INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN
Lehrstuhl für Programmierung und Softwaretechnik

Projektarbeit

Migration und Erweiterung des MagicDraw-Plugins MagicUWE zur Entwicklung von Web-Anwendungen

Marianne Busch
busch@cip.ifi.lmu.de

Aufgabensteller:	Prof. Dr. Alexander Knapp
Betreuer:	Dr. Nora Koch
Abgabetermin:	31. März 2009

Hiermit versichere ich, dass ich die vorliegende Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 31. März 2009

.....
(Marianne Busch)

Zusammenfassung

UWE ist eine UML-basierte Methode zur Entwicklung von Webanwendungen. Sie beruht auf einer konservativen Erweiterung der UML und verfolgt den Ansatz einer getrennten Modellierung der Inhalts-, Navigations-, Präsentations- und Prozessschicht einer Webanwendung. Im Rahmen einer Diplomarbeit wurde ein Plugin für die Software MagicDraw 12.0 mittels der bereitgestellten Open API Schnittstelle implementiert, das die Notation von UWE und einige der Schritte des methodischen Vorgehens von UWE unterstützt.

MagicDraw integriert das UML2 Metamodel und ermöglicht es, seine grundlegenden modellierenden Funktionalitäten mit Hilfe von Drittentwicklern zu erweitern. Die Implementierung und das Verwenden eines solchen UWE Plugins für ein so viel genutztes Modellierungs- und Entwicklungswerkzeug wie MagicDraw bietet sich an. Die Anpassung des Plugins an die aktuellen Versionen der Hosting-Software muss jedoch ebenfalls gewährleistet werden.

In dieser Projektarbeit wurde das UWE-Tool *MagicUWE* auf die Version 15.1 des Werkzeuges MagicDraw migriert, verbessert und erweitert. Die Erweiterung besteht in der Integration neuer Stereotypen und Tags zur Modellierung von Webanwendungen, der Erstellung einer benutzerfreundlichen Plugin-GUI und der Unterstützung des UWE Entwicklungsprozesses mit semi-automatischen Generierungsschritten. Darüber hinaus wurde auf Wartungsfreundlichkeit und gute Erweiterbarkeit des Java-Quellcodes besonderen Wert gelegt.

Inhaltsverzeichnis

1 Einleitung

1.1 Über UWE

UWE [13] steht für UML-based Web Engineering und ist ein Ansatz zur Entwicklung von Webanwendungen. „UML-based“ bedeutet, dass alle UWE Modelle UML (Unified Modelling Language [12]) Notation und UML Diagrammtypen verwenden. Überdies definiert UWE eine Erweiterung fürs Web, die im wesentlichen fünf Sichten umfasst: Ein Contentmodell (auch Inhaltsmodell genannt) skizziert die in der Applikation verwendeten Klassen und in einem Navigationsmodell werden die Verknüpfungen zwischen Knoten und Kanten dargestellt, wobei man sich die Knoten fürs erste als Teile einer Webseite und die Kanten als Links vorstellen kann. Das Präsentationsmodell beschreibt den Aufbau von Webseiten und unterstützt ebenfalls spezielle Elemente fürs Web, wie z.B. Formulare.

Alle bisher genannten Modelle werden durch UML2 Klassendiagramme abgebildet und auch die Beziehung zwischen verschiedenen, innerhalb der Webapplikation ablaufenden Prozessen, kann mit dieser Diagrammart in einem Prozessstrukturdiagramm dargestellt werden. Der Ablauf eines Prozesses selbst, wird wie üblich mit einem Aktivitätsdiagramm modelliert, das den Namen Prozess-Ablaufdiagramm (Process Flow Diagram) trägt. Transformationen zwischen den UWE-Diagrammtypen sind möglich und verknüpfen die Informationen der einzelnen Sichten auf die Anwendung. Mehr zu UWE findet sich unter anderem in Kapitel 4 und in den Dokumenten [6] und [7].

Die Verbreitung einer Modellierungstechnik wie UWE hängt unter anderem von folgenden Faktoren ab:

- Der Qualität der Methode an sich, wobei UWE den Vorteil hat, auf der weit verbreiteten UML zu basieren, ohne Veränderungen des UML-Metamodells nötig zu machen. Dies basiert auf den Erweiterungsmechnismen, die die UML selber anbietet, d.h. die Definition von sogenannten Profilen. UML Profile werden mittels der Spezifikation von Stereotypen, Tagged-Value Definitionen und Constraints erstellt.
- Der Unterstützung der Modellierung durch ein benutzerfreundliches Tool, das folglich am besten als Plugin auf bestehenden UML-CASE-Tools aufbaut.
- Der Dokumentation von UWE und von dem Plugin, wobei damit zum einen eine Art Referenz gemeint ist und zum anderen die Dokumentation des Quellcodes, um die Erweiterbarkeit und Wartbarkeit des Tools zu gewährleisten.
- Den Einstiegshilfen, sowohl in die Modellierungstechnik als auch in der mit der Alltagsarbeit verbundenen Software, womit die Hürde für das Erlernen von UWE möglichst gering gehalten werden soll.

1.2 Zielsetzung eines Plugins für UWE

Die eben genannten Stichpunkte lassen deutlich werden, wie eng der Erfolg einer Modellierungstechnik mit der Qualität der dazu angebotenen Software verknüpft ist. Im Fall von UWE setzte man zuerst auf das Plugin ArgoUWE [16], doch das zu Grunde liegende, freie ArgoUML wurde nicht an UML2 angepasst. Das komerzielle MagicDraw [8] hat den Vorteil UML2 zu unterstützen, daher ist das in dieser Projektarbeit beschriebene UWE-Tool namens MagicUWE [17], das seinen Ursprung (als uweMDPlugin) in einer vorhergehenden Diplomarbeit [1] hat, als Plugin für MagicDraw konzepiert worden.

Die Ziele, die MagicUWE verfolgt, ergeben sich im wesentlichen aus den im letzten Unterkapitel genannten Faktoren für die Verbreitung von UWE. Zum einen soll das Tool den Modellierer optimal bei seiner Arbeit unterstützen und gegebenenfalls etwas dabei anleiten und muss daher leicht bedienbar und konfigurierbar sein. Diesbezügliche Designentscheidungen sind in Kapitel 2 dieser Arbeit beschrieben.

Zum anderen erfüllt ein Plugin nur dann über die Jahre seinen Zweck, wenn es so programmiert ist, dass es sowohl leicht an neue Versionen der Hosting-Software MagicDraw angepasst werden kann, als sich auch optimal um neue Funktionalitäten erweitern lässt (mehr dazu in Kapitel 3).

Den oben genannte Punkt der problemlosen Einstiegsmöglichkeit kann man ebenfalls auf MagicUWE beziehen, wobei sich eine Art Tutorium - wie in Kapitel 4 - anbietet, das unter Verwendung der Funktionalitäten des Plugins, ein einführendes Beispiel in UWE modelliert.

Generell werden die Möglichkeiten zur Anpassung und Erweiterung nur dann ausgeschöpft, wenn MagicUWE kontinuierlich weiterentwickelt wird (Kapitel 5).

Da die englische Sprache für Fachbegriffe der Informatik besonders häufig Verwendung findet und MagicUWE momentan ebenfalls in Englisch verfasst ist, wird in dieser Arbeit nicht alles ins Deutsche übersetzt. Folglich wird auch MagicDraw meist in englischer Sprache genutzt, was sich auf Screenshots bemerkbar macht, wie z.B. bei der Übersicht über einige Funktionalitäten von MagicUWE in Abbildung 1.1.

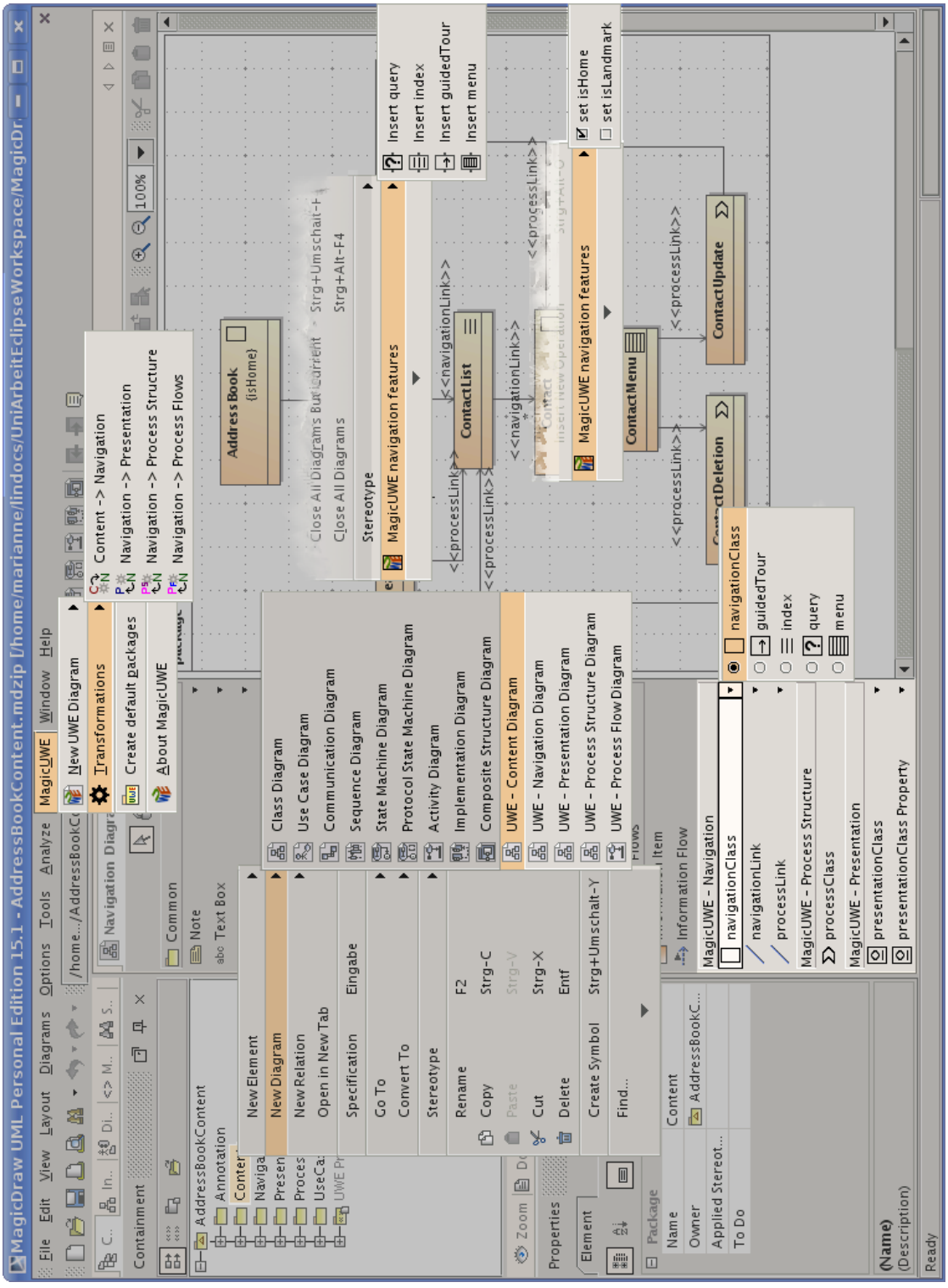


Abbildung 1.1: MagicDraw Screenshot mit hervorgehobenen MagicUWE Elementen

2 Designentscheidungen für MagicUWE

In diesem Kapitel werden die MagicUWE betreffenden Designentscheidungen beleuchtet, was sowohl die Beschreibung der grundlegenden Funktionalitäten des Plugins, als insbesondere auch Benutzerfreundlichkeit umfasst.

2.1 Installation des Plugins

Um MagicDraw zu installieren, lädt man die jar-Datei [17] herunter und führt sie mit `java -jar MagicUWEv1.2Installer.jar` aus, oder klickt doppelt darauf, sofern das verwendete Betriebssystem dies unterstützt. Der mit IzPack [11] erstellte Installer führt den Benutzer Schritt für Schritt durch eine komfortable Installation der einzelnen Komponenten, die sich anschließend in den jeweiligen Unterordnern des MagicDraw-Stammverzeichnisses wiederfinden. Die Deinstallation erfolgt ebenfalls über ein Java Programm, das in dem Ordner `Uninstaller` lokalisiert ist. Alternativ kann das Plugin auch im Eigenschaftsdialog von MagicDraw (de)aktiviert werden.

Zu beachten ist, dass zu einer Installation von MagicUWE das Plugin selbst genauso gehört, wie eine Vorlage zum einfachen Erstellen neuer UWE Projekte und das UWE-Profil [15] für MagicDraw. Wer vermeidbaren Arbeitsaufwand nicht scheut, kann die Vorlage und das Profil auch ohne das Plugin nutzen, was praktischerweise die problemlose Darstellung und Bearbeitung der Projektdateien an allen Rechnern (auf denen MagicDraw und das UWE-Profil verfügbar ist), möglich macht.

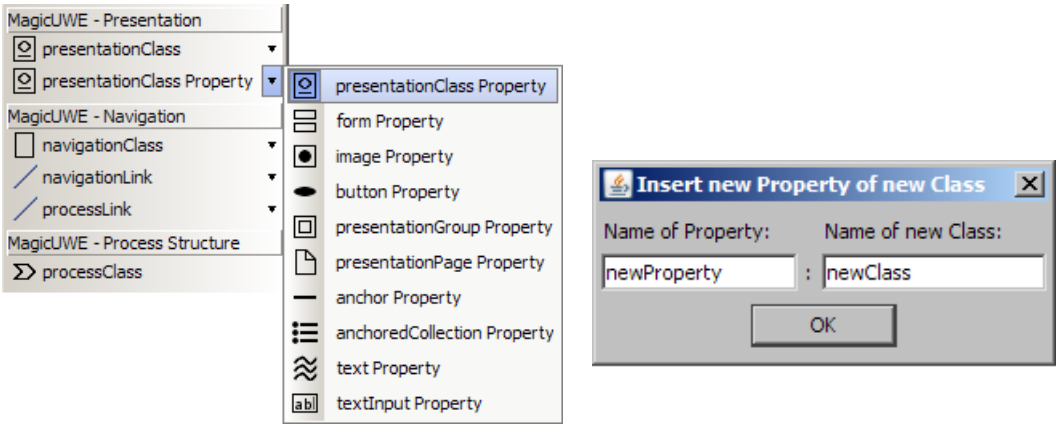
2.2 Die Menüs und deren Funktionalität

MagicUWE erweitert MagicDraw vor allem bezüglich des Menüangebots, sei es in der Toolbar von MagicDraw, der Menüleiste, oder in den verschiedenen Kontextmenüs (z.B. im Containment-Baum und in Navigationsdiagrammen). Eine tabellarische Referenz findet sich online unter [18].

2.2.1 Die Toolbar

Die sogenannte Toolbar bietet pro Diagrammtyp passende Modellierungselemente an und wird für gewöhnlich auf der linken Seite des offenen Diagramms eingeblendet. Dort wählt man ein Element aus, um es dann an die gewünschte Position im Diagramm zu zeichnen.

Mit MagicUWE stehen dort in der Standardkonfiguration die im Bild 2.1(a) dargestellten Einträge für Klassendiagramme zur Auswahl. Hier ist der Unterschied zu beachten,



(a) Die MagicUWE Toolbar (b) Benennung der einzufügenden Property

Abbildung 2.1: Toolbar zum Einfügen von Klassen, Assoziationen und Properties mit Stereotypen aus dem UWE-Profil

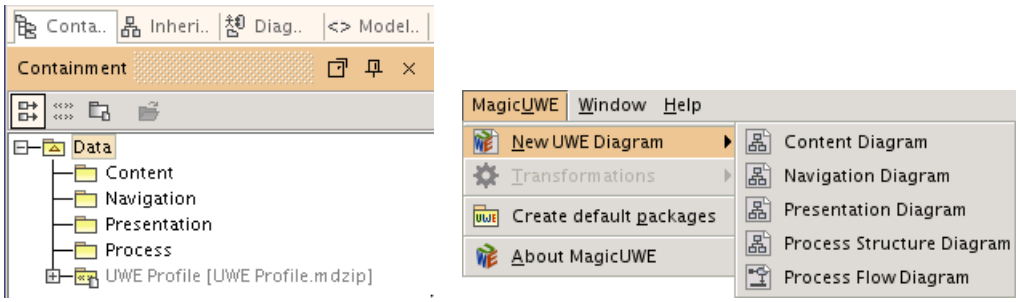
ob man eine mit einem Stereotyp versehene Klasse bzw. Assoziation, oder eine Property auswählt. Die beiden Ersteren kann man direkt ins Diagramm zeichnen, wohingegen zum Einfügen von Properties zuvor eine Klasse markiert worden sein muss. In diese Klasse wird die gewählte Property dann hineingezeichnet, nachdem für sie selbst eine neue Klasse erstellt wurde. Der Name dafür wird - ebenso wie die Benennung der Property - in einem Fenster abgefragt (Abb. 2.1(b)). Die Properties lassen sich (z.B. für Präsentationsdiagramme) verschachteln, in dem man statt einer Klasse eine schon vorhandene Property selektiert. Ein Beispiel dazu findet sich in Abschnitt 4.4.

Der UWE-Diagrammtyp wird von MagicUWE an Hand des übergeordneten Package-Namens erkannt. Die Unterscheidung macht es möglich, in den verschiedenen Klassendiagrammen, je nach UWE-Diagrammtyp Warnungen einzublenden, z.B. wenn in einem Navigationsdiagramm UWE Klassen aus dem Präsentationsmodell eingefügt werden. Die Erkennung der Parent-Package Namen kann in einer Konfigurationsdatei angepasst werden (s. Abschnitt 2.4).

Einfacher wäre es, wenn es in MagicDraw eine Möglichkeit gäbe, die Diagramme durch ein gut sichtbares Merkmal zu unterscheiden. Leider gibt es bisher nur die Möglichkeit komplett neue Diagrammtypen zu definieren, oder das UWE-Profil um zusätzliche Tagged Values (im Deutschen „Eigenschaften“) zu erweitern. Da UWE nicht über das Standard-UML hinausgehen soll, scheidet der erste Vorschlag aus - und da die grafische Repräsentation von Tags nicht aussagekräftig ist, auch die genannte Alternative, zumal es ein Ziel des Plugins ist, dass man an den UWE-Projekten bei Bedarf auch ohne geladenes Plugin weiterarbeiten kann. Dabei ist es wahrscheinlicher, dass der Benutzer seine Diagramme in passend benannten Paketen verwaltet, als dass er eher verborgene Tagged Values setzt.

2.2.2 Das MagicUWE-Hauptmenü

Im Gegensatz zur Toolbar erscheint das Menü namens „MagicUWE“ in MagicDraw, sobald das Plugin geladen ist. Wurde kein Diagramm geöffnet, bietet es lediglich die Möglichkeit, mit „Create default packages“ die UWE-Standardpakete anzulegen oder neue UWE-Diagramme erstellen, die vorzugsweise in eben diese Pakethierarchie eingeordnet werden (Abb. 2.2).



(a) Standardpakete (b) MagicUWE Menü

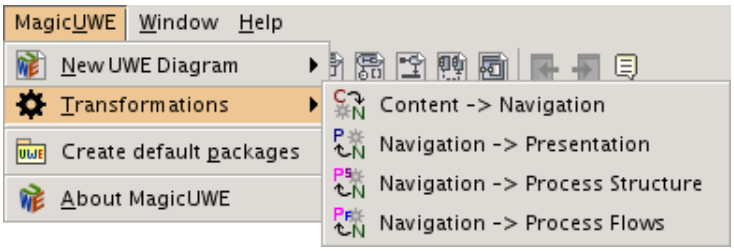
Abbildung 2.2: Standardansicht bei geschlossenen Diagrammen

Wurde bei „neues Projekt“ die UWE-Vorlage ausgewählt, so existieren die Pakete bereits und müssen nicht extra erstellt werden. Überdies sind darin die Methoden von Klassen standardmäßig ausgeblendet, da sie für die Modellierung (zumindest im ersten Schritt) nicht benötigt werden.

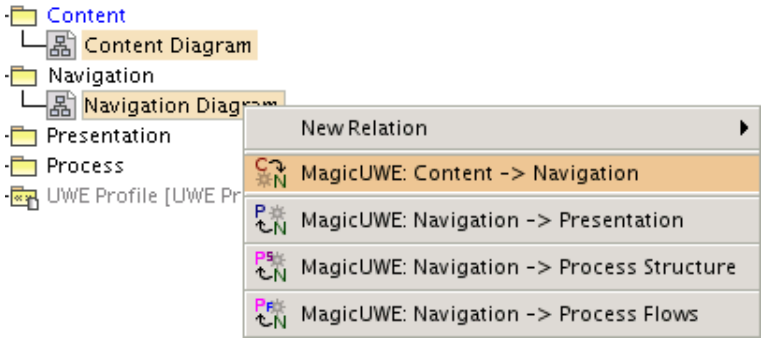
In UWE gibt es zwischen mehreren UWE-Modellen eine Transformationsmöglichkeit (Abb. 2.3(a)), die das Erstellen von neuen Diagrammen vereinfacht. **Transformationen** sind in vielen Varianten denkbar, einige davon bietet MagicUWE bereits an, wobei das geöffnete Diagramm die Quelle ist:

- **Content zu Navigation:** Erstellt für jede Klasse des Content Diagramms eine neue Klasse (im Navigations Paket) mit dem selben Namen und fügt den `«navigationClass»`-Stereotyp hinzu. Die neuen Klassen und angepassten Assoziationen werden alle in ein neues Navigationsdiagramm gezeichnet.
- **Navigation zu Präsentation:** Erstellt für jede Klasse des Navigationsdiagramms im „Presentation“ Paket eine gleichnamige Klasse mit einem Präsentationstereotyp. Jede neue Klasse ist gleich benannt wie die Ursprungsklasse, hat aber im Gegensatz zu ihr den `«presentationClass»`-Stereotyp.
- **Navigation zu Process-Structure:** Zeichnet jede Klasse eines Navigationsdiagramms, die einen `«processClass»`-Stereotyp hat in ein neues Process Structure Diagram. Neue Klassen werden dabei nicht erstellt.
- **Navigation zu Process-Flow:** Erstellt aus einem Navigationsdiagramm neue Prozessflussdiagramme (Process Flow) für jede `«processClass»` mit einem nichtleeren Namen. Die neuen Aktivitätsdiagramme werden nach dem ursprünglichen

Klassennamen, gefolgt von „ `Workflow"` benannt, so dass man leicht sehen kann welche Aktivität zu welcher Klasse gehört.



(a) Transformationsmenü



(b) Transformationen im Kontextmenü

Abbildung 2.3: Transformationen

2.2.3 Die Kontextmenüs

Wie eben beschrieben, transformiert man mit den Transformationsfunktionen aus dem MagicUWE-Menü immer das aktuell geöffnete Diagramm. Möchte man mehrere Diagramme eines Typs transformieren, so gibt es die Lösung, sie im Containment-Baum zu selektieren und das Kontextmenü (Abb. 2.3(b)) zu öffnen, wobei jedes der gewählten Diagramm beachtet wird, das den zur gestarteten Transformation passenden UWE Diagrammtyp hat.

Auch innerhalb der Diagramme kann ein Angebot von Kontextmenüs nützlich sein. Damit kann z.B. ein Zugriffselement (wie Index oder Query) zwischen vorhandenen, verbundenen Navigationsklassen hinzugefügt werden, wobei zuvor eine oder mehrere Assoziationen selektiert werden müssen. Eine neue Klasse des aus dem Kontextmenü gewählten Stereotyps wird zwischen einer ‚Kopfklasse' und allen anderen Klassen eingefügt. Notwendigerweise müssen alle Assoziationen an einem Ende zu ein und derselben Klasse führen. Multiplizitäten, Klassennamen und die Assoziationstypen werden dem gewählten Stereotyp entsprechend gesetzt, wie in Unterabschnitt 3.4.3 näher erläutert.

Als Veranschaulichung bietet sich ein Adressbuch wie in Abbildung 2.4 an, das mehrere Kontakte beinhaltet. In UWE ist für diesen Fall ein Index vorgesehen, der mit

dem (im Bild rechts) eingeblendeten Menü automatisch zwischen den beiden schon vorhandenen Klassen eingefügt wird, wobei beide resultierenden Assoziationen einen «navigationLink» Stereotyp zugewiesen bekommen. Dabei wird die Multiplizität „*“ an dem zur Kontaktklasse hin gelegenen Assoziationsende angebracht, um auszudrücken, dass der Index auf beliebig viele Contact-Elemente verweist. Ist, wie im Beispiel, eine Rolle („Contacts“) angegeben, so wird diese als Name des Indexes übernommen (Abb. 2.5).

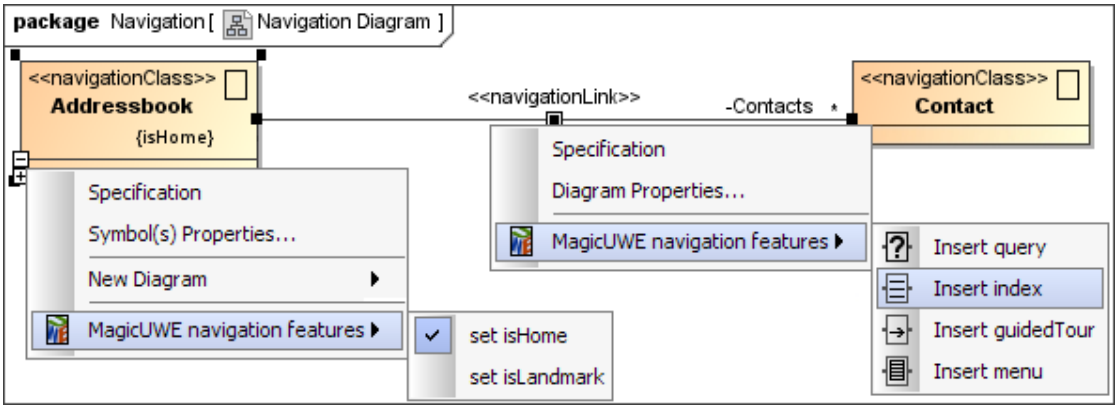


Abbildung 2.4: Die Kontextmenüs in Navigationsdiagrammen

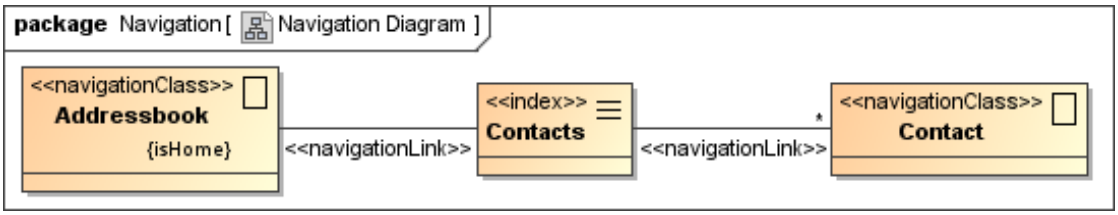


Abbildung 2.5: Fortführung des Beispiels aus Abbildung 2.4 mit eingefügtem Index

Auch Tagged Values (Eigenschaften) setzt man mit MagicUWE nicht umständlich über `(Rechtsklick)` / `Spezifikation` / `Eigenschaften` / `«navigationNode» isHome` bzw. `isLandmark / Wert anlegen`, sondern nutzt ebenfalls das Navigations-Kontextmenü, wie in Abbildung 2.4 (links) gezeigt.

2.3 Benutzerfreundlichkeit

Wie bereits angedeutet, liegt das Hauptaugenmerk auf der Benutzerfreundlichkeit von MagicUWE, da diese essenziell für die Verbreitung des Plugins und damit auch des UWE Ansatzes im Allgemeinen ist. Der Installer, der einem das mühsame Kopieren einzelner Plugin-Komponenten erspart, wurde in Abschnitt 2.1 schon vorgestellt.

Für die alltägliche Arbeit mit dem Plugin ist eine intuitive Benutzeroberfläche, welche die Bedienelemente von MagicDraw verwendet, wie z.B. in der Toolbar mit ihrer

‚zuletzt-benutzt' Funktion, von Vorteil. Der Anwender sollt daher keine nennenswerte Eingewöhnungszeit für MagicUWE benötigen, da er in gewohnter Umgebung arbeitet und das Plugin einem unauffälligen, minimalistischen Design folgt. Dazu gehört die effiziente Auswahl von Toolbarelementen und Menüeinträgen an Hand ihrer Symbole ebenso, wie das nicht-invasive Verhalten des Plugins. Damit ist gemeint, dass der Modellierer selbst bestimmt, in wie weit er von MagicUWEs Funktionen Gebrauch macht. Dies war in früheren Versionen des damaligen uweMDPlugins anders, wobei besonders die vorgeschriebene Anzeige der Stereotypen in Klassen mit Name und Icon für Unmut sorgte.

Nichtsdestotrotz soll ein derartiges Tool den Benutzer unterstützen und anleiten valide Diagramme zu erstellen, in dem zum Beispiel bei dem Versuch Processklassen in ein Präsentationsdiagramm zu zeichnen, eine warnende Nachfrage angezeigt wird. Den UWE-Diagrammtyp ermittelt MagicUWE dabei aus dem Namen des dem Diagramm übergeordneten Pakets, wobei die Konfigurierbarkeit dieser Abfrage sich an das Konzept der Benutzerfreundlichkeit anschließt.

2.4 Konfigurierbarkeit

Um das Plugin seinen Wünschen anzupassen, gibt es die Konfigurationsdatei[1], in der sich Folgendes einstellen lässt:

- Der Name der zu verwendenden UWE-Profildatei.
- Tastenkürzel (Shortcuts) für UWE-Einträge der Toolbar, wobei jeweils der Buchstabe zur „Grundtastenkombination“ `Alt`+`Shift` anzugeben ist.
- Welche Präsentationselemente als Klasse bzw. Property von der Toolbar aus eingefügt werden können. Wobei hier nicht alles auf „`true`“ gesetzt werden sollte, um die Übersichtlichkeit zu wahren.
- Der Pakete-Check: MagicUWE ermittelt aus dem Namen des Pakets, das ein Diagramm enthält, dessen Typ, was über einige Funktionalitäten und Warnungen entscheidet. Standardmäßig wird das Vorkommen folgender Substrings geprüft:

```
# Part of the package names. (case insensitive)
# Leaving one empty means the plugin does not care, if the diagram
# (e.g. for transformation) is located in an appropriate package.
packContent=cont
packNavigation=nav
packPresentation=pres
packProcess=proc
```

Leider sind die Möglichkeiten MagicUWE benutzerfreundlich zu gestalten durch die MagicDraw OpenAPI etwas beschränkt, dazu gegen Ende des nächsten Kapitels mehr.

[1] Die Konfigurationsdatei `magicUWE/settings/MagicUWE.properties`, befindet sich innerhalb des .jar - Archivs *`MagicDrawHome`*`/plugins/MagicUWE/MagicUWE.jar` und kann vom Nutzer bei Bedarf editiert werden. *MagicDrawHome* steht dabei für das Verzeichnis in dem MagicDraw installiert ist.

3 Implementierung

Nachdem die Funktionalität des Plugins und die dabei gesetzten Schwerpunkte im letzten Kapitel erläutert wurden, beschäftigt sich das Aktuelle mit der Implementierung von MagicUWE. Dabei geht es nicht darum, alle Überlegungen zum Quellcode zu erklären (das würde den Rahmen sprengen!), geschweige denn alle Veränderungen und Erweiterungen aufzuzählen, sondern vielmehr darum, einen Überblick über die Struktur zu geben und auf einige Entwicklungsschritte und -entscheidungen beispielhaft hinzuweisen.

3.1 Überblick über MagicUWE

Das Plugin wurde möglichst flexibel und erweiterbar konzipiert, denn nur so kann es neuen Anforderungen schnell gerecht werden, die ganz automatisch aus der Weiterentwicklung des UWE-Profils und des Tools MagicDraw entstehen. Dank des weitgehend modularen Code-Aufbaus ist es vorstellbar, dass sich neue Funktionalitäten besonders gut innerhalb einzelner Teams entwickeln lassen. Die Wiederverwendung und Verallgemeinerung von bestehenden Funktionen ist ein dazu oft kontroverses Ziel, was durch die Extraktion von Code in weitere Klassen bzw. Enums jedoch gut in den Griff zu bekommen ist.

MagicUWE nutzt den Plugin-Mechanismus von MagicDraws OpenAPI und wird - wie in Diplomarbeit [1] beschrieben - über die vom `ActionManager` überschriebene `init()`-Methode der Klasse `com.nomagic.magicdraw.Plugin` aufgerufen. Von dort registrieren sich alle UWE-Menüeinträge bei MagicDraw. Dazu werden diverse, teilweise selbst abgeleiteten *Configurator*s (Konfiguratoren), die sich um die Darstellung der Einträge kümmern und passende Aktionsklassen hinzufügen, genutzt. Da dieser Zusammenbau schnell unübersichtlich wird, gibt es eine abstrakte Klasse namens `PluginManagerActions`, von der die gewünschte Kapselung vollzogen wird, wobei es so auch möglich ist, ähnliche Funktionalität in private Methoden zu extrahieren.

Übersicht versucht auch die **Paketstruktur** von MagicUWE mit folgendem Aufbau zu erreichen:

- **actions**: Stellt die Funktionalität der Menüeinträge bereit und ist, genauso wie das Paket `configurators`, jeweils nochmal nach den verschiedenen Menüarten unterteilt (s. Abb. 3.1).
- **configurators**: Die Konfiguratoren werden bei MagicDraw pro Menü registriert, sorgen für die Menüanzeige und verknüpfen die Einträge mit den zugehörigen Aktionen.

- **core**: Das Kernpaket von MagicUWE enthält besagten Startmechanismus des Plugins, die Icons der Menüeinträge und insbesondere einen `ProjectListener`, der beim Laden eines Projekts prüft, ob das UWE Profil eingebunden ist und es (nach Nachfrage) lädt.

- **settings**: Die Einstellungen umfassen sowohl die Konfigurationsdatei (für den Anwender) `MagicUWE.properties` (s. Abschnitt 2.4), als auch den `PropertyLoader`, der selbige beim Laden des Plugins ausliest und die Werte auf gewünschte Art und Weise automatisch z.B. den Stereotypen-Enums zuweist. Die abstrakte Klasse `GlobalConstants` enthält momentan nur zwei Konstanten, darunter den Namen der UWE-Profil-Projektdatei, der ebenfalls aus der Textdatei stammt.

- **shared**: Enthält wie der Name sagt Funktionen die von fast jeder Klasse genutzt werden, z.B. den oben genannten `MessageWriter`, einige standardisierte UWE und MagicDraw Operationen und eine Enumeration, die den UWE-Diagrammtyp bestimmt und u.a. für Paketverwaltung und Diagrammerstellung zuständig ist.

- **stereotypes**: `UWEStereotypeWithKey` heißt das Interface, das (aufgeteilt nach Klassen, Assoziationen und Diagrammtyp) verschiedene Stereotypen-Enums implementieren. „WithKey“ bedeutet dabei, dass die Enumerations nicht nur für die simple Unterscheidung der Stereotypen zuständig sind, sondern auch alle nötigen Informationen und Operationen zu ihrer Verwendung, wie eben z.B. den gewünschten Shortcut für die Toolbar bereitstellen.

- **transformation**: Auch hier gibt es ein Enum (`TransformationType`), das die vorhandenen Transformationen auflistet und diese anstößt, wozu auf zahlreiche Teilfunktionen zugegriffen wird, welche teilweise auch in einer Hilfsklasse ausgelagert sind. Die `Action`-Klasse `TransformatorAction` muss demnach hauptsächlich dafür sorgen, dass von der vom Benutzer gewählten Transformation die Methode `launchTransformation(sourceDiagram, targetPackage)` aufgerufen wird.

Damit ergibt sich das in Abbildung 3.1 dargestellte, grobe Klassendiagramm des Plugins, woraus die zuerst beschriebene Aufteilung in Configurators und Actions genauso ersichtlich ist, wie die ausgeprägte Verwendung von Enumerations um die UWE-Stereotypen, Diagramme, Transformationen und Tags zu verwalten.

3.2 Grundlagen effizienter Entwicklung

Bereits während der Einarbeitungsphase wurde klar, dass man um effizient zu Testen und zu Debuggen nicht nur eine lauffähige Entwicklungsumgebung wie eclipse [3] benötigt, sondern auch eine gut eingerichtete – womit mehr als ein funktionierendes System zur Versionskontrolle gemeint ist.

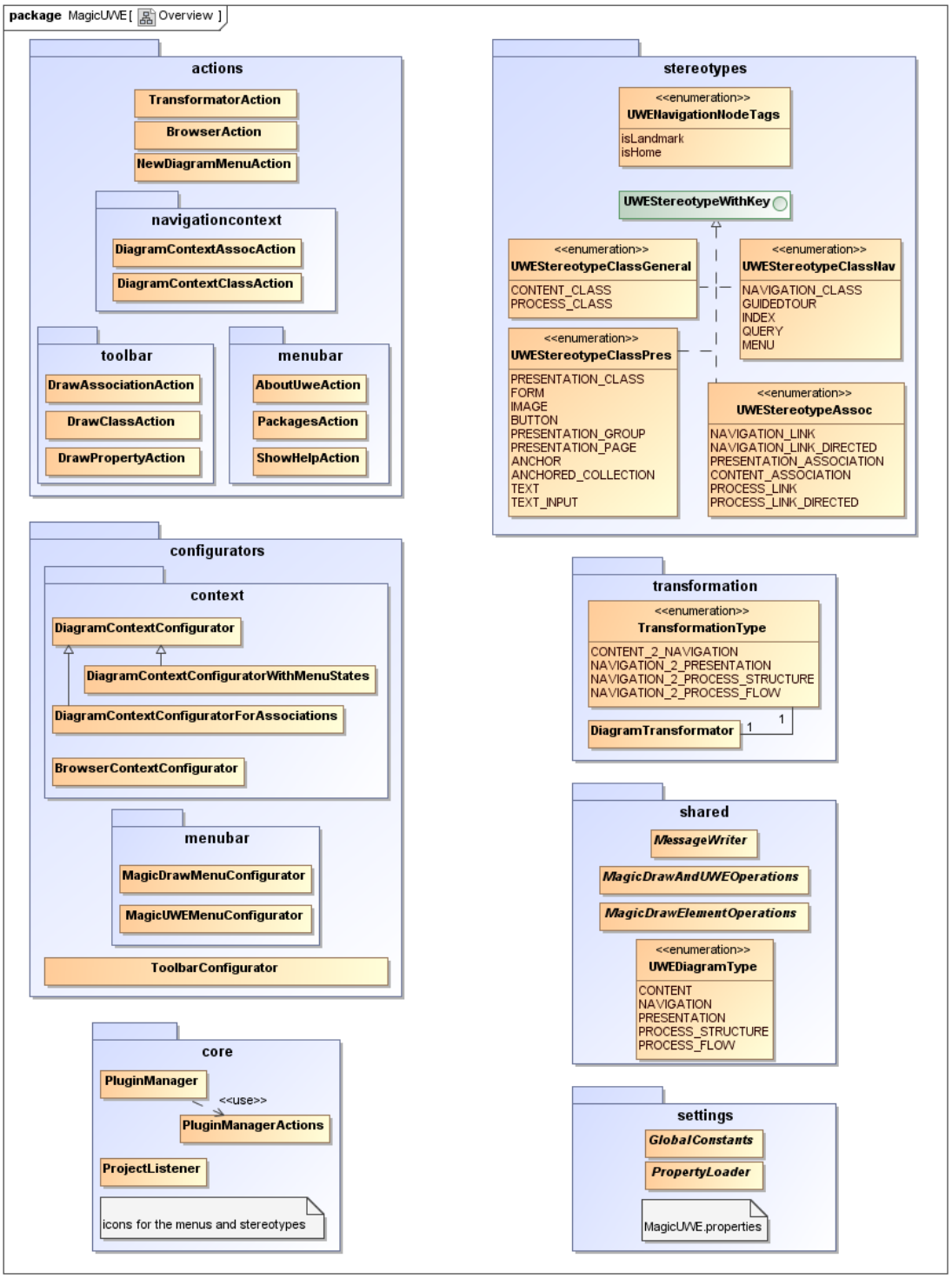


Abbildung 3.1: UML Klassendiagramm von MagicUWE - Übersicht

3.2.1 Debugging

Natürlich kann man Fehler in das Logfile[1] ausgeben lassen (MagicUWE implementiert dazu extra einen `MessageWriter`, der auch Fragen an den User und seine Antworten protokolliert), sofern das Logging in der Datei *MagicDrawHome*`/data/debug.properties` durch die Zeile `log4j.logger.magicUWE=DEBUG` eingeschaltet wurde.

Doch zur Fehlersuche ist es weitaus praktischer, den Debug-Modus von eclipse zu verwenden. Dazu muss eine neue Debug-Konfiguration angelegt werden, worauf es auf die Startklasse `com.nomagic.magicdraw.Main` und die Parameter der Virtual Machine `-Xmx800M -XX:PermSize=130M` ankommt. Außerdem ist es nötig, das MagicUWE Programmverzeichnis als *Working directory* einzustellen (sonst wird das Profil nicht gefunden) und die Bibliotheken aus *MagicDrawHome*`/lib` in den *Classpath* einzubinden.

3.2.2 Build-Skript

Bevor man das Plugin starten oder sogar debuggen kann, muss der aktuelle Quellcode inklusive der benötigten Bilder und Dateien in ein jar-Archiv gepackt werden und die alte Plugindatei in *MagicDrawHome*`/plugins/MagicUWE` damit überschrieben werden. Dazu ist ein Neustart von MagicDraw notwendig.

Weitaus einfacher gestalten sich die Arbeitsbedingungen, wenn man ein Build-Skript für diese Aufgabe verwendet, zum Beispiel ein Ant-Skript [4] namens `build.xml`.

In einer XML Datei kann man verschiedene, aufeinander aufbauende Targets (Ziele) angeben, die in der gegebenen Reihenfolge abgearbeitet werden. Ant ist in der Lage Code inklusive aller benötigten Bibliotheken zu kompilieren, jar-Dateien zu erstellen, Dateien zu kopieren, Javadoc zu generieren und nicht zuletzt fremde Programme auszuführen, was im Fall von MagicDraw besonders angenehm ist. Als Beispiel ist in Listing 3.1 das Target „build“ abgedruckt, das nur von „welcome“ (einer Begrüßungsanzeige) abhängt und die Aufgabe hat, die class-Dateien für das spätere jar-file, aus den neuen Sourcen heraus zu erstellen.

```
1  <target name="welcome" if="magicDraw.dir">
2      <echo level="info" message="Welcome. This is the ${ant.project.
           name} build-file"/>
3      <echo level="info" message="located in ${ant.file}"/>
4  </target>
5
6  <target name="build" depends="welcome" if="src.dir">
7      <echo level="info" message="Deleting old class-files.."/>
8      <delete includeemptydirs="true">
9          <fileset dir="${classes.dir}" includes="**/*"/>
10     </delete>
11     <echo level="info" message="Building ${ant.project.name}-classes
           from source files.."/>
12     <mkdir dir="${classes.dir}"/>
```

[1] Das Logfile von MagicDraw befindet sich standardmäßig im Homeverzeichnis des aktuellen Benutzers unter `.magicdraw_personal/15.1/md.log`

```
13      <javac classpathref="project.classpath" debug="on" srcdir="${src.
            dir}" destdir="${classes.dir}" encoding="utf-8" failonerror="
            yes"/>
14      <!-- non-class-files must be copied seperately-->
15      <copy todir="${classes.dir}">
16          <fileset dir="${src.dir}">
17                  <exclude name="**/*.java"/>
18                  </fileset>
19      </copy>
20  </target>
```

Listing 3.1: Auszug aus dem Ant Build-Skript

Und um damit nochmal auf das Debuggen aus dem letzten Kapitel zurückzukommen: in dem „build“-target ist für den Eclipse-Debugger die Option `debug="on"` nötig (Zeile 13), damit die Zeilennummern des Codes zur Laufzeit verfügbar sind. Es ist möglich, dieses Ant-Skript so flexibel zu gestalten, dass es sowohl unter Linux, als auch unter Windows läuft. Dazu gibt es die zusätzliche Datei `build.os.properties`, in der alle Pfade für die aktuelle Plattform definiert werden (u.a. `magicDraw.dir`, s. Zeile 1). In `build.properties` werden weitere Eigenschaften abgelegt, wie z.B. der Pfad zum Quellcode (`src.dir`) und die Parameter für den Installer.

Zur Auslieferung des Projekts, wird der Installer IzPack [11] verwendet. Da er umfangreiche Konfigurationsmöglichkeiten bietet, fällt es leicht, die einzelnen Schritte anzupassen, das Sprachfile zu verändern und Versionen mit und ohne Quellcode und Javadoc automatisch mit dem Ant-Skript zu erstellen. Für das bei der Installation angezeigte Lizenzfeld, fiel die Entscheidung auf die *Common Public License* [2], da sie das Plugin als Open Source definiert ohne zu verlangen, dass die Hosting-Software MagicDraw ebenfalls frei sein muss.

3.3 Migration

Nachdem die Grundlagen für effektives Arbeiten geschaffen waren, stand zunächst die Migration des mdUWEPlugins [1] von MagicDraw 12.0 auf MagicDraw 14.0 und später 15.1 an, wobei Refactoring und Fehlerbehebung ein notwendiger Bestandteil waren. Außerdem wurde (angelehnt an Magic*Draw*) „Magic*UWE*“ als prägnanter, neuer Name des Tools eingeführt.

3.3.1 Anpassung an die MagicDraw 15.1 OpenAPI

Glücklicherweise gibt es ein Dokument namens *OpenAPI user's guide* im Verzeichnis *`MagicUWEHome`*`/openapi/docs`, in dem im Kapitel „Plugins migration to MagicDraw 15.0 and later OpenAPI“ (fast) alle Änderungen der OpenAPI beschrieben sind. Dies ist auch dringend nötig, da die API nicht abwärtskompatibel ist und so z.B. die Funktion `uml.BaseElement.getProject()` nicht länger anbietet, dafür jedoch als guten Ersatz `core.Project.getProject(BaseElement)` bereithält. Etwas mühsam gestaltete sich die

Auswahl der benötigten Libraries aus dem `lib`-Verzeichnis, da half hauptsächlich Ausprobieren, bis im Projekt und im Ant-Skript je die richtigen 7 von 44 Bibliotheken eingebunden werden konnten.

Zunächst schien alles zu funktionieren, nur kam es hin und wieder zu beinahe nicht reproduzierbaren Abstürzen, nicht nur des Plugins, sondern auch von MagicDraw. Sie hinterließen keinerlei Spuren im logfile und es dauerte eine ganze Weile, bis die Auslöser gefunden werden konnten: sie entpuppten sich als Aufrufe einer OpenAPI-Funktion, die neue Diagrammtypen definiert.

So z.B. `Application.getInstance().addNewDiagramType(processDiagram)`, wobei alle Diagrammarten, wie im Beispiel `processDiagram` von `DiagramDescriptor` erbten, was nicht erwünscht ist, da UWE-Modelle unter Verwendung von gewöhnlichen UML Klassen- und Aktivitätsdiagrammen erstellt werden sollen. Daher wurden alle abgeleiteten Diagrammtypen durch eine zweckdienliche Enumeration namens `UWEDiagramType` ersetzt. Sie sorgt unter anderem für die Erstellung von Diagrammen des jeweiligen UWE-Typs und verwendet dazu die Methode `ModelElementsManager.getInstance() .createDiagram(umlDiagramType, destPackage)`, wobei `umlDiagramType` eine Konstante aus `DiagramTypeConstants` ist, die im Konstruktor der einzelnen Enum-Elemente angegeben wird.

3.3.2 Umstieg auf das UWE-Profil v1.7

Der Wechsel auf das UWE-Profil v1.7 verlief dagegen problemlos, es waren in MagicUWE lediglich die Namen der Stereotypen und deren Pfad anzupassen. Da für die Toolbar zukünftig 16x16 Pixel große Icons verwendet werden sollten (s. Unterabschnitt 3.4.1) und dieses Format auch in den Klassen von MagicDraw unskaliert und damit schöner dargestellt wird, fiel der Entschluss alle UWE-Stereotypen neu zu zeichnen. Als besonders hübsch erwies sich das `png`-Format mit durchsichtigem Hintergrund und halbdurchscheinenden Pixeln zur Kantenglättung. Die Originale zur einfachen Weiterbearbeitung befinden sich als `xcf`-Dateien (GIMP [5]) im Ordner `MagicUWE/res/originalIcons`.

3.3.3 Bugfixing und Refactoring

Fehlersuche und -behebung darf mit Recht als eine der Hauptaufgaben dieser Projektarbeit dargestellt werden, wobei es nicht nur um vergessene Test-Codezeilen ging, sondern auch um etliche `null`-Pointer-Exceptions, z.B. bei den damals implementierten Transformationen. Verwirrende Meldungen an den Benutzer waren auszubessern, ganze Blöcke offensichtlich ungetesteten Codes (weil unfunktionsfähig) zu entfernen, bzw. zu ersetzen und nicht zuletzt ist es unpraktisch, wenn nach dem Einfügen spezieller UWE-Elemente das am Ende gespeicherte Projekt plötzlich Inkonsistenzen aufweist. Letztere entstehen mit MagicDraws OpenAPI sehr leicht, u.a. wenn Elemente erzeugt werden ohne ihnen mit der `setOwner(Element)`-Methode einen Platz in der Hierarchie des Containment-Baums zuzuweisen. Leider machen sich derartige Bugs oft erst beim Beenden von MagicDraw bemerkbar, was das Debugging erschwert.

So mühsam die Fehlersuche sein kann, so schön ist es, wenn am Ende funktionierender

Code herauskommt, und da man am meisten Fehler während des Refactorings findet, wurden verteilte Stereotypenklassen zu kompakten Enums, Konstruktoren vereinheitlicht, Klassen extrahiert oder zusammengeführt und unter anderem auch die Warnmeldungen von eclipse konfiguriert und dementsprechend eliminiert. Es hat sich bestätigt, dass ein übersichtlicher, gut kommentierter, auto-formatierter Code oft die Voraussetzung ist, um überhaupt zu sehen, wo Code-reuse möglich ist – was die Zeilenmenge in den ersten Wochen der Weiterentwicklung zunächst erheblich schrumpfen ließ.

3.4 Beispiele der OpenAPI-Nutzung

Bereits gegen Ende der Fehlerbehebungsphase wurde MagicUWE um neue Features erweitert. Drei Weiterentwicklungen werden im Folgenden beispielhaft beschrieben, um zu zeigen, wie MagicDraws OpenAPI genutzt wurde und auf welche Art auftretende Schwierigkeiten gelöst werden konnten.

Die Javadoc-Dokumentation der MagicDraw OpenAPI enthält leider nur eine Auswahl an Klassen und Funktionsnamen nebst Parametern und verzichtet weitestgehend auf Erläuterungen. Auf diese Art und Weise wird man als Pluginentwickler genötigt vieles auszuprobieren, wobei erschwerend dazukommt, dass es trotz dem vollautomatischem Build-Skript auf heute handelsüblichen Rechnern eine gute Minute dauern kann, bis MagicDraw inklusive einem vordefinierten Beispielprojekt geöffnet ist. Eine Alternative gibt es in Form der OpenAPI-Newsgroup [9] von MagicDraw, in der man für gewöhnlich innerhalb von ein bis zwei Werktagen eine Antwort bekommt.

3.4.1 Konfigurieren der Menüs

Wie einleitend erwähnt, wird der Zusammenbau von Menüs und ihrer Aktionen durch Funktionen der Klasse `PluginManagerActions` übernommen. Die OpenAPI sieht vor, dass Konfiguratoren dem MagicDraw `ActionsManager` Aktionen (in Form von Objekten des Typs `ActionsCategory`) hinzufügen, oder schon vorhandene Kategorien zu verändern, was verwendet wird um MagicDraw Menüs zu erweitern.

Als simples Beispiel bietet sich das Kontextmenü an, dass es erlaubt, aus dem Containment-Baum heraus neue UWE-Diagramme zu erstellen (Listing 3.2). Die `init()`-Methode registriert dabei den `BrowserContextConfigurator`, der wiederum aus einer Funktion der Klasse `PluginManagerActions` eine (erweiterte) `MDAction` übernimmt und zu dem MagicDraw Menü mit der ID `NEW_DIAGRAM` hinzufügt, was das gewünschte Menü für neue UML-Diagramme ist. Hinter der `MDAction` verbirgt sich die Klasse `NewDiagramMenuAction`, deren überschriebene `actionPerformed`-Methode die Erstellung neuer UWE-Diagramme startet.

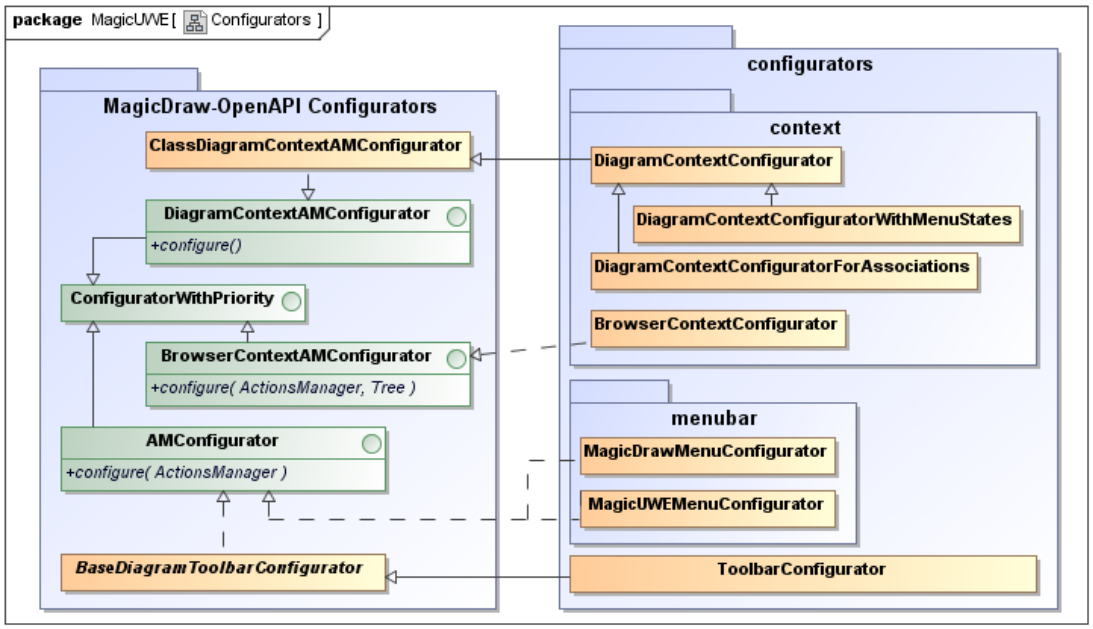
Die abgedruckte `configure`-Methode (ab Zeile 10) ist überschrieben und wird von MagicDraw aufgerufen, wenn ein Menü initialisiert und/oder dargestelltwerden soll. Besonders zu beachten ist in Zeile 5 der einfache Zugriff auf die (sich in einer Enumeration befindlichen) Diagrammtypen und in Zeile 19 das Hinzufügen des gerade bearbeiteten Menüeintrags.

```
1  // In der init()-Methode der Klasse PluginManager:
2  ActionsConfiguratorsManager manager = ActionsConfiguratorsManager.
       getInstance();
3
4  // add MagicUWE diagrams to containment browser new diagram context
       menu
5  for (UWEDiagramType dgTyp : UWEDiagramType.values()) {
6    manager.addContainmentBrowserContextConfigurator(new
         BrowserContextConfigurator(PluginManagerActions.
         getNewDiagramAction("UWE - ", dgTyp, true), ActionsID.
         NEW_DIAGRAM, null));
7  }
8
9  // In der Klasse BrowserContextConfigurator:
10 public void configure(ActionsManager mngr, Tree tree) {
11   // don't show the menu, if the selected element is read only or not
         an element
12   if (canBeUsed(tree)) {
13     if (actionsID == null) {
14       mngr.addCategory((ActionsCategory) action);
15     } else {
16       // Show menu within another (see actionsID)
17       NMAction magicDrawAction = mngr.getActionFor(actionsID);
18       if (magicDrawAction instanceof ActionsCategory) {
19         ((ActionsCategory) magicDrawAction).addAction(action);
20       } else {
21         logger.debug("Something went wrong with the menu, can't find
               MagicDraw's \"" + actionsID + "\" menu");
22         return;
23       }
24     }
25   }
26 }
```

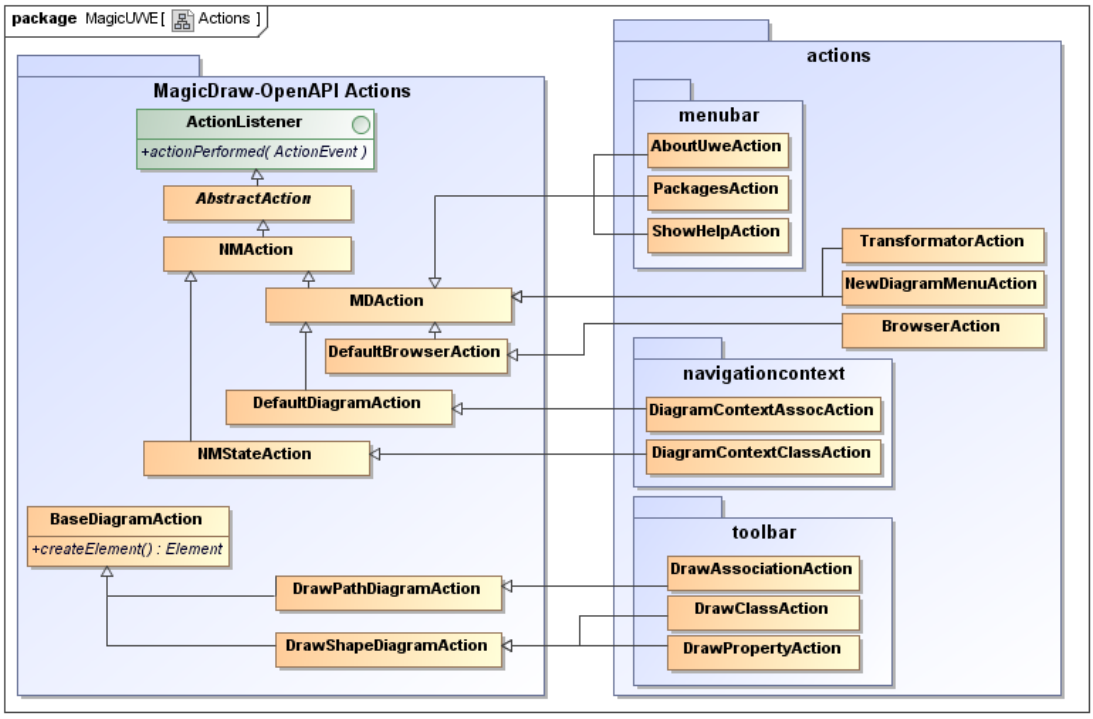
Listing 3.2: Beispielcode zum Konfigurieren des „new Diagram“-Kontextmenüs

Da die Aktionsklassen und Kategorien der OpenAPI schwer im Kopf zu behalten sind, findet sich in Abbildung 3.2 eine Übersicht in Form eines Klassendiagramms. In Bild 3.2(a) ist unter anderem der `BrowserContextConfigurator` zu sehen, und in Diagramm 3.2(b) die Klasse `NewDiagramMenuAction` aus obigem Beispiel, die von `MDAction` abgeleitet ist.

Bei der Menükonfiguration (`PluginManagerActions`) werden auch Icons gesetzt, wodurch bei der Benutzung mit einem Blick die gewünschte Funktionalität ausgewählt werden kann. Für die Toolbar war zu beachten, dass es zwar sehr wohl möglich wäre, die Stereotypenbilder aus dem UWE-Profil zu laden, es aber in der Implementierung der OpenAPI den Mangel gibt, dass die Toolbarelemente, von beim Laden des Projekts offenen Diagrammen, kein Icon bekommen. Folglich wurde auf diese Option verzichtet und alle Icons werden einheitlich als `png`-Datei aus dem Paket `core/icons` geladen. Der Da-



(a) Configurators



(b) Actions

Abbildung 3.2: UML - Vererbung der Configurators & Actions für die Menüs

teiname wird dabei meist automatisch z.B. in der Toolbar aus der jeweiligen Stereotyp-Enumeration generiert, was den Vorteil hat, dass sobald ein neuer Enum-Eintrag erstellt wird, sofort das passende Bild, der richtige Name und der gewünschte Shortcut geladen wird und der Entwickler sich auf die Programmierung der hinzukommenden Funktionalität konzentrieren kann.

Manchmal kann man sich kaum sicher sein, ob von einem Bug der OpenAPI zu sprechen ist, oder nicht. So wird beim einmaligem Öffnen eines Kontextmenüs z.B. im Navigationsdiagramm die `configure` Methode beliebige Male ausgeführt. Eine Nachfrage in der Newsgroup ergab, dass das bei MagicDraw intern ebenso ist, wodurch sich die Behäbigkeit der Kontextmenüs in diesem Tool von selbst erklärt. Als Pluginentwickler kann man sich dem nicht enziehen, denn liefert die Funktion bei mehreren Aufrufen hintereinander nicht jedes Mal das Gleiche, so wird das Menü nicht zuverlässig angezeigt.

3.4.2 Einfügen von Properties

Properties können aus der Toolbar heraus eingefügt werden, wie in Unterabschnitt 2.2.1 beschrieben. Dazu tritt die Klasse `DrawPropertyAction` (s. Abb. 3.2(b)) in Aktion, die als Attribut einen speziellen `UWEStereotypeWithKey` hält. Darüber bestimmt sie auch, ob ein Hinweis bezüglich des falschen Diagrammtyps angezeigt werden soll. Die von `DrawShapeDiagramAction` überschriebene `createElement`-Methode braucht sich dank OpenAPI nur um das Präparieren der Zielklasse und die Erstellung der neuen Klasse nebst Property kümmern; das Einzeichnen des Elements wird geerbt.

Listing 3.3 zeigt einen kleinen Ausschnitt der unspektakulären Erstellung der neuern Klasse und der zugehörigen Property, wobei in Zeile 8 der `UWEStereotypeWithKey` namens `classStereotype` den Namen des UWE-Stereotyps liefert. Die Variable `names` (Zeilen 4 und 10) stammt aus dem Eingabedialogfeld, in das der Benutzer die gewünschten Namen eingegeben hat.

```
1  // Extrahierte Unterfunktion von DrawPropertyAction.createElement()
2  // create property
3  Property property = project.getElementsFactory().
       createPropertyInstance();
4  property.setName(names[0]);
5  property.setAggregation(AggregationKindEnum.COMPOSITE);
6
7  // create class and set it as property type
8  Class clazz = MagicDrawElementOperations.createClass(classStereotype.
       toString());
9  clazz.setOwner(presentationElement.getElement().getOwner());
10 clazz.setName(names[1]);
11 property.setType(clazz);
12 return property;
```

Listing 3.3: Erstellung der neuen Klasse und Property

Hier nicht abgebildet ist die zuvor getroffene Entscheidung, ob die Property in eine andere Property hineingeschachtelt werden soll, oder ob ihr künftiger Container eine

Klasse ist, von der zunächst der Anzeigetyp so gesetzt werden muss, dass Properties in sie hineingezeichnet werden können, s. Listing 3.4 (Achtung, „property“ bezieht sich im Code auf die Eigenschaft des angezeigten Elements, in diesem Fall soll `suppress structure` auf `false` gesetzt werden). Übrigens unterscheidet MagicDraw `PresentationElements` von `Elements`, wobei Ersteres die Darstellung eines Elements im Diagramm wiedergibt und Letzteres das dahinterstehende Element, das im Containment Baum zu finden ist.

```
1 public static void setSupressStrctureOfClass(PresentationElement
      presentationElement, boolean val) {
2   PropertyManager propertyManager = new PropertyManager();
3   propertyManager.addProperty(new BooleanProperty(PropertyID.
        SUPPRESS_STRUCTURE, val));
4   try {
5     PresentationElementsManager.getInstance().
          setPresentationElementProperties(presentationElement,
          propertyManager);
6   } catch (ReadOnlyElementException e) {
7     logger.error("Can't modify the SUPPRESS_STRUCTURE property of the
          element!");
8   }
9 }
```

Listing 3.4: Setzen der „suppress structure“ Eigenschaft einer Klasse

Problematisch bei der Übernahme der Zeichenfunktion von der OpenAPI ist, dass man trotz vorangegangener, eindeutiger Selektion, beim Zeichnen dann doch zusätzlich in ein anderes Element einfügen darf, wie in Bild 3.3 dargestellt. Im Code hat man keinen Zugriff auf diese Funktion. Die Folge ist, dass die Klasse zwar richtig erstellt und hinzugefügt wird, die Erstellung des gezeichneten Elements jedoch nicht funktioniert.

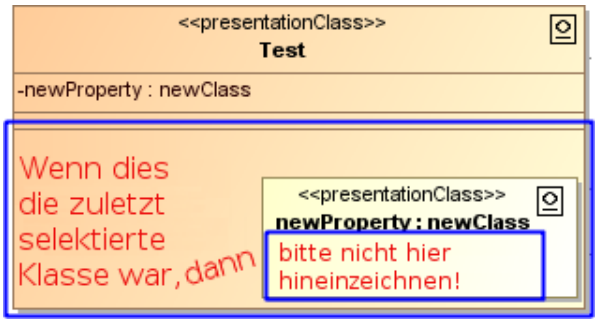


Abbildung 3.3: Gefahr der Auswahl des falschen Zielcontainers für eine Property

Ein weiteres Problem bei allen Operationen über Toolbar ist, dass wenn man während des (nicht abgeschlossenen) Zeichnens ein neues UWE-Diagramm erstellt, ein genereller Inkonsistenzfehler auftritt, der zu einer unschönen Messagebox von MagicDraw führt (Abb. 3.4). Leider gibt es für diesen Bug keinen Workaround. Bleibt zu hoffen, dass in den nächsten Versionen von MagicDraw die Anfragen der Newsgroup berücksichtigt werden und vielleicht künftig generell darauf verzichtet wird in der OpenAPI andere

Methoden bereitzustellen wie in MagicDraw intern, denn das Einfügen von Diagrammen aus MagicDraw heraus funktioniert einwandfrei.

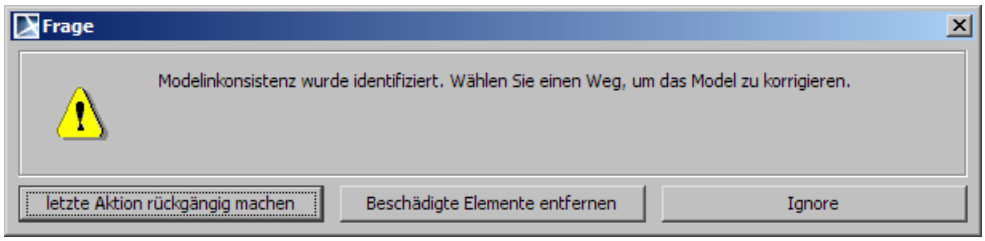


Abbildung 3.4: Noch nicht abgeschlossene Toolbaroperationen führen durch gleichzeitiges Einfügen von UWE-Diagrammen zu Modellinkonsistenz

3.4.3 Einfügen von Zugriffselementen (Access Primitives)

Auch wenn man einen Index, ein Menu, eine Query oder eine GuidedTour einfügt, wird auf Stereotypen-Enumerations zurückgegriffen. Darüber hinaus muss im Gegensatz zu den Properties hinter jedem Eintrag ein anderes Verhalten stecken, worum sich die Klasse `DiagramContextAssocAction` kümmert, die von der OpenAPI-Klasse `DefaultDiagramAction` erbt und deren `actionPerformed`-Methode überschreibt. In ihr steckt die Logik, die dafür sorgt, dass:

- Die Richtungen in der die Assoziationen ursprünglich gezeichnet wurden beachtet wird, wozu ein HashMap Verwendung findet (`HashMap<AssociationView, Boolean>`). Bei diesem steht `false` dafür, dass sich die „Kopfklasse“ (zu der die eingefügte Klasse später genau eine Verbindung haben wird) auf der „Client“-Seite der Assoziation befindet. (Die beiden Enden einer Assoziation werden intern als „Client“ und „Supplier“ geführt.)

  Dies ist wichtig, da der Benutzer gegebenenfalls eine Richtung für das Einfügen auswählen muss, wenn diese nicht, zum Beispiel durch die Multiziplität „*“ (nur bei Index) oder durch die Wahl von mehreren Assoziationen, die zu unterschiedlichen Klassen führen, erkenntlich ist. In besagtem HashMap hängt die spätere Einfügerichtung vom ersten Element ab, was zur Folge hat, dass bei der Richtungsermittlung alle Booleans nochmal gekippt werden müssen, wenn sich z.B. erst bei der letzten selektierten Assoziation die Richtung entscheidet. Das sollte in der Praxis jedoch selten der Fall sein, da es ungewöhnlich ist, zwei Klassen untereinander mit vielen Assoziationen zu verknüpfen.

- Bei Index, Query oder GuidedTour eine Nachfrage erscheint, wenn mehr als eine Assoziation gewählt war, da dies den einzufügenden Zugriffselementen nicht entspricht (z.B. sollte dann statt einem `«index»` besser ein `«menu»` gewählt werden).

- Der Rollenname der Assoziation als Name der neuen Klasse verwendet wird, sofern dieser angegeben war.

- Bei Index (sofern noch nicht vorhanden) ein „*“ an das von dem eingefügten Element abgewandten Assoziationsende platziert wird.
- Alle Assoziationen den Stereotyp «navigationLink» bekommen, außer bei Query, da dort «processLink»s angebracht sind.
- Das Menü mit einer Komposition an die Kopfklasse gebunden ist und keinen Stereotyp hinzugefügt bekommt.
- Und nicht zuletzt die neue Klasse samt ihrer geänderten bzw. neuen Assoziationen eingefügt und auf der Diagrammfläche ausgerichtet wird.

Leider ist das UWE-Kontextmenü in Navigationsdiagrammen nur schwer zu öffnen, da MagicDraw beim Rechtsklick auf das Ende einer selektierte Assoziation diese wieder deselektiert und daraufhin das Menü nicht einblendet (statt dessen ist das Menü für das Editieren der Rollen zu sehen). Abhilfe ist, entweder gezielt in die Mitte zu klicken, oder noch einfacher die Assoziation(en) zu selektieren und dann davon entfernt den Rechtsklick auf die freie Diagrammfläche zu setzen.

Normalerweise werden die ausgeführten Aktionen des Plugins in die Rückgängig-Liste von Magic-Draw integriert, was durch einen SessionManager ermöglicht wird (Listing 3.5). Werden durch die OpenAPI jedoch Assoziationen entfernt oder geändert, so löscht MagicUWE bei einer Rückgängig-Operation zwar die neuen Assoziationen, stellt jedoch die alten nicht wieder her.

```
1 SessionManager.getInstance().createSession("Inserting new element
      between the selected associations");
2   \\ .. Ausfuehrung der geplanten Operationen
3 SessionManager.getInstance().closeSession();
```

Listing 3.5: Nicht immer funktionierende Sitzungsverwaltung der OpenAPI

3.5 Beschränkung der OpenAPI

Wie in den vorigen Kapiteln angedeutet, ist die MagicDraw OpenAPI nicht abwärtskompatibel, auch bei alltäglichen Routinen nicht fehlerfrei und unflexibel, da der Quellcode nicht zur Verfügung steht und daher nicht verbessert oder erweitert werden kann. Durch die mangelhafte Dokumentation der API kommt es zudem zu nicht vernachlässigbarem Overhead, weil für die Verwendung einer noch unbekannten Funktion MagicDraws zuerst in den mitgelieferten Beispielprojekten gesucht werden muss. Darauf folgt meist ein Suchvorgang im Javadoc nach erratenen Funktionsnamen (wobei die Trefferquote in manchen Fällen erstaunlich hoch ist, dafür dass wichtige Funktionen und Klassen dort gar nicht auftauchen und man muss anschließend nur noch die jeweilige Funktionsweise erforschen). Letztendlich muss meist doch in der Newsgroup gefragt werden, wobei man nicht selten die Antwort bekommt, dass etwas zwar in MagicDraw selbst, nicht jedoch über MagicDraws OpenAPI verfügbar ist (- oder anders funktioniert, wie am Beispiel des Zeichnens von Elementen aus der Toolbar in Unterabschnitt 3.4.2 zu sehen ist).

Ein Beispiel für eine sehr angenehme Erweiterung wäre nicht nur die bereits genannte Möglichkeit von der Toolbar aus auf Elemente im Diagramm zeigen zu können, bevor die Zeichenfunktion (z.B. für Properties) aktiviert wird, sondern auch die Interaktion mit dem Plugin per Drag&Drop.

So wäre für die Transformation von der Navigations- zur Präsentationsschicht ein Fenster mit Checkboxen in einer Tabelle denkbar, bei der links die Klassen des zu transformierenden Navigationsdiagramms und oben die häufigsten Präsentations-Stereotypen (wie `«page»` und `«presentationClass»` aufgetragen sind. Problem bei der statischen Auflistung aller Klassen des aktuellen Diagramms wäre, dass das bei vielen (unbenannten) Klassen zu unübersichtlich wird. Dies ließe sich vermeiden, wenn Drag&Drop von Klassen in das Plugin-Fenster von der OpenAPI unterstützt würde, denn dann könnte die Tabelle dynamisch und sehr intuitiv von Nutzer selbst, je nach Bedarf, gefüllt werden.

4 Modellierung eines Beispiels mit MagicUWE

Dieses Kapitel ist eine Einführung in UWE an Hand eines Web-Adressbuchs, welches - möglichst mit dem Leser gemeinsam - Schritt für Schritt mit MagicUWE modelliert wird. Eine mehrsprachige Onlineversion findet sich unter dem Namen „UWE-Tutorial“ in leichter Abwandlung auf der UWE-Webseite [14].

4.1 Vorbereitungen für das Adressbuch-Beispiel

Wir erstellen ein neues Projekt in MagicDraw und nutzen dabei die „UWE“-Vorlage.

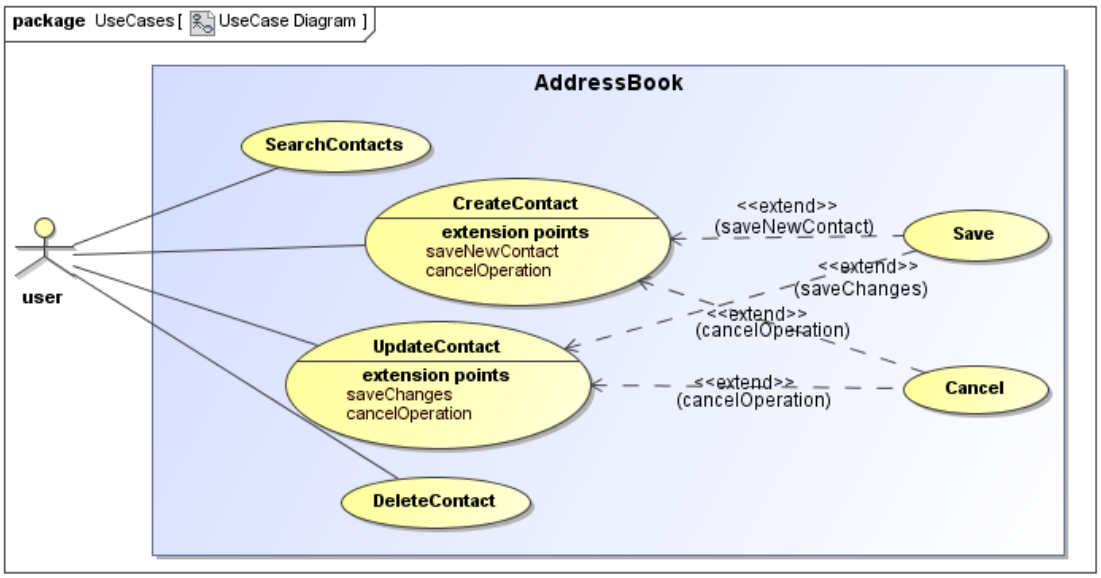


Abbildung 4.1: Adressbuch - Anwendungsfalldiagramm

Zum Beispiel kann man mit einem normalen UML Anwendungsfalldiagramm (Use Case) anfangen (Abb. 4.1). Planen wir doch zunächst die Grundfunktionalitäten unseres Adressbuchs: Der Benutzer soll Kontakte suchen und löschen können, er soll Kontakte erstellen oder ändern und sich nach seinen Eingaben entscheiden können, ob er Speichern oder Abbrechen möchte. Natürlich können Sie in Ihren Diagrammen weitere Funktionalitäten modellieren, wir beschränken uns hier jedoch - der Übersichtlichkeit halber - auf die eben genannten.

4.2 Content Model (Inhaltsmodell)

Erstellen Sie ein neues Content Diagramm aus dem MagicUWE Hauptmenü. Es ist ein normales UML Klassendiagramm, daher müssen wir überlegen, welche Klassen wir für unser Beispiel benötigen. Es bietet sich eine Adressbuchklasse („AddressBook“) an, die eine Menge von Kontakten enthält. Jeder benannte Kontakt soll eine E-Mailadresse, zwei Telefonnummern und zwei Adressen speichern. Name und Mailadresse sind Zeichenketten (Strings), Telefonnummer und Adresse sind separate Klassen, die einige Zusatzinformationen umfassen (Abb. 4.2).

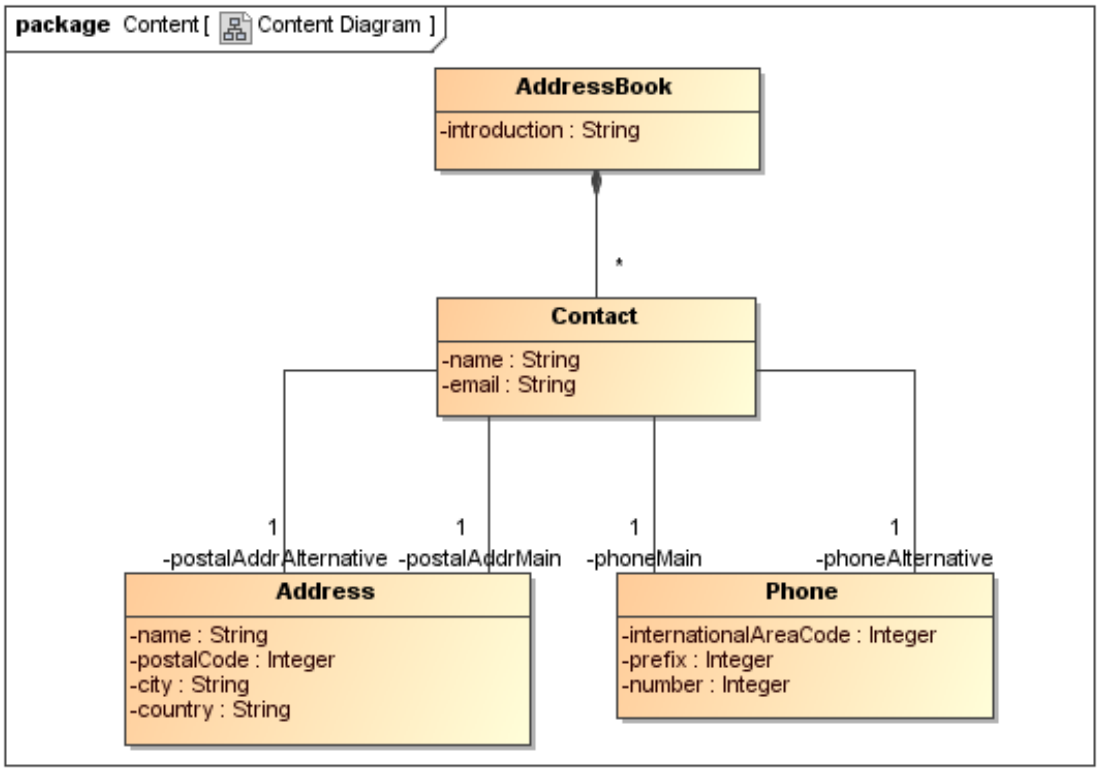


Abbildung 4.2: Adressbuch - Content Diagram

Wozu brauchen wir das „introduction“-Attribut in der Klasse AddressBook? - Dort speichern wir den Begrüßungstext, der auf der Startseite angezeigt werden soll.

4.3 Navigation Model (Navigationsmodell)

Es wäre gut, wenn man eine Möglichkeit hätte, darzustellen, wie Webseiten miteinander verbunden sind. Daraus folgt, dass man eine Diagrammart benötigt, die Nodes (Knoten) und Links (Verknüpfungen) beinhaltet.

Nodes sind Navigationselemente, die durch Links verbunden sind. Ein Node muss

keiner ganzen Webseite entsprechen, es handelt sich vielmehr um einen bestimmten Aspekt davon, der auch auf verschiedenen Webseiten dargestellt werden kann.

UWE hält mehrere Stereotypen für Navigationsknoten bereit, was man am besten am Beispiel erläutert: Zunächst erstellt man ein Navigationsdiagramm mit der Content-to-Navigation Transformation. In unserem Fall erhalten wir damit ein Diagramm mit mehr Nodes als wir benötigen (Abb. 4.3). Für Nodes und Links werden die Stereotypen `«navigationClass»` und `«navigationLink»` benutzt.

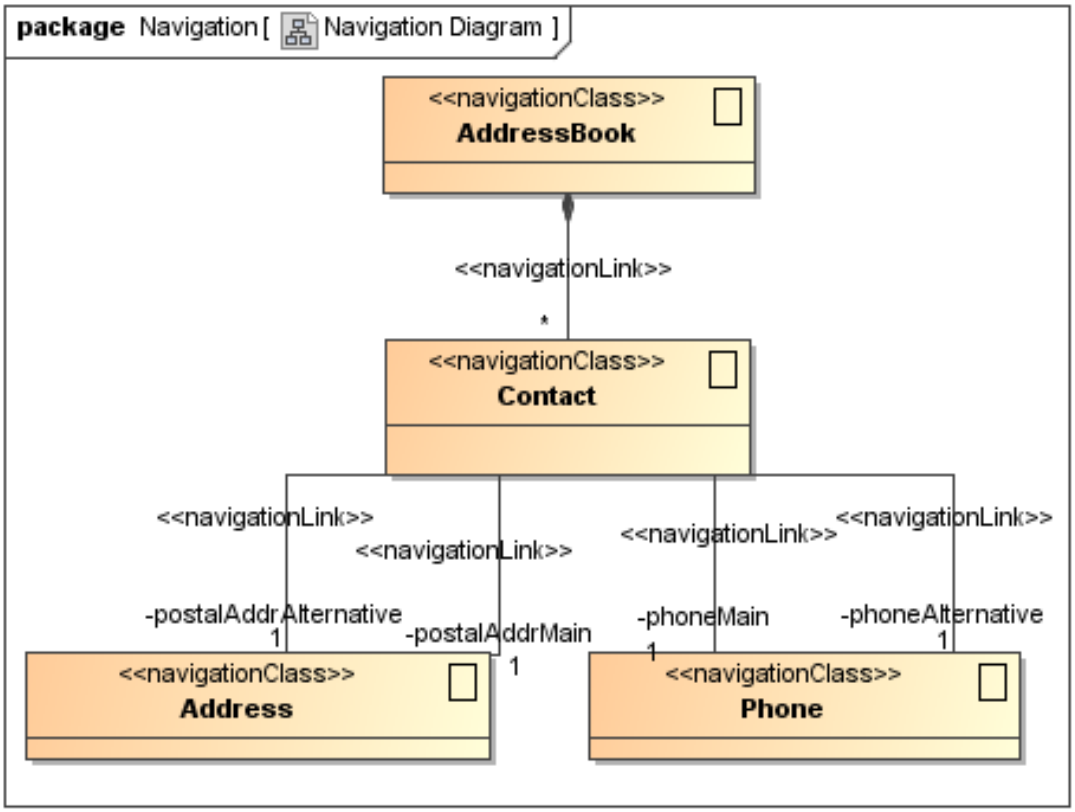
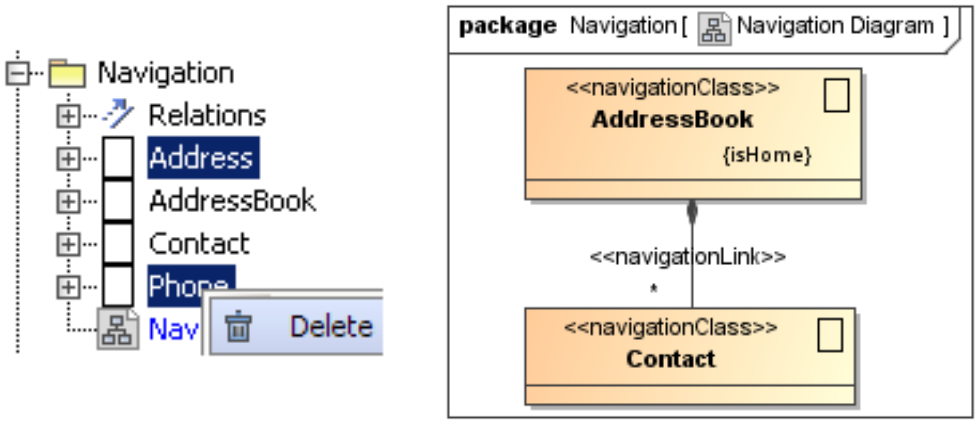


Abbildung 4.3: Adressbuch - Transformation von Content zu Navigation

Wollen wir überhaupt eine Navigationsmöglichkeit vom Kontakt zur Adresse oder der Telefonnummer darstellen? - Nein, denn für die Navigation ist das nicht wirklich relevant. Wir können die beiden Klassen also, wie im Screenshot 4.4(a) gezeigt, aus dem Containment-Baum löschen, wodurch sich das Bild 4.4(b) ergibt.

AddressBook soll unsere Hauptseite sein, daher der Tagged Value {`isHome`}, der über das MagicUWE-Kontextmenü der Klasse gesetzt werden kann. (Tags werden in MagicDraw mit „Eigenschaften“ übersetzt.)

Was halten Sie von einem Adressbuch, das immer alle Kontaktinformationen auf der gleichen Webseite anzeigt? - Nicht ganz das, was wir wollen. Schließlich sollte unsere Anwendung die Operationen unseres Anwendungsfalldiagramms bereitstellen; deswegen



(a) Löschen nicht benötigter Klassen (b) resultierendes Navigationsdiagramm

Abbildung 4.4: Adressbuch - nicht benötigte Klassen aus dem Navigationsdiagramm entfernen

brauchen wir eine Hauptseite mit Verbindungen zu mehreren Navigationsknoten:

1. ContactList - um jeden Kontakt von einem Link der Hauptseite aus zu erreichen.
2. (Contact)Search - um nach Kontakten suchen zu können.
3. ContactCreation - um neue Kontakte zu erstellen und anschließend anzuzeigen.

UWE nutzt ein Menü (`«menu»`), um zu verschiedenen Klassen zu navigieren. Fügen wir eines über das Assoziations-Kontextmenü von MagicUWE ein und nennen es „Main-Menu“ (Abb. 4.5).

1. Die Kontaktliste können wir fast genauso einfügen, der Stereotyp `«index»` gibt an, dass es sich um eine Liste von Objekten des selben Typs handelt.

Um die anderen beiden Klassen einzufügen, verwenden wir die MagicUWE Toolbar:

2. Die Klasse für die Suche wird mit dem Stereotyp `«query»` versehen. Zu suchen heißt Code auszuführen, also binden wir die Klasse mit einer `«processLink»` Assoziation an.
3. Das Erstellen eines Kontakts ist auch ein Prozess, jedoch kein vordefinierter, daher wird hier der Stereotyp `«processClass»` verwendet. (Später werden wir dazu auch die zugehörige Aktion modellieren.)

Es scheint sinnvoll, dass ein neu erstellter Kontakt anschließend angezeigt wird. Und wenn man nach etwas sucht, dann erwartet man eine Kontaktliste mit den Ergebnissen. Für diese ausgehenden Assoziationen verwendet man ebenfalls den `«processLink»`-Stereotyp, allerdings diesmal in gerichteter Form. So werden ungewollte Duplikate vermieden, die sonst leicht durch das Zurückwechseln entstehen könnten (Abb. 4.6).

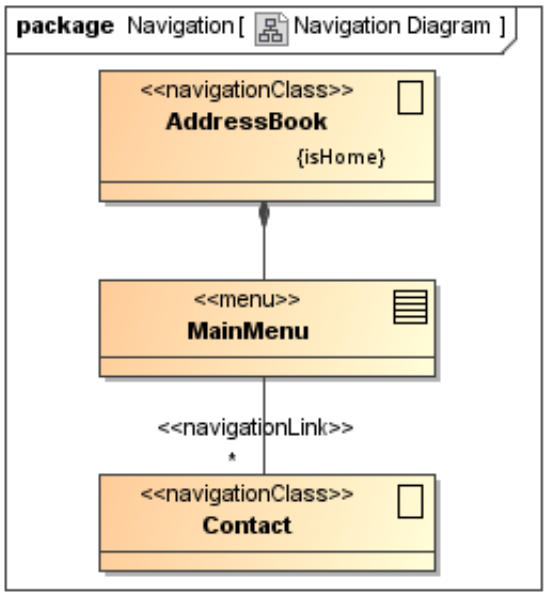


Abbildung 4.5: Adressbuch - Navigation mit Menü

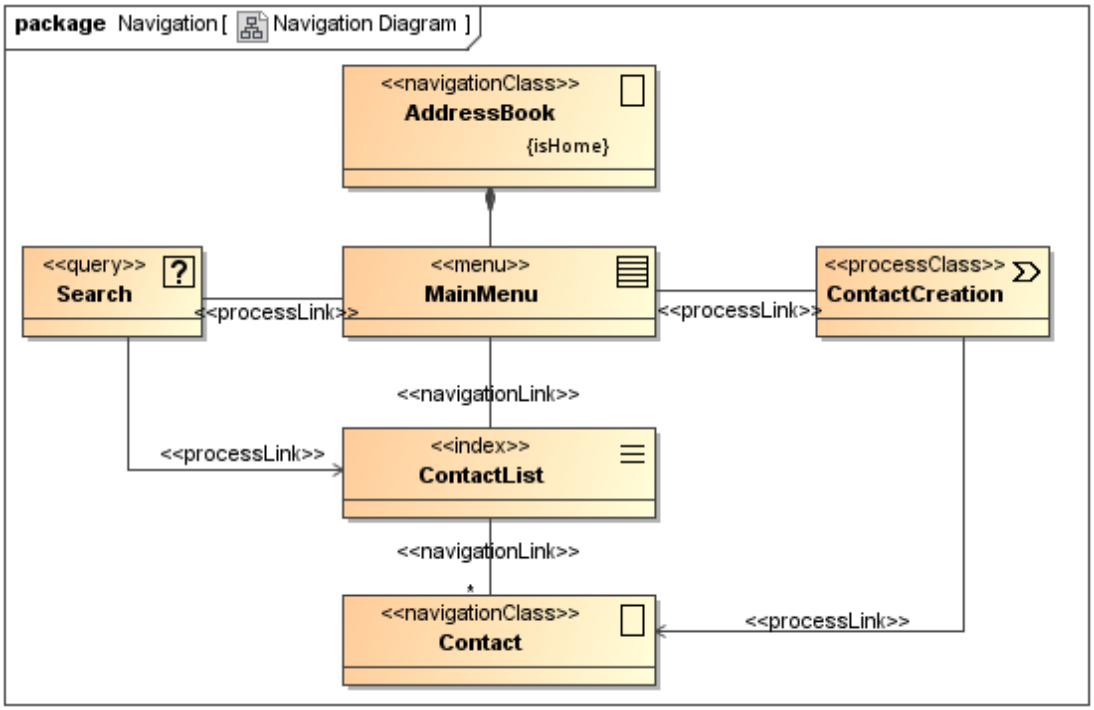


Abbildung 4.6: Adressbuch - Zwischenstand des Navigationsdiagramms

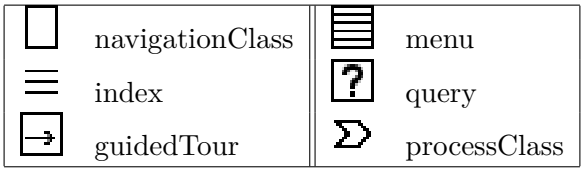


Abbildung 4.7: Namen und Symbole der Navigations-Stereotypen

In den einigen Diagrammen dieses Beispiels sind die Stereotypen nur durch ihre Symbole dargestellt. Man kann MagicDraw jedoch auch so konfigurieren, dass sowohl Symbole als auch Namen angezeigt werden. (Hier wird jweils nur eine Auswahl an Stereotypen vorgestellt (insbes. Abb. 4.7 und 4.9). Die Spezifikation aller UWE- Stereotypen kann dem Dokument „User Guide and Reference“ [7] entnommen werden.)

Um unser Navigationsmodell zu vervollständigen, müssen wir die noch fehlenden Funktionalitäten - wie „Kontakt löschen“ und „Kontakt ändern“ hinzufügen (auch hierzu sei wieder auf das Anwendungsfalldiagramm aus Abbildung 4.1 verwiesen). Zu beiden Klassen wird von einem konkreten Kontakt aus navigiert, daher brauchen wir hier wieder ein Menü - und nennen es „ContactMenu“ um auszudrücken, dass es auf der Webseite eines jeden Kontakts angezeigt wird (s. Abb. 4.8).

4.4 Presentation Model (Präsentationsmodell)

Das Navigationsmodell zeigt nicht, welche Navigations- oder Prozessklassen zu welcher Webseite gehören. Um diese Information darzustellen, nutzen wir ein Präsentationsdiagramm, das wir als neues Diagramm aus dem Kontextmenü des `Presentation`-Pakets im Containment Baum anlegen.

Wir fügen eine `«presentationPage»` Klasse ein und zeichnen, ebenfalls unter Benutzung der Toolbar, Properties mit UWE-Stereotypen hinein, um auszudrücken, dass das Element auf dem übergeordneten Element - wie z.B. einer Webseite - platziert ist. Properties können verschachtelt werden, indem als Zielcontainer eine schon vorhandene Property selektiert wird. Auf diese Art enthält jeder Kontakt im Adressbuch (`«presentationClass»`-Property) weitere Elemente, wie Texte und Buttons. Folglich werden für jeden Kontakt die zugehörige Mailadresse sowie Telefonnummer und Adressinformationen angezeigt.

Erinnern wir uns an das „introduction“-Attribut in unserem Content Diagramm und fügen eine Adressbuch-Hauptseite hinzu, die den Begrüßungstext enthält (Stereotyp `«text»`). Außerdem ist ein Formular (form) nötig, dass die textInput-Eingabefelder für das Suchkriterium und die zugehörige „Suchen“-Schaltfläche (button) enthält. Eine beliebige Anzahl Kontakte können angezeigt werden, was durch die Multiplizität „*“ ausgedrückt wird.

Message, Confirmation und ValidationError sind in Abbildung 4.10 modelliert, obwohl sie - der Übersicht wegen - nicht in unserem Navigationsdiagramm zu sehen waren. Es sind lediglich einfache Webseiten, die eine Nachricht oder eine Frage beinhalten.

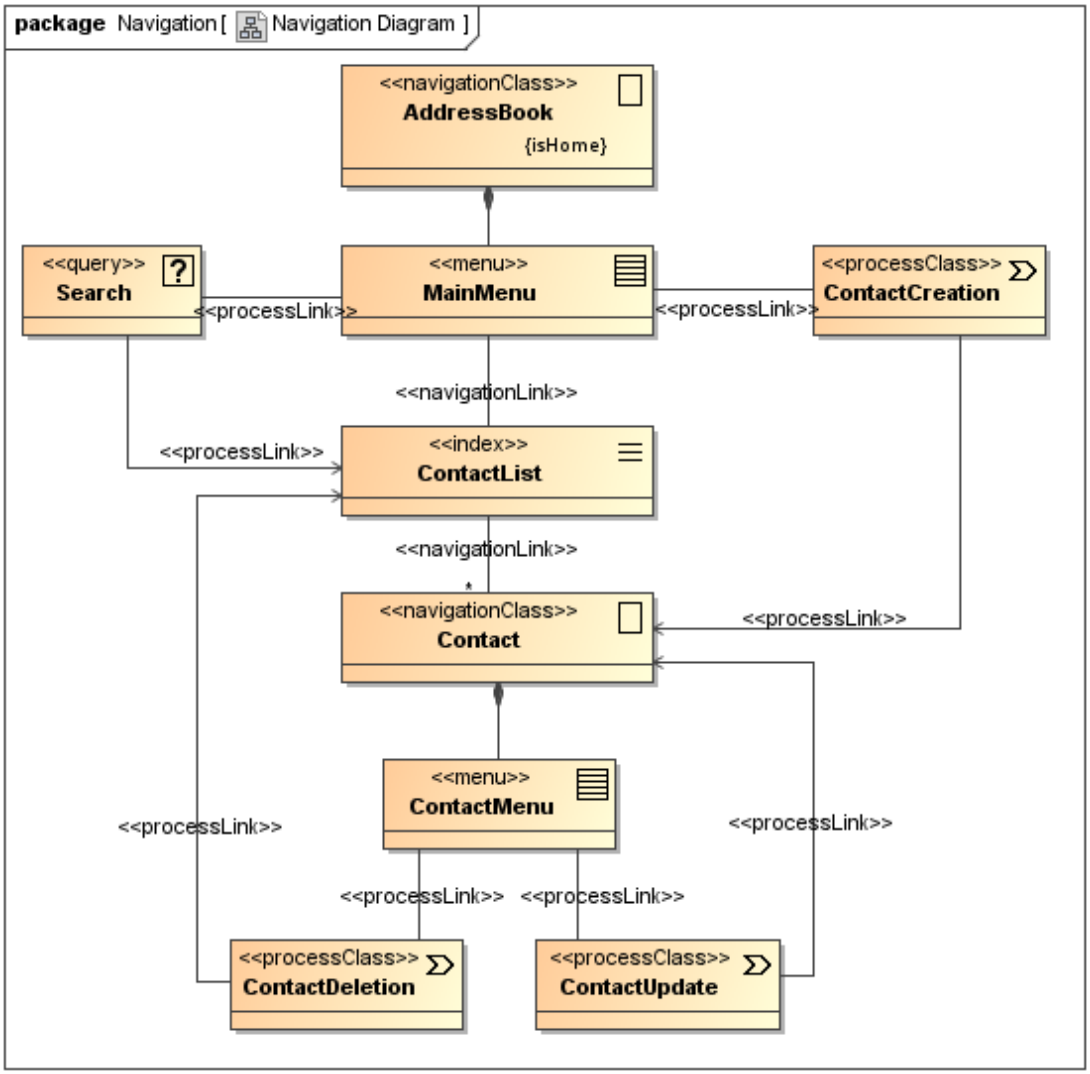


Abbildung 4.8: Adressbuch - vollständiges Navigationsdiagramm

Wie im Diagramm 4.11 zu sehen ist, sind ContentCreation und ContentUpdate vom Aufbau her äquivalent, nur die Namen der Seiten und die „ok“-Schaltflächen wurden ihrer jeweiligen Funktionalität angepasst.

4.5 Process Model (Prozessmodell)

Mit den bisherigen Modellen können wir bereits einige Merkmale unserer Webseite abbilden. Bisher wurde allerdings noch nicht festgelegt, wie die Aktionen, die hinter unseren Prozessklassen (Stereotyp «`processClass`») stecken, ablaufen. Das hier vorgestellte Prozessmodell (Process Model) umfasst:

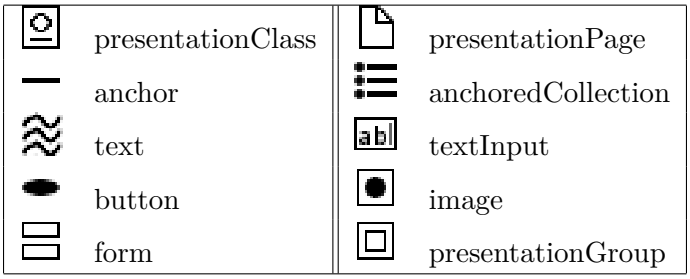


Abbildung 4.9: Namen und Symbole der Präsentations-Stereotypen

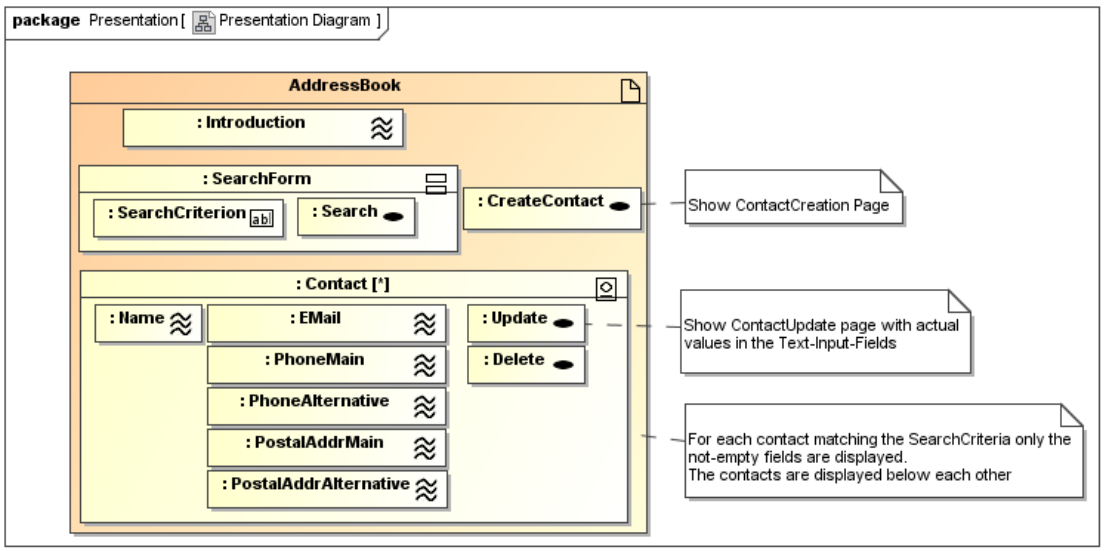


Abbildung 4.10: Adressbuch - Präsentationsmodell, Teil 1

- das Process Structure Model (Prozess-Strukturmodell), das die Beziehung zwischen verschiedenen Prozessklassen beschreibt und
- das Process Flow Model (Prozess-Ablaufmodell), das die Aktivitäten, die mit jeder «`processClass`» verknüpft sind, spezifiziert.

4.5.1 Process Structure Model (Prozess-Strukturmodell)

Um die Beziehung zwischen verschiedenen Prozessklassen zu beschreiben, erstellen wir mit der Navigation zu Process-Structure Transformation ein Klassendiagramm. Daraus folgt ein Diagramm, dass nur die in der Abbildung 4.12 dargestellten, rot umrandeten Klassen beinhaltet.

Wie man sehen kann haben wir noch andere Klassen hinzugefügt, um auszudrücken, dass alle drei Operationen eine Bestätigung mit Fragetext (Klasse Confirmation mit Attribut question) enthalten. (siehe unser Präsentationsdiagramm in Abschnitt 4.4).

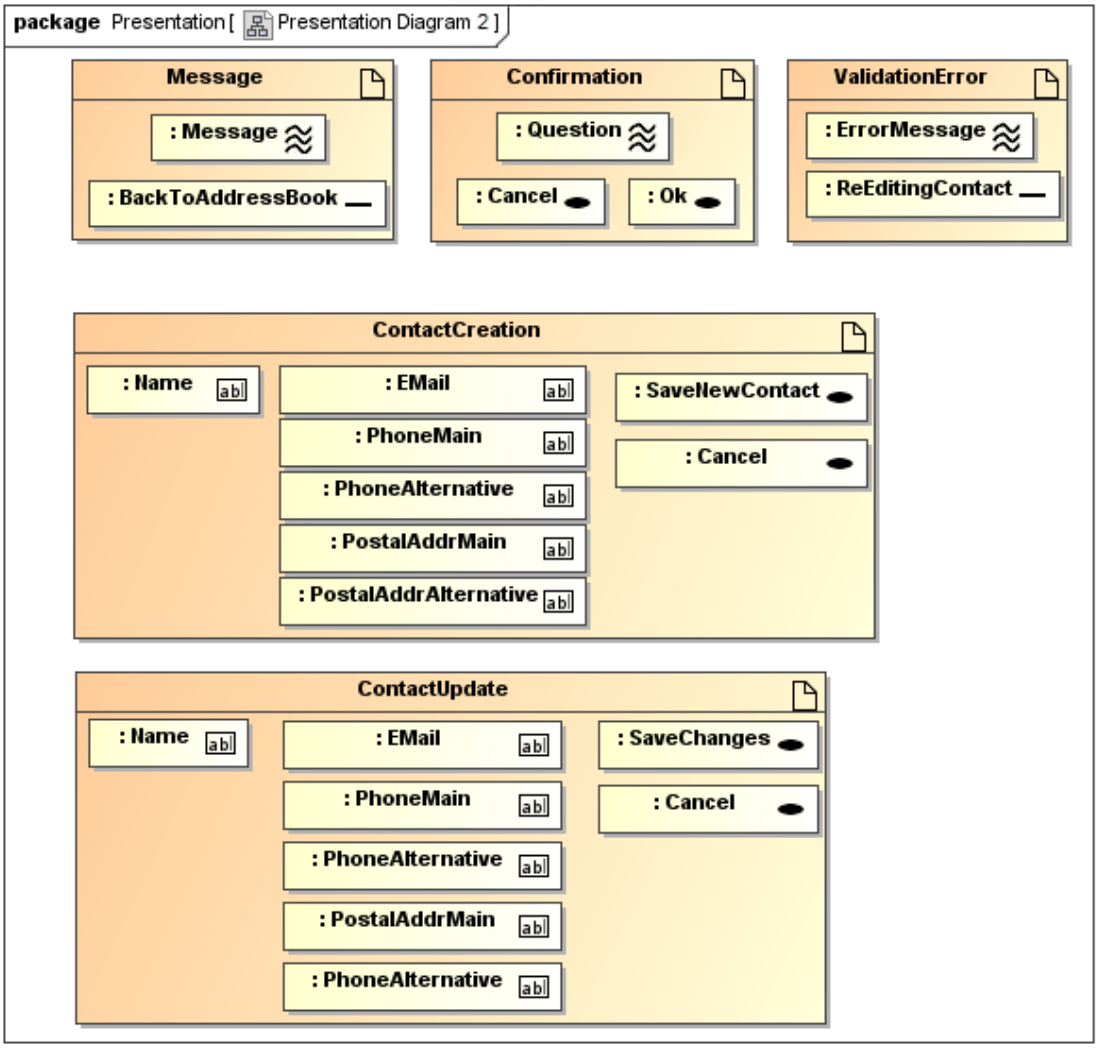


Abbildung 4.11: Adressbuch - Präsentationsmodell, Teil 2

Das heißt, wenn der Benutzer einen Kontakt löschen will, bekommt er eine Nachricht angezeigt und erst nachdem er auf „ok“ geklickt hat, wird die Löschaktion ausgeführt. ContactCreation und ContactUpdate verhalten sich sehr ähnlich: beide erben von der abstrakten Klasse ContactProcessing, was die Eingabe gültiger Werte für die Attribute von ContactDataInput sicherstellt. Dies trägt zum Beispiel zur Vermeidung von leeren Namen und damit unsinnigen Einträgen bei. Sobald die Benutzereingaben verifiziert wurden, tritt natürlich kein Fehler auf und die jeweilige Bestätigungsseite wird angezeigt. Der nächste Abschnitt beschäftigt sich mit den Details der jeweiligen Aktivitäten.

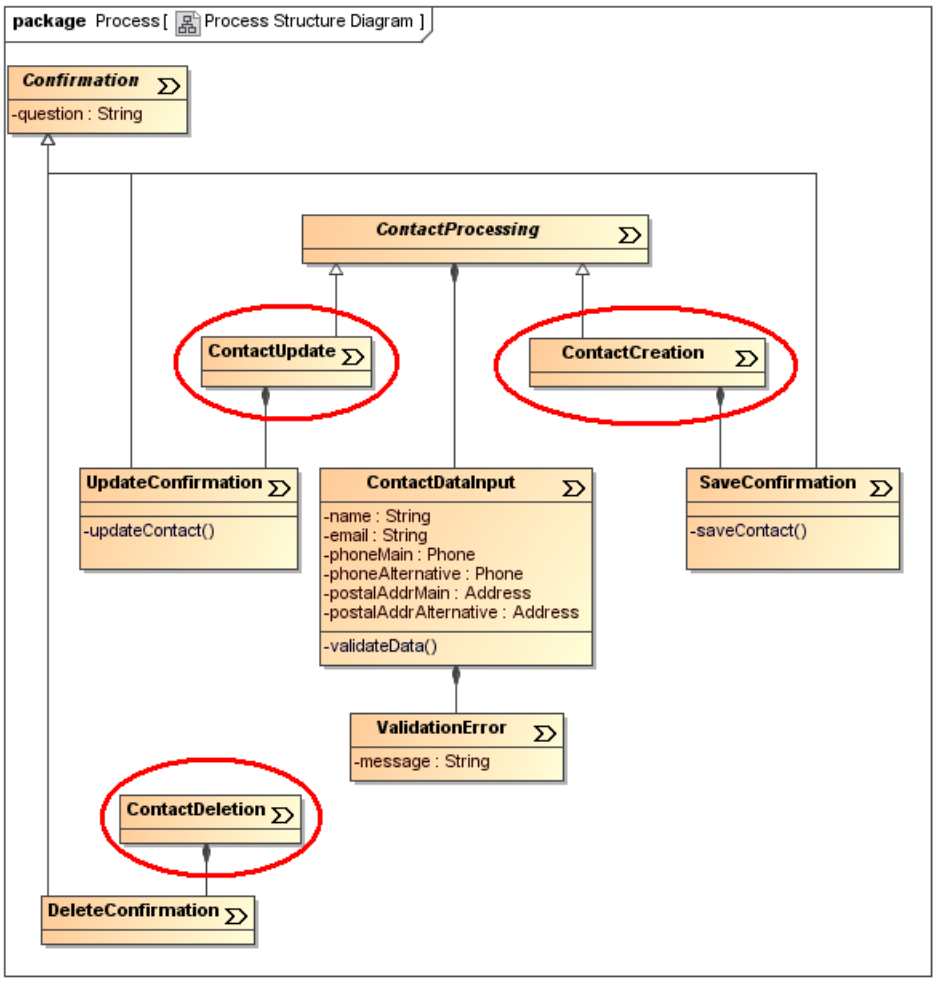


Abbildung 4.12: Adressbuch - Process Structure Diagramm

4.5.2 Process Flow Model (Prozess-Ablaufmodell)

Ein Process Flow (Prozessfluss), auch Process Workflow (Prozessarbeitsfluss) genannt, ist ein Aktivitätsdiagramm, das das Verhalten von Prozessklassen beschreibt. So wird zum Beispiel dargestellt was passiert, wenn der Anwender zu einer Prozessklasse (wie ContactCreation) navigiert.

Öffnen wir unser Navigationsdiagramm und führen die *Navigation to Process-Flows* Transformation aus, die uns drei leere Aktivitätsiagramme namens `ContactCreation`, `ContactDeletion` und `ContactUpdate` erstellt. Sie sollen dem Modellierer als Hinweis dienen, dass er dieses Verhalten modellieren muss, um die Web-Anwendung vollständig zu spezifieren.

Der Stereotyp `«user Action»` wird für Benutzeraktionen auf der Webseite benutzt, die einen Prozessablauf anstoßen oder eine Antwort auf eine konkrete Informationsan-

forderung abfragen.

Für die drei Process Flow Diagramme wird lediglich gewöhnliches UML benötigt. Diagramm 4.14 führt das Hinzufügen eines Kontakts aus, die darauf folgenden Abbildungen das Entfernen (Abb. 4.13) und Modifizieren (Abb. 4.15) eines Kontakteintrags.

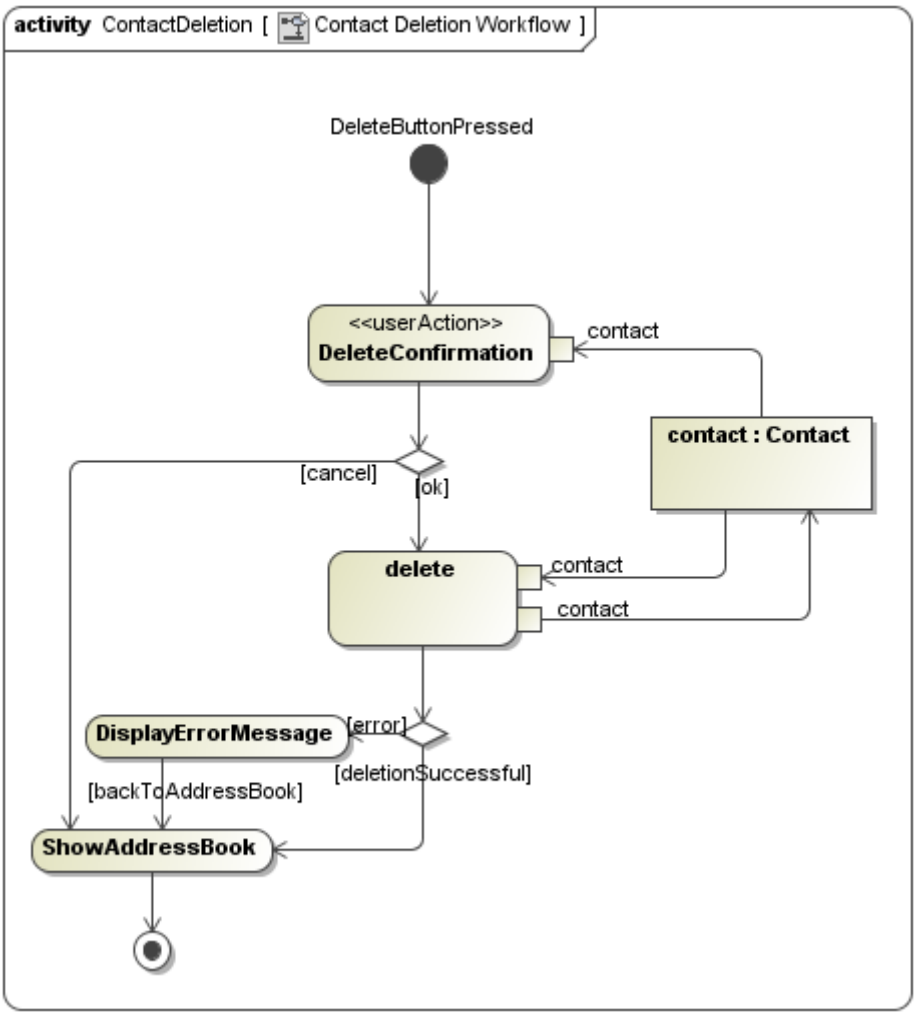


Abbildung 4.13: Adressbuch - Process Flow Diagramm, Entfernen

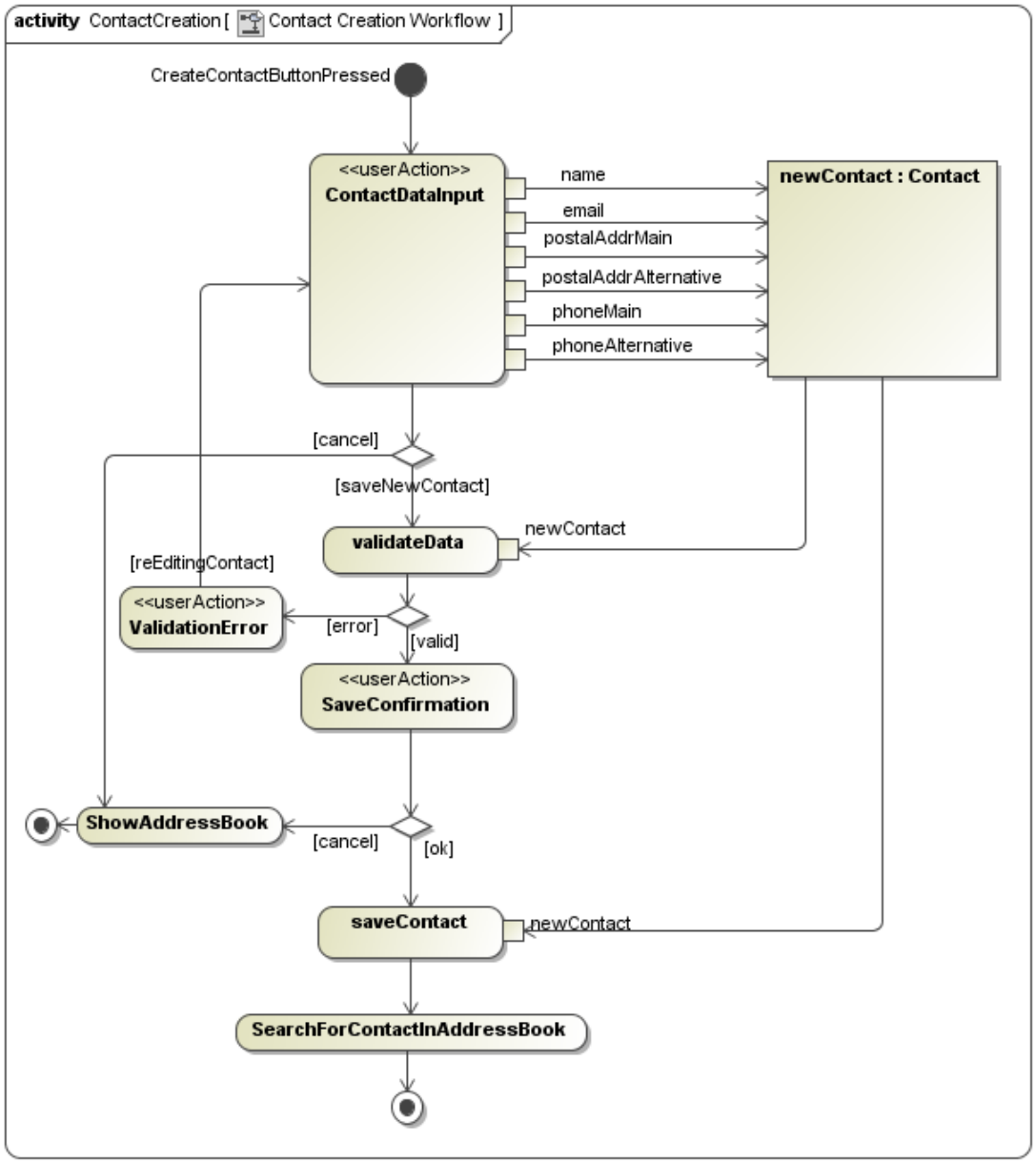


Abbildung 4.14: Adressbuch - Process Flow Diagramm, Hinzufügen

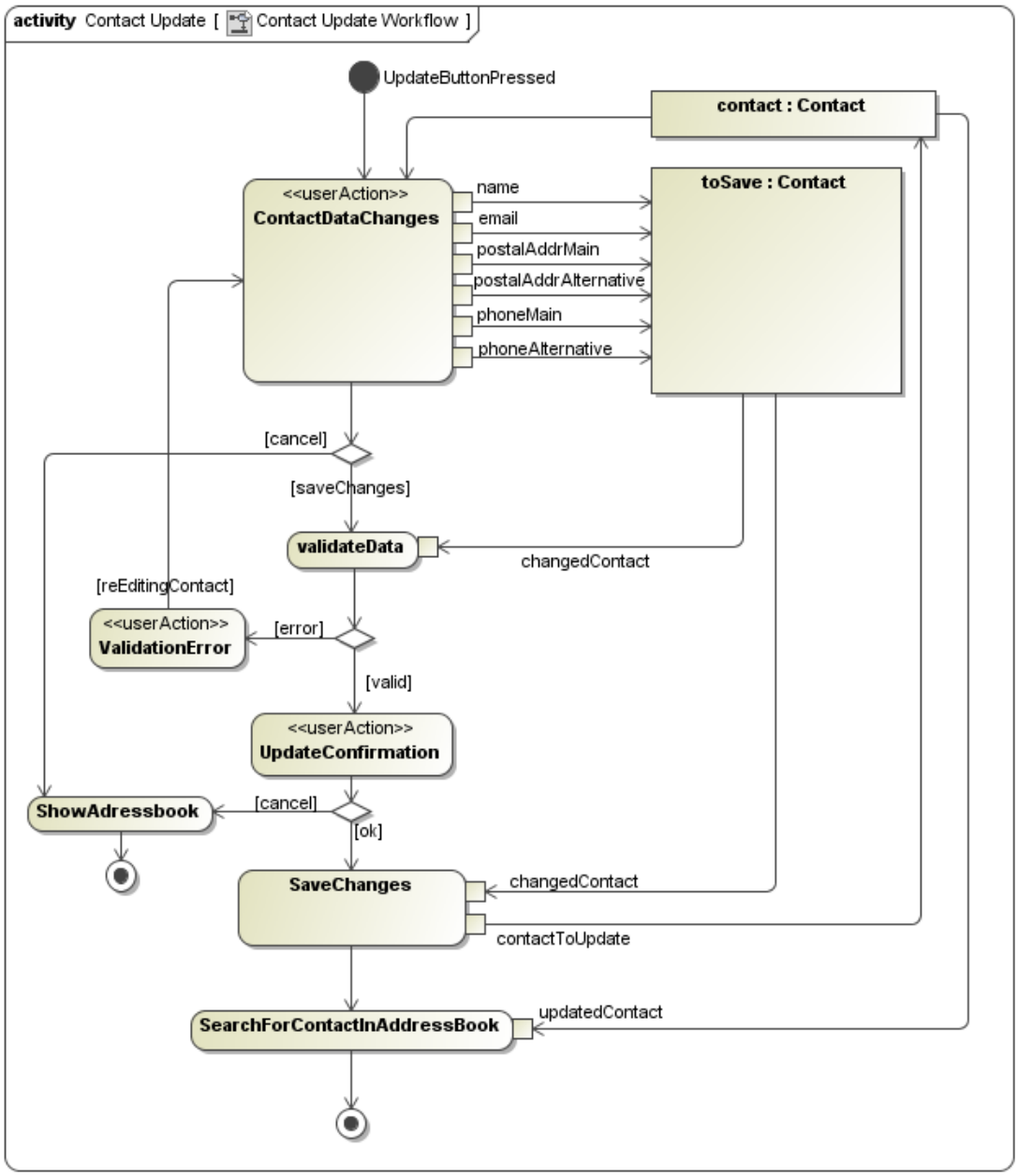


Abbildung 4.15: Adressbuch - Process Flow Diagramm, Modifizieren

5 Zusammenfassung und Ausblick

MagicUWE vereinfacht die Modellierung von Webanwendungen mit UWE und wie es so oft mit Vereinfachungen ist, finden sich immer neue Aspekte, die man gut in das Plugin einbauen könnte. So zum Beispiel eine Unterstützung für die Modellierung von Rich Internet Applications mit UWE, zu der erst kürzlich eine Diplomarbeit fertiggestellt wurde[10]. Hier wäre besonders der Einsatz von vorgefertigten Modellen für häufige asynchrone Vorgänge wie z.B. das interaktive Suchen-Dialogfeld zu überlegen.

Außerdem lassen sich die Transformationen verfeinern, in dem man z.B. im UWE-Profil einen Tagged Value für Content Klassen festlegt, der bestimmt, ob die Klasse bei einer Content zu Navigation Transformation beachtet wird. Dadurch könnte man sich das nachträgliche Löschen der nicht benötigten Klassen ersparen, das in dem Beispiel aus Abschnitt 4.3 nötig war.

Zusammenfassend kann man sagen, dass MagicUWE wegen der Weiterentwicklung von UWE und MagicDraw immer wieder einiger Anpassungen und Erweiterungen bedürfen wird, die dank der Implementierung - insbesondere wegen der ausführlichen Dokumentation, der der komfortablen Arbeitsbedingungen mit dem zum Projekt gehörenden Ant-Skript und des modularen Code-Aufbaus - leicht zu meistern sein sollten. Darüber hinaus ist das Plugin in der momentanen Version 1.2 voll einsatzfähig und erweitert MagicDraw 15.1 um eine Menge Features, angefangen von dem effizienten Einfügen von Elementen mit UWE Stereotypen, bis hin zu den jeweiligen Transformationen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Blagoev, Petar: *MagicDraw-Plugin zur Modellierung und Generierung von Web-Anwendungen*, LMU - Ludwig-Maximilians-Universität München, Institut für Informatik, Lehrstuhl für Programmierung und Softwaretechnik, Diplomarbeit, 2007

[2] Open Source Initiative: *Common Public License*. `http://www.opensource.org/licenses/cpl1.0.txt`. Version: 1.0, Abruf: 14. 3. 2009

[3] Eclipse Foundation, Inc.: *Eclipse.org home*. `http://www.eclipse.org/`. Version: 3.4.2, Abruf: 14. 3. 2009

[4] The Eclipse Foundation (Apache Software Foundation, Inc.): *Platform Ant Project*. `http://www.eclipse.org/eclipse/ant/index.php`. Version: 1.7.0, Abruf: 28. 3. 2009

[5] The GIMP Team: *GIMP - The GNU Image Manipulation Program*. `http://www.gimp.org/`. Version: 2.6, Abruf: 28. 3. 2009

[6] Koch, Nora ; Knapp, Alexander ; Zhang, Gefei ; Baumeister, Hubert: UML-Based Web Engineering: An Approach Based on Standards. Version: 2008. `http://www.pst.ifi.lmu.de/veroeffentlichungen/uwe.pdf`. In: Olsina, Luis (Hrsg.) ; Pastor, Oscar (Hrsg.) ; Rossi, Gustavo (Hrsg.) ; Schwabe, Daniel (Hrsg.): *Web Engineering: Modelling and Implementing Web Applications* Bd. 12. Springer, Berlin, 2008, Kapitel 7, 157–191

[7] Kroiss, Christian ; Koch, Nora: *UWE Metamodel and Profile*. `http://www.pst.ifi.lmu.de/projekte/uwe/download/UWE-Metamodel-Reference.pdf`. Version: 2008, Abruf: 14. 3. 2009

[8] No Magic, Inc.: *UML 2 diagramming, OO software modeling, Source code engineering Tool MagicDraw UML from No Magic*. `http://www.magicdraw.com/`. Version: Personal Edition 15.1, Abruf: 13. 6. 2008

[9] No Magic, Inc.: *MagicDraw OpenAPI related questions and discussions*. `news://news.nomagic.com/nomagic.products.magicdrawuml.openapi`, Abruf: 13. 6. 2008

[10] Morozova, Tatiana: *Modellierung und Generierung von Web 2.0 Webanwendungen*, LMU - Ludwig-Maximilians-Universität München, Institut für Informatik, Lehrstuhl für Programmierung und Softwaretechnik, Diplomarbeit, 2008

[11] Ponge, Julien: *IzPack - Package once. Deploy everywhere.* `http://izpack.org/`. Version: 3.11.0, Abruf: 14. 3. 2009

[12] Object Management Group (OMG): *OMG Unified Modeling Language (OMG UML), Superstructure.* `http://www.uml.org/`. Version: 2.1.2, Abruf: 26. 11. 2008

[13] LMU - PST: *UWE - UML-based web engineering.* `http://www.pst.ifi.lmu.de/projekte/uwe/`, Abruf: 14. 3. 2009

[14] LMU - PST: *UWE - Tutorial (German).* `http://www.pst.ifi.lmu.de/projekte/uwe/teachingTutorialGerman.html`, Abruf: 14. 3. 2009

[15] LMU - PST: *UWE - Metamodel and Profile.* `http://www.pst.ifi.lmu.de/projekte/uwe/publicationsMetamodelAndProfile.html`, Abruf: 14. 3. 2009

[16] LMU - PST: *UWE - ArgoUWE.* `http://www.pst.ifi.lmu.de/projekte/uwe/toolargoUWE.html`, Abruf: 14. 3. 2009

[17] LMU - PST: *UWE - MagicUWE.* `http://www.pst.ifi.lmu.de/projekte/uwe/toolMagicUWE.html`, Abruf: 14. 3. 2009

[18] LMU - PST: *UWE - MagicUWE - Referenz (German).* `http://www.pst.ifi.lmu.de/projekte/uwe/toolMagicUWEReferenceGerman.html`, Abruf: 14. 3. 2009