elementare Prozessmodell

Um zu verstehen was ein Prozessmodell ist, wollen wir zunächst uns veranschaulichen, wie der zu Grunde liegende Geschäftsprozess definiert wird. Einfach gesagt, kann ein Geschäftsprozess als eine Reihe von Aktivitäten angesehen werden, die Beteiligte als Reaktion auf ein auslösendes Event verrichten. Weiter wird vom Prozess festgelegt wie eine bestimmte Aktivität auszuführen ist [9]. Weitere Erklärungen, was einen Prozess beziehungsweise sein Modell auszeichnen, geben Thomas H. Davenport und Herbert Stachowiak. Nach Thomas H. Davenport ist ein Prozess eine festgelegte Reihenfolge von Aktivitäten zeit- und ortsübergreifend mit einem Start- und End-Event [10]. Das Prozessmodell ist nun die Umsetzung des realen Geschäftsprozess in einer Beschreibungssprache wie BPMN; beispielhaft wird in Abbildung 2.3 ein Prozessmodell illustriert. Die Aufgabe eines Modelles kann dabei sein den Realweltanteil eines Prozesses auf eine verständliche und deskriptive Weise zu veranschaulichen. Nach Herbert Stachowiak zeichnet sich ein Prozessmodell dadurch aus, dass es entweder eine tatsächliche Abbildung eines Originals oder dessen Verkürzung ist. Mit Pragmatismus wird daneben das zeitlich begrenzte Ersetzen eines Originals mit dem Modell bezeichnet [11].
Beim Prozessmodell spielen weiter auch ***Syntax*** und ***Semantik*** eine wichtige Rolle. Die Syntax ist kohärent mit dem korrekten formalen Aufbau des Modells bezüglich der Elemente der verwendeten Modellierungssprache. Mit Semantik wird weitläufig die Korrektheit eines Modells bezüglich seines Realweltausschnittes bezeichnet. Ein Prozessmodell kann man auf verschiedene Art und Weise betrachten. Zum einen gibt es die Kontrollfluss Perspektive. Sie beschreibt die Aktivitäten und ihre Ausführungsreihenfolge durch verschiede-

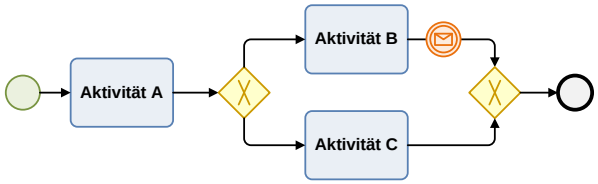

Abbildung 2.3: Beispiel für ein Prozessmodell

ne Konstrukte, welche den Fluss der Ausführung kontrollieren, wie zum Beispiel Sequenz, Auswahl, Parallelität oder Synchronisierung. Die Daten Perspektive hingegen beinhaltet Geschäfts- und Prozessdaten der Kontrollperspektive, wie zum Beispiel Geschäftsdokumente und andere Objekte, die zwischen den Aktivitäten, ausgetauscht werden [12].
Im Folgenden betrachten wir nun die grundlegenden Elemente eines Prozessmodells (siehe Abbildung 2.4).

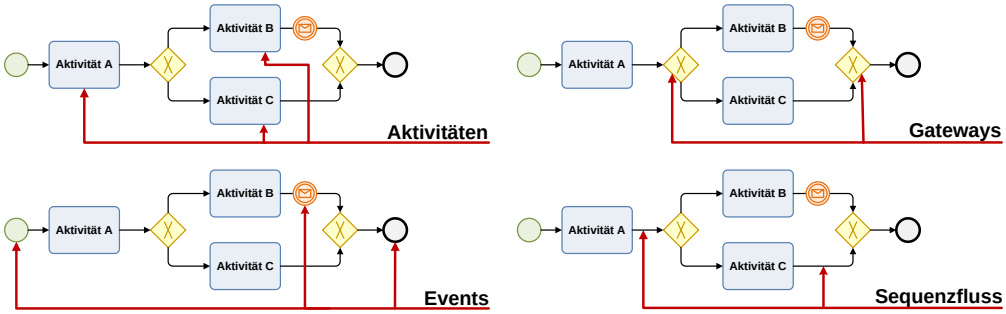

Abbildung 2.4: Beispiele für in einem Prozessmodell enthaltenen Elemente

Aktivitäten können gleichzeitig atomare Arbeitsschritte im Prozess sein oder komplexere Aufgabenstellungen beinhalten. Ist dies der Fall wird eine Aktivität auch als *Subprozess* bezeichnet. *Subprozesse* werden verwendet um die Komplexität eines Prozessmodells zu minimieren, da der Detailgrad durch das Ausblenden von Aufgaben reduziert wird.

Gateways unterscheiden sich grob in zwei Gruppen. Der *Split* stellt den Zeitpunkt von Entscheidungen im Prozess dar. Der *Join* hingegen bündelt alle Kontrollflüsse wiederum in einen.

Events bilden Ereignisse, die während dem Ablauf des Prozesses eintreten können, ab. Eine besondere Rolle spielen dabei die *Start-* und *End-Events*. Sie stoßen ein Prozess an oder beenden ihn. Das *Start-Event* zeichnet sich durch die Eigenschaft aus, dass keine Sequenzflüsse in das Event münden. Das *End-Event* besitzt dagegen nur hineinlaufende Sequenzflüsse. Sie können daher auch als Quelle und Senke von Sequenzflüssen bezeichnet werden.

Sequenzflüsse verbinden alle oben genannten Elemente miteinander. Sie stellen dabei die Ablaufkontrolle dar, welche die richtige Reihenfolge der Ausführung von Aktivitäten gewährleistet.

3
Meta-Meta-Modell

Dieses Kapitel beschäftigt sich mit der Frage, wie ein System realisiert werden kann, das in der Lage ist beliebig verschiedene Prozess-Meta-Modelle zu integrieren, um auf Grundlage derer Prozesse zu modellieren. Um ein besseres Verständnis für die Problematik zu bekommen betrachten wir zuerst das Meta-Modellhierachieschema der Unified Modeling Language (UML) [13] um eine Basis für die Systemanforderungen zu erarbeiten, denen unsere Umgebung gerecht werden muss. Das Ziel ist es, ein Meta-Meta-Modell zu erstellen, das die komplexe Aufgabe meistert. Ein weiteres Thema wird die Realisierung der Speicherung der modellierten Prozesse sein, wofür im gleichen Maße eine spezielle Modellierungssprache entwickelt wird, die grafische Elemente auf eine abstrakte und generische Weise darstellt und speichert.

3.1 Anforderungsanalyse

Für die Analyse aller Anforderungen, die an eine Metasprache gestellt werden, wollen wir zuerst die Modellierungsprache UML betrachten. Sie ist modular aufgebaut und deshalb geeignetes Vergleichsobjekt für unsere generische Meta-Metasprache.
UML wird als Vereinigung der Ansätze von Boch [14], Rumbaugh mit OMT [15] und Jacobson mit OOSE [16] seit 1990 entwickelt. Sie stellt eine prozess-unabhängige und objektorientierte Universalsprache dar. UML, mit seiner großen Ausdrucksmächtigkeit, hat einen breiten Anwendungsbereich, welcher ein umfangreiches Spektrum von Anwendungen abdeckt, die im Gebiet der informationellen Modellierung angesiedelt sind. Dies verlangt, dass

die Sprache eine gewisse Modularität besitzt und die Möglichkeit bietet nur diejenigen Teile der Sprache zu gebrauchen, welche auch nur tatsächlich im gegenwärtigen Arbeitsbereich benötigt werden. Die große Flexibilität, sich an den jeweiligen Anwendungsbereich anzupassen, verhalf ihr zu einem weit verbreiteten Gebrauch bei Analyse- und Entwurfsentscheidungen. UML bietet eine große Palette an Modelltypen und ihren Diagrammelementen an. Wir betrachten im folgenden eine Zusammenstellung dieser angebotenen Strukturen, um einen besseren Eindruck über die Ausdrucksmächtigkeit von UML zu bekommen.

So können mit Hilfe von UML *Klassendiagramme* erstellt werden. Dieser Typ von Diagramm wird zur Modellierung statischer Strukturen eines Systems verwendet. Ein Modell hat dabei im allgemeinen drei Bestandteile. Klassen bilden die statischen Komponenten einer Systemarchitektur ab. Während *Assoziationen* die Beziehungen der beteiligten Objekte untereinander repräsentieren. Weiter gibt es Spezialisierungs- und Generalisierungsbeziehungen zwischen einzelnen Klassen [13, 17, 18].

Will man sich hingegen einen Überblick über die wesentlichen Systemfunktionen verschaffen, bietet UML die *Anwendungsfalldiagramme* an. Sie beschreiben sämtliche angebotenen Funktionen aus Sicht des Benutzers. Der Informationsaustausch in einem System ist ein wichtiger Punkt. *Sequenzdiagramme* bilden diesen Austausch zwischen einzelnen Kommunikationspartnern ab. Ein wichtiger Aspekt ist dabei die korrekte zeitliche Abfolge. *Aktivitätsdiagramme* stellen zuletzt eine Kontrollinstanz dar. Sie haben in UML die Aufgabe den Kontroll- und Datenfluss aller Aktionen abzubilden [13, 17, 18].

Bei der allgemeinen Modellierung unterscheiden wir grundsätzlich zwischen Meta-Modellen und Modellen. Dabei kann ein Modell, welches eine Instanz eines Meta-Modells repräsentiert, wiederum auf eine rekursive Weise als Meta-Modell eines weiteren Modell fungieren [13]. Ist dies der Fall, so wird das gemeinsame Muttermodell auch als Meta-Meta-Modell bezeichnet. Typischerweise enthält ein Modell Elemente. Diese werden durch die Instanziierung der Meta-Modellelemente erzeugt. Die Rolle eines Meta-Modells ist die, die Semantik wie Modellelemente in einem Modell instantiiert werden, zu definieren [13]. In Abbildung 3.1 wird ein Beispiel einer Meta-Modell-Hierarchie mit vier Ebenen illustriert. Sie zeigt wie die einzelnen Ebenen miteinander korrelieren.

Ein weiterer Bestandteil von UML sind die eingebetteten Kontrollbedingungen. Die Bedingungen werden in UML auf zwei Ebenen klassifiziert, die ***Abstract Syntax Compliance*** und die ***Concrete Syntax Compliance***. Erst genannte ist für die Meta-Klassen und deren strukturellen Beziehungen untereinander verantwortlich. Während die Concrete Syn-

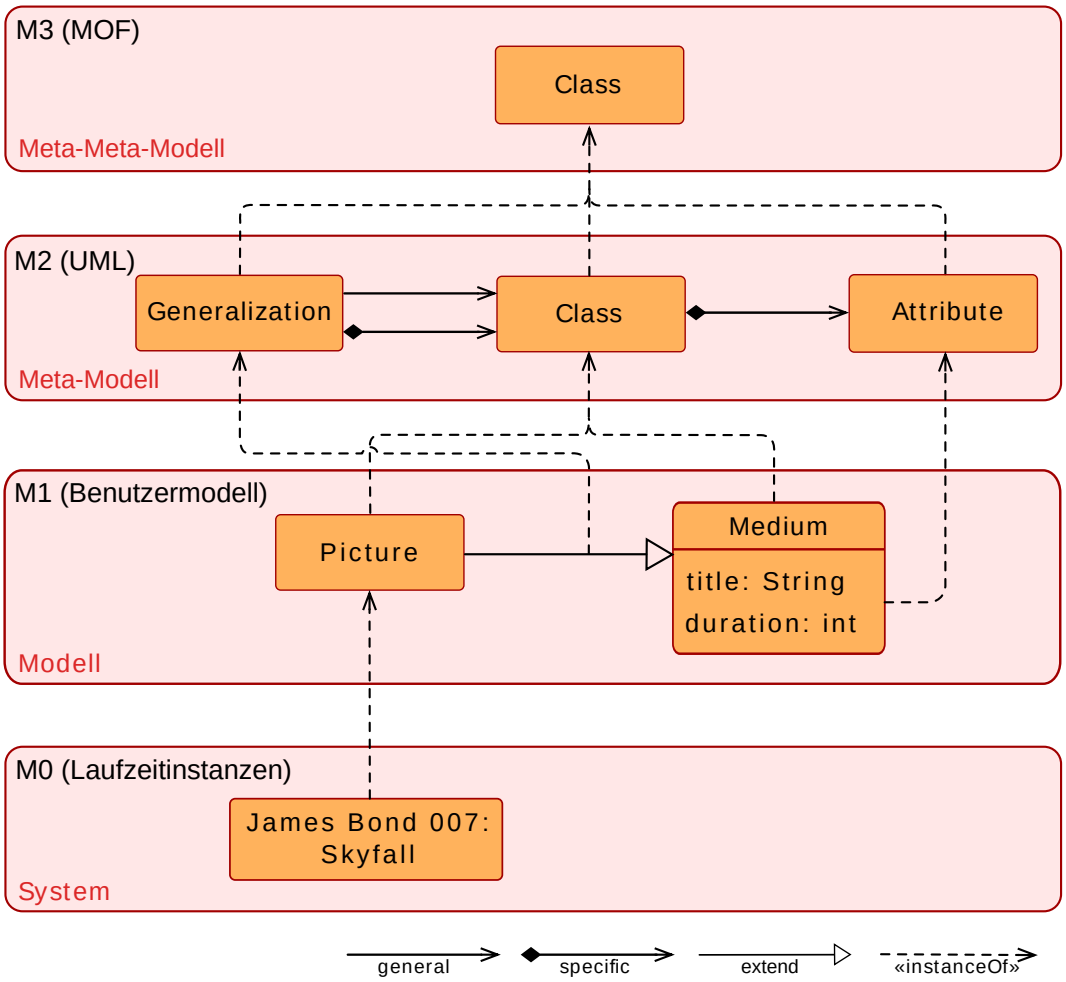

Abbildung 3.1: UML Referenz-Meta-Modellhierarchie basierend auf [13]

tax Compliance die konkrete Regelung zur Notation für Elemente in den einzelnen Meta-Modellen bereitstellt [13].
Konzentrieren wir uns nun auf unsere Meta-Metasprache, die wir für Chameleon benötigen. Dafür postulieren wir zunächst eine Liste mit gewünschten Eigenschaften, die wir von UML entlehnen. Die Liste dient uns im späteren Verlauf dazu, eine XML-basierte Meta-Metasprache zu entwickeln.

- Die vollständige Beschreibung der Elemente
- Eine Möglichkeit zur Bildung von Elementgruppen
- Die Elemente sollen eindeutig im Meta-Modell identifizierbar sein

Die Vollständigkeit einer Beschreibungssprache ist eine essentielle Eigenschaft, ohne die nicht die Gesamtheit einer Meta-Modellierungssprache gewährleistet wird. Die Elemente eine Meta-Modells müssen in Gruppen klassifizierbar sein, da so eine erste syntaktische Abgrenzung erreichbar wird. Ein Beispiel, wie die Gruppierung verstanden wird, ist in Abbildung 3.2 dargestellt. Eine Gruppenbildung ermöglicht weiter eine hierarchische Ordnung in der, durch Vererbung, Gruppeneigenschaften gekapselt weitergegeben werden können.

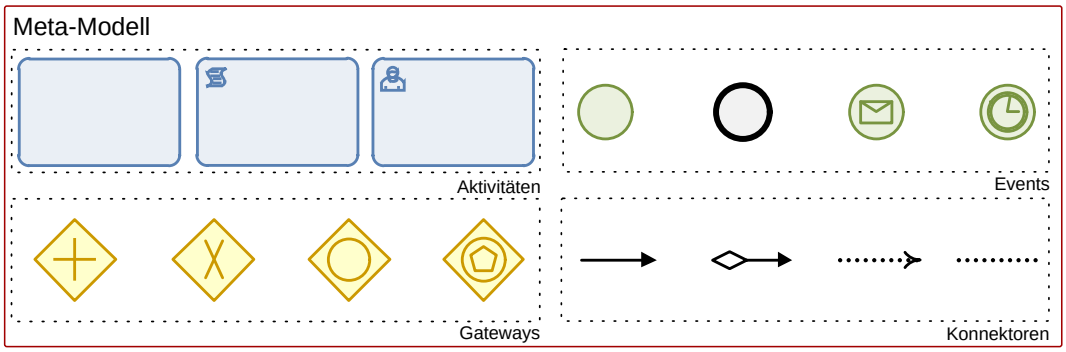

Abbildung 3.2: Beispiel für eine mögliche Gruppierung unter BPMN

Die eindeutige Identifizierung einzelner Elemente innerhalb einer Meta-Modellierungssprache ist unabdingbar für die spätere Modellierung, zum Beispiel eines Prozessmodells. Zusätzlich müssen für erste syntaktische Verifikationen die Elemente ohne weiteres erkennbar und interpretierbar sein. Ein wichtiger Aspekt ist dabei allerdings auch, dass unsere Meta-Metasprache nicht nur Prozessmodelle charakterisieren kann, sondern eine Vielzahl verschiedener Modelltypen. Im weiteren Verlauf wird zwar der Schwerpunkt auf Prozessmodelle gerichtet, dennoch soll für zukünftige Projekte eine stabile Basis gelegt werden.

3.2 Meta Model Description Language (MMDL)

Dieser Abschnitt stellt einen ersten Entwurf der Meta-Metasprache, *Meta Model Description Language (MMDL)*, an Hand eines vollständigen Beispiels vor. Um ein Meta-Modell darzustellen entwickeln wir ausgehend von den, im vorherigen Kapitel aufgestellten Anforderungen, eine konkrete Sprache die ein Modell charakterisiert. Wie im vorherigen Kapitel sind Vollständigkeit, Gruppenbildung und eindeutige Identifizierung von Elementen Grundvoraussetzung für unsere Meta-Metasprache. Wir werden ein XML Schema [19, 20] erarbeiten, das unsere Meta-Metasprache repräsentiert und keine Kenntnisse über die späteren modellierten Diagrammtypen besitzen muss.

Ein Modell muss in eine oder mehrere Gruppen unterteilbar sein, die Modellelemente oder weitere Gruppen beinhalten. Gruppen sowie die einzelnen Elemente müssen einen Namen und eine, im Meta-Modell, eindeutige Identifizierung besitzen. Ein Meta-Modell selber wird über seinen Name identifiziert. Folgendes UML-Diagramm in Abbildung 3.3 repräsentiert ein vollständiges Beispiel eines Meta-Modells, das mit Hilfe der aufgestellten Eigenschaften einer Meta-Metasprache entwickelt wurde.

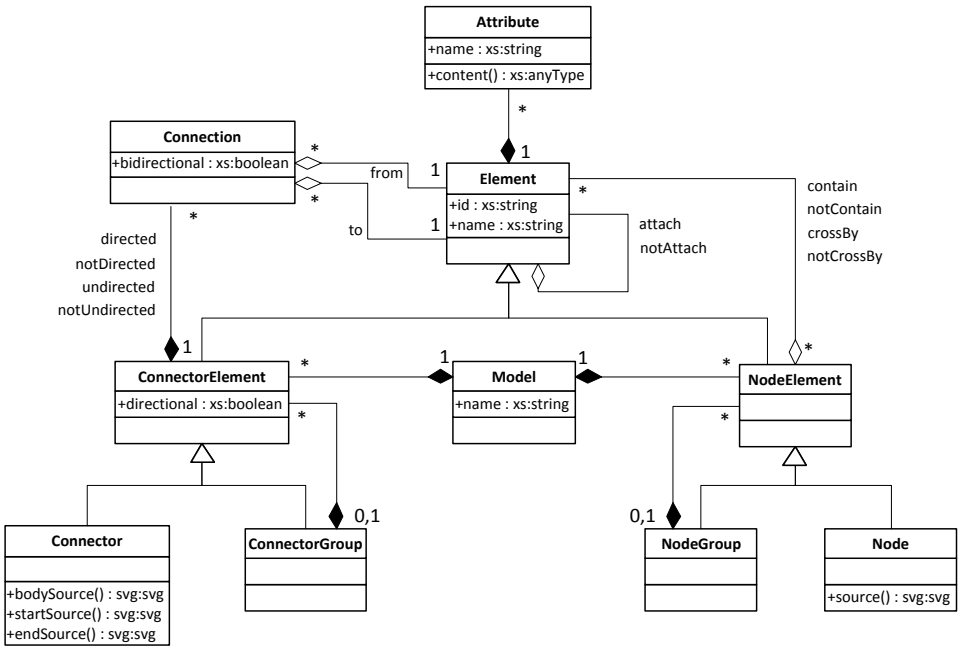

Abbildung 3.3: Visuelle Darstellung des MMDL XML-Schema

Es besteht aus den beiden Gruppen *nodeGroup* und *connectorGroup*. Jede Gruppe ist in der Lage beliebig viele weitere Untergruppen aufzunehmen. Dies schafft die Möglichkeit komplexe Strukturen in einem Meta-Modell zu realisieren. Zu jeder Gruppe gehören die entsprechenden Elemente *node* und *connector*. Mit Hilfe dieser Bausteine werden sämtliche Bestandteile eines Modells abgedeckt. So nimmt *node* beispielhaft in Prozessmodellen die Rolle von Aktivitäten, Attachments, Events oder Swimlanes ein. Mit dem Element *connector* können alle vorhanden Elemente miteinander verbunden werden. Jeder *nodeGroup* und jedem *node* wird die Möglichkeit geboten Einschränkungen bezüglich, welche Elemente sie aufnehmen wollen, welche Elemente an ihnen haften dürfen oder von welchen sie gekreuzt werden dürfen, zu machen. Durch die Kapselung in Gruppen wird erreicht, dass Eigenschaften innerhalb einer Gruppe automatisch weiter vererbt werden oder in Untergruppen

und einzelnen Elementen auf eine andere Art und Weise spezifiziert werden. Weiter sind die jeweiligen grafischen Objekte direkt im Modell integriert. Das komplette Schema, der in Abbildung 3.3 vorgestellten Metasprache ist im Anhang A.1 nochmals als XML-Schema aufgeführt.

Um ein tieferes Verständnis über *MMDL* zu erlangen werden wir im weiteren ein Meta-Model mit ihr erstellen und visualisieren. Es besitzt Start- und End-Events, eine Aktivität, eine Nachrichten-Attachment und einen Sequenzfluss. Die genaue XML-Datei kann im Anhang A.2 eingesehen werden.

3.3 Model Description Language (MDL)

Im Zuge der Meta-Modellentwicklung muss auch die Repräsentation der modellierten Prozessmodelle diskutiert werden. Die graphischen Elemente entsprechen den Gruppenelementen von MMDL. Alle Elemente sind in den modellierten Prozess eindeutig identifizierbar. Die Zuordnung von im Prozessmodell verwendeten Elementen zum Meta-Modell geschieht über ein Class-Attribute. Weiter besitzen alle Elemente eine Position, ein Feld in dem eine Beschriftung der jeweiligen Elemente erfolgen kann und über die Möglichkeit beliebige weitere Attribute für sich zu definieren. Die Abbildung 3.4 illustriert die obige informelle Ausführung in einer formalen und strukturierten Weise. Für eine detailliertere Betrachtung ist im Anhang A.3 das Modell als XML-Schema nochmals aufgeführt.

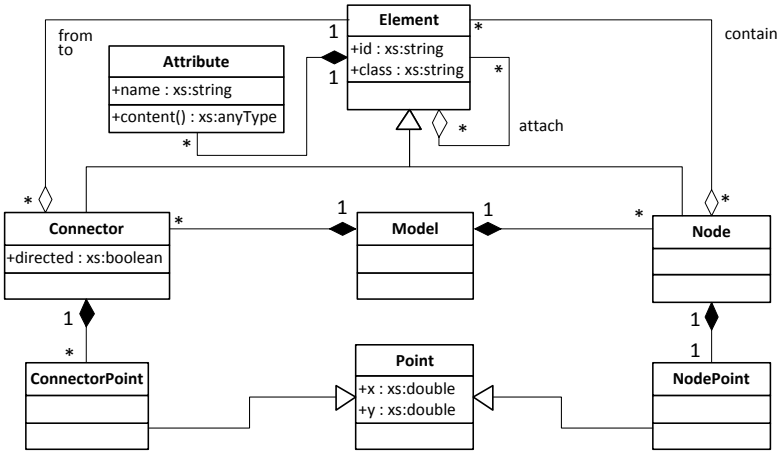

Abbildung 3.4: Visuelle Darstellung des MDL XML-Schema

Für eine bessere Vorstellung, wie das XML-Schema im Anhang A.3 anzuwenden ist, wird beispielhaft eine rudimentäre Modellierung im Auszug 3.1 angegeben. Sie stellt einen einfachen modellierten Prozess dar.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <model name="Mein erstes Modell" xmlns:xsi="http://www.w3.org
       /2001/XMLSchema-instance" xmls="www.uni-ulm.de/mdl/"
       xsi:noNamespaceSchemaLocation="mdl.xsd">
3      <node id="D2Hqbq72_#T!" identifier="3&CzJvQ,]L#V" x="171" y=
           "122">
4          Aktivität A
5      </node>
6      <node id="JP-+S{*DfQ66" identifier="3&CzJvQ,]L#V" x="431" y=
           "64">
7          Aktivität B
8      </node>
9      <node id="avJE+%%_B(p4" identifier="3&CzJvQ,]L#V" x="431" y=
           "180">
10         Aktivität C
11     </node>
12     <node id="g?_jAm7#-qzh" identifier="$Hj8\buc}xdz" x="" y=""/
           >
13   <node id="M&w#eUL6GkjR" identifier="_?!!Hwjr$Gv{" x="62" y="
       122" />
14     <node id="g?_jAm7#-qzh" identifier="3&CzJvQ,]L#V" x="648" y=
           "122" />
15     <node id="Sq+ShQtCNf=C" identifier="MumwSe(}+U4." x="117" y=
           "122" />
16     <node id="ScA[XchUa&gs" identifier="MumwSe(}+U4." x="540" y=
           "122" />

18     <connector id="J@7\Q6MAk\%L" identifier="kC*=./,/ugTd" from=
           "M&w#eUL6GkjR" to="D2Hqbq72_#T!" directional="true"
           bidirectional="false"/>
19     <connector id="J@7\Q7MAk\%L" identifier="kC*=./,/ugTd" from=
           "D2Hqbq72_#T!" to="Sq+ShQtCNf=C" directional="true"
           bidirectional="false" />
20     <connector id="J@7\Q8MAk\%L" identifier="kC*=./,/ugTd" from=
           "Sq+ShQtCNf=C" to="JP-+S{*DfQ66" directional="true"
           bidirectional="false" />
21     <connector id="J@7\Q9MAk\%L" identifier="kC*=./,/ugTd" from=
           "Sq+ShQtCNf=C" to="avJE+%%_B(p4" directional="true"
           bidirectional="false" />
```

```
22      <connector id="J@8\Q6MAk\%L" identifier="kC*=./,/ugTd" from=
            "JP-+S{*DfQ66" to="g?_jAm7#-qzh" attach="g?_jAm7#-qzh"
            directional="true" bidirectional="false" />
23      <connector id="J@9\Q6MAk\%L" identifier="kC*=./,/ugTd" from=
            "avJE+%%_B(p4" to="g?_jAm7#-qzh" directional="true"
            bidirectional="false" />
24      <connector id="J@8\Q7MAk\%L" identifier="kC*=./,/ugTd" from=
            "g?_jAm7#-qzh" to="g?_jAm7#-qzh" directional="true"
            bidirectional="false" />

26  </model>
```

Listing 3.1: Generic Model Language

Wird das Beispiel im Auszug 3.1 mit dem, im Anhang A.2, vorgestellten Meta-Modell als Grundlage gerendert, so ist das Resultat wie in Abbildung 3.5 zu sehen.

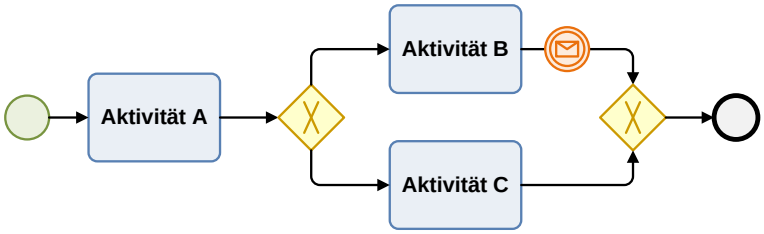

Abbildung 3.5: Beispiel für ein Prozessmodell

4
Systembeschreibung

Dieses Kapitel befasst sich mit der detaillierten Beschreibung der Entwicklungsumgebung Chameleon. Diese gliedert sich in die Schilderung des Systemaufbaus von der grundlegenden Einteilung der Architektur über das Skizzieren einzelner Klassen, die eine wichtige Rolle im System einnehmen, und deren Zusammenspiel bis hin zum konkreten Verhalten Chameleons in bestimmten Situationen. Ein weiterer Punkt in diesem Kapitel wird sein, die erste Beschreibungen der GUI des späteren Programms, modelliert mit Hilfe von Mockups, zu geben.

4.1 Die Systemarchitektur

Zuerst wollen wir den grundsätzlichen Aufbau von Chameleon schildern. Chameleon soll eine Web-Plattform werden. Als logische Kosequenz wird eine dreigeteilte Aufteilung aller Packages erarbeitet. Das Server Package wird zur Laufzeit vollständig auf der Serverseite liegen und dort die systemrelevanten Arbeiten übernehmen. Das Client Package ist für die Bereitstellung von grafischen Oberflächen zur Repräsentation der im System enthaltenen Daten zuständig. Es stellt über Interfaces die Systemfunktionalität auf der Clientseite bereit. Das Shared Package wird für beide Seiten zur Verfügung gestellt. Es enthält alle Datentypen, die beiderseitig Verwendung finden, und Überprüfungsmethoden, die für eine inkorrupte Nachrichtenverbindung zwischen Server und Client unabdingbar sind. In Abbildung 4.1 ist der strukturelle Aufbau von Chameleon illustriert.

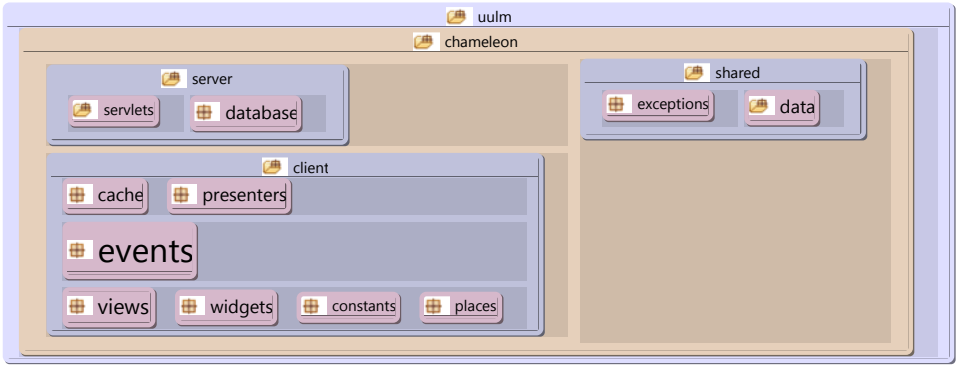

Abbildung 4.1: Systemaufbau von Chameleon

Der Vorteil eines logisch geteilten Arbeitsbereiches ist, dass Methoden und Datentypen, die auf beiden Seiten benötigt werden, nicht redundant programmiert werden müssen und so leichter ein einheitliches Schema beibehalten werden kann.

4.1.1 Klasse: Chameleon

Die Klasse Chameleon, wie sie in Abbildung 4.2 dargestellt ist, ist die zentrale Stelle im System auf Clientseite. Sie ist der Ankerpunkt, dort laufen alle Kommunikationsnachrichten der Clientseite zusammen und werden verteilt. Sie initialisiert alle weiteren wichtigen Funktionalitäten. Sie ist die zentrale Kontrollinstanz für einen reibungslosen Ablauf bei der Bedienung von Chameleon. Für die clientseitige Kommunikation wird ein Bussystem verwendet, das alle Komponenten mit Nachrichten versorgt. Das Package places ist für die Seitensteuerung im Browser zuständig. Dort laufen alle Seitenaufrufe zusammen. All diese wichtigen Komponenten werden von Chameleon am Anfang eingerichtet und gestartet.

4.1.2 Klasse: Cache

Die Klasse Cache aus Abbildung 4.3 ist die Schnittstelle für die im System notwendige Server-Client-Kommunikation. Über ihr werden alle Verbindungsaufrufe gebündelt, um eine zentrale Anlaufstelle für Serveranfragen zu bekommen. So soll der Implementierungsaufwand in den restlichen Teilen des Systems auf ein Minimum reduziert werden, da ein ständiger Verbindungsaufbau der einzelnen Komponenten mit dem Server entfällt und dies alleine der Cache übernimmt. Für die Kommunikation werden einfache High-Level Interfa-

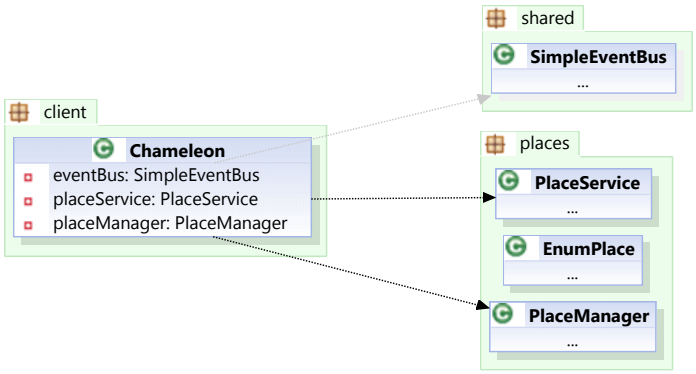

Abbildung 4.2: UML-Diagramm der Klasse Chameleon

ces vom Cache zur Verfügung gestellt um selbst komplexe Anfragen leicht zu realisieren. Der Cache selber teilt sich nochmals auf die großen Arbeitsgebiete von Chameleon auf, um auch hier die Komplexität der Programmierung so gering wie möglich zu halten. Ein detaillierter Einblick in die Arbeitsweise der Cache-Schicht wird im folgenden bei der Diskussion der Komponenten im Kapitel 4.2 gegeben.

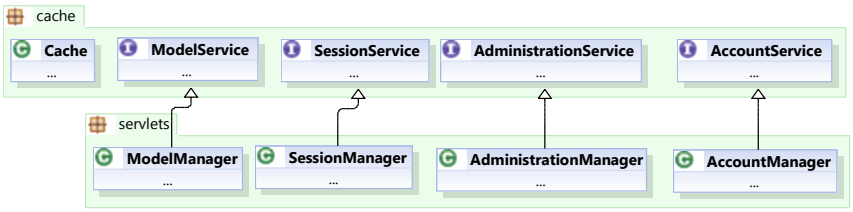

Abbildung 4.3: UML-Diagramm der Klasse Cache

4.2 Die Systemkomponenten

Dieser Abschnitt widmet sich den einzelnen Komponenten von Chameleon und bietet einen Überblick über die Funktionalitäten des Systems. Es wird von den einzelnen Oberflächen und den auf ihnen angebotenen Funktionen ausgehend der Weg der Funktionsaufruf durch das System und seinen Komponenten vollzogen. Die Komponentendiagramme sind auf die drei im System vorhandenen Benutzergruppen Modeler, Manager und Administrator aufgeteilt. Sie bauen aufeinander auf und erweitern den Funktionsausschnitt der vorangegangenen Diagramme.

Komponenten: Modeler

In diesem Unterkapitel werden alle grundlegenden Bereiche des Funktionsumfangs, die angeboten werden, beleuchtet. Wie man aus dem Komponentendiagramm im Anhang B.1 ersehen kann, ist das System in vier logische Ebenen aufgeteilt. Dabei befinden sich die zwei obersten Schichten auf der Clientseite und die zwei untersten teilen die Serverseite unter sich auf. Die oberste Schicht ist für die reine Visualisierung der im System vorhandenen Daten und der einfachen Bereitstellung der Funktionsumfangs verantwortlich. Die zweite Schicht ist für die Server-Client-Kommunikation zuständig. Sie bündelt sämtliche Aufrufe der Clientseite und leitet sie an der Server weiter. Diese Schicht wurde zur Kapselung der zur Kommunikation wichtigen Komponenten eingeführt. So können in den verteilten Bereichen der Oberfläche von Chameleon High-Level-Anfragen getätigt werden ohne mit der Komplexität des Verbindungsaufbaus belastet zu werden. Ein weiterer Vorteil dieser Art von Kommunikationsverwaltung ist ein gewisses Geheimnisprinzip. Komplexere Anfragen, die ein Hand-shake-Verfahren benötigen, können so leicht über abstraktere Schnittstellen bewältigt werden, ohne die gegebenenfalls komplizierte Anfragestruktur an den Frontend Entwickler weiterzugeben. Die dritte Schicht enthält sämtliche anfrageabhängige Logik. Die gesamten Anfragen, die von Clientseite gestellt werden, laufen dort auf der Serverseite zusammen und werden weiterverarbeitet. Sie dient gleichzeitig als Sicherheitsschicht, auf der die korrekte Kommunikation zwischen Client und Server sichergestellt wird, und als Trennschicht zur reinen Datenhaltungsschicht. Die letzte Schicht enthält ausschließlich anfrageunabhängige Logik. Sie liefert an die oberen Schichten die, im System, gespeicherten Daten aus. Sie ist vollständig von der Logik der anwendungsspezifischen Funktionalität getrennt und kümmert sich nur um die Kohärenz der gespeicherten Daten.

Komponenten: Manager

Der Manager ist für die Verwaltung der einzelnen Meta-Modelle im System zuständig. Dafür kann er auf spezielle Methoden zurückgreifen, die es ihm ermöglichen die Modelle zuerst in Chameleon einzuspeisen und später zu verwalten. Wie man im Anhang B.2 erkennen kann, ist der Modell-Upload auf direktem Weg realisiert. Die hoch geladenen Daten werden auf dem Server auf ihre Richtigkeit, im Bezug auf ein implementiertes Plugin, geprüft und in das System eingebettet. Wie schon zuvor erwähnt bauen die Oberflächen und somit

auch die Funktionalität aufeinander auf. Der Manager ist damit in der Lage den gesamten Funktionsumfang der Oberfläche des Modeler mit zu benutzen.

Komponenten: Administrator

Der Administrator besitzt die Aufgabe das System zu administrieren. Eine wichtige Rolle nimmt dabei die Logik-Schicht aus dem Anhang B.3 ein. Sie stellt sicher, dass die Administration nur von berechtigten Benutzern durchgeführt wird. Sie prüft bei jedem administrativen Aufruf die Privilegien des Benutzers, der die Anfrage abgesetzt hat. Sie leistet so einen Beitrag zur sicheren Benutzung von Chameleon. Der Administrator hat die höchste Freigabestufe. Er kann den gesamten Umfang, der angebotenen Anwendungen, nutzen.

4.3 Das Systemverhalten

In diesem Unterkapitel wird ein detaillierter Blick auf das Verhalten des Systems in ausgewählten Situationen geworfen. So werden interessante Szenarien aufgegriffen und an Hand von Sequenzdiagrammen illustriert. Die einzelnen Abbildungen und ihre korrespondierenden Beschreibungen sollen ein tieferes Verständnis der Arbeitsweise von Chameleon geben.

4.3.1 Aufgabe: Login

Der Login am System ist ein Hand-shake-Verfahren. Es teilt sich in zwei Phasen. Zuerst wird beim Login der Passwordsalt des jeweiligen Benutzers abgefragt, um später diesen mit dem eingegebenen Passwort gehasht an der Server zur Verifikation zurück zu schicken. Diese Aufsplittung des Vorgangs dient zur sicheren Übertragung von Passwörtern zum Server-System.

Diese Vorgehensweise wurde bei allen sicherheitskritischen Vorgängen angewandt, um private Daten der Benutzer vor Dritten zu schützen.

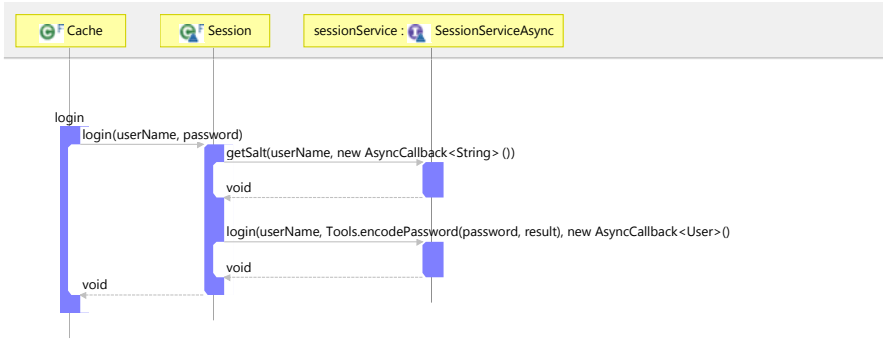

Abbildung 4.4: Sequenzdiagramm für das Anmelden am System

4.3.2 Aufgabe: Meta-Modell importieren

Der Meta-Modell-Import ist der einzige Vorgang der an der zuvor angesprochenen Kommunikationsschnittstelle des Caches vorbei eine Verbindung zum Server aufbaut und das ausgewählte Plugin hoch lädt.

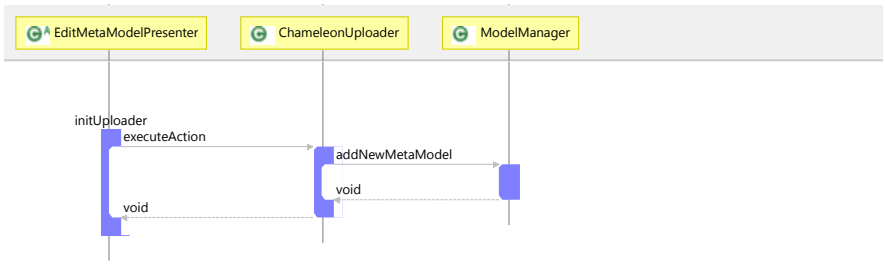

Abbildung 4.5: Sequenzdiagramm für das Importiere von Meta-Modellen

Der Chameleon-Uploader übernimmt den Upload und überprüft ob die hoch geladenen Dateien einem Meta-Modell-Plugin entsprechen und übergibt sie daraufhin an den Model-Manager. Dieser pflegt anschließend das Plugin in das System ein.

4.3.3 Aufgabe: Prozessmodel bearbeiten

Wie zuvor gesehen wird ein Prozessmodell im entsprechenden Editor bearbeitet. Nach der Bearbeitung wird das Modell an den Server geschickt, der die Änderungen des entsprechenden Modells übernimmt.
Der modellierte Prozess wird aus dem Editor in die zuvor angesprochene MDL exportiert und über die Kommunikationskomponente an den Server weitergeleitet.

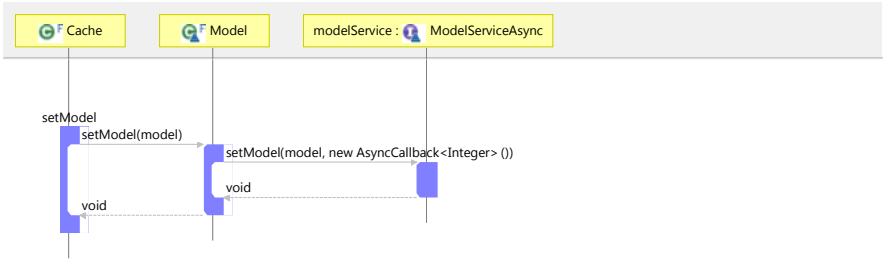

Abbildung 4.6: Sequenzdiagramm für die Bearbeitung eines Prozessmodells

4.4 Graphical User Interface

Widmen wir uns nun der graphischen Oberfläche des zu entwickelnden Systems. Wir werden an Hand von Mockups die Funktionalität verschiedener Komponenten erklären. Dies soll dem besseren Verständnis der Verwendung des späteren Systems dienen. Hier werden auch Komponenten vorgestellt, die bis jetzt noch nicht in den aktuellen Prototypen eingebunden wurden. Somit wird dieses Teilkapitel auch als ein Anstoß für weitere Arbeiten an und mit Chameleon verstanden.

GUI: Anmelden

Die Login-Oberfläche aus Abbildung 4.7 ist die Standardseite, falls man sich noch nicht am System angemeldet hat. Hier findet der Benutzer die Möglichkeiten sich in das System einzuloggen oder auf die Registrierungsseite zu gelangen. Beim Login wird der Anwender auf sein entsprechendes Profil geleitet. Dabei werden zwischen drei Benutzergruppen unterschieden, der Modellierer, der die im System vorhandenen Prozess-Meta-Modelle verwenden kann, um mit Hilfe ihrer Prozesse zu modellieren, der Manager, der für die Verwaltung und Pflege der einzelnen Prozess-Meta-Modelle verantwortlich ist und der Administrator, der sämtliche administrative Aufgaben, angefangen bei der Benutzerverwaltung, erledigt. Nach dem Abmelden gelangt man automatisch von neuem auf diese Oberfläche.

GUI: Registrieren

Diese Oberfläche aus Abbildung 4.8 ist im ausgeloggten Zustand über die Login-Seite zu erreichten. Hier wird dem Benutzer die Möglichkeit geboten, sich am System zu registrie-

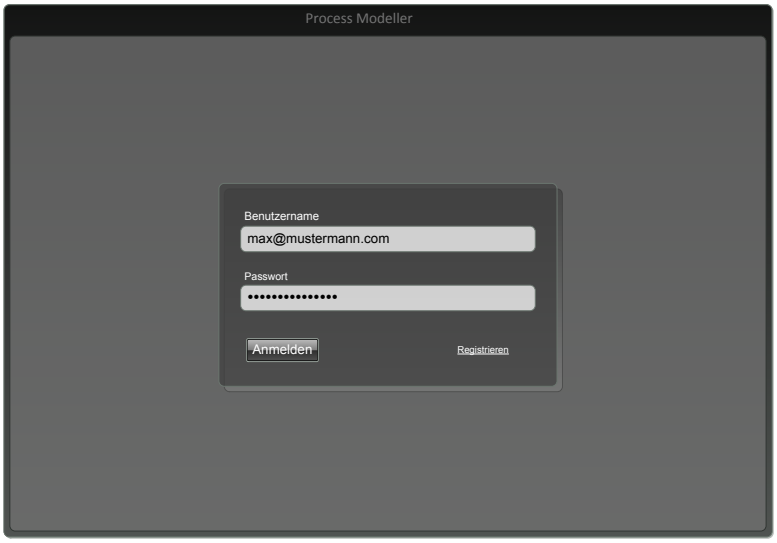

Abbildung 4.7: Mockup für die Logout-Ansicht

ren. Dem neuen Anwender stehen zwei Benutzergruppen bei der Registrierung zur Auswahl, der Modellierer und der Manager. Für letzteren wird noch zusätzlich eine Bestätigung durch einen der Administratoren benötigt. Administratoren hingegen können sich nicht explizit am System registrieren. Sie werden von bereits existierenden Administratoren in diese Benutzergruppe aufgenommen, dazu wird schon bei der Einrichtung des Systems ein Administrator initial angelegt. Im weiteren Verlauf der Nutzung von Chameleon wird ständig mindestens ein Administrator im System vorhanden sein, um nie eine unkontrollierte Situation entstehen zu lassen.

GUI: Account-Profil

Auf dieser Oberfläche aus Abbildung 4.9 stehen dem Anwender gewisse Funktionen zur Verfügung, um sein eigenes Profil zu administrieren. Hier kann er seinen bei der Registrierung angegebenen Benutzername ändern sowie ein neues Passwort für sein Account festlegen. Außerdem kann er dort sein Profil vollständig aus dem System löschen. Falls modellierte Prozesse mit dem zu löschenden Profil eines Benutzers verknüpft sind, werden diese ebenfalls aus dem System gelöscht. Das Löschen von Manager- bzw. Administratorenaccounts, soweit mit diesen Prozess-Meta-Modelle verknüpft sind, hat die Folge das die Meta-Modelle anderen Accounts gleicher Gruppe zu geordnet werden. Ein Admi-

Abbildung 4.8: Mockup für das Registrieren

nistrator kann dabei sein Profil nur löschen, solange mindestens ein weiterer Administrator im System verbleibt. Die Seite ist ständig über den Benutzernamenlink im oberen rechten Bereich des Fensters erreichbar.

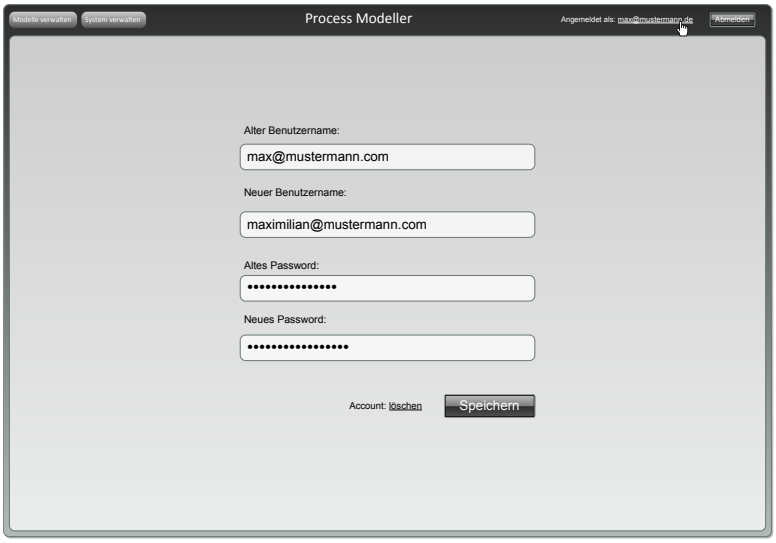

Abbildung 4.9: Mockup für das Profil

GUI: Administration

Diese Oberfläche aus Abbildung 4.10 ist einzig Administratoren zugänglich. Hier können sie die einzelnen Benutzer, die am System registriert sind, einsehen und verwalten. Es stehen administrative Funktionen, wie das Löschen von Benutzern oder das Ändern ihrer Gruppenzugehörigkeit, zur Verfügung. Außerdem werden Funktionen zum Bannen und Freischalten von Benutzeraccounts bereitgestellt. Administratoren haben zusätzlich die wichtige Aufgabe Manager-Accounts freizuschalten.

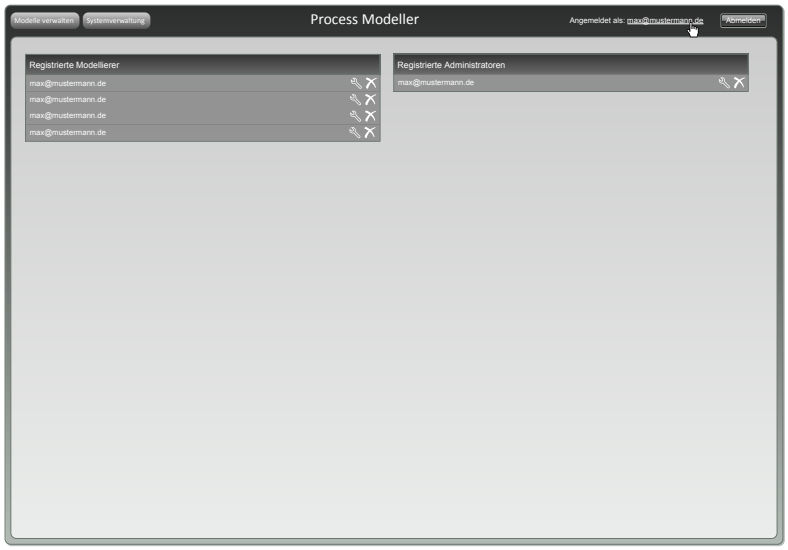

Abbildung 4.10: Mockup für die Administration durch den Systemverwalter

GUI: Modellierungsumgebung

Dies ist das Herzstück der Entwicklungsumgebung Chameleon aus Abbildung 4.11. Der Anwender kann sich ein Prozess-Meta-Modell auswählen, nach welchem er gerne seinen Prozess modellieren möchte, und gelangt anschließend auf diese Oberfläche. Hier stehen ihm die vom Meta-Modell bereitgestellten Elemente in ihre Gruppen eingeteilt zur Verfügung. Er kann die einzelnen Elemente auswählen und in der Zeichenfläche nach seinen Wünschen bearbeiten und mit ihnen modellieren.

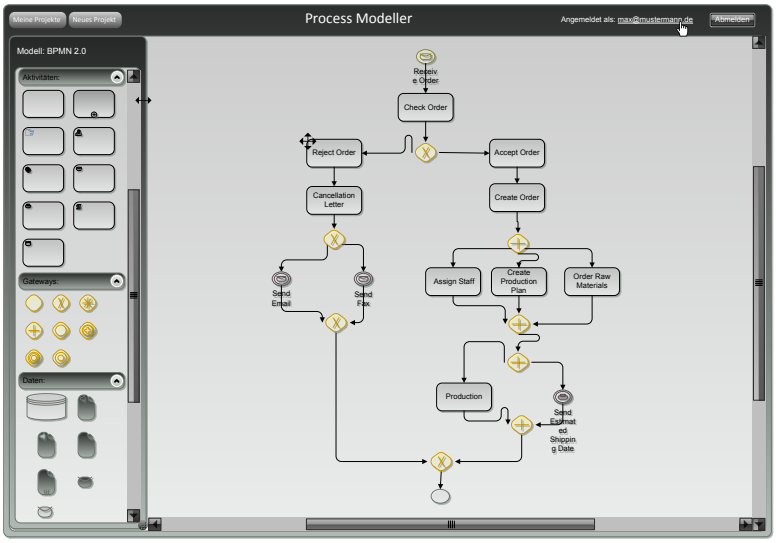

Abbildung 4.11: Mockup für die Modellierungsumgebung

GUI: Model-Creator

Zuletzt im Bereich der Oberflächenbeschreibungen möchten wir zwei Anwendungsbereiche vorstellen, die in zukünftigen Projekten in das bestehende System eingepflegt werden können. Der Modell-Creator aus Abbildung 4.12 ist für die leichte Erstellung von Meta-Modellen vorgesehen. Hier können neue Meta-Modelle erstellt oder bestehende bearbeitet werden. Es können die Meta-Modelle mit ihren Elementen bearbeitet werden, indem Elemente leicht verschoben, gruppiert oder gelöscht werden können. Es soll auch möglich sein syntaktische Regeln in die Meta-Modelle einzubetten.

GUI: Verification

Die Verifikationsansicht, wie man sie in dem Mockup aus Abbildung 4.13 sieht, ist ebenfalls noch nicht in den bis jetzt bestehenden Prototypen integriert. Hier kann der Anwender ein oder mehrere bestehende Modelle auswählen und mit den gewünschten Tests analysieren. Dabei werden fehlerhafte Stellen im Modell in einer Miniaturansicht illustriert sofern dies die Implementierung des jeweiligen Tests unterstützt. Ebenfalls werden bestandene wie verletzte Kriterien des ausgeführten Tests aufgelistet, um detailliertere Analysen zu erlauben.

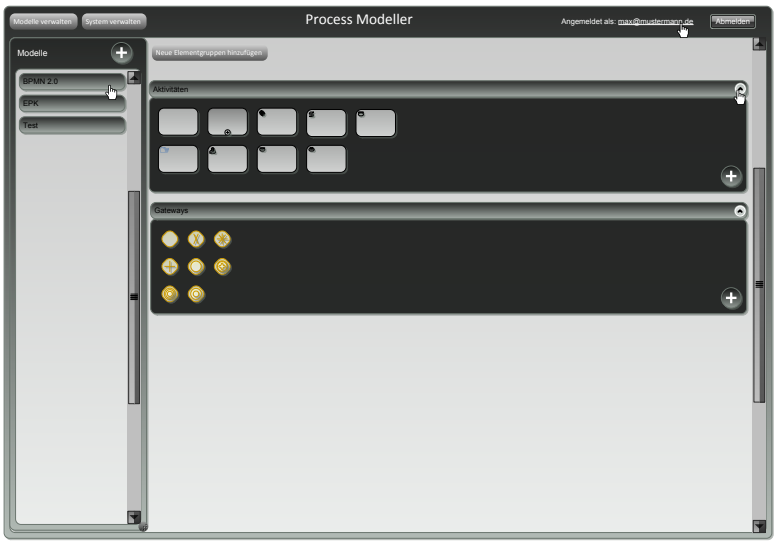

Abbildung 4.12: Mockup für den Model-Creator

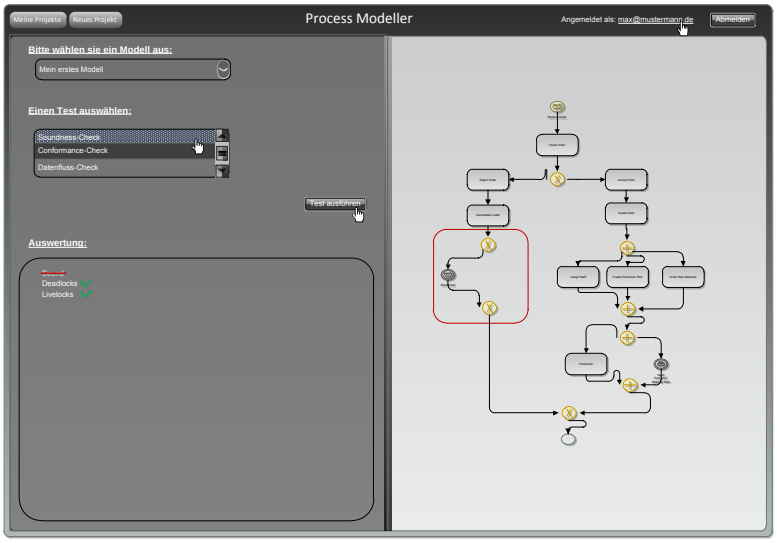

Abbildung 4.13: Mockup für den Verification

5 Implementierung

Die Dokumentation der Entwicklung von Chameleon wollen wir in diesem Kapitel mit der Beschreibung einer ersten Implementierung abschließen. Hierzu werden wir zuerst die verwendeten Implementierungstechnologien vorstellen. Weiter greifen wir ausgewählte Implementierungsaspekte heraus und erklären sie exemplarisch an Quellcodebeispielen.

5.1 Eingesetzte Implementierungstechnologien

Für die Entwicklung von Chameleon wird auf die Umgebung *Eclipse IDE* [21] zurückgegriffen. Zur Erstellung einer Webapplikation wird das *Google Web Toolkit 2.5* [22] verwendet das zusammen mit dem *Google Plugin for Eclipse 4.3* [23] in Eclipse eingebunden wurde. Für sichere Verbindungen während der Nutzung von Chameleon wird das *gwtcrypto* Framework [24] in Kombination mit dem *Apache Commons Codec* [25] Packet verwendet, die sich zusammen für die Verschlüsselung bei der Kommunikation kümmern. Der Datei-Upload wird mit dem Framework *gwtupload* [26] und den Paketen *Apache Commons Fileupload* und *IO* [25] realisiert. Für den Modelleditor wird auf das *gwt-links* [27] Framework zurückgegriffen. E-Mailbenachrichtigungen werden mit dem Packet *JavaMail* [28] erledigt. Für die Datenhaltung wird eine MySQL-Datenbank [29] verwendet, die mit dem *MySQL Connector/J* [30] in den Quellcode eingebunden wurde.

5.2 Ausgewählte Implementierungsaspekte

Nun wollen wir einzelne Aspekte bei der Implementierung von Chameleon herausgreifen und an Hand von Codeauszügen einer konkreten Implementierung erläutern. Die Auszüge stellen nicht die komplette Implementierung der Teilaspekte dar. Sie sollen nur das Prinzip der Arbeitsweise verdeutlichen.

5.2.1 Aspekt: Login

Wie schon im Kapitel 4.3.1 angesprochen ist der Login ein Hand-shake-Verfahren. Es besteht aus zwei Phasen. Die erste Phase holt sich ein Passwortsalt, der für eine sichere Übertragung des Passworts zum Server benötigt wird. Die zweite Phase nimmt den erhaltenen Salt und hasht ihn mit dem eingegebenen Passwort und schickt das Resultat zum Abschließen des Vorgangs zum Server.

```
1  /**
2   * <code>{@link SessionService#getSalt(String userName)}</code>
3   * <code>{@link SessionService#login(String userName, String
        password)}</code>
4   */
5  static void login(final String userName, final String password) {
6     sessionService.getSalt(userName, new AsyncCallback<String>() {
7
8        @Override
9        public void onFailure(Throwable caught) {
10          cache.failureEvent(caught);
11       }
12
13       @Override
14       public void onSuccess(String result) {
15          sessionService.login(userName, Tools.encodePassword(
               password, result), new AsyncCallback<User>() {
16
17             @Override
18             public void onFailure(Throwable caught) {
19                cache.failureEvent(caught);
20             }
21
22             @Override
23             public void onSuccess(User result) {
```

```
24                  if(result != null) {
25                      cache.fireLoginEvent(result);
26                  }
27              }
28          });
29      }
30  });
31  }
```

Listing 5.1: Programmcode für das Einloggen auf der Clientseite

Auf der Clientseite wird über ein Interface ein Remote Procedure Call (RPC) [31, 32] abgesetzt, welcher die entsprechenden Methoden auf der Serverseite anspricht.

```
1  /**
2   * Returns the salt of user who is registered
3   *
4   * @param userName the user-name
5   * @throws InternalServerException for any error on the server
6   */
7  String getSalt(String userName) throws InternalServerException;
8
9  /**
10  * Login a user. Creates a new session
11  *
12  * @param userName the user-name
13  * @param password the account password
14  * @throws InternalServerException for any error on the server
15  */
16 User login(String userName, String password) throws
       InternalServerException;
```

Listing 5.2: Programmcode für das Einloggen als Interface

Auf der Serverseite werden zum einen die getSalt zum anderen die login Methode aufgerufen, die die eingehenden Aufrufe bearbeiten. Die getSalt Methode liefert den jeweiligen Salt, der für jeden Benutzer eindeutig ist, zurück. Die login Methode überprüft ob die Kombination aus Benutzername und Passworthash in der Datenbank vorhanden ist, und gibt bei erfolgreicher Authentifizierung, den Benutzer an die Clientseite zurück.

```
1  @Override
2  public User login(String userName, String password) throws
       InternalServerException {
3     HttpSession session = getThreadLocalRequest().getSession();
4     User user;
5     try {
6        user = userData.login(userName, password);
7        if(user != null) {
8           userData.deleteNewPassword(userName);
9           switch(user.getAccountStatus()) {
10          case ACTIVE:
11             SessionTools.setCurrentUser(session, user);
12             return user;
13          case LOCKED:
14             throw new InternalServerException(ExceptionStatus.
                  ACCESS_DENIED_LOCKED);
15          case UNCONFIRMED:
16             throw new InternalServerException(ExceptionStatus.
                  ACCESS_DENIED_UNCONFIRMED);
17          }
18       }
19       throw new InternalServerException(ExceptionStatus.
            INVALID_LOGIN);
20    } catch (SQLException e) {
21       throw new InternalServerException(ExceptionStatus.DATABASE);
22    }
23 }
24
25 @Override
26 public String getSalt(String userName) throws
       InternalServerException {
27    try {
28       return userData.getSalt(userName);
29    } catch (SQLException e) {
30       throw new InternalServerException(ExceptionStatus.DATABASE);
31    }
32 }
```

Listing 5.3: Programmcode für das Einloggen auf der Serverseite

Ist kein Benutzer in der Datenbank vorhanden oder sein Accountstatus ist gerade nicht für eine Authentifizierung ausreichend, wird der Loginversuch mit einer entsprechenden Exception abgelehnt.

5.2.2 Aspekt: Ein Meta-Modell importieren

Der Chameleon-Uploader liefert eine, zum Upload ausgewählte, Datei zum Server. Dort wird sie in der executeAction Methode entgegengenommen und zum Einpflegen an den Model-Manager weitergereicht. Der Model-Manager prüft anschließend das einzuspeisende Plugin und legt in der Datenbank ein neues Meta-Modell an. Zusätzlich speichert er die hoch geladene Datei in einen speziellen Plugin-Ordner, um es bei Gebrauch bereitstellen zu können.

```
1  @Override
2  public String executeAction(HttpServletRequest request,
3        List<FileItem> sessionFiles) throws UploadActionException {
4     String response = "";
5     FileItem fileItem = null;
6     String modelName = "", modelDescription = "";
7     for (FileItem item : sessionFiles) {
8        if (item.isFormField() && "modelName".equals(item.
             getFieldName())) {
9           modelName = item.getString();
10       }
11       if (item.isFormField() && "modelDescription".equals(item.
             getFieldName())) {
12          modelDescription = item.getString();
13       }
14       if (!item.isFormField()) {
15          fileItem = item;
16       }
17    }
18    try {
19       if(fileItem.getName().endsWith(".jar")) {
20          ModelManager manager = new ModelManager();
21          manager.addNewMetaModel(fileItem, SessionTools.
                getCurrentUser(request.getSession()), modelName,
                modelDescription);
22       }
23       response = fileItem.getName();
24    } catch (Exception e) {
```

```
25          throw new UploadActionException(e);
26      }
27
28      // / Remove files from session because we have a copy of them
29      removeSessionFileItems(request);
30
31      // / Send your customized message to the client.
32      return response;
33  }
```

Listing 5.4: Programmcode für den Upload von Meta-Model-Plugins

5.2.3 Aspekt: Das Plugin-System

Als Plugin wird eine JAR Datei verwendet, die alle benötigten Dateien beinhaltet. Der Aufbau solch eines Plugin ist in Abbildung 5.1 illustriert. Der Name des Plguins und der Klasse, in der alle nötigen Informationen für das Erstellen des Meta-Modells im System enthalten sind, müssen übereinstimmen. Das System wird an Hand dieser Klasse das Meta-Modell aufbauen und in ihm einbetten. Das eingebettete Meta-Modell bietet durch ein implementiertes Interface die Möglichkeit auf den modellierten Prozessen einfache Tests aufzurufen.

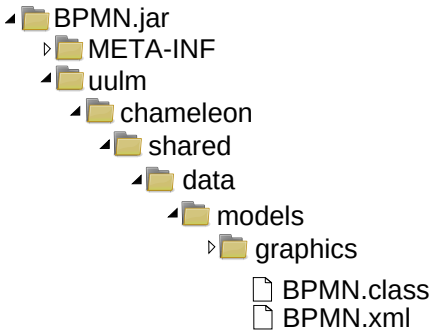

Abbildung 5.1: Aufbau des Meta-Modell-Plugins

5.2.4 Aspekt: Der Modell Editor

Der Editor für die Modellierung von Prozessen bildet die Hauptfunktionalität von Chameleon. Im Programmauszug 5.5 wird die Group-View des Editor initialisiert. Die Gruppen und Elemente des Meta-Modells, das vom Server geliefert wird, werden visuell dargestellt. Die, vom Anwender ausgewählten, Elemente werden auf der Modellierungsfläche zur weiteren graphischen Bearbeitung bereitgestellt.

```
1  @Override
2  public void onModelInfos(ModelInfosEvent e) {
3     if(display.isActive() && flag) {
4        EditorView editor = new EditorView();
5        processModelEditorPresenter.setEditor(editor);
6        editorPresenter = new EditorPresenter(editor, model.getModel
             (), (MetaModel) e.getModel());
7        flag = false;
8        MetaModel model = (MetaModel) e.getModel();
9        Set<String> groups = model.getGroups().keySet();
10       for(String groupName : groups) {
11          final Group group = model.getGroups().get(groupName);
12          Set<String> elements = group.getElements().keySet();
13          AbsolutePanel elemList = new AbsolutePanel();
14          elemList.getElement().setAttribute("style", "padding: 6px
                ;");
15          for(String elementName : elements) {
16             final Element elem = group.getElements().get(
                   elementName);
17             Label elemLabel = new Label();
18             elemLabel.addStyleName("element");
19             elemLabel.getElement().setAttribute("style", elem.
                   getElement());
20             elemLabel.addClickHandler(new ClickHandler() {
21
22                @Override
23                public void onClick(ClickEvent event) {
24                   editorPresenter.addElement(elem.getName(), elem.
                         getElement());
25                }
26             });
27             elemList.add(elemLabel);
28          }
29          display.addNewGroup(elemList, groupName);
```

```
30          }
31      }
32  }
```

Listing 5.5: Programmcode des GroupsPresenters

Der Auszug 5.6 zeigt die Verwaltung der Modellierungsfläche. Dort wird alles zur Steuerung der graphischen Modellierung benötigte initialisiert. Hier wird das Drag-and-Drop der Elemente auf der Modellierungsfläche gesteuert und überwacht.

```
1  private Display display;
2  private DiagramController controller;
3  private PickupDragController dragController;
4  private MetaModel metaModel;
5  private double originalWidth = 1400;
6  private double originalHeight = 800;
7
8  public EditorPresenter(Display display, String model, MetaModel
       metaModel) {
9     this.display = display;
10    this.metaModel = metaModel;
11    controller = new DiagramController((int)originalWidth, (int)
          originalHeight);
12    controller.showGrid(true);
13    this.display.setDiagramm(controller.getView());
14    dragController = new PickupDragController(controller.getView(),
           true);
15    controller.registerDragController(dragController);
16    controller.addNewFunctionHandler(new NewFunctionHandler() {
17       @Override
18       public void onNewFunction(NewFunctionEvent event) {
19          dragController.makeDraggable(event.getFunction());
20       }
21    });
22    controller.getContextMenu().clear();
23    setModel(model);
24    bind();
25 }
```

Listing 5.6: Programmcode für den Modell Editor

In Abbildung 5.2 sieht man einen Screenshot des implementierten Editors. Wie zuvor beschrieben wurde, baut er sich aus zwei Komponenten auf. Am linken Rand ist die GroupView angeordnet. Sie listet alle, im Meta-Modell, enthaltenen Elemente sortiert nach ihren

Gruppen auf, um sie für den Anwender zur Modellierung bereitzustellen. Der Hauptbereich bildet die Modellierungsfläche, die zur visuellen Darstellung und Manipulation von Elementen dient. Die modellierten Prozesse werden dort angezeigt und stehen hier für die Modellierung zur Verfügung.

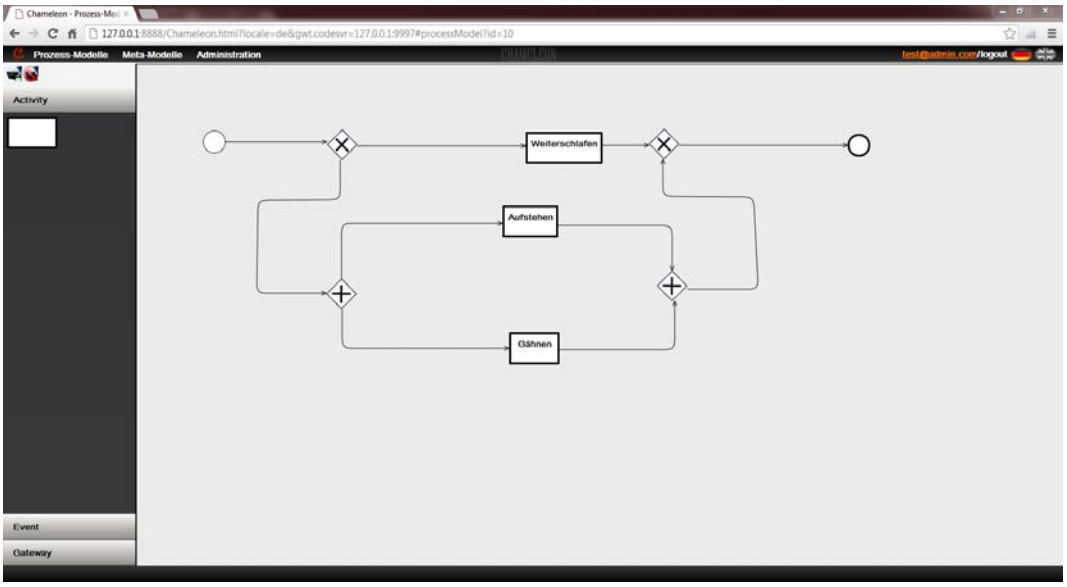

Abbildung 5.2: Screenshot der implementierten Editor Oberfläche

6
Zusammenfassung und Ausblick

Abschließend befasst sich dieses Kapitel noch einmal mit den in dieser Arbeit behandelten Themen und stellt sie in einen Gesamtzusammenhang. Außerdem zeigen wir auf, wie die Ergebnisse dieser Arbeit in zukünftigen Projekten und Arbeiten genutzt und erweitert werden können.

Es galt eine Umgebung zu schaffen, die für zukünftige, neuartige Forschungsprototypen im Rahmen des C^{3}Pro die Grundlage bilden soll, um die Problematik der Einhaltung von Constriants in der Geschäftsmodellierung einzudämmen. Hierfür wurde ein modulares Programm entwickelt, welches durch eine eigene Meta-Metasprache verschiedene Prozessmodellierungssprachen über ein Plugin-System aufnehmen kann. Dabei werden die, vom zugrundeliegenden Meta-Modell, bereitgestellten Elemente dem Anwender zum Modellieren zur Verfügung gestellt. Es ist eine Schnittstelle dem Anwender verfügbar, mit Hilfe derer man eine einfache syntaktische Verifikation auf die erstellten Modelle anwenden kann. Komplexere Tests und die entsprechenden Algorithmen können in das Meta-Meta-Modell-Plugin integriert werden. Chameleon ist somit ein Programm, das mehr als ein reines Zeichentool für Prozessmodelle darstellt. Es kann als Startpunkt für weitere neuartige Prototypen von Programmen im Gebiet der Modellierung und Verifikation von Prozessmodellen dienen.

In dieser Arbeit konnten nicht alle Aspekte einer generischen Modellierungsumgebung aufgegriffen und ausgiebig detailliert analysiert werden. So fehlt beispielsweise eine Komponente, welche die Erstellung von Meta-Modellen direkt in das System integriert. Diese würde grundlegend das Erstellen der Plugins für die Meta-Modelle erleichtern. Ebenfalls bislang nur angedacht wurde eine separate Verifikationsansicht, über welche auch

Tests durchführbar sind die Modelle unterschiedlicher Meta-Modelle umfassen; zum Beispiel Compliance-Prüfung mit Prozessmodellen und Compliance Regeln.
Ein weiterer Aspekt, um welchen Chameleon erweitert werden könnte, ist die Simulation der Ausführung von Prozessmodellen. Das System könnte mit einer Komponente erweitert werden, welche generische Definitionen von Ausführungssemantiken auf Basis der Elemente eines Meta-Modells unterstützt und dafür eine graphische Modellierung anbietet. Ebenso ist eine Komponente zur Meta-Modell basierten graphischen Modellierung von Modelltransformationen angedacht.

A

XML-Schemata und Instanzen von MMDL und MDL

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified" xmlns:svg="http://www.w3.org
       /2000/svg">
3    <xs:import namespace="http://www.w3.org/2000/svg"
         schemaLocation="svg.xsd" />
4    <xs:element name="model">
5      <xs:complexType mixed="true">
6        <xs:sequence>
7          <xs:element ref="nodeGroup" minOccurs="0" maxOccurs="
               unbounded" />
8          <xs:element ref="connectorGroup" minOccurs="0" maxOccurs=
               "unbounded" />
9          <xs:element ref="attribute" minOccurs="0" maxOccurs="
               unbounded" />
10       </xs:sequence>
11       <xs:attribute name="name" use="required" />
12       <xs:attribute name="xmls" use="required" type="xs:anyURI" /
             >
13     </xs:complexType>
14   </xs:element>
15   <xs:element name="nodeGroup">
16     <xs:complexType mixed="true">
17       <xs:complexContent>
18         <xs:extension base="element">
```

```
19          <xs:sequence>
20            <xs:element ref="nodeGroup" minOccurs="0" maxOccurs="
                unbounded" />
21            <xs:element ref="node" minOccurs="0" maxOccurs="
                unbounded" />
22          </xs:sequence>
23        </xs:extension>
24      </xs:complexContent>
25    </xs:complexType>
26  </xs:element>
27  <xs:element name="connectorGroup">
28    <xs:complexType mixed="true">
29      <xs:complexContent>
30        <xs:extension base="element">
31          <xs:sequence>
32            <xs:element ref="connection" minOccurs="0" maxOccurs="
                unbounded" />
33            <xs:element ref="connectorGroup" minOccurs="0"
                maxOccurs="unbounded" />
34            <xs:element ref="connector" minOccurs="0" maxOccurs="
                unbounded" />
35          </xs:sequence>
36        </xs:extension>
37      </xs:complexContent>
38    </xs:complexType>
39  </xs:element>

41  <xs:element name="node">
42    <xs:complexType mixed="true">
43      <xs:complexContent>
44        <xs:extension base="element">
45          <xs:sequence>
46            <xs:element ref="source" minOccurs="0" maxOccurs="
                unbounded" />
47          </xs:sequence>
48        </xs:extension>
49      </xs:complexContent>
50    </xs:complexType>
51  </xs:element>
52  <xs:element name="connector">
53    <xs:complexType mixed="true">
54      <xs:complexContent>
55        <xs:extension base="element">
```

```
56          <xs:sequence>
57            <xs:element ref="connection" minOccurs="0" maxOccurs="
                unbounded" />
58            <xs:element ref="bodySource" minOccurs="0" maxOccurs="
                unbounded" />
59            <xs:element ref="startSource" minOccurs="0" maxOccurs=
                "unbounded" />
60            <xs:element ref="endSource" minOccurs="0" maxOccurs="
                unbounded" />
61          </xs:sequence>
62        </xs:extension>
63      </xs:complexContent>
64    </xs:complexType>
65  </xs:element>

67  <xs:element name="connection">
68    <xs:complexType>
69      <xs:sequence>
70        <xs:element ref="attach" minOccurs="0" maxOccurs="
            unbounded" />
71        <xs:element ref="notAttach" minOccurs="0" maxOccurs="
            unbounded" />
72      </xs:sequence>
73      <xs:attribute name="from" type="xs:string" use="required" /
          >
74      <xs:attribute name="to" type="xs:string" use="required" />
75      <xs:attribute name="bidirectional" type="xs:boolean" use="
          required" />
76          <xs:attribute name="directed" type="xs:boolean" use="
              required" />
77          <xs:attribute name="notDirected" type="xs:boolean" use
              ="required" />
78          <xs:attribute name="undirected" type="xs:boolean" use=
              "required" />
79          <xs:attribute name="notUndirected" type="xs:boolean"
              use="required" />
80    </xs:complexType>
81  </xs:element>

83  <xs:complexType name="element" mixed="true">
84    <xs:complexContent>
85      <xs:extension base="capabilities">
86        <xs:sequence>
```

```
87          <xs:element ref="attribute" minOccurs="0" maxOccurs="
                unbounded" />
88        </xs:sequence>
89        <xs:attribute name="name" type="xs:string" use="required"
              />
90        <xs:attribute name="id" type="xs:string" use="required" /
              >
91      </xs:extension>
92    </xs:complexContent>

94  </xs:complexType>
95  <xs:element name="attribute">
96    <xs:complexType mixed="true">
97      <xs:sequence>
98        <xs:element name="content" type="xs:anyType" minOccurs="0
              " maxOccurs="1" />
99      </xs:sequence>
100     <xs:attribute name="name" type="xs:string" use="required" /
            >
101   </xs:complexType>
102 </xs:element>

104 <xs:element name="source">
105   <xs:complexType>
106     <xs:sequence>
107       <xs:element ref="svg:svg" />
108     </xs:sequence>
109   </xs:complexType>
110 </xs:element>
111 <xs:element name="bodySource">
112   <xs:complexType>
113     <xs:sequence>
114       <xs:element ref="svg:svg" />
115     </xs:sequence>
116   </xs:complexType>
117 </xs:element>
118 <xs:element name="startSource">
119   <xs:complexType>
120     <xs:sequence>
121       <xs:element ref="svg:svg" />
122     </xs:sequence>
123   </xs:complexType>
124 </xs:element>
```

```
<xs:element name="endSource">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="svg:svg" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="capabilities" mixed="true">
  <xs:sequence>
    <xs:element ref="attach" minOccurs="0" maxOccurs="unbounded
        " />
    <xs:element ref="notAttach" minOccurs="0" maxOccurs="
        unbounded" />
    <xs:element ref="contain" minOccurs="0" maxOccurs="
        unbounded" />
    <xs:element ref="notContain" minOccurs="0" maxOccurs="
        unbounded" />
    <xs:element ref="crossBy" minOccurs="0" maxOccurs="
        unbounded" />
    <xs:element ref="notCrossBy" minOccurs="0" maxOccurs="
        unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:element name="attach">
  <xs:complexType mixed="true">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="notAttach">
  <xs:complexType mixed="true">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="contain">
  <xs:complexType mixed="true">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="notContain">
  <xs:complexType mixed="true">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

```
162   <xs:element name="crossBy">
163     <xs:complexType mixed="true">
164       <xs:attribute name="id" type="xs:string" use="required" />
165     </xs:complexType>
166   </xs:element>
167   <xs:element name="notCrossBy">
168     <xs:complexType mixed="true">
169       <xs:attribute name="id" type="xs:string" use="required" />
170     </xs:complexType>
171   </xs:element>
172 </xs:schema>
```

Listing A.1: Meta-Model Description Language

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <model name="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance" xmls="www.uni-ulm.de/mmdl/"
       xsi:noNamespaceSchemaLocation="mmdl.xsd">

4    <nodeGroup id="&mH(%QBKbTx?" name="Elements">
5      <node id="3&CzJvQ,]L#V" name="Activity">
6        <source>
7        <svg width="744.09447" height="1052.362179" xmlns="http:
           //www.w3.org/2000/svg">
8          <metadata id="metadata27096">image/svg+xml</metadata>
9            <g>
10             <title>Layer 1</title>
11               <g id="layer1">
12                 <g id="g28674">
13                 <path fill="#eaeff5" fill-rule="evenodd" d="m237
                     .49117,499.403811122.40379,0l0.46802,0.15622c5
                     .15198,0 9.44598,-4.21552 9.44598,-9.44571c0,0
                     0,0 0,0l0.078,0.54639l0,-76.18988l-0.078,0.078
                     c0,-5.15219 -4.29401,-9.44568 -9.44598,-9.44568
                     l-0.46802,0.62451l-122.40379,0l0.078,-0.62451c
                     -5.15218,0 -9.44566,4.29349 -9.44566,9.44568l0
                     .6245,-0.078l0,76.18988l-0.6245,-0.54639c0
                     ,5.23019 4.29349,9.44571 9.44566,9.44571l
                     -0.078,-0.15622z" id="path28676"/>
14                 <path fill="none" stroke="#5880b3" stroke-width="
                     2.49803138px" stroke-linecap="round" stroke-
                     linejoin="round" d="m237.49117,499.403811122
                     .40379,0l0.46802,0.15622c5.15198,0
                     9.44598,-4.21552 9.44598,-9.44571c0,0 0,0 0,0l0
```

```
                .078,0.54639l0,-76.18988l-0.078,0.078c0
                ,-5.15219 -4.29401,-9.44568 -9.44598,-9.44568l
                -0.46802,0.62451l-122.40379,0l0.078,-0.62451c
                -5.15218,0 -9.44566,4.29349 -9.44566,9.44568l0
                .6245,-0.078l0,76.18988l-0.6245,-0.54639c0
                ,5.23019 4.29349,9.44571 9.44566,9.44571l
                -0.078,-0.15622z" id="path28678"/>
            </g>
          </g>
        </g>
      </svg>
      </source>
    </node>
    <node id="$Hj8\buc}xdz" name="Message">
        <source>
          <svg width="744.09447" height="1052.362179" xmlns="
              http://www.w3.org/2000/svg">
        <metadata id="metadata27096">image/svg+xml</metadata>
        <g>
           <title>Layer 1</title>
           <g id="layer1">
            <g id="g28674">
              <g id="g28694">
                <path fill="#ffffff" fill-rule="evenodd" d="m252
                    .78313,442.19443l0,27.47836l38.7195,0l0
                    ,-27.47836l-38.7195,0z" id="path28696"/>
                <path fill="none" stroke="#000000" stroke-width=
                    "2.49803138px" stroke-linecap="round" stroke-
                    linejoin="round" d="m252.78313,469.67279l38
                    .7195,0l0,-27.47836l-38.7195,0l0,27.47836z"
                    id="path28698"/>
                <path fill="none" stroke="#000000" stroke-width=
                    "2.49803138px" stroke-linecap="round" stroke-
                    linejoin="round" d="m252.78313,442.19443l118
                    .73524,13.73917l19.98425,-13.73917" id="
                    path28700"/>
              </g>
            </g>
           </g>
        </g>
      </svg>
        </source>
     </node>
```

```
40      <node id="_?!!Hwjr$Gv{" name="Start Event">
41        <source>
42        <svg width="744.09447" height="1052.362179" xmlns="http:
              //www.w3.org/2000/svg">
43          <metadata id="metadata27096">image/svg+xml</metadata>
44          <g>
45             <title>Layer 1</title>
46             <g id="layer1">
47               <g id="g28674">
48                 <g id="g2993">
49                   <path fill="#ebf1df" fill-rule="evenodd" d="m202
                         .41119,689.75647c0,-13.03662
                         -10.53857,-23.6532 -23.57516,-23.6532c
                         -13.03661,0 -23.65323,10.61658
                         -23.65323,23.6532c0,13.03662 10.61662,23.5752
                          23.65323,23.5752c13.03662,0
                         23.57516,-10.53857 23.57516,-23.5752" id="
                         path2995"/>
50                   <path fill="none" stroke="#789440" stroke-width=
                         "2.49803138px" stroke-linecap="round" stroke-
                         linejoin="round" d="m202.41119,689.75647c0
                         ,-13.03662 -10.53857,-23.6532
                         -23.57516,-23.6532c-13.03661,0
                         -23.65323,10.61658 -23.65323,23.6532c0
                         ,13.03662 10.61662,23.5752 23.65323,23.5752
                         c13.03662,0 23.57516,-10.53857
                         23.57516,-23.5752" id="path2997"/>
51                 </g>
52               </g>
53             </g>
54          </g>
55        </svg>
56        </source>
57      </node>
58      <node id="3&CzJvQ,]L#V" name="End Event">
59        <source>
60        <svg width="744.09447" height="1052.362179" xmlns="http:
              //www.w3.org/2000/svg">
61          <metadata id="metadata27096">image/svg+xml</metadata>
62          <g>
63             <title>Layer 1</title>
64             <g id="layer1">
65               <g id="g28674">
```

```
66              <g id="g3010">
67                <path fill="#f2f2f2" fill-rule="evenodd" d="m227
                      .44833,398.17426c0,-13.03656
                      -10.53857,-23.57513 -23.57516,-23.57513c
                      -13.03659,0 -23.57516,10.53857
                      -23.57516,23.57513c0,13.03656
                      10.53857,23.57513 23.57516,23.57513c13
                      .03659,0 23.57516,-10.53857
                      23.57516,-23.57513" id="path3012"/>
68                <path fill="none" stroke="#000000" stroke-width=
                      "5.23025322px" stroke-linecap="round" stroke-
                      linejoin="round" d="m227.44833,398.17426c0
                      ,-13.03656 -10.53857,-23.57513
                      -23.57516,-23.57513c-13.03659,0
                      -23.57516,10.53857 -23.57516,23.57513c0
                      ,13.03656 10.53857,23.57513 23.57516,23.57513
                      c13.03659,0 23.57516,-10.53857
                      23.57516,-23.57513" id="path3014"/>
69              </g>
70            </g>
71          </g>
72        </g>
73      </svg>
74      </source>
75    </node>
76      <node id="MumwSe(}+U4." name="XOR">
77        <source>
78          <svg width="744.09447" height="1052.362179" xmlns="
                http://www.w3.org/2000/svg">
79        <metadata id="metadata81496">image/svg+xml</metadata>
80        <g>
81           <title>Layer 1</title>
82           <g id="layer1">
83            <g id="g81503">
84              <path fill="#ffffcc" fill-rule="evenodd" d="m467
                    .8847,463.79074l34.97244,-34.97244l34
                    .97244,34.97244l-34.97244,34.97244l
                    -34.97244,-34.97244z" id="path81505"/>
85              <path fill="none" stroke="#cc9900" stroke-width="
                    2.49803138px" stroke-linecap="round" stroke-
                    linejoin="round" d="m467.8847,463.79074l34
                    .97244,-34.97244l34.97244,34.97244l
```

```
                -34.97244,34.97244l-34.97244,-34.97244z" id="
                path81507"/>
86            <path fill="none" stroke="#cc9900" stroke-width="
                2.49803138px" stroke-linecap="round" stroke-
                linejoin="round" d="m495.36307,447.55353l14
                .98819,32.47443" id="path81509"/>
87            <path fill="none" stroke="#cc9900" stroke-width="
                2.49803138px" stroke-linecap="round" stroke-
                linejoin="round" d="m510.35126,447.55353l
                -14.98819,32.47443" id="path81511"/>
88          </g>
89        </g>
90      </g>
91    </svg>
92      </source>
93    </node>
94  </nodeGroup>

96    <connectorGroup id="G(93==)KCBaB" name="Connector">
97      <connector id="kC*=./,/ugTd" name="Sequence">
98        <source>
99          <svg width="744.09447" height="1052.362179" xmlns="
              http://www.w3.org/2000/svg">
100       <metadata id="metadata27096">image/svg+xml</metadata>
101       <g>
102         <title>Layer 1</title>
103         <g id="layer1">
104           <g id="g28674">
105             <g id="g28714">
106               <path fill="none" stroke="#000000" stroke-width=
                    "2.49803138px" stroke-linecap="round" stroke-
                    linejoin="round" d="m316.54196,716.64789149
                    .96063,0" id="path28716"/>
107               <path fill="#000000" fill-rule="evenodd" d="m365
                    .25357,710.40283l13.7392,6.24506l
                    -13.7392,6.2450610,-12.49011z" id="path28718"
                    />
108             </g>
109           </g>
110         </g>
111       </g>
112     </svg>
113       </source>
```

```
    </connector>
  </connectorGroup>
</model>
```

Listing A.2: Eine Instanz eines Meta-Modells in MMDL

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="model">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="node" minOccurs="0" maxOccurs="unbounded
          " />
      <xs:element ref="connector" minOccurs="0" maxOccurs="
          unbounded" />
      <xs:element ref="attribute" minOccurs="0" maxOccurs="
          unbounded" />
    </xs:sequence>
    <xs:attribute name="name" use="required" />
    <xs:attribute name="xmls" use="required" type="xs:anyURI" /
        >
  </xs:complexType>
 </xs:element>

 <xs:element name="node">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="nodeElement" />
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="connector">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="connectorElement">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="nodeElement">
  <xs:complexContent>
    <xs:extension base="capabilities">
```

```
34        <xs:sequence>
35          <xs:element name="decoration" minOccurs="0" maxOccurs="
              unbounded" type="xs:string" />
36          <xs:element ref="attribute" minOccurs="0" maxOccurs="
              unbounded" />
37        </xs:sequence>
38        <xs:attribute name="id" use="required" type="xs:string" /
            >
39        <xs:attribute name="class" use="required" type="xs:string
            " />
40        <xs:attribute name="x" use="required" type="xs:string" />
41        <xs:attribute name="y" use="required" type="xs:string" />
42      </xs:extension>
43    </xs:complexContent>
44  </xs:complexType>

46  <xs:complexType name="connectorElement">
47    <xs:sequence>
48      <xs:element ref="point" minOccurs="0" maxOccurs="unbounded"
            />
49      <xs:element ref="attach" minOccurs="0" maxOccurs="unbounded
            " />
50      <xs:element name="decoration" minOccurs="0" maxOccurs="
            unbounded" type="xs:string" />
51      <xs:element ref="attribute" minOccurs="0" maxOccurs="
            unbounded" />
52    </xs:sequence>
53    <xs:attribute name="from" type="xs:string" use="required" />
54    <xs:attribute name="to" type="xs:string" use="required" />
55    <xs:attribute name="directional" type="xs:boolean" use="
          required" />
56    <xs:attribute name="bidirectional" type="xs:boolean" use="
          required" />
57  </xs:complexType>

59  <xs:complexType name="capabilities">
60    <xs:sequence>
61      <xs:element ref="attach" minOccurs="0" maxOccurs="unbounded
            " />
62      <xs:element ref="contain" minOccurs="0" maxOccurs="
            unbounded" />
63    </xs:sequence>
64  </xs:complexType>
```

```
65    <xs:element name="attach">
66      <xs:complexType mixed="true">
67        <xs:attribute name="id" type="xs:string" use="required" />
68      </xs:complexType>
69    </xs:element>
70    <xs:element name="contain">
71      <xs:complexType>
72        <xs:attribute name="id" type="xs:string" use="required" />
73      </xs:complexType>
74    </xs:element>

76    <xs:element name="decoration">
77      <xs:complexType>
78        <xs:sequence>
79          <xs:element name="content" type="xs:string" minOccurs="0"
                maxOccurs="1" />
80        </xs:sequence>
81        <xs:attribute name="x" type="xs:double" use="required" />
82        <xs:attribute name="y" type="xs:double" use="required" />
83      </xs:complexType>
84    </xs:element>
85    <xs:element name="attribute">
86      <xs:complexType>
87        <xs:sequence>
88          <xs:element name="content" type="xs:anyType" minOccurs="0
              " maxOccurs="1" />
89        </xs:sequence>
90        <xs:attribute name="name" type="xs:string" use="required" /
            >
91      </xs:complexType>
92    </xs:element>
93    <xs:element name="point">
94      <xs:complexType>
95        <xs:attribute name="x" use="required" />
96        <xs:attribute name="y" use="required" />
97      </xs:complexType>
98    </xs:element>
99  </xs:schema>
```

Listing A.3: Generic Model Language

B

Komponentendiagramme des Systems Chameleon

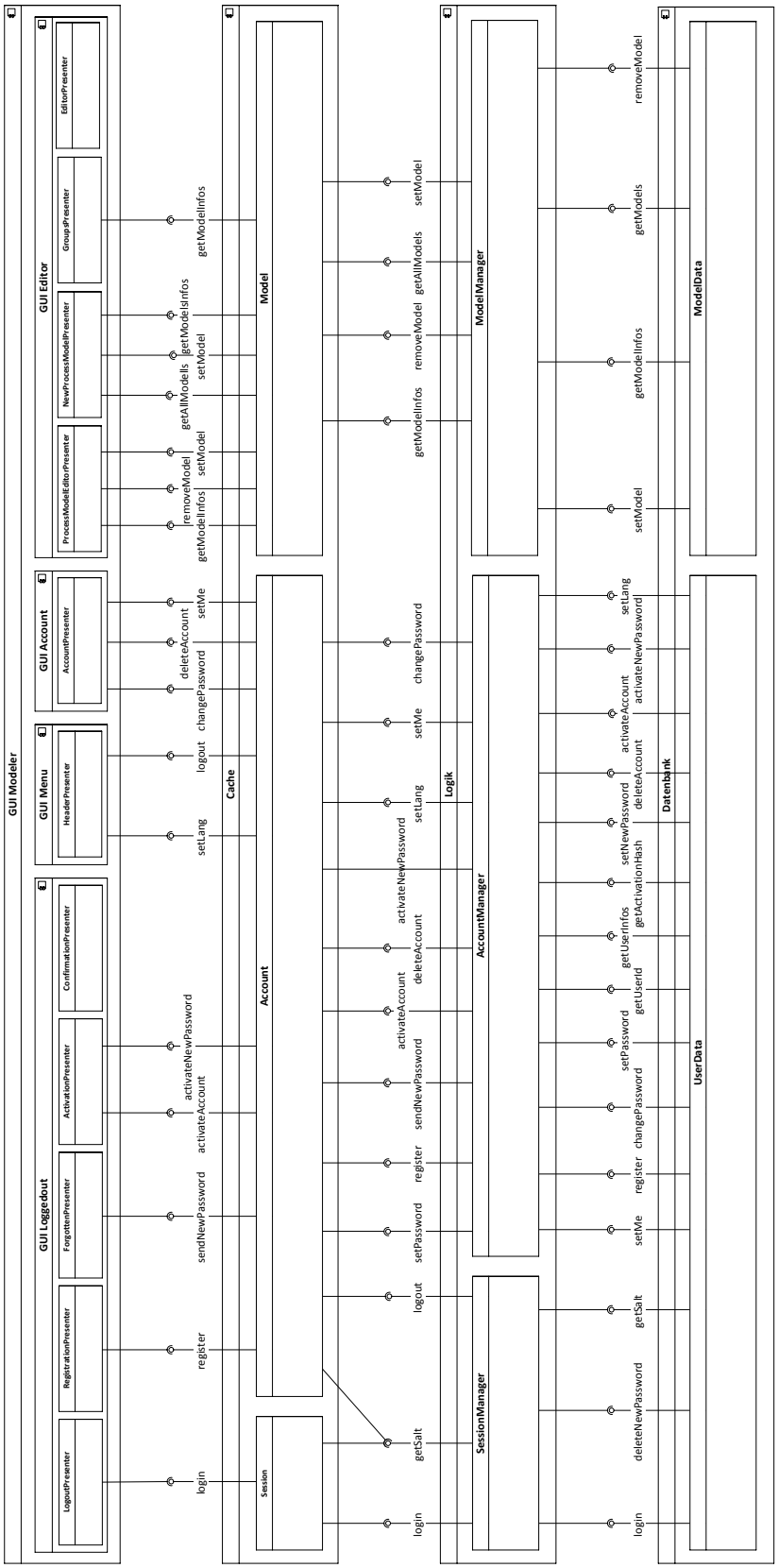

Abbildung B.1: Komponentendiagramm der Oberfläche des Modeler

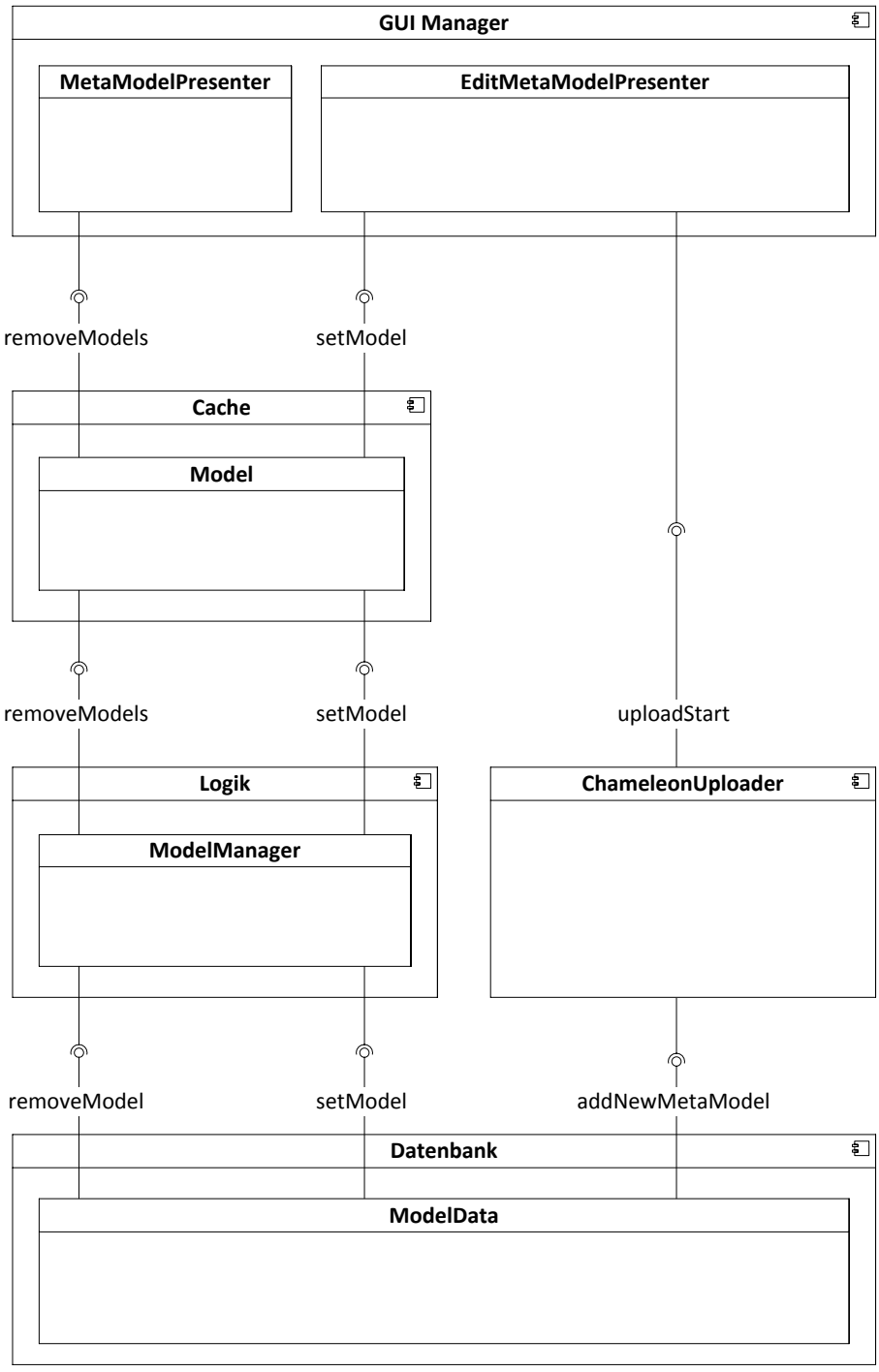

Abbildung B.2: Komponentendiagramm der Oberfläche des Managers

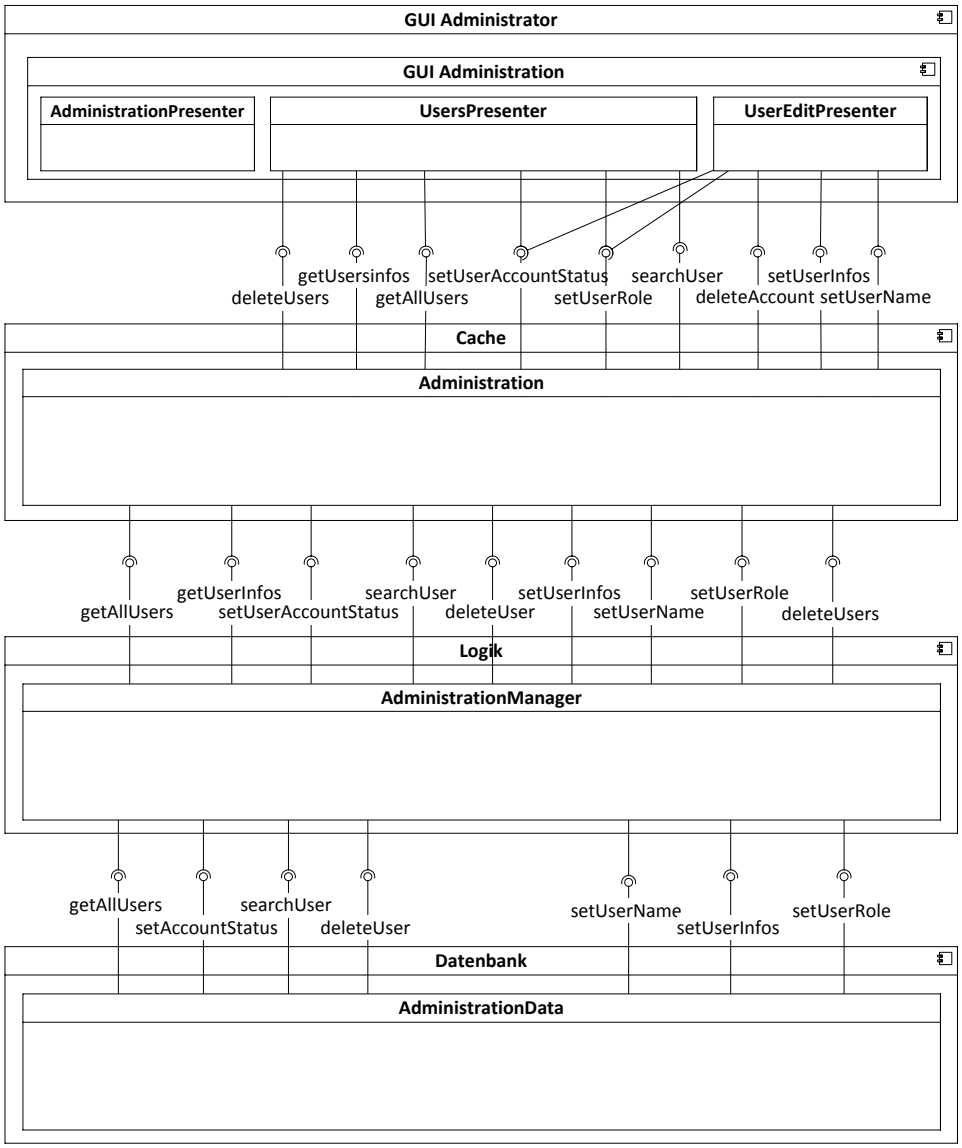

Abbildung B.3: Komponentendiagramm der Oberfläche des Administrators

C

Weitere Screenshots der implementierten Umgebung

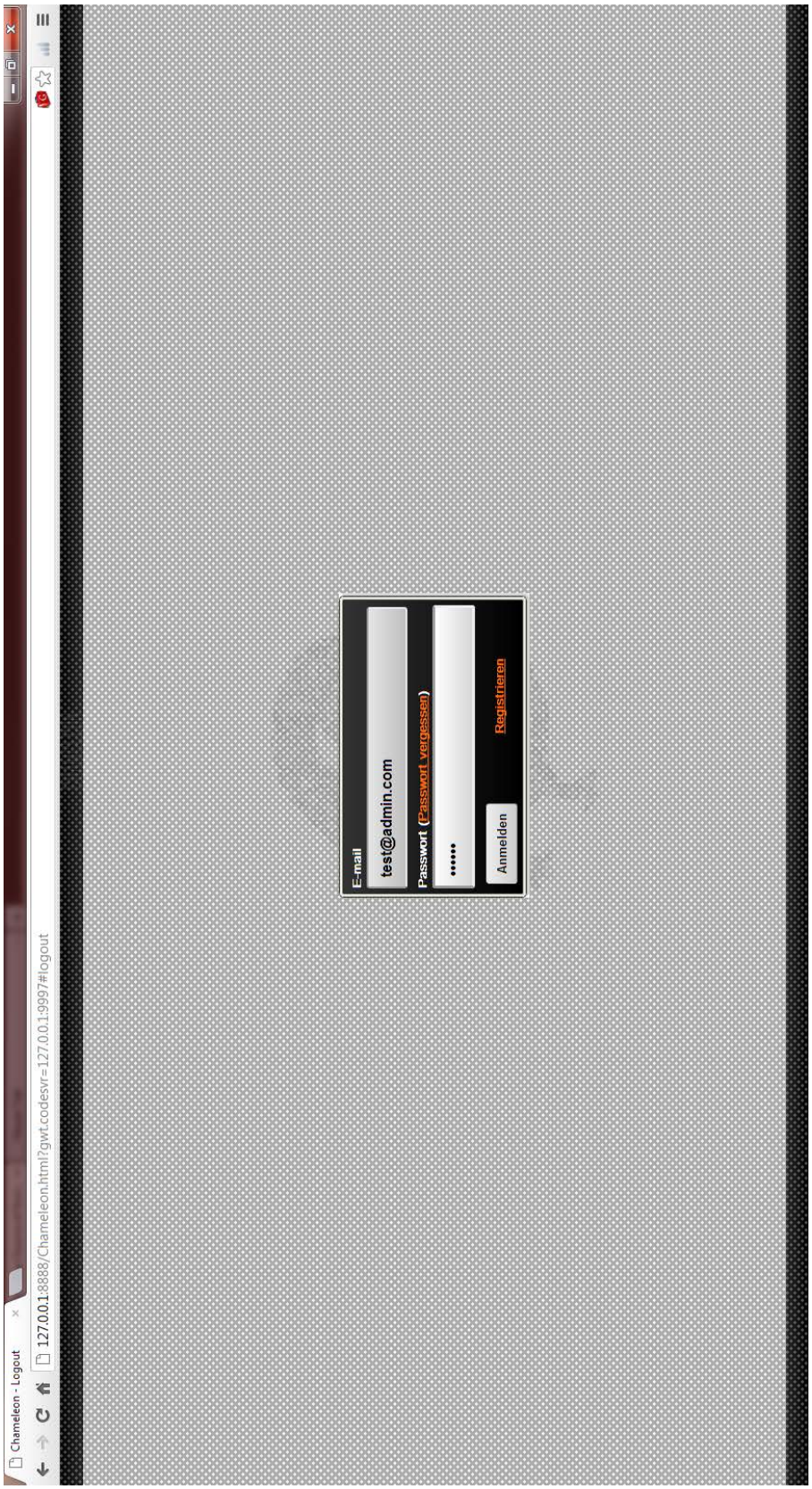

Abbildung C.1: Screenshot der implementierten Login Oberfläche

Abbildung C.2: Screenshot der implementierten Hauptoberfläche

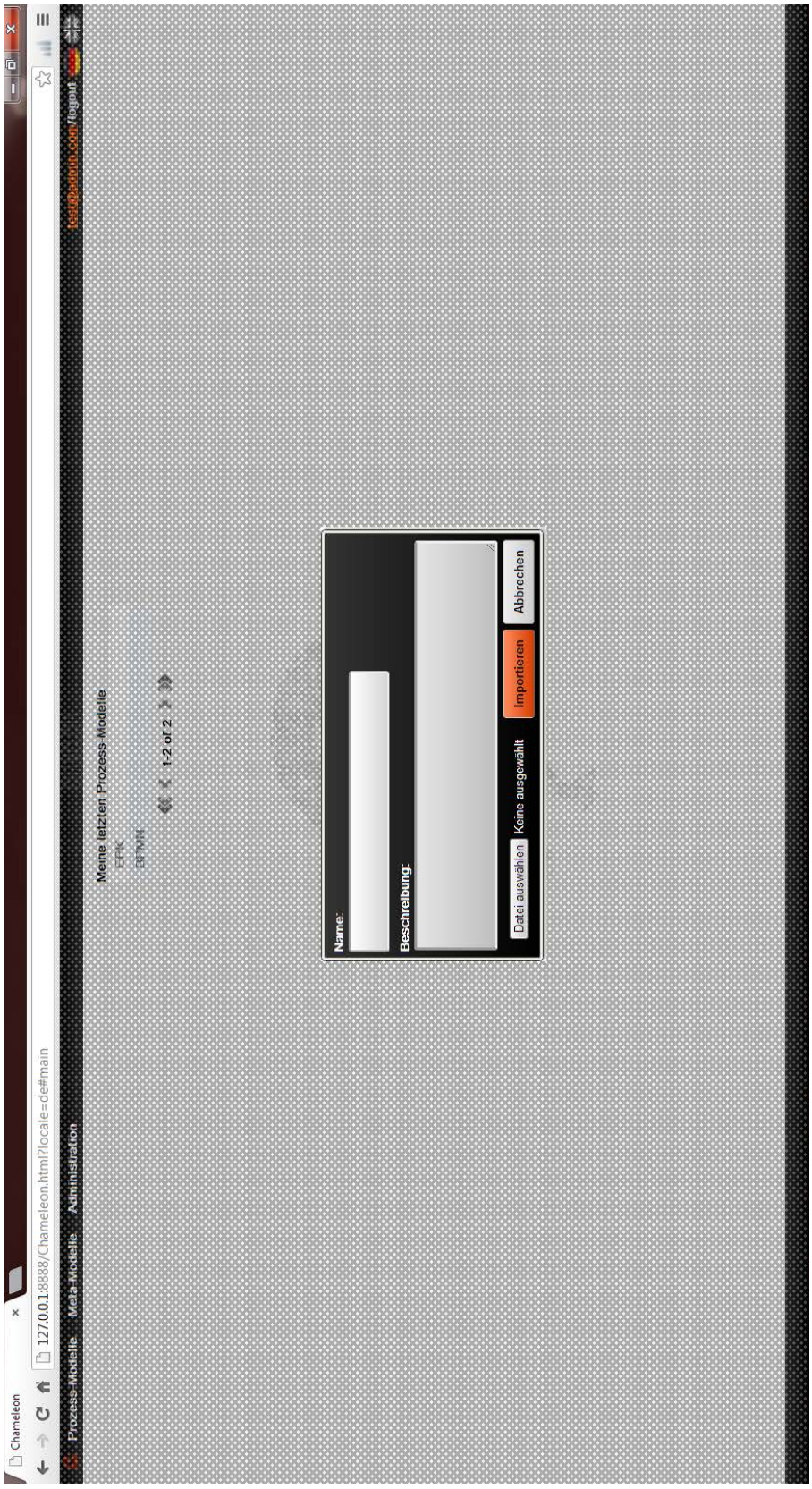

Abbildung C.3: Screenshot des implementierten Meta-Modellimports

Abbildung C.4: Screenshot der implementierten Administrationsoberfläche

Abbildungsverzeichnis

Literaturverzeichnis

[1] POPPENDIECK, Mary ; POPPENDIECK, Tom: *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006

[2] WEBER, Barbara ; REICHERT, Manfred ; RINDERLE-MA, Stefanie: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. In: *Data and Knowledge Engineering* (2008)

[3] ULM, Universität: *Projektseite des Forschungsprojektes C³Pro*. `http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/c3pro.html`. Version: Dezember 2012

[4] WIEN, Universität: *Projektseite des Forschungsprojektes C³Pro*. `http://www.wst.univie.ac.at/communities/c3pro/`. Version: Dezember 2012

[5] REICHERT, Manfred ; RINDERLE, Stefanie: On Design Principles for Realizing Adaptive Service Flows with BPEL. In: *Proc. Workshop "Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen"(EMISA'06)*, Koellen-Verlag, 2006

[6] KNUPLESCH, David ; REICHERT, Manfred ; MANGLER, Jürgen ; RINDERLE-MA, Stefanie ; FDHILA, Walid: Towards Compliance of Cross-Organizational Processes and their Changes. In: *1st International Workshop on Security in Business Processes (SBP'12), BPM'12 Workshops*, Springer, 2012

[7] LY, Linh T. ; RINDERLE-MA, Stefanie ; KNUPLESCH, David ; DADAM, Peter: Monitoring Business Process Compliance Using Compliance Rule Graphs. In: *19th International Conference on Cooperative Information Systems (CoopIS 11)*, Springer, 2011

[8] REICHERT, Manfred ; WEBER, Barbara: *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Berlin-Heidelberg : Springer, 2012

[9] REDBOOKS, IBM: *Improving Business Performance Insight– With Business Intelligence and Business Process Management* -. Vervante, 2006

[10] DAVENPORT, Thomas H.: *Process Innovation - Reengineering Work Through Information Technology*. Boston, Massachusetts : Harvard Business Press, 1993

[11] STACHOWIAK, Herbert: *Allgemeine Modelltheorie*. 1. Auflage. Springer, 1973

[12] VAN DER AALST, W. M. P. ; TER HOFSTEDE, A. H. M. ; KIEPUSZEWSKI, B. ; BARROS, A. P.: Workflow Patterns. In: *Distrib. Parallel Databases* (1999)

[13] OMG (HRSG.): Documents Associated With UML Version 2.4.1. Version: 2011. `http://www.omg.org/spec/UML/2.4.1/`. OMG, 2011. – Technical Report

[14] BOOCH, Grady ; MAKSIMCHUK, Robert A. ; CONALLEN, Jim ; HOUSTON, Kelli A. ; ENGLE, Michael W. ; YOUNG, Bobbi J. P.: *Object-Oriented Analysis and Design with Applications*. 3. Auflage. Pearson Education, 2007

[15] RUMBAUGH, James ; RUMBAUGH, Jim ; EDDY, Frederick: *Object-oriented modeling and design*. United States ed. Prentice Hall, 1991

[16] JACOBSON, Ivar ; CHRISTERSON, Magnus ; JONSSON, Patrik: *Object-oriented software engineering - a use case driven approach*. 4. Auflage. ACM Press, 1992

[17] SOMMERVILLE, Ian: *Software Engineering*. 8. Auflage. Pearson Education Lim., 2007

[18] PRESSMAN, Roger S. ; ROGER, Pressman: *Software engineering - a practitioner's approach*. 7. Auflage. McGraw-Hill Higher Education, 2010

[19] GAO, Shudi ; SPERBERG-MCQUEEN, C. M. ; THOMPSON, Henry S. ; MENDELSOHN, Noah ; BEECH, David ; MALONEY, Murray: *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. 2008

[20] PETERSON, David ; GAO, Shudi ; MALHOTRA, Ashok ; SPERBERG-MCQUEEN, C. M. ; THOMPSON, Henry S.: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. 2008

[21] *Projektseite von Eclipse IEE*. `http://eclipse.org/`. Version: Dezember 2012

[22] GOOGLE INC.: *Projektseite von Google Web Toolkit*. https://developers.google.com/web-toolkit/. Version: Dezember 2012

[23] GOOGLE INC.: *Projektseite von Goolge Plugin for Eclipse 4.2*. https://developers.google.com/eclipse/docs/install-eclipse-3.7. Version: Dezember 2012

[24] PAPPIN, John: *Projektseite von GWTcrypto*. http://code.google.com/p/gwt-crypto/. Version: Dezember 2012

[25] APACHE SOFTWARE FOUNDATION: *Projektseite von Apache Commons*. http://commons.apache.org/. Version: Dezember 2012

[26] CARRASCO, Manuel: *Projektseite von GWTupload*. http://code.google.com/p/gwtupload/. Version: Dezember 2012

[27] RENAUDIN, Pierre: *Projektseite von GWT-links*. http://code.google.com/p/gwt-links/. Version: Dezember 2012

[28] ORACLE CORPORATION: *Projektseite von JavaMail*. http://www.oracle.com/technetwork/java/javamail/index.html. Version: Dezember 2012

[29] ORACLE CORPORATION: *Projektseite von MySQL Community Edition 5.5.24*. https://www.mysql.com/products/community/. Version: Dezember 2012

[30] ORACLE CORPORATION: *Projektseite von MySQL Connector/J 5.1.18*. http://dev.mysql.com/downloads/connector/j/. Version: Dezember 2012

[31] WHITE, James E.: A high-level framework for network-based resource sharing. In: *AFIPS National Computer Conference*, 1976

[32] WHITE, James E.: *A High-level framework for network-based resource sharing*. RFC 707, 1975 (Request for Comments)

Name: Raphael Herfort Matrikelnummer: 698621

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Raphael Herfort