Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Erweiterung eines Prozessmodellierungsframeworks um Containerelemente

Bachelorarbeit

im Studiengang Informatik

von

Dimitri Gatowski

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Wolfgang Nejdl
Betreuer: Dipl.-Math. Thomas Flohr

Hannover, 31.07.2007

Danksagung

Besonders danken will ich meinem Betreuer Thomas Flohr,
der mich während meiner Arbeit hervorragend betreut hat.

Zusammenfassung

Das ProFLOW Framework wurde am Fachgebiet für Software Engineering an der Leibnitz Universität Hannover entwickelt. Es dient zur Implementation von bekannten aber auch eigenen grafischen Notationen mit Semantik. Das Framework stellt Entwicklern Elemente der Informationsflusssprache FLOW zur Verfügung. FLOW ist ein Forschungsprojekt vom Fachgebiet für Software Engineering, mit dessen Hilfe Softwareprozesse optimiert werden können. Als Grundkonzept dienen Informations- und Erfahrungsflüsse zwischen Personen und Dokumenten.

In dieser Bachelorarbeit wurde ProFLOW um Containerelemente erweitert. Containerelemente können andere Elemente enthalten und sind in vielen grafischen Notationen vorhanden. Mit ProFLOW lassen sich nun komplexe Elemente erstellen, die mehr Möglichkeiten und Vorteile beim Modellieren schaffen. Container können ein- und ausklappbar sein. Prozessmodellierer können sich durch einem Mausklick die Verfeinerung des Prozesses betrachten und genauso zurück zur Grobansicht zurückkehren.

Am Schluß wird die Weiterentwicklung von ProFLOW vorbereitet, indem Verbesserungsvorschläge gesammelt werden.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

ProFLOW ist ein Prozessmodellierungsframework und wurde am Fachgebiet für Software Engineering entwickelt. Es bietet eine Plattform an, auf dessen Basis grafische Notationen (z.B. UML Aktivitätsdiagramme, ereignisgesteuerte Prozessketten) mit ihrer Syntax und Semantik implementiert werden können. Diese Notationstypen stehen dann im ProFLOW Editor zur Verfügung und können von Prozessmodellierern verwendet werden.

Das wichtigste Feature vom ProFLOW Framework ist jedoch die Möglichkeit, FLOW in die Prozesse einzubetten. FLOW ist ein Forschungsprojekt vom Fachgebiet für Software Engineering, mit dessen Hilfe Softwareprozesse optimiert werden können. Als Grundkonzept dienen Informations- und Erfahrungsflüsse zwischen Personen und Dokumenten. [8] ProFLOW stellt für jede Notation alle Elemente von FLOW zur Verfügung.

ProFLOW befindet sich aktuell in der Entwicklungsphase. Unterstützt werden bereits einfache (nicht zusammengesetzte) Elemente, Transitionen und Annotationen. Zudem lassen sich semantische Regeln definieren und eigene Funktionen einbinden, die Operationen auf dem vorliegenden Prozess durchführen oder beliebige andere Aufgaben (z.B. Export in XML) erfüllen können.

Das Framework wird schon für studentische Arbeiten und Lehrveranstaltungen eingesetzt. Es wurden nicht nur Bugs, sondern auch Wünsche und Anregungen der Studenten, die mit ProFLOW gearbeitet haben gesammelt, um ProFLOW stetig zu verbessern.

1.2 Zielsetzung

Ziel der Bachelorarbeit ist das Erweitern des Frameworks um Containerelemente. Diese sind für den Großteil aller gängiger Notationen notwendig (z.B. Systemabgrezung in UML UseCase Diagrammen, zusammengesetzte Aktivitäten in UML Aktivitätsdiagrammen). Auch in FLOW spielen sie eine zentrale Rolle.

Die Container müssen sich wie die schon vorhandenen Elementtypen in ProFLOW eingliedern. Beim Modellieren von Prozessen sollen sie einfach und intuitiv eingesetzt werden können. Zusätzlich soll die Weiterentwicklung von ProFLOW vorbereitet werden. Dazu werden Verbesserungsvorschläge und weitere noch nicht vorhandene Features gesammelt und dokumentiert.

1.3 Definitionen

Container

Container können andere Elemente enthalten. Die Semantik der Container hängt von der jeweiligen Sprache ab. Meistens fassen Container andere Elemente zusammen (Zugehörigkeit) oder die enthaltenen Elemente beschreiben den Container im Detail (Verfeinerung).

In ProFLOW haben Container noch weitere Eigenschaften, die das Modellieren von Prozessen erleichtern. Sie können zum Beispiel mitsamt ihrem Inhalt verschoben werden. Weitere Features werden in diesem Kapitel beschrieben.

Notation

In diesem Dokument wird der Begriff „Notation“ sehr oft verwendet. Damit ist hier die graphische Notation einer Sprache gemeint (zum Beispiel ist für den Begriff „Aktivität“ der Sprache UML ein Rechteck mit abgerundeten Ecken als graphisches Notationselement definiert).

1.4 Gliederung

In Kapitel 2 werden gängige graphische Notationen untersucht und Anforderungen für Containerelemente gesammelt. Diese werden anschließend priorisiert und zu einer Liste von funktionalen Anforderungen zusammengefasst. Für Anwendungsfälle, wo Benutzerinteraktion eine Rolle spielt, wurden UseCases erstellt. Anschließend werden noch nichtfunktionale Anforderungen aufgeführt, auf die ich während der Bachelorarbeit geachtet habe.
In Kapitel 3 werden wichtige Konzepte, die im ProFLOW Framework verwendung finden, beschrieben und diskutiert. Am Schluss begründe ich jeweils meine Entscheidung für eines der vorgestellten Konzepte.
Der Entwurf für die Implementation der Container wird in Kapitel 4 vorgestellt. Es wird auf die vorhandene Architektur, die zu einem Teil aus dem Entwurf von GEF hervorgeht, und auf meine Erweiterungen eingegangen.
Kapitel 5 enthält Details zur Implementierung. Es wird zusammengefasst, in welchem Umfang die funktionalen Anforderungen umgesetzt und wie weit die nichtfunktionalen Anforderungen eingehalten wurden. Außerdem werden bekannte Bugs/Probleme aufgelistet, die in der weiteren Entwicklung von ProFLOW behoben werden sollten.
In Kapitel 6 „Verbesserungsvorschläge und Fazit“ fasse ich meine Ideen und Anregungen zusammen, wie es mit ProFLOW weitergehen sollte und formuliere ein Fazit.

2 Anforderungsermittlung

2.1 Benutzerprofile

In der Arbeit verwende ich häufig die Begriffe „Prozessmodellierer“ und „Diagrammentwickler“. Diese Begriffe stehen für zwei verschiedene Benutzerprofile, die ich nun vorstellen will.

Prozessmodellierer

Prozessmodellierer verwenden das Tool ProFLOW, um Prozesse zu modellieren. Sie nutzen dabei die bereits implementierten Diagrammtypen wie z.B. das Aktivitätsdiagramm, welches alle graphischen Elemente zum Modellieren bereitstellt. Prozessmodellierer kennen sich mit Sprachen und deren graphischen Notationen aus. Ihnen steht das Kapitel „Benutzung“ aus dem ProFLOW Tutorial zur Verfügung, für den Fall dass sie keine Erfahrung mit Modellierungstool und speziell ProFLOW haben.

Diagrammentwickler

Diagrammentwickler nutzen das ProFLOW Framework, um neue Diagrammtypen zu implementieren. Sie legen für jedes Notationselement das Aussehen und Verhalten fest und implementieren Operationen, die dem Prozessmodellierer bei der Arbeit helfen (z.B. Export in XML). Diagrammentwickler haben Erfahrung in Java und Grundkenntnisse vom Eclipse Framework und GEF. Ihnen steht das Kapitel „Erweiterung“ aus dem ProFLOW Tutorial zur Verfügung.

2.2 Funktionale Anforderungen

In dem folgenden Abschnitt werden zunächst bekannte und häufig verwendete Notationen kurz vorgestellt. Zusätzlich nehme ich die graphische Notation von FLOW hinzu, da diese in ProFLOW eine besondere Rolle spielt. Die einzelnen Anforderungen für Containerelemente, die die einzelnen Notationen erfordern, werden dann gesammelt und zu einer Liste von funktionalen Anforderungen zusammengefasst werden.

2.2.1 Analyse vorhandener Notationen

FLOW

In FLOW werden Aktivitäten (z.B. Qualitätstechniken wie ein Review oder ein Walkthrough) zu einer FLOW Signatur abstrahiert. Die Aktivität wird dabei von einem FLOW Block repräsentiert, der mehrere Schnittstellen nach außen hat [8]. Ein abstrakter Block ist in Abb. 2.1 zu sehen:

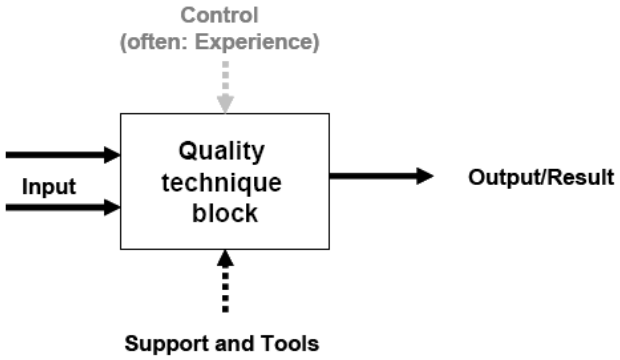

Abbildung 2.1: Abstrakter FLOW Block einer Qualitätstechnik [8]

Außerhalb des Blocks sind Personen und Dokumente, die als Quelle bzw. Senke für Wissens- oder Erfahrungsflüsse dienen. In Abb. 2.2 ist die Signatur von einem Walkthrough in FLOW modelliert:

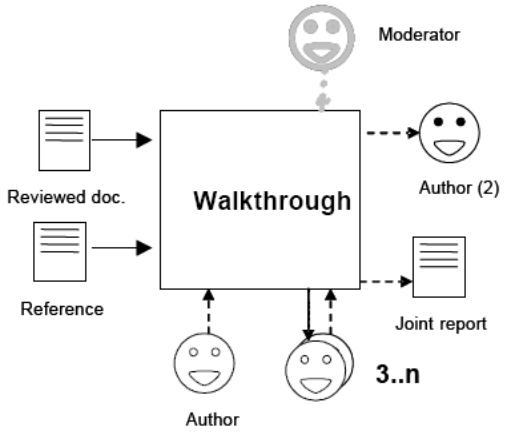

Abbildung 2.2: FLOW Signatur von einem Walkthrough [8]

Wenn man das Walkthrough verfeinern will, müssen nach dem Aufklappen des Blocks die einzelnen Schnittstellen zu Außenwelt nach innen hin sichtbar sein. Man muss z.B. den Wissenfluss aus dem „Reviewed doc.“ im Walkthrough aufgreifen können und weiterleiten können. Hierfür ist ein „Ankerelement“ nötig, welches innerhalb des Containers liegt und sichtbar ist, wenn der Container aufgeklappt ist. Das Element

nimmt Flüsse von außen an und verteilt diese nach innen und auch umgekehrt. Diese Anforderung wird in Kapitel 3.1 diskutiert und näher erläutert.

Entity-Relationship-Diagramm

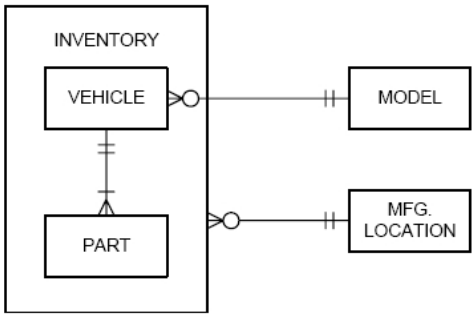

Abbildung 2.3: Beispiel für ein ER-Diagramm mit Verschachtelung [7]

In Abb. 2.3 ist ein ER-Diagramm in Krähenfußnotation abgebildet. Mit ER-Diagrammen kann man einen Ausschnitt der realen Welt modellieren und werden zum Erstellen von Datenbankdesigns verwendet [1].

Interessant ist hierbei die Entität „Inventory“. Sie ist aus den Entitäten „Vehicle“ und „Part“ zusammengesetzt, die miteinander in Beziehung stehen. Zusätzlich gibt es zwei weitere Beziehungen: eine zwischen „Vehicle“ und „Model“, wobei „Model“ außerhalb von „Inventory“ liegt, und eine zwischen „Inventory“ und „Mfg. Location“.

„Inventory„ ist hier ein Container und enthält Entitäten, wobei „Inventory„ selbst auch eine Entität ist und an Beziehungen mit anderen Entitäten teilnehmen kann. Beziehungen zwischen Entitäten sind Transitionen, die zwischen Entitäten und zusammengesetzten Entitäten bestehen und auch über mehrere Ebenen gehen können.

Business Process Modeling Notation

Die Business Process Modeling Notation (BMPN) ist eine graphische Spezifikationssprache, mit der Geschäftsprozesse modelliert werden können. In der BMPN gibt es „Pools“, die Benutzer oder Benutzerrollen repräsentieren. „Lanes“ unterteilen diese Pools [10, S. 87-90]:

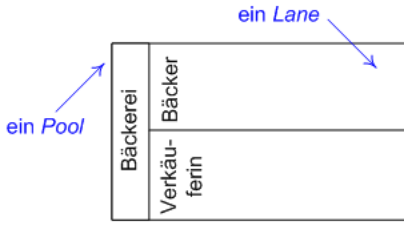

Abbildung 2.4: Pool mit zwei Swimlanes [4]

Ein Pool kann mehrere Swimlanes beinhalten und diese Swimlanes andere Diagrammelemente. Dazu ist eine Schachtelung der Containerelemente erforderlich.

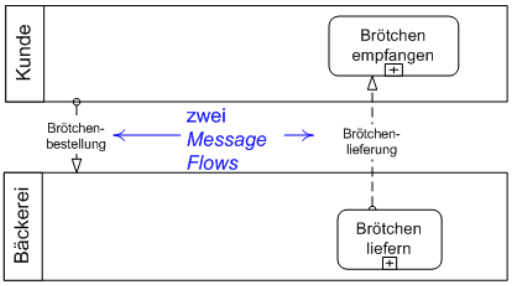

Abbildung 2.5: Message Flows zwischen Swimlanes [4]

In Abb. 2.5 sind Message Flows zu sehen, die einen Austausch von Meldungen repräsentieren. Diese Message Flows können Pools oder andere Elemente miteinander Verbinden. Es ist also möglich, Transitionen zwischen Containern und zwischen Elementen, die innerhalb von zwei verschiedenen Containern liegen, zu zeichnen. Problematisch ist die Positionierung von Message Flows zwischen Pools. Da diese in der Regel lang sind und auch mehrere Message Flows existieren können, ist ein „Ankerelement“ nötig, um den Message Flow an einer gewünschten Stelle zu positionieren.

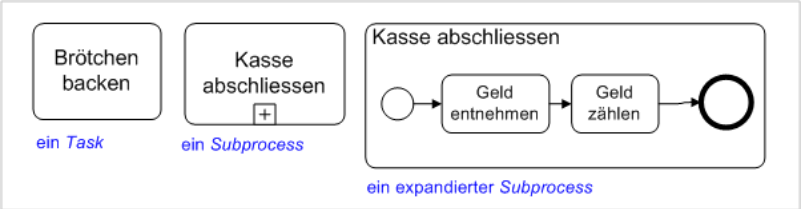

Abbildung 2.6: Beispiel für Aktivitäten [4]

In der BPMN gibt es Aktivitäten, welche die Aufgabe beschreiben, die durch den Geschäftprozess zu erledigen ist. Es wird zwischen einfachen (Task) und zusammengesetzten (Subprocess) Aktivitäten unterschieden. Ein Subprocess besteht aus einem oder mehreren Tasks oder Subprocesses und lässt sich auch aufgeklappt anzeigen (expandierter Subprocess in Abb. 2.6).

Auch hier müssen Container geschachtelt werden können. Zudem muss ein Container ein- und wieder ausklappbar sein, um die beiden Darstellungsformen in Abb. 2.6 zu unterstützen.

Aktivitätsdiagramm (UML 2)

Ein Aktivitätsdiagramm stellt den Ablauf von elementaren Aktivitäten eines Systems dar. Es eignet sich um den Ablauf eines Anwendungsfalls zu beschreiben.

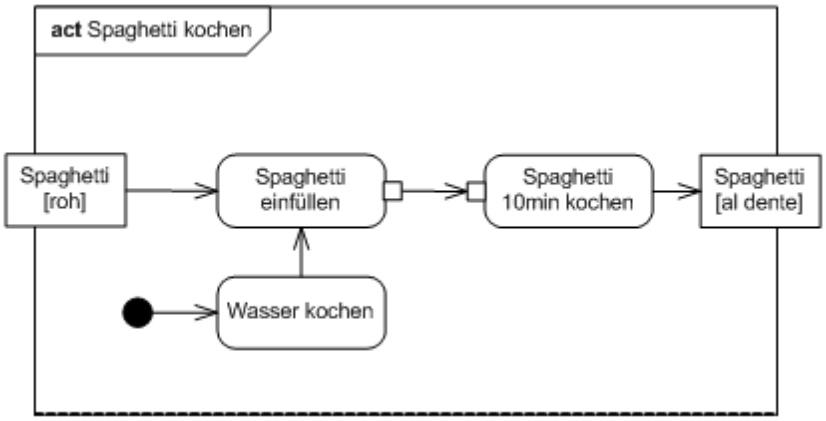

Abbildung 2.7: Aktivitätsdiagramm [5]

In Abb. 2.7 ist eine Aktivität zu sehen, die aus mehreren Aktivitäten besteht. Diese können elementar aber auch komplex sein. Eine Aktivität kann also, wie ein Container, mit Diagrammelementen gefüllt werden.

Eine Besonderheit sind hier die Ein- und Ausgabepins, die eine Schnittstelle nach außen bilden. Über diese Schnittstelle können Werte in die Aktivität übergeben und auf der Ausgabeseite wieder entnommen werden. Die Aktivität aus Abb. 2.7 kann in

einem zugeklappten Zustand modelliert werden. Die Pins sind dann weiterhin sichtbar:

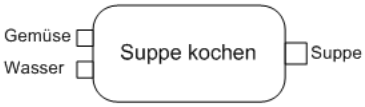

Abbildung 2.8: Aktion mit Ein- und Ausgabepins [5]

In UML 2 Aktivitätsdiagramme sind auch Swimlanes spezifiziert. Sie partitionieren das System und ordnen einzelne Aktivitäten bestimmten Partitionen zu. In Abb. 2.9 ist ein Beispiel dafür zu sehen. Der Startknoten und die darauf folgende Aktivität sind der Partition A zugeordnet. Der nächste Schritt wird in der Partition B durchgeführt.

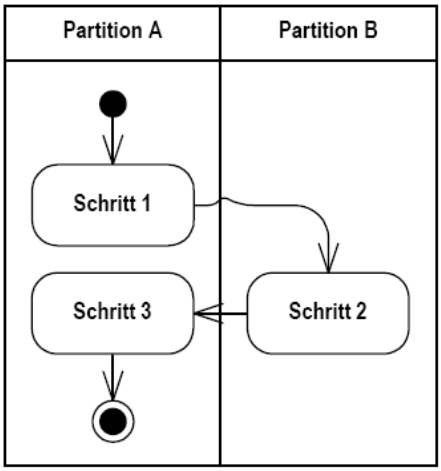

Abbildung 2.9: Beispiel für Swimlanes [6]

Datenflussdiagramm (DeMarco)

In einem Datenflussdiagramm (DFD) werden Flüsse von Daten zwischen Prozessen dargestellt. Für DFDs gibt es verschiedene Notationen, wobei hier die bekannte Notation nach Tom DeMarco betrachtet wird. Eine Kernidee dieser Notation ist große Systeme, die nicht auf einer Seite gezeichnet werden können, in Subsysteme zu zerlegen [2]. Dieser Vorgang wird Dekomposition genannt.

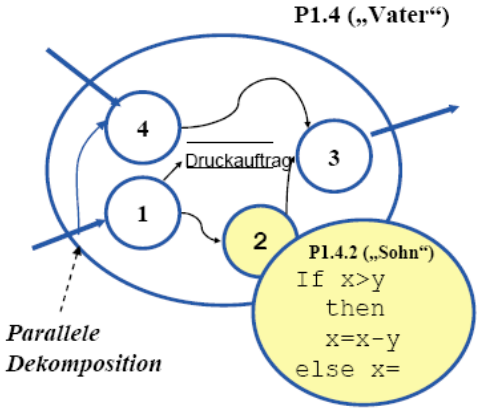

Abbildung 2.10: DFD nach DeMarco [9]

In Abb. 2.10 ist die Dekomposition des Prozesses P1.4 modelliert. Es ist eine Verfeinerung von P1.4 erkennbar, da nun die Subprozesse und die Datenflüsse zwischen ihnen sichtbar sind. Dies betrifft auch Datenflüsse von außen: der Datenfluss nach Prozess 1.4.1 teilt sich innerhalb von P1.4 auf und fließt auch zu 1.4.4.

Prozesse sind in DFDs Container, die Prozesse und Datenflüsse enthalten. Sie haben eine Art Schnittstelle für Datenflüsse von außen, um sie innerhalb des Prozesses aufteilen zu können. Nach dem Aufklappen von einem Prozess gehen die Datenflüsse in das Innere des Prozesscontainers. Für die parallele Dekomposition ist ein Schnittstellenelement nötig, an dem man den Datenfluss trennen kann.

Anwendungsfalldiagramm (UML 2)

Ein Anwendungsfalldiagramm beschreibt das Verhalten eines Systems. Es werden Akteure und Anwendungsfälle dargestellt und Beziehungen zwischen Akteuren und einzelnen Anwendungsfällen gezeichnet. Zusätzlich wird das System mit Hilfe einer Box abgegrenzt.

In Abb. 2.11 ist das System „Mobilfunkbetreiber“ mit zwei Anwendungsfällen dargestellt. Es gibt zwei Akteure außerhalb des Systems, die an den beiden Anwedungsfällen interessiert sind.

Ein System ist ein Container für Anwendungsfälle und haben einen Bezeichner, der angezeigt wird. Die Elemente im Container können untereinander und mit Diagrammelementen außen verbunden werden.

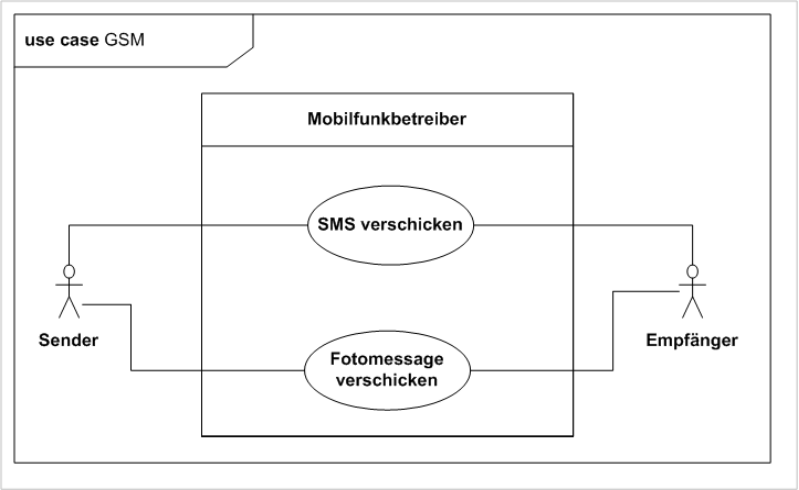

Abbildung 2.11: Beispiel für ein Anwendungsfalldiagramm [5]

2.2.2 Zusammenfassung der Anforderungen

Anforderungen von Prozessmodellierern

Im folgenden Abschnitt werden Anforderungen für Container, die im vorigen Abschnitt gesammelt wurden, zusammengefasst. Sie werden hier verallgemeinert dargestellt, d.h. jede Anforderung kommt in einer Notation nur dann zum Einsatz, wenn die Sprache es erlaubt. Die Anforderung „Container können Elemente beinhalten“ ergibt sich aus der Definition eines Containers und ist in allen betrachteten Notationen erforderlich. Deswegen ist sie nicht weiter aufgeführt.

1. Container sollen auch Container beinhalten (Schachtelung)
2. Elemente aus einem Container können mit Elementen aus einem anderen Container verbunden werden
3. auch Container können mit beliebigen Elementen verbunden werden
4. Container sind einklappbar
5. im eingeklappten Zustand sind die Transitionen, die mit dem Container verbunden sind immer noch sichtbar
6. Darstellung von Transitionen, die ein Element innerhalb des eingeklappten Containers mit einem Element außerhalb des Containers oder umgekehrt verbinden
7. Container haben einen Namen, der im eingeklappten und ausgeklappten Zustand des Containers anzeigbar sein muss
8. Ankerelemente für Container, um eine Schnittstelle zu modellieren

Priorisierung der Anforderungen

Die oben genannten Punkte werden hier tabellarisch zusammengefasst. Links sind die einzelnen Notationen aufgeführt, oben die Nummern der Anforderungen aus der Auflistung. Wenn in einer Notation eine Anforderung benötigt wird, ist dies mit einem X gekennzeichnet.

Sprache / Notation	1	2	3	4	5	6	7	8
FLOW		X	X	X	X	X	X	X
Entity-Relationship-Diagramm	X	X	X	X	X	X	X	
Business Process Modeling Notation	X	X	X	X	X		X	X
UML 2 Aktivitätsdiagramm	X		X	X	X		X	
Datenflussdiagramm (DeMarco)	X	X	X	X	X	X	X	X
UML 2 Anwendungsfalldiagramm		X					X	

Anhand der Häufigkeit der Kreuze in jeder Spalte ergibt sich folgende Priorisierung, die ich in Iterationen aufgeteilt habe. Es mussten neben der Häufigkeit auch funktionale Abhängigkeiten beachtet werden (z.B. kann die Darstellung von Transitionen zu eingeklappten Containern erst dann implementiert werden, wenn es einklappbare Container gibt).

1. Iteration
   - Elemente aus einem Container können mit Elementen aus einem anderen Container verbunden werden
   - auch Container können mit beliebigen Elementen verbunden werden
   - Container haben einen Namen, der im eingeklappten und ausgeklappten Zustand des Containers anzeigbar sein muss
   - Container sollen auch Container beinhalten
2. Iteration
   - Container sind einklappbar
   - im eingeklappten Zustand sind die Transitionen, die mit dem Container verbunden sind immer noch sichtbar
3. Iteration
   - Ankerelemente für Container, um eine Schnittstelle zu modellieren
   - Darstellung von Transitionen, die ein Element innerhalb des eingeklappten Containers mit einem Element außerhalb des Containers oder umgekehrt verbinden

Anforderungen von Diagrammentwicklern an das Framework

Diagrammentwickler implementieren neue Diagramme auf der Basis des ProFLOW Frameworks. Containerelemente müssen also an den Diagrammtyp anpassbar sein, optisch und funktionell.

1. Container haben wie *SimpleElements* eine Figure-Klasse, die das Aussehen festlegt
2. es muss wie schon bei *SimpleElements* möglich sein, semantische Regeln festzulegen, die Containerelemente betreffen; dazu gehören:
   a) erlaubte Elementtypen, die in einen Container gelegt werden dürfen, festlegen
   b) Transitionen zwischen oder ausgehend von einem Container erlauben oder verbieten
   c) Einklappen von Containern erlauben oder verbieten
3. es muss möglich sein, das Verhalten von Containern im Diagramm anzupassen oder zu erweitern, wobei eine strikte Trennung zwischen Framework und Applikation bestehen muss; dazu gehört:
   a) Positionsänderungen
   b) Größenänderungen
   c) Anker für Transitionen anbieten (Anzahl, Position direkt am Container oder im Container; siehe Kapitel 3.3)

2.2.3 UseCases

Die Anforderungen der Prozessmodellierer werden hier in Form von UseCases genauer spezifiziert.

UseCase-Übersicht

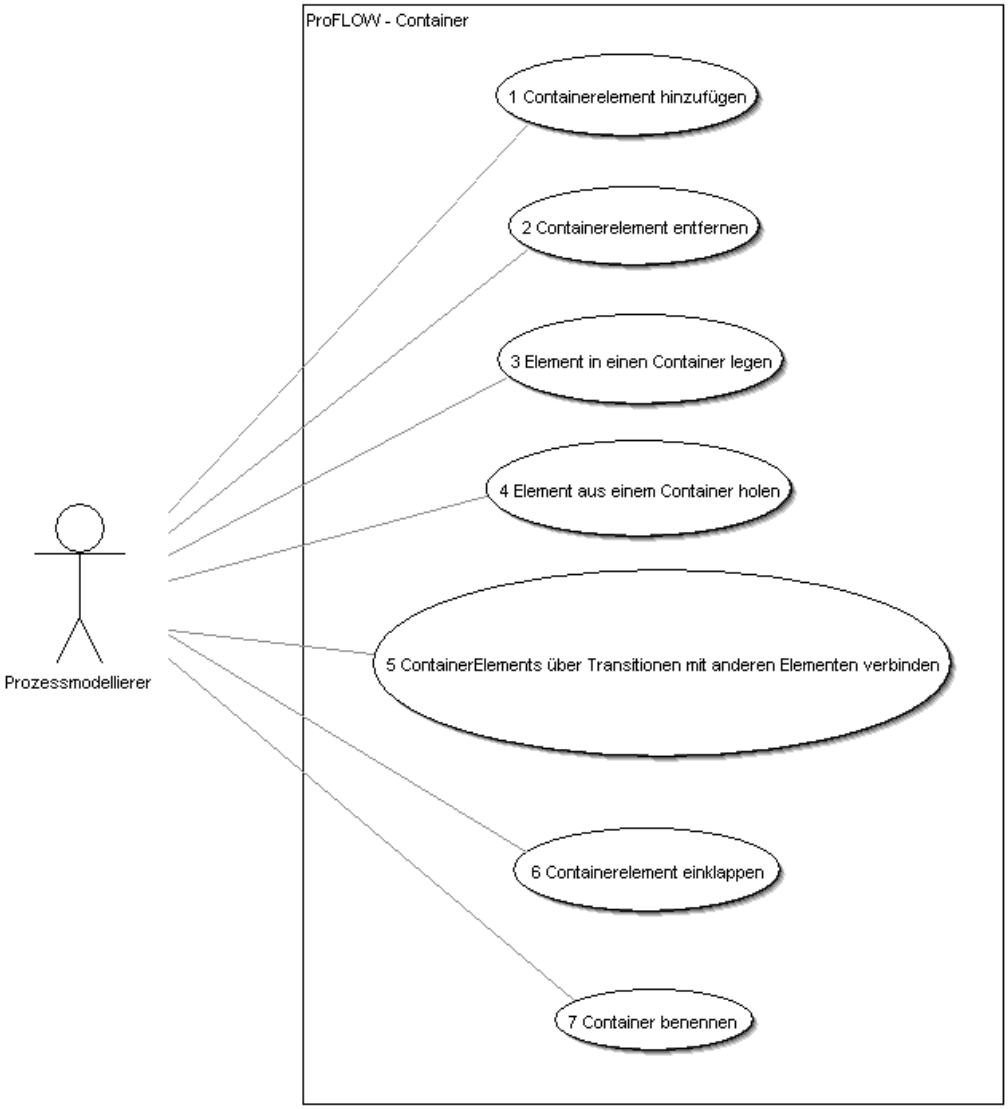

Abbildung 2.12: UseCase Übersicht

Containerelement hinzufügen

<table>
<tr><td>Use Case Nr.1</td><td>Containerelement hinzufügen</td></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Zum Hinzufügen stehen nur die im Diagramm bereitgestellten Containerelemente zur Verfügung.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td><table><tr><td>Prozessmodellierer</td><td>Containerelement im Diagramm verwenden</td></tr></table></td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet</td></tr>
<tr><td>Garantien</td><td>./.</td></tr>
<tr><td>Erfolgsfall</td><td>Containerelement wurde an gewünschter Position hinzugefügt</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer wählt Containerelement in der Palette aus und wählt eine Stelle im Diagramm an</td></tr>
<tr><td>Beschreibung</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>1</td><td>Prozessmodellierer wählt Containerelement in der Palette aus</td></tr><tr><td>2</td><td>Prozessmodellierer wählt auf eine Stelle im Diagramm</td></tr><tr><td>3</td><td>Containerelement erscheint im Diagramm</td></tr></table></td></tr>
<tr><td>Erweiterungen</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>3a</td><td>WENN Einfügen an dieser Stelle nicht erlaubt, DANN wird das Containerelement nicht hinzugefügt</td></tr></table></td></tr>
</table>

Containerelement entfernen

<table>
<tr><th>Use Case Nr.2</th><th>Containerelement entfernen</th></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Es können nur Elemente aus dem gerade geöffneten Diagramm entfernt werden.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td>
<table>
<tr><td>Prozessmodellierer</td><td>Nicht benötigte Containerelemente löschen</td></tr>
</table>
</td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet, welches min. ein Containerelement enthält</td></tr>
<tr><td>Garantien</td><td>Containerelement wird gelöscht</td></tr>
<tr><td>Erfolgsfall</td><td>Containerelement, alle im Container enthaltenen Elemente und alle beteiligten Transitionen wurden gelöscht</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer wählt Containerelement im Diagramm aus und initiiert die Löschkaktion</td></tr>
<tr><td>Beschreibung</td><td>
<table>
<tr><td>Schritt</td><td>Aktion</td></tr>
<tr><td>1</td><td>Prozessmodellierer wählt Containerelement im Diagramm aus</td></tr>
<tr><td>2</td><td>Prozessmodellierer initiiert die Löschaktion</td></tr>
<tr><td>3</td><td>Containerelement, alle im Container enthaltenen Elemente und alle beteiligten Transitionen werden gelöscht</td></tr>
</table>
</td></tr>
<tr><td>Erweiterungen</td><td>./.</td></tr>
</table>

Element in einen Container legen

<table>
<tr><th>Use Case Nr.3</th><th>Element in einen Container legen</th></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Es können nur im Diagramm verfügbare Elemente in den Container gelegt werden.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td><table><tr><td>Prozessmodellierer</td><td>Elemente einem Container unterordnen.</td></tr></table></td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet, welches min. ein Containerelement enthält.</td></tr>
<tr><td>Garantien</td><td>./.</td></tr>
<tr><td>Erfolgsfall</td><td>Element befindet sich an der gewünschten Stelle im Container.</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer zieht ein Element aus dem Diagramm über das Containerelement</td></tr>
<tr><td>Beschreibung</td><td><table><tr><th>Schritt</th><th>Aktion</th></tr><tr><td>1</td><td>Prozessmodellierer zieht ein Element aus dem Diagramm über das Containerelement.</td></tr><tr><td>2</td><td>Das betroffene Containerelement wird hervorgehoben.</td></tr><tr><td>3</td><td>Das Element erscheint an der gewünschten Stelle im Container.</td></tr></table></td></tr>
<tr><td>Erweiterungen</td><td><table><tr><th>Schritt</th><th>Aktion</th></tr><tr><td>3a</td><td>WENN das Einfügen dieses Elements in den Container nicht erlaubt ist, DANN bleibt das Element an seiner alten Position</td></tr></table></td></tr>
</table>

Element aus einem Container holen

<table>
<tr><td>Use Case Nr.4</td><td>Element aus einem Container holen</td></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Diese Funktion ist auf Elemente beschränkt, die sich innerhalb eines Containerelements befinden.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td><table><tr><td>Prozessmodellierer</td><td>Elemente aus einem Container holen.</td></tr></table></td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet, welches min. ein nicht leeres Containerelement enthält.</td></tr>
<tr><td>Garantien</td><td>./.</td></tr>
<tr><td>Erfolgsfall</td><td>Element befindet sich an der gewünschten Stelle im Diagramm außerhalb des Containers.</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer zieht ein Element aus dem Container heraus.</td></tr>
<tr><td>Beschreibung</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>1</td><td>Prozessmodellierer zieht ein Element aus dem Container heraus.</td></tr><tr><td>2</td><td>Das Element erscheint an der gewünschten Stelle außerhalb des Containers.</td></tr></table></td></tr>
<tr><td>Erweiterungen</td><td>./.</td></tr>
</table>

ContainerElements über Transitionen mit anderen Elementen verbinden

<table>
<tr><td>Use Case Nr.5</td><td>ContainerElements über Transitionen mit anderen Elementen verbinden</td></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Es stehen nur die im Diagramm bereitgestellten Transitionen zur Verfügung und das Containerelement muss eingeklappt sein.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td><table><tr><td>Prozessmodellierer</td><td>Transition zwischen Containerelement und einem anderen Element zeichnen.</td></tr></table></td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet, welches min. ein Containerelement und ein anderes Element beinhaltet.</td></tr>
<tr><td>Garantien</td><td>./.</td></tr>
<tr><td>Erfolgsfall</td><td>Das Containerelement wird mit dem anderen Element durch eine Transition verbunden.</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer wählt eine Transition in der Palette aus, wählt in entsprechender Reihenfolge ein Element und ein Containerelement im Diagramm an.</td></tr>
<tr><td>Beschreibung</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>1</td><td>Prozessmodellierer wählt eine Transition in der Palette aus.</td></tr><tr><td>2</td><td>Prozessmodellierer wählt entweder ein Containerelement oder ein SimpleElement im Diagramm an.</td></tr><tr><td>3</td><td>Prozessmodellierer wählt anschließend ein weiteres Element (auch Containerelement) an.</td></tr><tr><td>4</td><td>Die Transition erscheint zwischen den angewählten Elementen im Diagramm.</td></tr></table></td></tr>
<tr><td>Erweiterungen</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>4a</td><td>WENN eine Transition zwischen diesen Elementen wegen Diagrammregeln nicht erlaubt ist, DANN wird die Transition nicht hinzugefügt.</td></tr></table></td></tr>
</table>

Containerelement einklappen

<table>
<tr><th>Use Case Nr.6</th><th>Containerelement einklappen</th></tr>
<tr><td>Umfeld</td><td>Computer mit Eclipse IDE und ProFLOW</td></tr>
<tr><td>Systemgrenzen</td><td>Es können nur ausgeklappte Containerelemente aus dem Diagramm eingeklappt werden.</td></tr>
<tr><td>Ebene</td><td>Hauptaufgabe</td></tr>
<tr><td>Hauptakteur</td><td>Prozessmodellierer</td></tr>
<tr><td>Stakeholder u. Interessen</td><td><table><tr><td>Prozessmodellierer</td><td>Grob- und Detailansicht auf ein Diagramm ermöglichen.</td></tr></table></td></tr>
<tr><td>Voraussetzungen</td><td>Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet</td></tr>
<tr><td>Garantien</td><td>./.</td></tr>
<tr><td>Erfolgsfall</td><td>Containerelement wurde eingeklappt, alle Elemente im Container sind nicht mehr sichtbar. Transitionen sind weiterhin sichtbar, wobei Transitionen, die an einem Element im Container hängen, in das Containerelement hineinragen.</td></tr>
<tr><td>Auslöser</td><td>Prozessmodellierer selektiert ein Containerelement im Diagramm aus und initiiert das Einklappen des Containerelements.</td></tr>
<tr><td>Beschreibung</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>1</td><td>Prozessmodellierer wählt Containerelement im Diagramm aus.</td></tr><tr><td>2</td><td>Prozessmodellierer initiiert ’Einklappen’.</td></tr><tr><td>3</td><td>Containerelement erscheint eingeklappt im Diagramm.</td></tr></table></td></tr>
<tr><td>Erweiterungen</td><td><table><tr><td>Schritt</td><td>Aktion</td></tr><tr><td>3a</td><td>WENN das Einklappen des Containers wegen Diagrammregeln nicht erlaubt ist, DANN erfolgt keine Aktion.</td></tr></table></td></tr>
</table>

Containerelement bennen

Use Case Nr.7	**Containerelement benennen**
Umfeld	Computer mit Eclipse IDE und ProFLOW
Systemgrenzen	Es können nur Container in einem geöffneten Diagramm benannt werden.
Ebene	Hauptaufgabe
Hauptakteur	Prozessmodellierer
Stakeholder u. Interessen	Prozessmodellierer: Einem Containerelement einen Namen zuweisen.
Voraussetzungen	Prozessmodellierer hat ein ProFLOW Diagramm in Eclipse geöffnet
Garantien	./.
Erfolgsfall	Containerelement wird mit seinem neuen Namen im Diagramm angezeigt.
Auslöser	Prozessmodellierer klickt doppelt auf ein Containerelement im Diagramm.
Beschreibung	
Erweiterungen	./.

Schritt	Aktion
1	Prozessmodellierer klickt doppelt auf ein Containerelement im Diagramm.
2	Eingabefeld erscheint.
3	Prozessmodellierer tippt den Namen ein und bestätigt.
4	Containerelement wird mit seinem neuen Namen im Diagramm angezeigt.

2.3 Nichtfunktionale Anforderungen

Usability

Prozessmodellierer erwarten eine intuitive Bedienung der Containerelemente. Dazu gehört z.B. das Einfügen oder Entfernen von Elementen per Drag&Drop aus dem Container. Das Einklappen der Container erfolgt über einen Button in der Linken oberen Ecke des Containers. Dieser erscheint nur dann, wenn der Container einklappbar ist. Ist der Container eingeklappt, stellt der Button ein „Plus“-Symbol dar und löst das Ausklappen aus. Dies hat einen Informationsgewinn zu Folge, was dem Benutzer durch das „Plus“-Symbol suggeriert wird. Im umgekehrten Fall, enthält der Button ein „Minus“-Symbol.

Messung
Diese Qualitätsanforderung ist nicht einfach zu messen. Eine relativ gute Möglichkeit dafür wäre, jede mit Containern verbundene Funktion mit deren Umsetzung in anderen weit verbreiteten Applikationen zu vergleichen. Die Benutzer müssten sich dann nicht auf eine ungewohnte Situation einstellen, sondern wüssten sofort, wie sie mit Containern umgehen können. Sollwerte für die Messung wären hier die Umsetzung dieser Funktionen in weit verbreiteten Modellierungstools.

Erweiterbarkeit und Wartbarkeit

Das ProFLOW Framework wird in Zukunft von Studenten und wissenschaftlichen Mitarbeitern weiterentwickelt und gepflegt werden. Hierfür werde ich mich an bestehende Konzepte, die in ProFLOW angewandt wurden, halten. Dazu gehört auch die im bestehenden Code angewandte Codekonvention. So bleibt der Code für die zukünftigen Entwickler lesbar und verständlich.

Messung
Diese Qualitätsanforderung ist dann erfüllt, wenn die betroffenen Klassen und Methoden gut dokumentiert sind und der Code lesbar ist. Betroffen sind Model- und Controller-Klassen, die im Kapitel „Entwurf“ vorgestellt werden.

Kompatibilität

Die Erweiterung von ProFLOW um Containerelemente darf die Funktion von bestehenden ProFLOW Diagrammen nicht beeinträchtigen. Ein Update auf die neueste Version, die Containerelemente unterstützt, soll keine Änderungen am Code der Diagramme zur Folge haben.

Messung
Dies überprüfe ich, indem ich ProFLOW Diagramme, die auf Basis der Version 1.0 des Frameworks implementiert wurden, mit der neuen Frameworkversion ausführe und

nach Fehlfunktionen suche. Die Anforderung ist komplett erfüllt, wenn keine Anpassungen der ProFLOW Diagramme an das neue Framework nötig sind.

Flexibilität

Viele Notationen haben, neben dem bestimmten Aussehen der einzelnen Elemente, spezielle Eigenschaften, die man bei anderen Notationen nicht findet. Das Verhalten eines Containers ist im Framework vorgegeben, sollte aber nicht „final“ sein. Der Diagrammentwickler soll die Möglichkeit haben, das Verhalten der Container an diese speziellen Eigenschaften anzupassen.

Messung
Dies wird gewährleistet, indem das Verhalten der Container zu einem Teil oder, wenn notwendig, komplett geändert bzw. erweitert werden kann. Die Architektur dazu wird im Kapitel zum Entwurf genauer erläutert. Diese Anforderung ist komplett erfüllt, wenn das Verhalten des Containers für jedes Szenario änderbar bzw. erweiterbar ist.

3 Konzepte

Im folgenden Abschnitt werden Konzepte erläutert und diskutiert, welche bei der Umsetzung von bestimmten Anforderungen eine Rolle spielen.

3.1 Ankerelemente

Jeder Elementtyp in ProFLOW, der als Quelle oder Ziel für eine Transition dient, hat eine bestimmte Menge an Ankerpunkten, an die die Transitionen andocken können. Es wird stets der Punkt gewählt, so dass die Länge der Transition minimal ist. Folglich ist diese Art von Ankern nicht geeignet, um eine definierte Schnittstelle eines Containers darzustellen, dessen Position eine Semantik hat. Je nach Position im Diagramm, dockt die Transition an einer anderen Stelle am Element an.

Eine Lösung für dieses Problem ist das Einführen von Ankerelementen. Ankerelemente sind *SimpleElements* die eine eigene Position im Diagramm haben und nur in Containern hinzugefügt werden. In Abb. 3.1 sieht man Ankerelemente am Beispiel eines FLOW Diagramms.

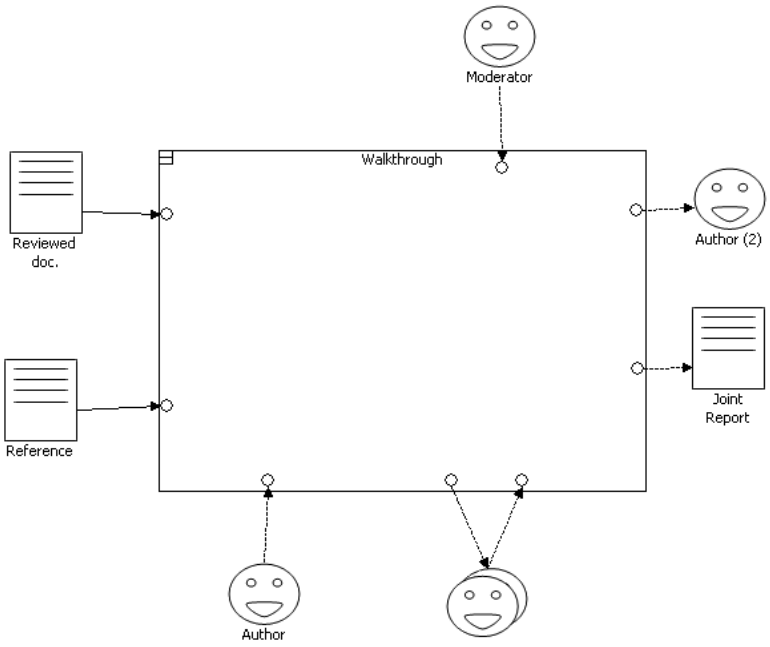

Abbildung 3.1: FLOW Signatur eines Walkthroughs (ausgeklappt)

Die Wissensflüsse sind an den Ankerelemente innerhalb des Containers angedockt. Von dort aus kann man den Fluss zu Elementen, die im Block „Walkthrough“ liegen weiterleiten und den Prozess so verfeinern. Klappt man den Block ein (siehe Abb. 3.2), docken die Wissensflüsse am Walkthrough an.

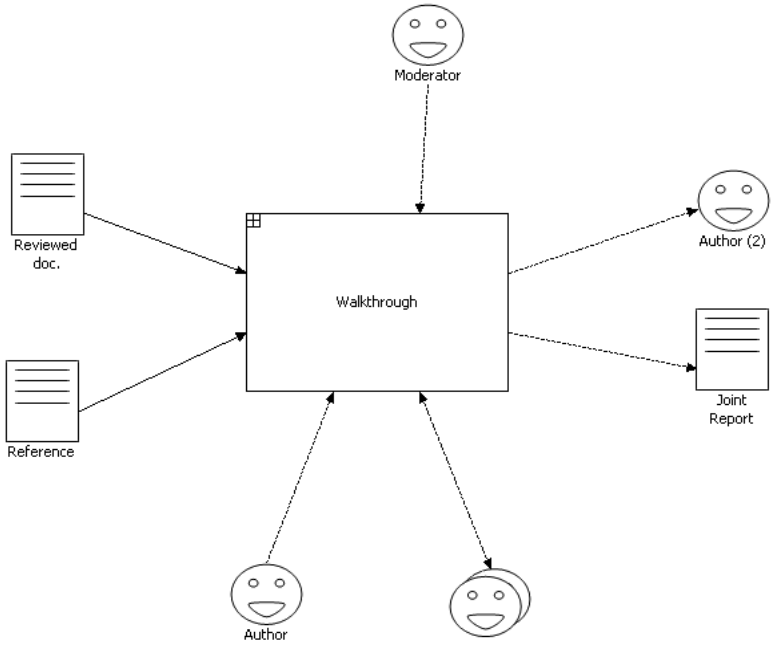

Abbildung 3.2: FLOW Signatur eines Walkthroughs (eingeklappt)

3.2 Ein- und Ausklappen von Containern

Das Ein- und Ausklappen von Containern wird mit dem Plus bzw. Minusknopf ausgelöst. Dieser Button erscheint nur bei einklappbaren Containern in der linken oberen Ecke. Das Plus zum Ausklappen suggeriert dem Benutzer, dass mehr Informationen sichtbar werden, nachdem der Plus-Button bedient wurde. Dies trifft auch zu, da der Containerinhalt erscheint. Beim Minus ist das Gegenteil der Fall. (Siehe Abb. 3.1 und Abb. 3.2)

Das Konzept sollte den meisten Benutzern bekannt sein, da es in vielen Anwendungen verwendet wird. Bei Verzeichnisstrukturen, die in Baumform angezeigt werden, haben Verzeichnisse mit Unterverzeichnissen ein Plus-Symbol. Damit kann man das Verzeichnis ausklappen und die Unterverzeichnisse werden sichtbar. Über das Minus-Symbol lässt sich die Aktion rückgängig machen. Die Analogie zu den Plus- bzw. Minus-Button bei den Container ist also vorhanden.

3.3 Darstellung von Transitionen bei zugeklappten Containern

In den meisten Fällen docken Transitionen direkt am Rand eines Elements an. Es können aber Sonderfälle auftreten.

Ein Sonderfall kommt im ER-Diagramm vor, welches in Kapitel 2.2.1 vorgestellt wurde. Betrachten wir nun den Fall, in dem in Abb. 2.3 „Inventory“ eingeklappt ist. Die Subentitäten und die Beziehungen zwischen ihnen werden ausgeblendet. Die Beziehung zwischen „Inventory“ und „Mfg. Location“ bleibt bestehen. Für die Beziehung zwischen „Vehicle“, welches nicht mehr sichtbar ist, und „Model“ gibt es nun mehrere Möglichkeiten:

1. die Beziehung wird ebenfalls ausgeblendet
2. das Aussehen des Pfeils, der die Beziehung repräsentiert, wird geändert (z.B. gestrichelte Linie, andere Farbe)
3. die Beziehung geht direkt von „Model“ zu „Inventory“
4. man lässt die Beziehung in „Inventory“ hineinlappen, um deutlich zu machen, dass eine Subentität statt „Inventory“ an der Beziehung beteiligt ist

Die erste Möglichkeit hätte einen unerwarteten Informationsverlust zur Folge, da ein Element außerhalb des Containers an der Beziehung teilnimmt. Zu erwarten wäre das Ausblenden von Elementen und Transitionen, die komplett im Container liegen. Transitionen liegen nur dann komplett im Container, wenn sie zwei Elemente verbinden, die sich in diesem Container befinden. Dieser Ansatz ist also nicht optimal.

Die zweite Möglichkeit führt ein nicht spezifiziertes Notationselement ein. Das Abweichen von der Spezifikation bzw. von Standards sollte möglichst vermieden werden, da außenstehende Betrachter ohne Legende die Bedeutung dieses Elements nicht verstehen würden. Standardisierte Notationen sind dafür da, um genau diesen Fall zu vermeiden.

Option 3 wäre nicht mehr äquivalent zum ER-Diagramm mit aufgeklapptem „Inventory“. „Model“ stände nun mit „Inventory“ in Beziehung. Option 3 ist hier also ausgeschlossen.

Die letzte Möglichkeit ist für diese Situation die optimale. Es gibt keinen unerwarteten Informationsverlust, da die Beziehung bestehen bleibt. Der Verstoß gegen die Spezifikation ist nicht so grob wie in Option 2 und das Hineinlappen unterscheidet sich von einem direkten Andocken an das Element. Es wird signalisiert, dass die Transition in den Container hineingeht und das sich im Container etwas befinden muss. Dies ist hier erwünscht. Deswegen habe ich mich für die letzten Option entschieden.

4 Entwurf

ProFLOW ist ein Eclipse Plugin und ist in Java geschrieben. Es basiert auf dem Graphical Editing Framework (GEF), das nach dem Model View Controller Pattern ein Controller Framework und ein View (Draw2D) liefert. Mit diesem Controller Framework lässt sich ein beliebiges Model auf das View abbilden und es können Regeln festgelegt werden, wie Interaktionen mit dem View sich auf das Model auswirken. [3]

Das ProFLOW Framework hat eine bestimmte Paketstruktur, die Model-, Controller- und View-Klassen voneinander trennen. Diese sollte bei der Entwicklung eines neuen Diagrammtypen eingehalten werden. Im diesem Abschnitt unterscheide ich zwischen verschiedenen Ebenen: der GEF-, ProFLOW Framework- und ProFLOW Diagrammebene. Meine Arbeit fand hauptsächlich auf der ProFLOW Frameworkebene statt.

In den folgenden Klassendiagrammen sind von mir erstellte oder erweiterte Klassen grau gefärbt. Es sind auch nur die wesentlichen Methoden aufgeführt, um die Funktion der Klassen zu erläutern. Im Text sind Klassen kursiv dargestellt.

4.1 View

Das View wird in GEF von Figure Klassen repräsentiert. Ein Figure-Objekt kann beliebig viele Kind-Figures enthalten. Das Diagramm, welches der Benutzer sieht, besteht also aus einem Figure-Baum. Für jedes Element lässt sich ein View mit Hilfe von einer oder mehrere Figures erstellen. Dies geschieht auf der ProFLOW Diagrammebene, da in jeder Notation die Elemente eine unterschiedliche Erscheinung haben.

Für oft verwendete Formen wie Polygone, Rechtecke oder ganze Bilder stehen schon vorgefertigte Figure-Klassen bereit, die sich auch weiter anpassen lassen. Diese waren schon in der finalen Version von Andreas Meissner, dem Entwickler von ProFLOW, implementiert, so dass ich auf diesen Teil nicht weiter eingehen werde.

4.2 Model

Das Model des ProFLOW Frameworks hat eine Oberklasse *ModelObject*, alle Elemente erben von dieser Klasse. Sie enthält Metadaten wie Name, Größe und Position von Elementen im Diagramm. *Element* repräsentiert Elemente im Diagramm, welche mit Hilfe von Transitionen mit anderen Elementen verbunden werden können. Ein *Element* kann auch Annotationen haben.

Von *Element* erben *SimpleElement* und *ContainerElement*. Ein *SimpleElement* ist in

ProFLOW ein atomares Element. In Abb. 4.1 sieht man links das Model Package der Implementation des Aktivitätsdiagramms. Eine nichtzusammengesetzte Aktivität (*Activity*) erbt von *SimpleElement*.

ContainerElement repräsentiert zusammengesetzte Elemente, die Container. Es können Elemente hinzugefügt und gelöscht werden, der Container kann einklappbar sein und seine Kindelemente ausgeben. Auf einer weiteren Ebene wird zwischen *Process* und *SubProcess* unterschieden. Beides sind Container.

Der *Process* ist das Diagramm, es enthält alle Elemente, die der Prozessmodellierer hinzufügt. Ein *Process* hat kein richtiges Aussehen, er stellt die Zeichenfläche dar. Ein *SubProcess* ist ein Container, der dem Prozessmodellierer als Notationselement zur Verfügung stellt. Es ist im Diagramm sichtbar, kann Elemente enthalten und der Benutzer kann mit diesem Container interagieren. In Abb. 4.1 wird dies deutlich. Das *ActivityDiagram*, das Diagramm, ist ein *Process*, eine *ComplexActivity* ist ein *SubProcess*, der ein untergeordnetes Aktivitätsdiagramm enthält.

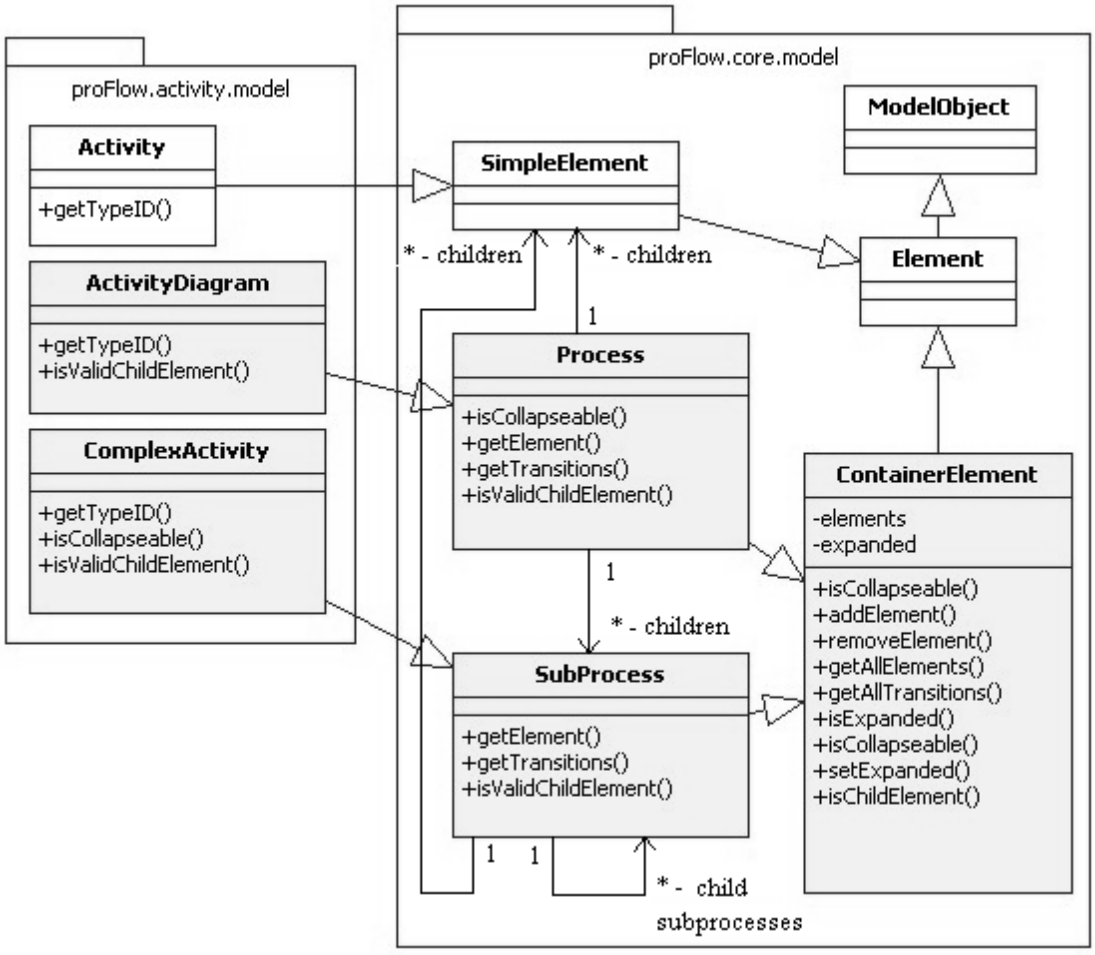

Abbildung 4.1: Model vom ProFLOW Framework und ProFLOW Activity

4.3 Controller

Der Controller ist in GEF der umfangreichste und mächtigste Teil. Jedes im Diagramm dargestellte Element, mit dem der Benutzer interagieren kann, hat einen EditPart. Ein EditPart ist jeweils der Model-Instanz eines Elements im Diagramm zugeordnet und reagiert auf Änderungen im Model, die es dann dem View mitteilt (z.B. Name des Elements wurde geändert). Dies wurde anhand des Observer Patterns von Andreas Meissner entworfen und umgesetzt. Dieses Entwurfsschema habe ich bei den Containern weiterverfolgt.

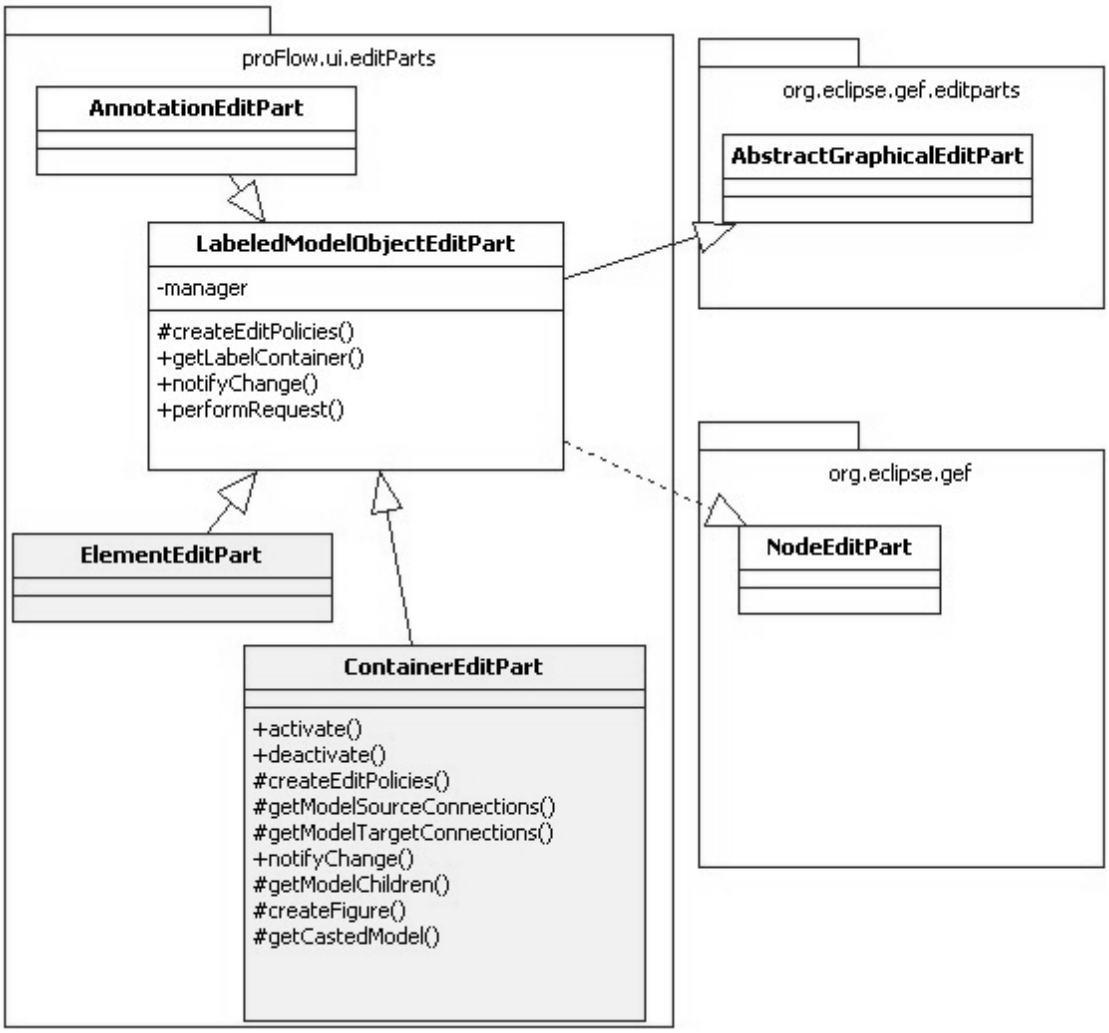

Abbildung 4.2: EditParts im ProFLOW Framework

Im ProFLOW Framework ist *LabeledModelObjectEditPart* die Oberklasse aller vom Elementtyp abhängigen EditParts. *LabeledModelObjectEditPart* teilt dem View Änderungen wie z.B. Größen-, Positions, und Namensänderungen mit. Es implementiert die Schnittstelle *NodeEditPart* aus GEF und hat so die Rolle einer grafischen Node im

Diagramm. Über diese Schnittstelle werden die vom Benutzer ausgelösten Requests mitgeteilt.

ElementEditPart und *ContainerEditPart* erben von dieser Klasse, wobei sie das Verhalten entsprechend erweitern. Ein *ContainerEditPart* reagiert auf Events wie z.B. das Ein- oder Ausklappen des Containers und das Hinzufügen oder Entfernen eines Elements. Auf der nächsten Ebene kommen schon die EditPart- Klassen, die sich auf der ProFLOW Diagrammebene befinden. Zu der *ComplexActivity* gehört dann die Klasse *ComplexActivityEditPart*, welches von *ContainerEditPart* abgeleitet ist. In *ComplexActivityEditPart* ist schließlich das Aussehen des Containers definiert. Zu *Activity* gehört dann entsprechend *ActivityEditPart*, welches von *ElementEditPart* abgeleitet ist und Verhalten für *SimpleElements* festlegt.

Zusätzlich werden im EditPart sogenannte EditPolicies eingerichtet. Diese reagieren auf Requests, die vom Benutzer ausgelöst werden. Jede Interaktion mit dem Diagramm (z.B. das Verschieben eines Elements, Größenänderung eines Elements) löst einen bestimmten Request aus. Die EditPolicies geben anhand des Requests eine bestimmtes Kommando (*Command*) zurück, welches ausgeführt wird und Änderungen am Model durchführt.

In Abb. 4.3 ist das Klassendiagramm für das ProFLOW EditPolicy Paket dargestellt. Ich habe ausschließlich neue EditPolicies für ContainerElemente implementiert und das vorhandene *ProcessXYLayoutEditPolicy* umstrukturiert. Dabei habe ich Methoden nach *ContainerXYLayoutEditPolicy* verschoben, die für Elemente des Typs *Process* und *SubProcess* gleich sind, da sie sich gleich verhalten, wenn z.B. Elemente hinzugefügt werden. In *ContainerElementXYLayoutEditPolicy* ist noch zusätzlich das Verhalten für die Fälle spezifiziert, wo Elemente im Container Verschoben oder deren Größe geändert wird. Alle diese Klassen sind ein Untertyp von *XYLayoutEditPolicy* aus GEF. Dies sorgt dafür, dass die Elemente im Diagramm in einem kartesischen Koordinatensystem gezeichnet werden und jedes Element eine Position relativ zum Nullpunkt besitzt.

ContainerHighlightEditPolicy sorgt für eine farbliche Hervorhebung des Containers, wenn ein bestimmter Fall eintritt (z.B. Benutzer zieht ein Element über einen Container, um es in den Container zu legen).

ContainerElementComponentEditPolicy sorgt dafür, dass ContainerElemente gelöscht werden können.

In Abb. 4.4 ist eine Übersicht der Command Klassen in ProFLOW. *AddToContainerCommand* und *MoveToContainerCommand* sind von mir implementiert und führen wie die Namen andeuten das Hinzufügen eines neuen Elementes bzw. eines schon im Diagramm vorhandenen Elements in einen Container durch. *CreateTransitionCommand* und *ChangeTransitionCommand* habe ich modifiziert, so dass auch Container als eine Quelle bzw. ein Ziel für Transitionen dienen können.

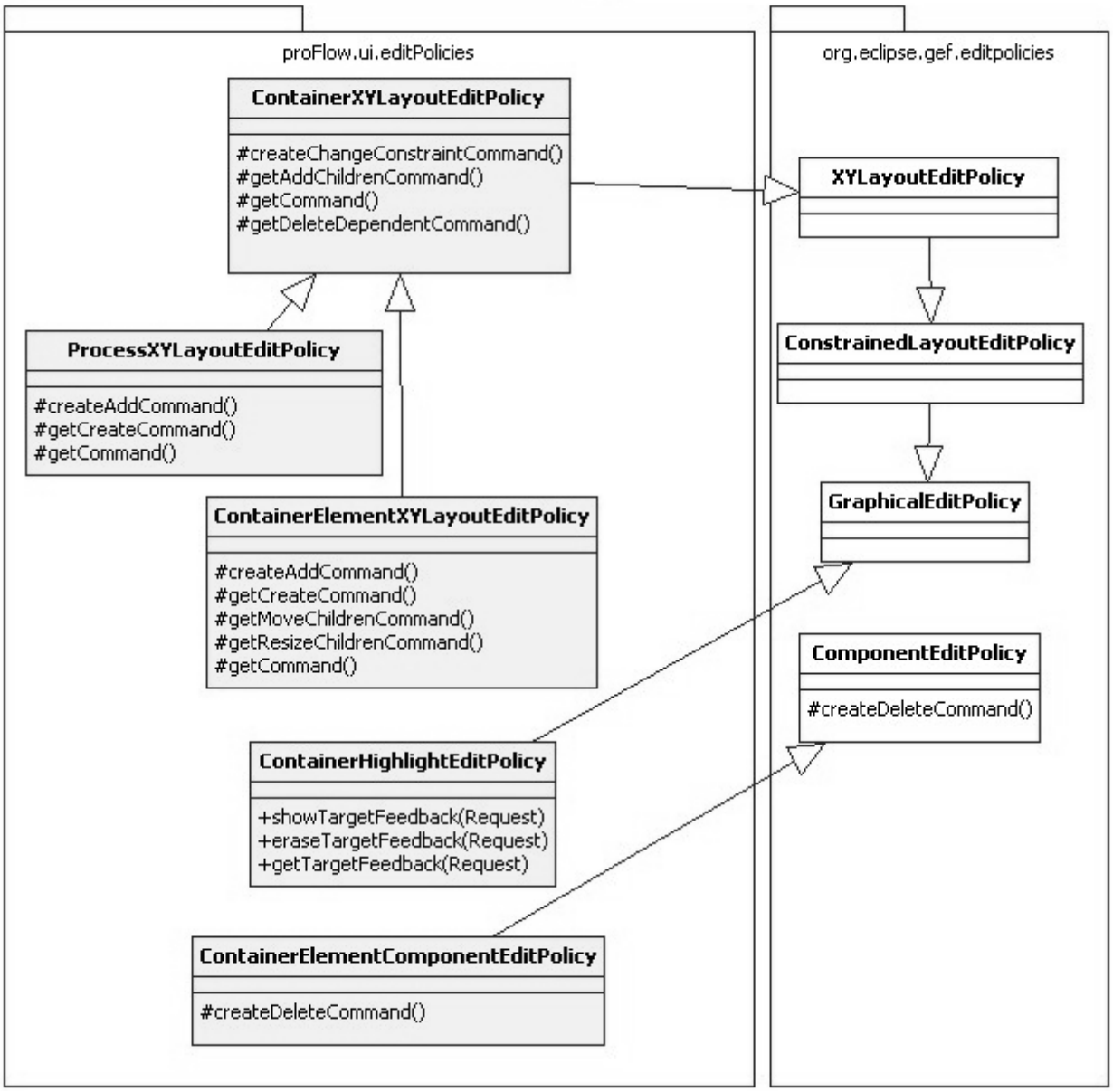

Abbildung 4.3: EditPolicies im ProFLOW Framework

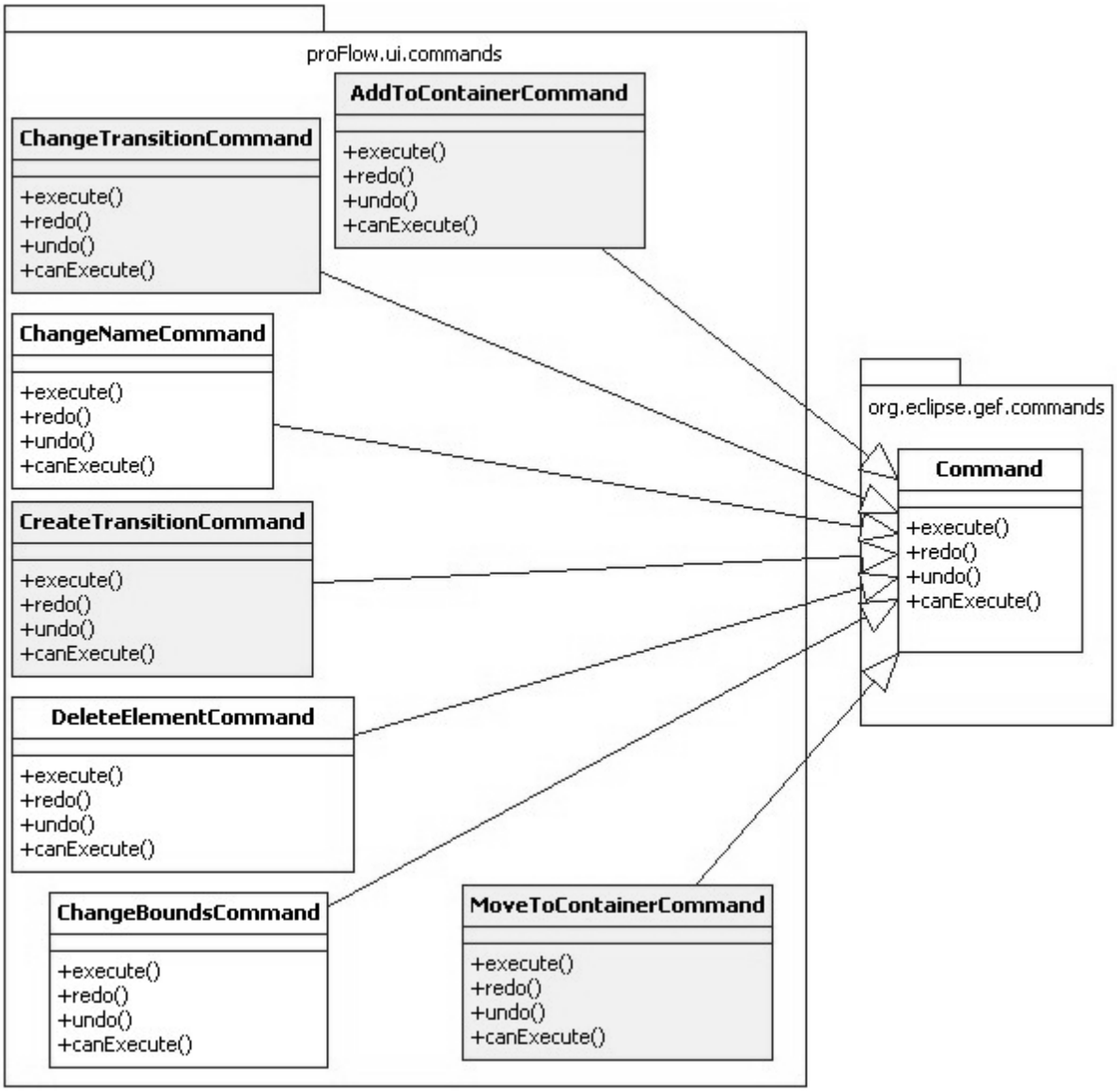

Abbildung 4.4: Commands im ProFLOW Framework

5 Implementierung

In diesem Abschnitt fasse ich zusammen, in welchem Umfang die funktionalen Anforderungen umgesetzt worden sind. Außerdem wird geprüft, ob die Qualitätsanforderungen eingehalten wurden und welche bekannten Probleme bzw. Bugs existieren.

5.1 Funktionale Anforderungen

Alle funktionalen Anforderungen konnten gemäß dem Zeitplan in der Implementationsphase fertiggestellt werden. Bis auf einige Bugs, die erst später aufgefallen sind und behoben wurden, musste die Implementationsphase nicht weiter verlängert oder auf Funktionen verzichtet werden.

5.2 Qualitätsanforderungen

In diesem Abschnitt führe ich die Messungungen für der einzelnen Qualitätsanforderungen durch und fasse kurz zusammen, in welchem Umfang diese erfüllt worden sind.

Usability

Untersucht werden die Funktionen, die in Kapitel 2.2.3 mit einem UseCase spezifiziert worden sind. Die Anforderungen in der folgenden Liste sund nach den UsaCase Nummern nummeriert. Als Referenzapplikationen werden die Modellierungstools StarUML und Omondo betrachtet, da sich Prozessmodellierer an die Bedienung von gängigen Modellierungstools gewöhnt haben.

1. **Containerelement hinzufügen**
   Diesen Vorgang kennt man aus oben aufgezählten Tools, wo in der Toolbar das verwendete Element ausgewählt und auf der Zeichenfläche verwendet wird. Dieses Prinzip wurde auch in ProFLOW angewandt. In Omondo kann man beim Hinzufügen von Packages einen Rahmen um schon vorhandene Klassen ziehen, so dass nach dem Hinzufügen die Klassen zum Package hinzugefügt werden. Dies ist in ProFLOW jedoch nicht möglich.
2. **Containerelement entfernen**
   Ein Container wird markiert und über eine Aktionstaste (Entf) das Entfernen ausgelöst. Der Inhalt wird ebenfalls gelöscht. Genauso verläuft das Löschen von

Packages in Omondo, die man zuerst markiert und dann mit der Taste „Entf“ löscht, wobei die im Package enthaltenen Klassen auch gelöscht werden.

3. **Element in einen Container legen**
4. **Element aus einem Container holen**
   Die Funktionen 3 und 4 sind auch analog zu dem Verschieben von Klassen von einem Package ins andere in Omondo implementiert. Dabei markiert man einen oder mehrere Elemente (in Omondo Klassen) und zieht diese aus oder in den Container (in Omondo verhalten sich Packages wie Container). Der Container wird auch in Omondo dabei farblich hervorgehoben.
5. **ContainerElements über Transitionen mit anderen Elementen verbinden**
   Das Zeichnen einer Transition ist in vielen Tools so implementiert, dass nacheinander Elemente ausgewählt werden. Danach erscheint die Transition im Diagramm. Bei gerichteten Transitionen spielt die Reihenfolge, nach der Elemente ausgewählt werden, eine Rolle. Dies findet man in oben genannten Modellierungstool (z.B. eine Vererbung zwischen zwei Klassen in Form eines gerichteten Pfeils, der zur Oberklasse zeigt). Genauso werden in ProFLOW Transitionen gezeichnet.
6. **Containerelement einklappen**
   In ProFLOW wird zum Einklappen (und Aufklappen) von Containern das Konzept aus Kapitel 3.2 verwendet. In diesem Kapitel wurde erläutert, warum ich aus Gründen der Usability mich für dieses Konzept entschieden habe. Aus technischen Gründen muss der Container vorher ausgewählt werden, da sonst der Klick auf den Button nicht registriert wird. Dies weicht vom Sollwert ab, da man erwartet, dass der Button auch dann funktioniert, wenn der Container nicht ausgewählt ist.
7. **Containerelement bennen**
   Um den Namen eines Containers zu ändern ist ein Doppelklick nötig. In StarUML lässt sich der Name von Klassen oder Packages auch mit einem Doppelklick ändern.

Zusammenfassend kann man sagen, dass die Anforderungen an die Usability bis auf zwei Kleinigkeiten den Sollwerten aus den bekannten Modellierungstools entsprechen. Die beiden abweichenden Punkte sollten im Rahmen der Weiterentwicklung eingepflegt werden, um das Arbeiten mit ProFLOW intuitiver und bequemer zu gestalten.

Erweiterbarkeit und Wartbarkeit

Die Klassen und Methoden, um die ich das Framework erweitert habe, sind dokumentiert und ich habe mich an die in ProFLOW verwendeten Codekonventionen gehalten. Somit ist diese Anforderung erfüllt.

Kompatibilität

Die Überprüfung der Kompatibilität war für alle untersuchten Diagramme erfolgreich. Getestet wurden eEPK und die drei Diagrammtypen „UseCasediagram“, „Classdiagram“ und „Activitydiagram“, die im Rahmen der Bachelorarbeit „Ein grafischer Editor mit Annotationsmöglichkeit für SPEM-Modelle“ von Andelko Jovancevic entstanden sind. Es konnte für jeden Diagrammtyp ein neues Diagramm erstellt und alle Elemente zum Modellieren verwendet werden.

Die Anforderung ist ebenfalls erfüllt. Andere existierende Diagrammtypen sind älter und nicht kompatibel zur finalen Version von Andreas Meissner. Daher konnte ich sie nicht in die Tests einbeziehen.

Flexibilität

Im EditPart eines Containers kann die Methode createEditPolicies() überschrieben werden. In dieser werden die im Kapitel „Entwurf“ erläuterten EditPolicies gesetzt, die das Verhalten des Containers definieren. Es ist also möglich, eigene EditPolicies oder Erweiterungen der EditPolicies aus dem Framework zu verwenden. Ein Diagrammentwickler kann also für jeden Fall flexibel auf abweichende Anforderungen einer Notation reagieren.

5.3 Bekannte Probleme

1. Bei aufgeklappten Containern, die ein abgerundetes Rechteck (RoundedRectangle) als Figure haben, werden bei einer Verschiebung des Containers die enthaltenen Elemente an einer falschen Position gezeichnet. Das Problem wurde für einen betroffenen Container (ComplexActivity im Aktivitätsdiagramm) aus Zeitmangel nur notdürftig gelöst.
2. Das Ein- und Ausklappen von Containern wird nicht über Commands eingeleitet, so dass keine Änderung am Model registriert wird. Dadurch lässt sich das Diagramm nach einem Einklappen oder Ausklappen nicht speichern, sondern erst, wenn eine weitere Änderung im Diagramm getätigt wird.

6 Verbesserungsvorschläge und Fazit

Dies ist das letzte Kapitel meiner Bachelorarbeit. Hier werden meine Anregungen und Ideen für die Verbesserung von ProFLOW gesammelt. Abschließend formuliere ich ein Fazit.

6.1 Verbesserungsvorschläge

Die vorhandene Liste mit Feature Requests für ProFLOW wird hier um neue Ideen von mir ergänzt.

Input / Output Schnittstellen für Container

Ein gutes Beispiel für Schnittstellen sind die Input- und Outputpins von Aktivitäten vom UML 2 Aktivitätsdiagramm (in Abb. 2.7 und 2.8 gut zu sehen). Sie dienen als Andockpunkt für Transitionen und repräsentieren einen Ein- bzw. Ausgang für Datenflüsse.

Die in meiner Arbeit eingeführten Ankerelemente können solche Schnittstellen nur teilweise abdecken. Nämlich nur für den aufgeklappten Zustand des Containers, da die Ankerlemente innerhalb des Containers liegen müssen, um sie semantisch als Schnittstelle zu erkennen. Im eingeklappten Zustand sind die Ankerelemente nicht sichtbar und die Transitionen docken wie in Abschnitt 3.1 beschrieben an dem Container an.

Es sollte möglich sein, für einen Container Schnittstellen zu definieren, wobei diese:

- an einer gewünschten Seite positioniert werden können (oben, unten, links, rechts)
- als Andockpunkt für Transitionen dienen
- bei aus- und eingeklappten Container sichtbar sind

Usability von Containern verbessern

Um einen Container hinzuzufügen muss dieser in der Palette ausgewählt sein und wird dann per Mausklick ins Diagramm an der Stelle des Mauszeigers eingefügt. Wenn man nun schon vorhandene Elemente hat und diese in den Container legen will, ist ein weiterer Arbeitsschritt nötig.

Um diesen Arbeitsschritt einzusparen, sollte man beim Hinzufügen eines Containers mit gedrückter Maustaste ein Rechteck markieren können. Alle Elemente, die von diesem Rechteck eingeschlossen sind, werden direkt in den neuen Container gelegt.

Das Einklappen bzw. Ausklappen funktioniert aus technischen Gründen nur dann, wenn der Container vorher ausgewählt ist. Dies sollte auch verbessert werden, so dass das Auswählen des Containers vorher nicht mehr nötig ist.

Gegenseitige Überdeckung von Elementen verhindern

Zur Zeit ist es möglich, ein Element über ein anderes Element zu schieben, so dass nur das Element sichtbar ist, was eine Ebene über dem anderen gezeichnet wird. Um dies zu verhindern, sollte ein Element, welches beim Verschieben eines anderen Elements im Weg steht, zur Seite geschoben werden.

6.2 Fazit

Wie im Kapitel „Implementierung“ deutlich wird, wurde das Ziel, ProFLOW um Containerelemente zu erweitern, erreicht. Betrachtet man die Messungen der Qualitätsanforderungen und die bekannten Probleme sieht man, dass noch Arbeit vorhanden ist und Container in ProFLOW noch verbessert werden können.

Zusammenfassend kann man sagen, dass mit der Erweiterung um Container das Fundament für die Implementierung von grafischen Notationen vervollständigt wurde. Es ist nun möglich, alle Elemente von grafischen Notationen mit Aussehen und Verhalten umzusetzen. Dies ebnet den Weg für die Weiterentwicklung von neuen Features in ProFLOW.

7 Glossar

Annotation	Eine Anmerkung oder Kommentar; kann in ProFLOW an SimpleElements und Transitions angehängt werden
ContainerElement	zusammengesetztes Element in ProFLOW, kann andere Elemente enthalten
Eclipse	Umgebung, die es ermöglicht, Entwicklungstools auf einer Arbeitsoberfläche zu integrieren
GEF	Das Graphical Editing Framework ist ein Framework für Eclipse, auf dessen Basis graphisch gestützte Tools entwickelt werden können
SimpleElement	steht für ein nichtzusammengesetztes Element in ProFLOW
Transition	Übergang von einem Element zu einem anderen

Literaturverzeichnis

[1] Peter Pin-Shan Chen. The entity-relationship model - toward a unified view of data. *International Conference on Very Large Data Bases, Framingham, Mass., Sept. 22-24*, 1975.

[2] Tom DeMarco. Structured analysis and system specification. *Prentice-Hall*, 1979.

[3] Randy Hudson. *Create an Eclipse-based application using the Graphical Editing Framework*, 2006.

[4] Wikimedia Foundation Inc. Business process modeling notation. `http://de.wikipedia.org/wiki/BPMN`.

[5] Wikimedia Foundation Inc. Unified modeling language 2. `http://de.wikipedia.org/wiki/UML2`.

[6] oose.de GmbH. *Unified Modeling Language 2 - Notationsübersicht*, 2006.

[7] Nathan D. M. Robertson, Ivy Anderson, Adam Chandler, Sharon E. Farb, Timothy Jewell, Kimberly Parker, and Angela Riggio. Entity relationship diagram for electronic resource management. *Electronic Resource Management Initiative Deliverables*, 2004.

[8] Kurt Schneider and Daniel Lübke. Systematic tailoring of quality techniques. *World Conference on Software Quality (3rd WCSQ), Munich, Germany*, 3, 2005.

[9] Prof. Dr. Kurt Schneider. *Skript zu Vorlesung: Grundlagen der Softwaretechnik*, WS 2006/07.

[10] Stephen A. White. *Business Process Modeling Notation Specification*, 2006.

A ProFLOW Tutorial: Containerelemente

Erzeugen eines neuen Containers

Einen neuen Container zu erstellen ist sehr ähnlich zum Erstellen eines neuen Prozesselements. Es muss eine Model- sowie eine Controller-Klasse erstellt werden.

Die Model-Klasse unseres Containers muss die Framework-Klasse ProFLOW.core.model.SubProcess erweitern. Erzeugen sie eine Klasse ExampleContainer im Package proFlow.example.model. In dieser Klasse müssen neben der schon bekannten Methode getTypeID():String die Methoden isCollapsable():boolean und isValidChildElement(Element element):boolean implementiert werden. Mit isCollapsable wird festgelegt, ob der Container eingeklappt werden darf. IsValidChildElement entscheidet, welche Elemente der Container enthalten darf.

```
public class ExampleContainer extends SubProcess {

	public ExampleContainer() {
		this("Container");
	}

	public ExampleContainer(String name) {
		super(name);
		setMinDimension(new Dimension(100, 50));
		setMaxDimension(new Dimension(Integer.MAX_VALUE, Inte-
ger.MAX_VALUE));
		setDimension(new Dimension(200, 100));
	}

	public String getTypeID() {
		return "ProFLOW.example.Container";
	}
}
```

Damit der Container auch in unseren Prozess hinzugefügt werden darf, verfahren sie wie für das Prozesselement, indem sie die isValidChildElement-Methode des ExampleProcess erweitern.

Nun benötigen wir eine Controller-Klasse. Erzeugen sie dafür eine Klasse ExampleContainertEditPart im Package proFlow.example.ui.editParts, welche die Klasse ContainerEditPart erweitert. Um das Aussehen des Containers festzulegen muss die Methode createFigure():IFigure implementiert werden. Wie bei dem normalen Element benötigen wir auch die getLabelContainer-Methode, um das Editieren des Containernamens im CellEditor zu ermöglichen.

Nun müssen sie in der plugin.xml den Plug-In den neuen Container bekannt machen.

```
public class ExampleContainerEditPart extends ContainerEditPart {

      protected IFigure createFigure() {
            LabeledPolygon containerFigure =
                  new LabeledPolygon(getLabelTextBorderLayout());

            PointList pList = new PointList(4);
            pList.addPoint(0,0);
            pList.addPoint(100,0);
            pList.addPoint(100,100);
            pList.addPoint(0,100);

            containerFigure.setSize(
                  getCastedModel().getDimension().getWidth(),
                  getCastedModel().getDimension().getHeight());
            containerFigure.setLocation(new Point(
                  getCastedModel().getLocation().getX(),
                  getCastedModel().getLocation().getY()));
            containerFigure.setText(getCastedModel().getName());
            return containerFigure;
      }

      public ILabelContainer getLabelContainer() {
            return ((LabeledPolygon) getFigure())
                  .getLabelContainer();
      }
}
```

Analog zur Erstellung eines neuen Prozesselementtyps müssen sie für einen Container lediglich Extensions für die Extension Points ProFLOW.core.containerElements und ProFLOW.ui.containerElements im Plug-In Editor definiert werden. Die Spezifikation der Attribute der entsprechenden Extensions ist dabei genauso definiert, wie schon zuvor bei den Extensions eines neuen Prozesselements beschrieben. Die plugin.xml wird somit um folgende Einträge erweitert:

```
<extension
      point="ProFLOW.core.containerElements">
   <transition
         id="ProFLOW.example.Container"
         class="proFlow.example.model.ExampleContainer"/>
</extension>
<extension
      point="ProFLOW.ui.containerElements">
   <transition
         id="ProFLOW.example.Container"
         name="Container"
         editPart="proFlow.example.ui.editParts
                  .ExampleContainerEditPart"
         description="Create a new Container"
         icon="icons/element.gif"/>
</extension>
```

Bei ContainerElements und SimpleElements ist es möglich, benutzerdefinierte Ankerpunkte für Transitionen zu verwenden. Ankerpunkte sind die Stellen eines Elements, an die Transitionen andocken können. Dafür müssen sie die Methoden getSource-

ConnectionAnchor(ConnectionEditPart):ConnectionAnchor, getSourceConnectionAnchor(Request): ConnectionAnchor, getTargetConnectionAnchor(ConnectionEditPart): ConnectionAnchor und getTargetConnectionAnchor(Request):ConnectionAnchor überschreiben.

Berechnet werden die Punkte von der Klasse CustomTopBottomLeftRightAnchor, mit der man die Anzahl der Ankerpunkte an allen vier Seiten (oben, unten, links, rechts) spezifizieren kann. Ein weiterer (optionaler) Parameter ist ein Skalierfaktor, der dafür sorget, dass die Andockpunkte weiter innen (<1) oder weiter außen liegen(>1). Der Standardwert ist 1, so dass die Transitionen direkt am Element andocken.

```
public ConnectionAnchor getSourceConnectionAnchor(ConnectionEditPart
connection) {
      return new CustomTopBottomLeftRightAnchor(getFigure(),2,2,2,2,1);
}

public ConnectionAnchor getSourceConnectionAnchor(Request request) {
      return new CustomTopBottomLeftRightAnchor(getFigure(),2,2,2,2,1);
}

public ConnectionAnchor getTargetConnectionAnchor(ConnectionEditPart
connection) {
      return new CustomTopBottomLeftRightAnchor(getFigure(),2,2,2,2,1);
}

public ConnectionAnchor getTargetConnectionAnchor(Request request) {
      return new CustomTopBottomLeftRightAnchor(getFigure(),2,2,2,2,1);
}
```

Um das Verhalten von Container zu ändern oder zu erweitern, müssen sie die createEditPolicies()-Methode überschreiben oder erweitern. Näheres dazu finden sie im Kapitel „Verwendung von Commands“.

B Software CD

Die beiliegende CD enthält:

- das ProFLOW Framework und drei ProFLOW Diagramme als Eclipse-Plugins
  - als Binary ohne Quellcode
  - als vollständige Version inklusive Quellcode
- ProFLOW Tutorial mit Installations- und Bedienungsanleitung
  - als PDF
  - als Word Document zur Weiterentwicklung

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 31.07.2007

_____

Dimitri Gatowski