Ludwig-Maximilians-Universität München
Institut für Informatik

Diplomarbeit

Analyse-Patterns zur Modellierung und Generierung von Web-Systeme mit UWE

Aufgabensteller: Prof. Dr. Alexander Knapp

Betreuer: Dr. Nora Koch, Christian Kroiß

Bearbeiter: Bahruz Mechtiyev

Abgabedatum: 26. März 2009

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 26. März 2009 .....

Bahruz Mechtiyev

Kurzfassung

Einhergehend der zunehmende Verbreitung und Komplexität von Web-Anwendungen und des technologischen Fortschrittes des World-Wide-Web von einem statischen Informationsmedium zu einem dynamischen Anwendungsmedium, entwickelte sich in den letzten Jahren eine Reihe von Modellierungsansätzen für Web-Anwendungen.
Neue Herausforderungen an diese Ansätze ergeben sich aus dem aufkommenden Trend der modellgetriebenen Software-Entwicklung (*englisch* Model-Driven Software Development, MDSD). MDSD basiert in den meisten Fällen auf den graphischen Modellen, aus denen durch die Transformationen ein lauffähiges Sourcecode erzeugt wird. Bei MDSD haben Modelle nicht vorrangig dokumentarischen Wert, sondern sind gleichbedeutend mit Sourcecode, was anders ist als bei bisherigen Vorgehensweisen. Mit diesem Ansatz wird versucht, die Entwicklungszyklen zu verkürzen, die Qualität des Endprodukts zu erhöhen und die Entwicklungskosten zu senken.

Ziel dieser Arbeit ist es, die Analyse-Patterns für die Modellierung und Generierung von Web-Anwendungen mit UWE zu entwerfen und deren Verwendung bei der Modellierung zu diskutieren. Außerdem sind die große Vielfalt von den Entwurfsmustern und deren verschiedenen Interpretationen, die besonders für die Modellierung der Web-Anwendungen angepasst sind, in eine Bibliothek der Analyse-Patterns zusammengefasst.

Die Evaluierung soll aufzeigen, ob die Einsetzung der Analyse-Patterns mit aktuellen Modellierungsansätzen bereits möglich ist und in wie weit die Modellierung von Web-Anwendungen erleichtert wird.

Inhaltsverzeichnis

1 Einleitung

Das Erstellen ausschließlich statischer Websites zählt längst zur Vergangenheit. Heutzutage findet man kaum mehr eine Website, die lediglich aus einer Ansammlung statischer Dokumente ist und nur wenige Interaktionsmöglichkeiten berücksichtigen. Die Möglichkeit, Webseiten dynamisch in Abhängigkeit von Benutzereingaben und Datenbankinhalten zu generieren, machte das Web schnell zu einer universellen Plattform für alle Arten von Anwendungen. Ein wesentlicher Vorteil von Konstruktion einer Web-Anwendung für die Unterstützung der standardmäßigen Funktionen des Browsers besteht darin, dass die Funktionen unabhängig vom Betriebssystem des gegebenen Clients ausgeführt werden. Statt verschiedene Versionen für jede existierende Betriebssysteme zu schreiben, wird die Anwendung einmal erstellt und auf beliebiger Plattform ausgeführt.

Web-Anwendungen begegnen uns in verschiedenen Formen: Online Buchungssysteme, Suchmaschinen und Web Shops sind nur einige Beispiele für Einsatzgebiete mit unterschiedlichen Anforderungen. Mittlerweile verbreiten sich im WWW die Web-Anwendungen, die tatsächlich eine Synthese aus klassischen Software-Anwendungen und „ursprünglichen“ Websites darstellen.

Eine Web-Anwendung wird nach Prof. Dr. Knapp [13] wie folgt definiert:

> *„Eine Web-Anwendung ist ein Softwaresystem, das auf Spezifikationen des World Wide Web Consortium (W3C) beruht und Web-spezifische Ressourcen wie Inhalte und Dienste bereitstellt, die über eine Benutzerschnittstelle, dem Web Browser, verwendet werden.“*

In der Zeit des großen Internet-Booms um die Jahrtausendwende erlebten Web-Anwendungen einen gewaltigen Schub. Viele Unternehmen, unabhängig von deren Größe, bauten ihr Geschäftsmodell auf einer Web-Anwendung auf. In dieser Zeit wurden aber auch Web-Anwendungen wie z.B. Ebay, Yahoo und Google geboren, die heute zu einem selbstverständlichen Teil des Web-Lebens geworden sind.

1.1 Web Engineering als eigenständige Disziplin

Heutzutage stellen Web-Anwendungen komplexe und vollständige Softwaresysteme dar. Die Entwicklung dieser Web-Anwendungen setzt ein methodisches und ingenieurmäßiges Vorgehen voraus. Ein klarer Trend der letzten Jahre ist die zunehmende Strukturierung des Entwicklungsprozesses von Web-Anwendungen. Während in der Vergangenheit die meisten Web-Anwendungen in einer Mischung aus Darstellung (statischen HTML-Dateien) und Anwendungslogik entwickelt wurden, finden sich heute mehr und mehr Vorgehensmodelle und Technologien, die sich an die klassische Softwareentwicklung anlehnen. Wie in die klassische Softwareentwicklung spielen Anforderungsanalyse, Entwurf, Implementierung, Test, Betrieb und Wartung der Web-Anwendung eine Große Rolle. Allerdings unterscheiden sich die Web-Anwendungen von den traditionellen Software-Anwendungen, indem diese auf ihren dokumentzentrierten Charakter und die Hypertextstruktur des dargestellten Inhalts aufweisen. Diesen Besonderheiten in der Entwicklung prägen sich in der relativ jungen Disziplin Web Engineering aus.

Web Engineering: Definition nach G. Kappel [11]:

> *„Web Engineering ist die Anwendung systematischer und quantifizierbarer Ansätze (Konzepte, Methoden, Techniken und Werkzeuge) um Anforderungsbeschreibung, Entwurf, Implementierung, Test, Betrieb und Wartung qualitativ hochwertiger Web-Anwendungen kosteneffektiv durchführen zu können*."

Analog zum Software Engineering ist Web Engineering kein einmaliges Ereignis, sondern erstreckt sich über den gesamten Lebenszyklus einer Web-Anwendung [13].

1.2 Analyse-Pattern für die Entwicklung von Web-Anwendungen

Analyse-Pattern beschreiben praktisch bewährte und wieder verwendbare Vorlagen für die Entwicklung einer Web-Anwendung und bieten dem Entwickler abstrakte Lösungen für die Geschäftsprozesse. Analyse-Pattern repräsentieren bereits in der Praxis einen bewährten Modellierungsansatz von Teilen einer Web-Anwendung.

> *„Analyse Patterns bringen in den gesamten Entwicklungsprozess Expertenwissen in Form von konkreten Beispielen und abstrakten Modellen ein, und beschleunigen damit den Erstellungsprozess von Software durch die Möglichkeit der Wiederverwendung von Analysemodellen und den damit verbundenen Designvorschlägen. Somit wird der Software-Lebenszyklus massiv beschleunigt, wodurch weniger Kosten anfallen."*

schreibt M. Hahsler in seiner Dissertation [22].

In einem Workshop für die Softwarearchitektur betrachtet Fernandes [21] Analyse-Pattern mehr von der technischen Seite:

> *„An analysis pattern is a set of classes and associations that have some meaning in the context of an application; that is, it is a conceptual model of a part of the application. However, the same structure may be valid for other applications, and this is the aspect that makes them very valuable for reuse and composition."*

Die beiden oberen Aussagen beziehen sich für Analyse-Pattern in der klassischen Softwareentwicklung.

Heutzutage sind Webpatterns[5] in der Entwicklung von Web-Anwendungen sehr verbreitet:

> *WebPatterns* are *design patterns for web development.*

Die meisten davon schlagen nur die Lösungen für die Benutzeroberflächengestaltung von der Web-Anwendung und die grobe Beschreibung der Funktionalität von Geschäftsprozessen vor [2][4][5][6][7].

Die Analyse-Patterns, die in Rahmen dieser Diplomarbeit entstanden sind, beinhalten nicht nur Lösungen für die Präsentationsschicht der Web-Anwendung, sondern auch abstrakte Prozessabläufe für die entsprechenden Teilprozesse in Form eines Aktivitätsdiagramms.

In dieser Diplomarbeit werden die Analyse-Patterns im Web Engineering betrachtet, deren Verwendung mehrere Vorteile bei der Entwicklung von Web-Anwendung bringt:

- Einheitliches und kontrolliertes Vokabular für die Bezeichnung der Analyse-Pattern bringt effiziente Kommunikation zwischen der Entwicklern;
- Analyse-Patterns beschreiben abstrakte und flexible Lösungen, die aufgrund veränderter Anforderungen mit minimalen Anpassungen weiter verwendet werden können;
- Analyse-Patterns bringen Erfahrungen von Experten in die Modellierung ein, die für die Entwicklung weiteren Pattern und Teilmodellen effizient wieder verwendet werden können.

Ferner beschleunigt den Entwicklungsprozess einer Web-Anwendung die Verwendung des flexiblen Analyse-Patterns mit den Designvorschlägen, wodurch sich die Kosten und die Entwicklungszeit reduzieren.

1.3 Motivation und Zielsetzung

In letzter Zeit entwickelte sich das Web von einem reinen *Informationsmedium* zu einem *Anwendungsmedium*. Nicht mehr rein statische Webseiten, die durch Hyperlinks verbunden sind, sondern komplexe Web-Anwendungen bestimmen zunehmend das Wesen des Webs. Trotz dieser fundamentalen Änderungen werden die Web-Anwendungen oftmals in einer spontanen Art und Weise entwickelt, basierend auf den Erfahrungen und den Entwicklungspraktiken einzelner Entwickler ohne unzureichende Dokumentation von Entwurfsentscheidungen. In den meisten Fällen erfolgt die Wiederverwendung im Sinne von „Copy&Paste-Paradigmas“. Solche Web-Anwendungen sind meinst fehlerhaft und technologieabhängig, gekennzeichnet durch unzureichende Zuverlässigkeit, Leistungsfähigkeit und mangelnde Benutzerfreundlichkeit. Deswegen erfordern die ständig wachsenden Anforderungen an den modernen Web-Anwendungen den Einsatz von verschiedenen ingenieurmäßigen Methoden im Entwicklungsprozess, wie die Verwendung der Entwurfsmuster bei der Modellierung von verschiedenen Sichten der Web-Anwendung.

Ein wesentlicher Teil am Beginn des Lebenszyklus einer Web-Anwendung ist die Modellierung von Web-Anwendungen. Die Einsetzung der Entwurfsmuster löst nicht nur bekannte und wiederkehrende Entwurfsprobleme, sondern auch erleichtert den Modellierungsprozess für die Entwickler einer Web-Anwendung. Der Modellierung kommt nicht zuletzt aufgrund des Trends zur modellgetriebenen Software-Entwicklung hoher Bedeutung zu. Abstrakte Modelle werden dabei als Ausgangspunkt zur automatischen Generierung von Programmcode verwendet. Einen wesentlichen Beitrag zur Etablierung

modellgetriebener Software-Entwicklung leistet die *Object Management Group* (OMG) im Rahmen ihrer *Model Driven Architecture* (MDA) [25] Initiative.

Der Gegenstand dieser Arbeit ist einen Katalog von Analyse-Pattern zur Verfügung zu stellen, die Beschreibung deren Einsatzmöglichkeiten bei der Modellierung, und die automatische Generierung von Web-Anwendungen mit der Hilfe von Analyse-Pattern für den Transformationswerkzeug UWE4JSF (ausführlich im Kapitel 4 beschrieben). Dabei wird eine Kette von Modell-zu-Modell Transformationen durchgeführt, die das schon ausmodelliertes Modell von Analyse-Pattern in das Modell der Web-Anwendung integriert.

1.4 Aufbau der Arbeit

Die Diplomarbeit ist in sechs Hauptkapitel gegliedert.

Kapitel 2 gibt in Folge eine Einführung zum Thema des UWE (UML-basierte Web Engineering) und insbesondere einen Überblick über verschiedene Modelle für die Entwicklung von Web-Anwendungen und deren spezielle Charakteristika. Des Weiteren sollen Grundlagen zu deren Aufbau und Modellierung vermittelt werden. Außerdem werden in diesem Kapitel die Technologien für die Modell-zu-Modell Transformationen vorgestellt.

Kapitel 3 befasst sich mit der Darstellung und Beschreibung verschiedenen von Analyse-Pattern, die auf ein einheitliches Schema basieren und für die modernen Web-Anwendungen relevant sind. Ferner werden über die Vorteile des Ansatzes von Entwurfsmustern für die Web-Sytemen diskutiert.

Im Kapitel 4 wird der Ansatz des Entwurfsmusters für die Generierung einer Web-Anwendung in der Praxis unter Verwendung der Kombination aus MDUWE, ATL-Transformationen und UWE4JSF erklärt. Ferner werden auch die neuen Features wie die UWE-Metamodellerweiterung und Patternprofil beschrieben und den Einsatz des neuen Patternprofils begründet. Darüber hinaus wird in diesem Kapitel das Modellierungsbeispiel für die Benutzung des Analyse-Patterns gezeigt.

Die Integration des Entwurfsmusters mittels einer Modell-zu-Modell Transformation, wird im Kapitel 5 vorgestellt. Weiterhin sind der logische Aufbau und die sukzessive Entwicklung der ATL-Transformationen in diesem Kapitel ausführlich beschrieben und anhand einiger Beispiele erklärt.

Abschließend wird in Kapitel 6 eine Zusammenfassung über die gewonnenen Erkenntnisse präsentiert und Anregungen für zukünftige Arbeiten auf dem Gebiet gegeben.

2 Grundlagen und eingesetzte Technologien

In diesem Kapitel sind für ein Verständnis der nächstfolgenden Ausführungen die Grundlagen und eingesetzte Technologie kurz beschrieben.

2.1 Metamodellierung und die Meta Object Facility (MOF)

Metamodellierung ist die wohl wichtigste Voraussetzung für modellgetriebene Entwicklung. Metamodellierung ist Basis für Domänenspezifische oder architekturzentrierte Modellierung, Toolanpassung, Modellvalidierung, Modelltransformationen und Codegenerierung.

Ein Metamodell ist ein Modell eines Modells. Anders ausgedrückt: das Modell ist in der Sprache verfasst, die das Metamodell beschreibt. Ein Metamodell ist selbst ein Modell, das auf einem anderen Metamodell basiert.

Zur Beschreibung von Abstraktionsebenen im Zusammenhang von Metamodellen eignet sich das Vier-Schichten-Modell der Object Management Group [OMG]. Die Einteilung in die vier Ebenen M0 bis M3 wird dabei über den Abstraktionsgrad vorgenommen (siehe Abb. 1). Im Allgemeinen abstrahiert eine höher gelegene Ebene die darunter liegende Schicht, umgekehrt heißt das, dass die untere Ebene die höher gelegene Schicht spezialisiert.

- M3 stellt ein Meta-Metamodell dar, also eine Sprache zur Definition von Metamodellen. Meta-Metamodelle sind reflexiv (bzw. selbsterklärend) und werden in der von ihnen selbst beschriebenen Sprache definiert.
- M2 enthält das Metamodell, also die Modellierungssprache, die zur Erstellung von M1 verwendet wird. Beispielsweise liegt das Metamodell der Unified Modeling Language (UML) demnach auf dieser Ebene.
- M1 enthält das Modell der Anwendung, das in objektorientierten Systemen aus Klassen, Attributen, etc. besteht.
- M0 enthält Daten, die zur Laufzeit innerhalb einer Anwendung bestehen. In objektorientierten Systemen sind das Instanzen von Klassen aus dem Modell der Anwendung.

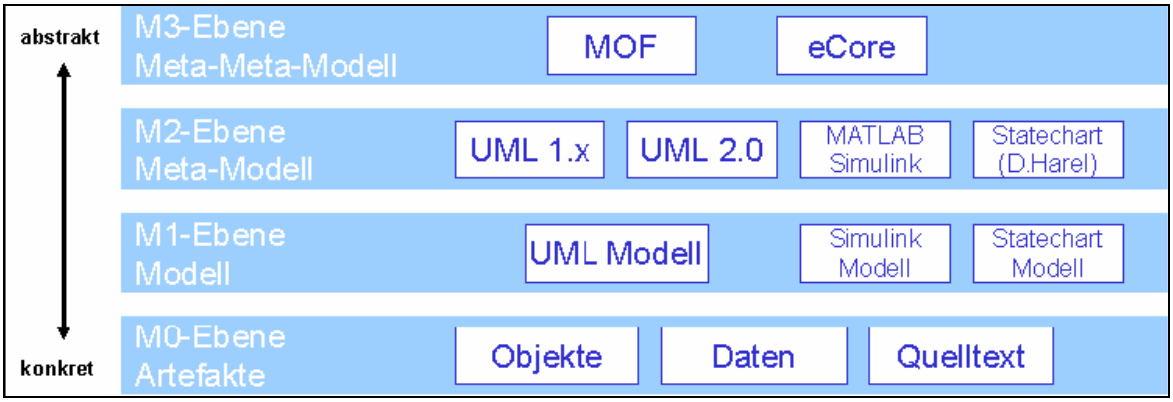

Abb. 1. Das Vier-Schichten-Modell der OMG[24]

Für die Ebene M3 existiert ein Standard der Object Management Group (OMG): die Meta Object Facility (MOF, siehe [25]). Wie oben angedeutet ist die MOF ein Meta-Metamodell, das selbst wiederum durch Verwendung der MOF definiert ist. Auf gleiche Ebene der Metameta-Modelle wird zur Modellierung des Metamodells ECore im Rahmen des Eclipse Modelling Framework (EMF) eingesetzt. Ecore entspricht, bis auf leichte Abweichungen in der Benennung der Elemente, der so genannten Essential MOF (EMOF), die wiederum eine Untermenge von der MOF ist.

MOF bildet eine Grundlage für die Model Driven Architecture (MDA) der OMG, auf die in Abschnitt 2.4 noch näher eingegangen wird. Dies ist vor allem dadurch begründet, dass die Verwendung eines einheitlichen Meta-Metamodells notwendig für die Definition von Modelltransformationen ist.

Zusätzlich existiert als unterstützender Standard der OMG das XML Metadata Interchange (XMI, siehe [26]) Format. Darin werden im Wesentlichen Regeln für die Serialisierung von MOF-konformen Modellen als XML-Dokumente definiert. Alle modernen Modellierungs- und Transformationswerkzeuge sind in der Lage, Dateien im XMI-Format zu verarbeiten. Auch in UWE4JSF dient XMI als Format zur Speicherung von Modellen in allen Stufen des Entwicklungsprozesses.

2.2 UML

Unified Modeling Language (**UML**, engl. *Vereinheitlichte Modellierungssprache,*[28]) ist eine von der Object Management Group (OMG) entwickelte und standardisierte Sprache für die Modellierung von Software und anderen Systemen. Heutzutage gehört die UML zu den dominierenden Sprachen für die Modellierung von betrieblichen Anwendungs- bzw. Softwaresystemen. Die UML ist eine Sprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme und Geschäftsmodelle.

Die UML definiert dabei den Bezeichner für die meisten Begriffe im Sinne einer Sprache, die für die Modellierung wichtig sind, und legt mögliche Beziehungen zwischen diesen Begriffen fest. Die UML definiert graphische Notationen für diese Begriffe und für Modelle von statischen Strukturen und von dynamischen Abläufen, die mit diesen Begriffen formuliert werden können [28].

Die graphische Notation ist jedoch nur ein Aspekt, der durch die UML geregelt wird. Die UML legt in erster Linie fest, mit welchen Begriffen und welchen Beziehungen zwischen diesen Begriffen so genannte *Modelle* spezifiziert werden – die Diagramme der UML zeigen nur eine graphische Sicht auf Ausschnitte dieser Modelle. Die UML schlägt weiter ein Format vor, in dem Modelle und Diagramme zwischen Werkzeugen ausgetauscht werden können.

2.3 UWE – UML basierte Web Engineering

UML-based Web Engineering (UWE, siehe [29][30]) ist auf die Unified Modeling Language (UML) aufgebaut und besteht aus einer Erweiterung der UML 2.0, die ein objektorientierter Ansatz zur modellgetriebenen Entwicklung von Web-Anwendungen bereitstellt. Dabei werden die separaten Teilmodelle, wie Navigations-, Prozess-, Inhalts- und Präsentationsmodelle entwickelt, die separate Sichten bei der Modellierung einer Web-Anwendung darstellen.

Die Daten einer Web-Anwendung werden anhand UWE Ansatzes in vier folgende Phasen modelliert:

- Anforderungsanalyse
- konzeptueller Entwurf
- Navigationsentwurf
- Präsentationsdesign

Entsprechend der obigen Reihenfolge entstehen aus einzelnen Entwurfsschritten folgende Entwurfsmodelle:

- Anforderungsmodell (Requirements)
- konzeptuelles Modell (Content Model)
- Navigationsmodell (Navigation Model)
- Prozessmodell (Process Model)
- Präsentationsmodell (Presentation Model)

Abb. 2 zeigt die Struktur des UWE-Metamodells, die der hierarchische Aufbau einzelnen oben aufgelisteten Teilmodellen darstellt. Solch eine Aufteilung bringt den größten Vorteil mit, indem im Entwicklungsprozess einer Web-Anwendung jeder Teilaspekt unabhängig von den anderen modelliert werden kann.

In der Anforderungsanalysephase werden die Anforderungen an die Web-Anwendung gesammelt, die mittels UML-Anwendungsfalldiagrammen repräsentiert werden. Basiert auf Anforderungsmodell wird das konzeptuelle Modell (Inhaltsmodell) erstellt und in der Form von UML-Klassendiagramm realisiert. Im Inhaltsmodell werden alle Daten dargestellt, die eine Web-Anwendung zu verwalten hat. Dies umfasst allgemeine Entitätsklassen, Klassen für Systemmetadaten und Geschäftsklassen, Klassen für Benutzer und Rollen.

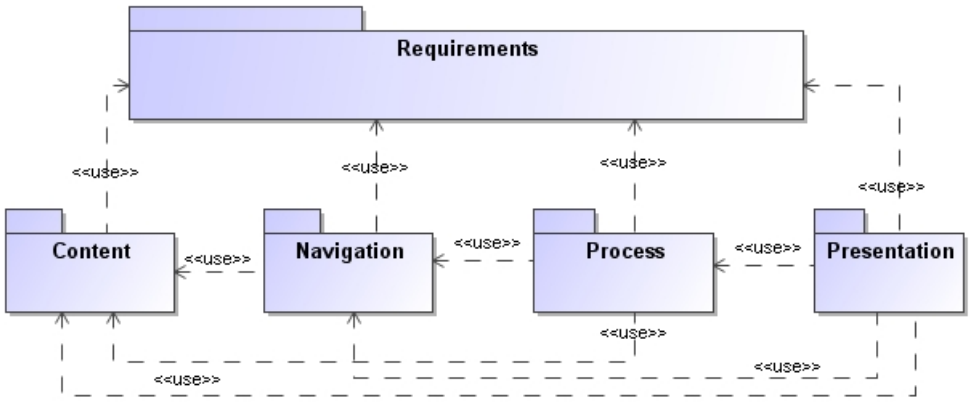

Abb. 2. Struktur des UWE-Metamodells (Überblick)

Aufbauend auf dem Inhaltsmodell wird das Navigationsmodell entwickelt, die ebenfalls auf den UML-Klassendiagrammen basiert ist und stellt abstrakte Navigationsmöglichkeiten in der Web-Anwendung dar. Das Navigationsmodell wird durch einen gerichteten Graph mit so genannten Navigationsknoten und Links als Kanten repräsentiert. Ein Navigationsknoten steht für eine navigierbare Einheit an Information oder Funktionalität in der Webapplikation.

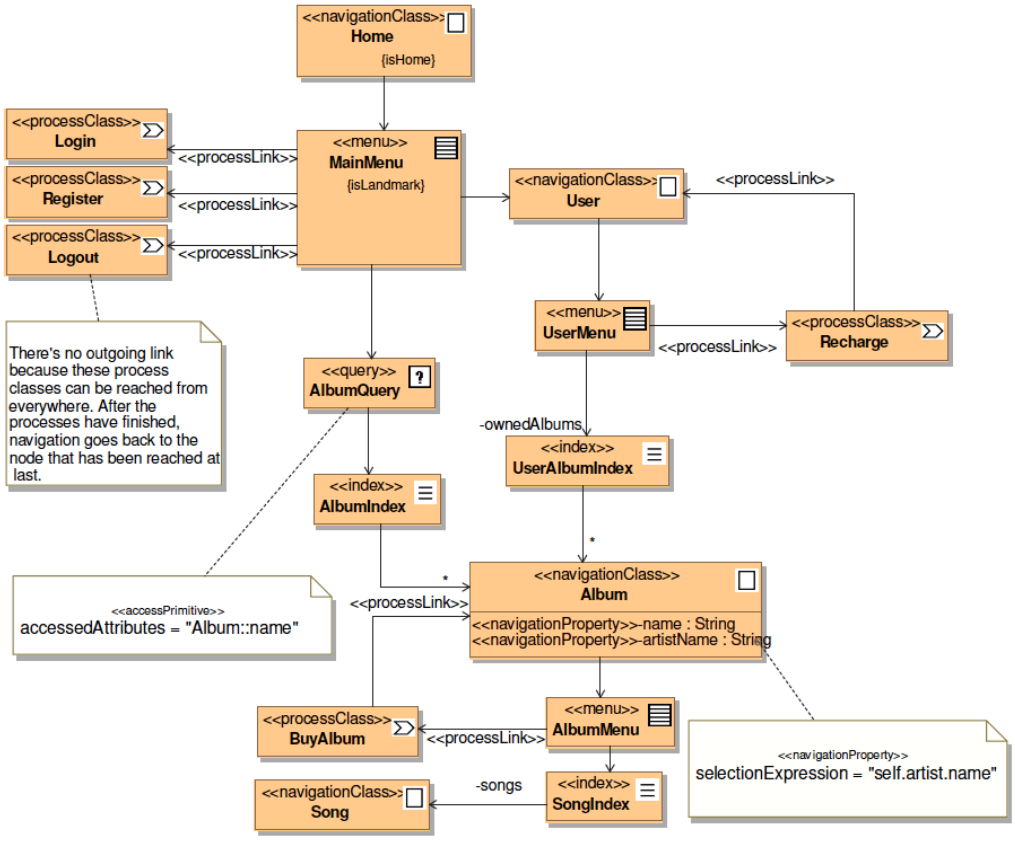

Abb. 3. Navigationsmodell für Musik Portal [31]

Die Abb. 3 schildert das Navigationsmodell für das Musikportal, der im Rahmen der Diplomarbeit „Modellbasierte Generierung von Web-Anwendungen mit UWE (UML-based Web Engineering" [31] entwickelt wurde.

Ausgehend aus den Inhalts- und Navigationsmodellen werden im Prozessmodell alle Systemfunktionalitäten in Form von Klassen- und Aktivitätsdiagrammen dargestellt. Das

Prozessmodell auf der Abb. 4 zeigt die Prozessklassen, die für Musikportal modelliert wurden. Dieses Prozessmodell beinhaltet auch die Aktivitätsdiagramme für die Prozessklassen „*Login*“, „*Register*“, „*BuyAlbum*“ und „*Recharge*“, die den Ablauf jeweiliger Prozesse beschreiben (siehe [35]).

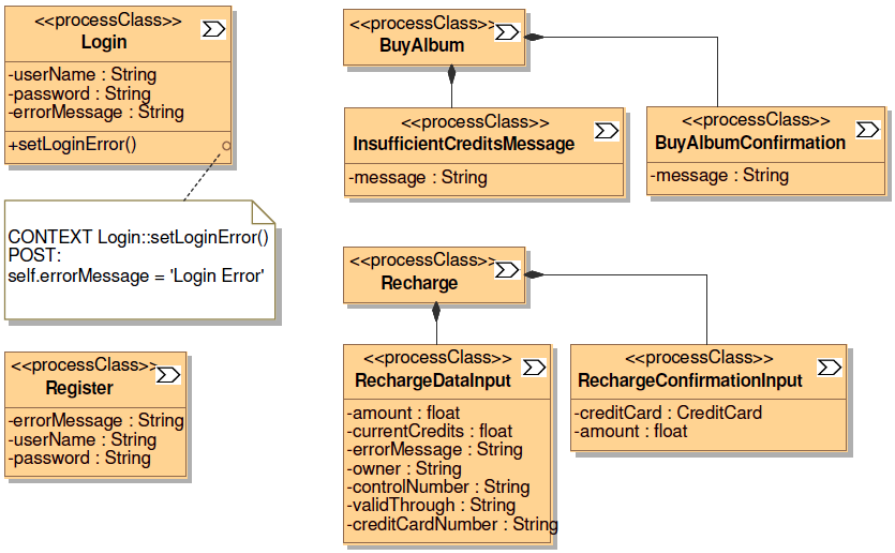

Abb. 4. Prozessmodell für Musikportal [31]

Die Stereotypen *<<processClass>>* und *<<processLink>>* wurden für das Prozessmodell (siehe Abb. 5) konzipiert, die eine Erweiterung von *<<node>>* und *<<link>>* Navigationsstereotypen darstellen. Der Stereotyp *<<userAction>>* kann auf eine Aktivität (Action) im Aktivitätsdiagramm angewendet werden.

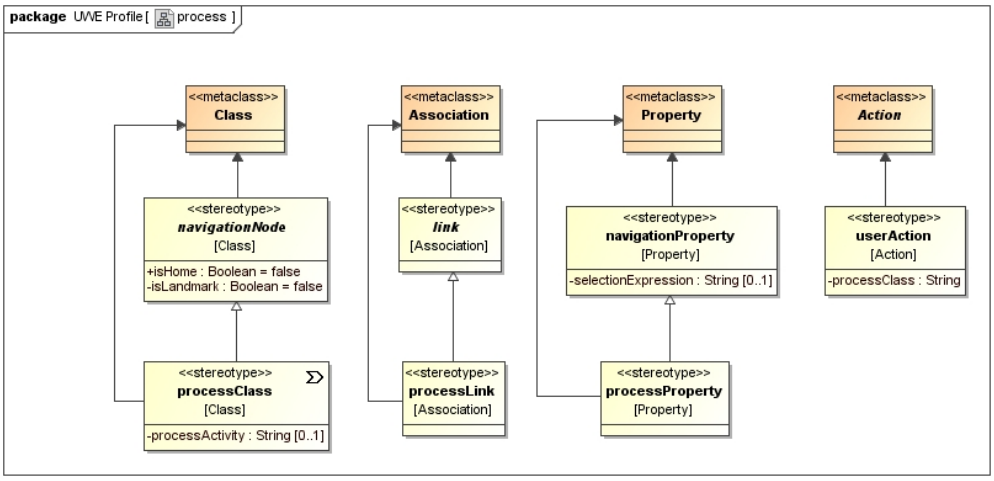

Abb. 5. UWE Metamodell: Prozesspaket

Als eine Skizze zur Benutzerschnittstelle von der Web-Anwendung wird das Präsentationsmodell erstellt. In diesem Modell können sowohl Instanzen von Navigationsklassen als auch Zugriffselemente dargestellt werden. Diese werden um Präsentationsobjekte wie Text, Bilder, Buttons, Anker etc. ergänzt und mit diesen strukturell

zu einem Benutzinterface der Web-Anwendung angeordnet. Die folgende Abb. 6 stellt die Seitenstruktur für den Benutzer Administrator im Musikportal dar.

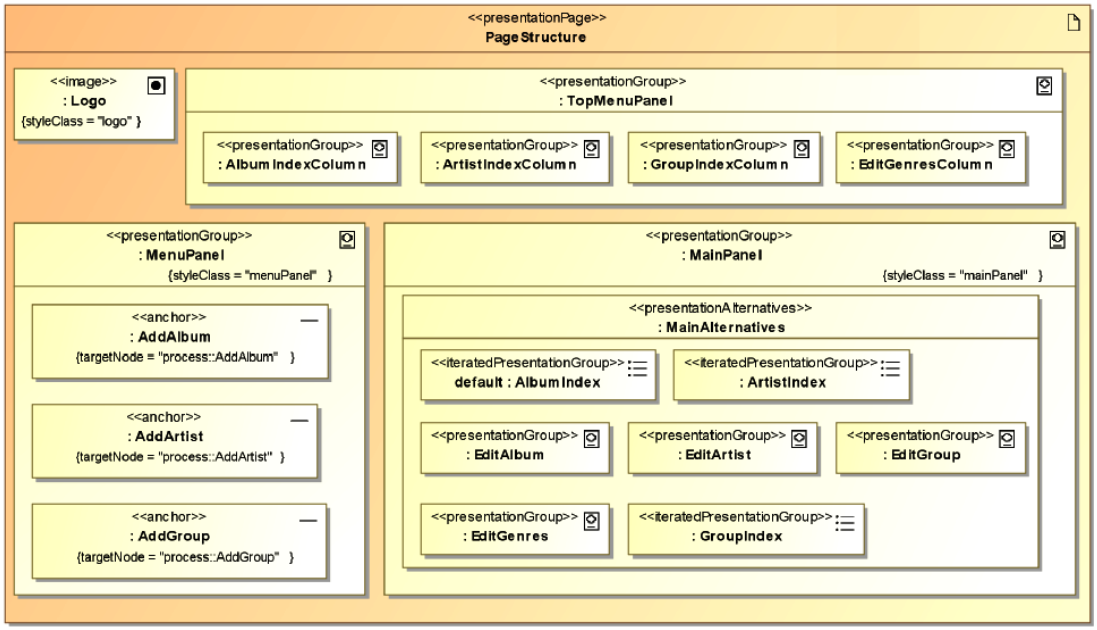

Abb. 6. Musikportal-Admin - Präsentation – Seitenstruktur [31]

Die vorgestellte Vielfältigkeit von Modellelementen macht UWE zu einer mächtigen und flexiblen Modellierungssprache, die sowohl eine robuste Stütze während der gesamten Designphase für den Entwickler bietet als auch eine reale Basis für die automatische Generierung des Quellkodes aus den Modellen darstellt.

Zurzeit wird UWE als ein Plug-In, das eine leichtgewichtige Erweiterung von UML in Form eines Profils darstellt und eine Menge von elementaren Erweiterungen des UML 2.x Metamodells - Stereotypen umfasst, für das kommerzielle Werkzeug „MagicDraw“ [32] angeboten.

Die Möglichkeiten zur modellgetriebenen Entwicklung von Web-Anwendungen mit UWE wurden in [17] bzw. [18] untersucht. Ein modellgetriebener Ansatz für die Entwicklung von Web-Anwendungen unterstützt durch eine in ATL (Abschnitt 2.5) realisierte Transformationskette den kompletten Entwicklungsprozess vom Analysemodell bis hin zum generierten Quelltext. Anders als bei UWE4JSF (Abschnitt 2.7) wird für die Realisierung der Funktionalität von Navigation und Prozessen der Web-Anwendung kein Java-Code generiert. Stattdessen werden XML-Dateien erzeugt, die die entsprechenden Abläufe definieren und von einer generischen Controller-Komponente interpretiert werden.

2.4 Modelgetriebene Softwareentwicklung und Model Driven Architecture

Die Model Driven Architecture (MDA) ist ein Ansatz für die modellgetriebene Softwareentwicklung, der von der Object Management Group (OMG) initiiert wurde. Die MDA ist ein Standardisierungsvorschlag die Software auf die Modellebene statt auf der

Programmcodeebene zu repräsentieren. Das elementare Ziel der MDA ist plattformspezifische Modelle möglichst automatisiert aus plattformunabhängigen Modellen abzuleiten, wodurch die Anpassung an neue Technologien und eine höhere Produktivität und einfache Softwareentwicklung ermöglichen werden.

Damit werden die Modelle als zentrale Elemente des Softwareentwicklungsprozesses betrachtet. Jedes Modell abstrahiert sich von einer konkreten Plattform, die abstrakt als eine Menge von Subsystemen und Technologien definiert ist. Dabei stellt diese Plattform die Dienste durch Schnittstellen und spezifizierte Anwendungsmuster bereit, ohne die Details ihrer Implementierung zu kennen [20].

Die Abstraktion von einer Plattform wird durch ein so genanntes plattformunabhängiges Modell erreicht (Platform Independent Model, PIM), das die fachliche Spezifikationen der Anwendung ohne einen Bezug auf konkrete Technologie beschreibt. Dieses Modell wird durch verschiedene Modelltransformationen in ein plattformspezifisches Modell (Platform Specific Model, PSM) überführt, das die plattformunabhängige Spezifikation des Systems auf die gewünschte Plattformen und Technologien abbildet. Der Abb. 7 stellt das Grundprinzip der MDA dar, das durch seine Schichten aufgebaut ist.

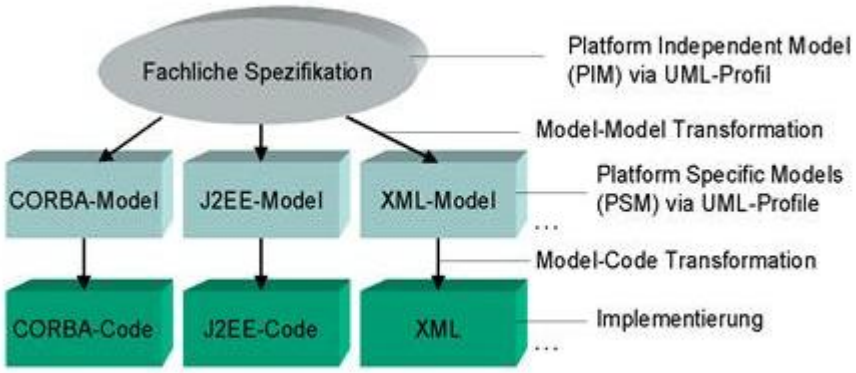

Abb. 7. MDA Grundprinzip([23])

Außerdem sowohl das PIM als auch das PSM basieren auf Metamodellen, die ihrerseits durch MOF definiert werden.

2.5 Die ATLAS Tranformation Language (ATL)

Die ATLAS Transformation Language (ATL, [14]) wurde von der ATLAS Group an der Universität Nantes (Laboratoire d“Informatique de Nantes Atlantique, LINA) ins Leben gerufen und ist eine Programmiersprache zum Transformieren von Modellen. ATL ermöglicht die Durchführung von automatischen Modell-zu-Modell Transformationen und ist eine hybride Sprache, die Konzepte imperativer und deklarativer Programmierung vereint. Mittlerweile steht ATL als stabiler Plug-In mit einem Editor und Debugger für die Softwareentwicklungsplattform Eclipse zu Verfügung und findet weit reichende Verwendung.

Der Bestandteil der Ein- und Ausgabe einer ATL-Transformation sind ein oder mehrere auf MOF-konformen Metamodellen basierende Modelle. Die Transformationsregeln bestehen aus einen deklarativen Modus mit Transformationsregeln, die als Matched Rules bezeichnet werden, und einen imperativen Modus mit Called Rules genannten Prozeduren [15].

Matched Rules spezifizieren die Elemente aus Quellmodellen, die generiert werden, und der Weg, wie die erzeugten Zielelemente initialisiert werden müssen. Dafür alle notwendige Anfrageausdrücke und Auswahlbedingungen innerhalb ATL werden auf die Object Constrant Language (OCL) definiert. Zur Strukturierung der Transformationen existieren in ATL Hilfsfunktionen, die dort Helper genannt werden und ebenfalls OCL-Ausdrücke enthalten. Diese können in spezielle ATL-Module, sogenannte Libraries, ausgelagert werden. Called Rules können als ein besonderer Typ von Helpers gesehen werden. Jedoch, im Vergleich zur Helpers, Called Rules können Zielmodellelemente erzeugen, wie Matched Rules.

Zusätzlich können komplexere Transformationen durch die so genannte Superimposition in mehrere Module aufgeteilt werden, wobei ein Basismodul existiert, dessen Regeln durch Regeln aus übergelagerten Modulen überschrieben werden können.

ATL-Transformationen stellen selbst Modelle dar, die auf einem MOF-konformen ATL-Metamodell basieren. Dadurch werden so genannte High-Order-Transformations möglich, deren Ausgabe aus einer neuen Modelltransformation besteht.

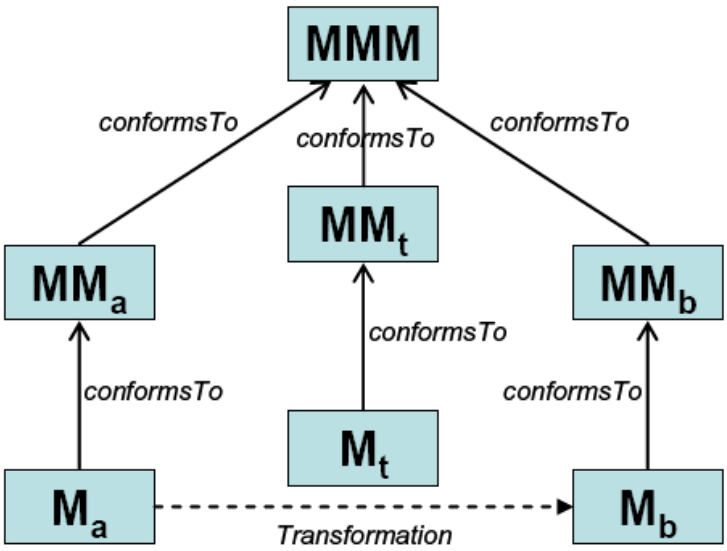

Abb. 8. Eine Übersicht der Modelltransformation[15]

Das obige Model (siehe Abb. 8) fasst den vollen Modelltransformationsprozess zusammen. Ein Modell Ma, der zu Metamodell MMa konform ist, wird in ein zu Metamodell MMb konformte Modell Mb transformiert. Die Transformation ist durch Transformationsmodell Mt definiert, die selber zur Tranformationsmetamodell MMt konform ist. Das letzte Metamodell, zusammen mit dem MMa und MMb Metamodellen, muss sich einem Metametamodell MMM (wie MOF oder Ecore) konform sein.

2.6 MDUWE

In Rahmen der Diplomarbeit von Christian Kroiß „Modellbasierte Generierung von Web-Anwendungen mit UWE (UML-based Web Engineering)“ ([31]) wurde eine erweiterte Version der Modellierungssprache UWE mit dem Namen MDUWE (Model Driven UML-based Web Engineering) entwickelt. Die beträchtliche Erweiterung für die Modellierung ist die Einführung des konkreten Präsentationsmodells, wodurch sich das plattformunabhängige Modell der Benutzeroberfläche sehr ausführlich und detailliert modellieren lässt. Außerdem wurde eine Konfigurationsschicht oberhalb des plattformunabhängigen Modells der

Benutzeroberfläche kreiert, wie die konkreten UI-Komponenten der verwendeten Zielplattform die abstrakten Präsentations-Elementen realisieren werden sollen. Ferner wird die Modellierung mit MDUWE für die Beschreibung aller relevanten Details der Anwendung durch Object-Graph Notation Language (ONGL, siehe [33]) unterstützt.

2.7 Transformationswerkzeug UWE4JSF

Am Lehrstuhl für Programmierung und Softwaretechnik wurde das Transformationswerkzeug UWE4JSF für die automatische Generierung von Web-Anwendungen entwickelt, das in Form von Plug-Ins in die Entwicklungsumgebung Eclipse (siehe [27]) integriert ist und die automatische Generierung ermöglicht, die auf den JavaServer Faces (JSF,siehe [34]) Standard aufbauen. Eine Kombination aus einem plattformunabhängigen MDUWE-Modell (PIM) und dem konkreten Präsentationsmodell sind als Eingabe erforderlich. Durch eine Kette aus Modelltransformationen, die in der Atlas Transformation Language (ATL, siehe [14]) definiert wurden, wird aus dieser Eingabe zunächst ein plattformspezifisches (PSM) Modell erzeugt. Anschließend erfolgt eine Modell-zu-Code-Transformation, die durch die Java Emitter Templates Technologie realisiert wurde und den Quelltext der Anwendung generiert. Der gesamte Generierungsprozess ist flexibel konfigurierbar. Insbesondere lassen sich an vielen Schnittstellen der generierten Anwendung manuell implementierte Java-Klassen einbinden, die Funktionalität zur Beschaffung von Daten oder für Prozessabläufe realisieren. Dieser Mechanismus wird beispielsweise eingesetzt, wenn die entsprechenden Abläufe zu aufwändig zu modellieren sind oder keine für das Modell interessanten Informationen liefern.

3 Entwurfmuster für das Prozessmodell

Noch ganz am Anfang des Entwurfs von einer Web-Anwendung soll vom Entwickler festgelegt werden, welche Funktionalitäten dem Benutzer im Rahmen dieser Anwendung zur Verfügung gestellt werden sollen. Nachdem die Anforderungen an die Web-Anwendung gesammelt sind, kommt die Frage, wie diese Funktionalitäten in der Web-Anwendung umgesetzt werden. Um die Antwort auf diese Frage zu erleichtern, wurde im Rahmen dieser Arbeit eine Menge von Entwurfsmuster im Prozessmodell zusammengestellt und beschrieben. Um der Beschreibung eine einheitliche Struktur zu verleihen, wurde jedes Muster einem Schema folgend dargestellt. Das Beschreibungsschema beinhaltet folgende Schritte:

Beschreibung: In diesem Schritt wird kurz die Funktionalität beschrieben, die anhand von diesem Pattern in der Web-Anwendung zur Verfügung gestellt wird.

Zweck: Während dieses Schrittes wird beschrieben, wozu dieses Entwurfmuster benutzt werden kann. Dadurch werden die praktische Nützlichkeit und die Lösungskraft des Patterns illustriert.

Auch bekannt als: In diesem Schritt werden mögliche weitere Namen für das entsprechende Entwurfsmuster vorgestellt, die für dieses Muster zur Zeit der Verfassung dieser Arbeit in der Literatur und anderen Quellen zu finden waren.

Umsetzung: An dieser Stelle wird eine mögliche Umsetzung des Entwurfsmusters in der Web-Anwendung beschrieben. Dabei werden sowohl die Umsetzung auf der Benutzeroberfläche, als auch ein mögliches Ablaufszenario der gesamten Funktion vorgestellt.

Modellierung: Dieser Teil des Schemas beinhaltet eine Beschreibung von nötigen Modellierungsschritten, sowie ein Beispiel des Aktivitätsdiagramms für den gesamten Prozess.

Reale Beispiele (optional): Hier werden einige reale Beispiele vorgestellt, die für das beschriebene Muster in zeitgenössischen Web-Anwendungen auf dem heutigen Markt zu finden sind.

Der Entwurfsmustername, der die wesentlichen Aspekte des Patterns ausdrückt und für die Kommunikation zwischen den Entwickler in der Fachsprache für die Analysephase übernommen wird, ist als Titel für den Unterkapitel dargestellt.

Das Beschreibungsschema abstrahiert von implementierungsspezifischen Details und verallgemeinert das Entwurfsmuster, so hat der Entwickler die Möglichkeit während des Entwurfs schnell nach einer Problemlösung zu suchen und selber entscheiden, welche Programmiersprache angewendet wird. Muster sind also nicht die Problemlöser schlechthin, sondern eine Ansammlung von praktischen Lösungsmöglichkeiten, die gut funktionieren. Da bekannte Entwurfsmuster bereits oft angewendet wurden, kann man sicher sein, dass dieser Ansatz praxistauglich ist. Darüber hinaus werden für die Gestaltung von Benutzeroberflächen bei der Beschreibung des Entwurfsmusters zahlreiche Beispiele vorgestellt.

3.1 Registrierung

Beschreibung: Um das vollständige Nutzen einer Web-Anwendung zu genießen, soll jeder Benutzer einmalig den Registrierungsprozess durchlaufen. Diese Funktionalität kann entweder auf der freiwilligen Basis oder als Pflicht für alle Benutzer eingeführt werden. Dabei wird der Benutzer in einem Eingabeformular nach seinen persönlichen Daten, sowie z.B. Name, Alter, Geschlecht und so weiter, abgefragt.

Zweck: Benutzerfreundlichkeit der Anwendung kann dadurch erhöht werden, dass dem Benutzer z.B. bei der Kaufabwicklung oder Reservierung die mehrfache Abrage nach persönlichen Daten erspart bleibt, denn es werden direkt Daten seines Benutzerprofils benutzt, die einmalig bei der Registrierung gespeichert wurden. Abgesehen davon können Inhalt und Gestaltung der Web-Anwendung an die Benutzergruppe angepasst werden, um noch mehr Benutzer zu dem Mitbenutzen der Web-Anwendung zu motivieren. Ein wichtiger Hinweis für den Entwickler ist die Menge der abgefragten Daten während der Registrierung in Grenzen zu halten. Außerdem ist es absolut wichtig, im Fall der optionalen Registrierung, die Vorteile der Registrierung ausgiebig zu erklären, denn für beide Parteien sowohl für den Benutzer als auch für den Web-Anwendungsbetreiber ist die Registrierung vom Vorteil.

Auch bekannt als: Registration, Register

Umsetzung: Auf der Benutzeroberfläche wird ein Formular zur Registrierung angeboten. Ein Beispiel für das mögliche Layout des Registrierungsformulars sieht man auf der Abb. 9.

Falls die Registrierung nicht notwendig ist, kann der Benutzer in einem passenden Moment, wie z.B. am Anfang oder am Ende eines Kaufprozesses, auf die Vorteile der Registrierung aufmerksam gemacht werden.

Nach der Registrierung kann der Benutzer sich entweder sofort Anmelden, oder einem Bestätigungsverfahren unterzogen werden. Im Rahmen dieses Verfahrens erhält der Benutzer eine E-Mail mit den Anmeldungsdaten und einen Aktivierungslink. Nur wenn der Benutzer diesem Link folgt, wird sein Benutzerkonto in der Datenbank tatsächlich aktiviert, sonst verfallen die Daten und die Registrierung wird annulliert. Abgesehen davon kann die erhaltene E-Mail auch Auskunft geben, wie der Benutzer sein Benutzerkonto auflösen kann.

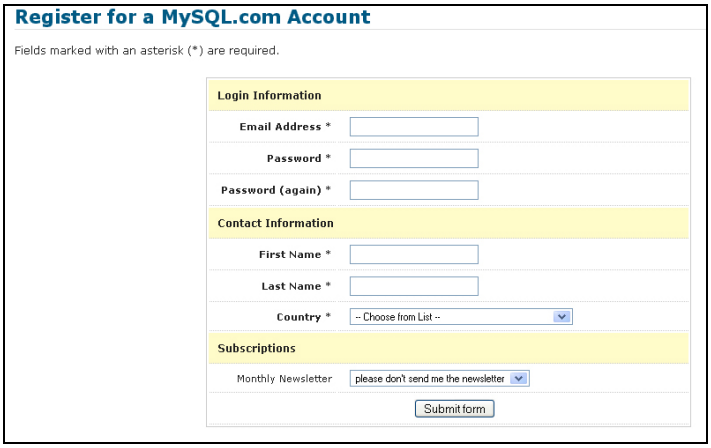

Abb. 9. Registrierungsformular bei www.mysql.com (25.06.08)

Die zweite Methode mit dem aufwendigeren Registrierungsprozess kann für die Web-Anwendungen mit sehr vielen Benutzern von Vorteil werden. Denn diese Methode kann mit ziemlicher Sicherheit garantieren, dass alle registrierten Benutzer tatsächliche Benutzer der Web-Anwendung sind und spart unnötige Einträge in der Datenbank.

Prozessbeschreibung:

1. Der Benutzer startet den Registrierungsprozess.

2. Dem Benutzer wird das Registrierungsformular zum ausfüllen angeboten.

3. Der Benutzer füllt das Formular aus und tätigt den Aktionsknopf („Submit form“ - Button auf der Abb. 9).

4. Der Benutzername darf nicht in der Datenbank schon enthalten sein und wird darauf überprüft.

5. Die beiden Passwörter werden auf die Übereinstimmung untersucht.

6. In diesem Prozess können auch weitere Aktivitäten für die Eingabedatenkontrolle eingebaut werden, wie z.B., ob die E-Mailaddresse das Zeichens @ beinhaltet oder die PLZ (Postleitzahl) nicht länger als 5 Zeichen ist.

7. In diesem Schritt kann abhängig von der Registrierungsart entweder die Anmeldung gestartet werden oder der Benutzer wird aufgefordert anhand von der erhaltenen E-Mail den Anmeldungsprozess zu beenden.

Modellierung: Dem Prozessmodell wird die Klasse „Register“ hinzugefügt sowie das Aktivitätsdiagramm „Registration“ (siehe Beispiel auf der Abb. 10), das den gesamten Registrierungsprozess beschreibt. Im Präsentationsmodell werden eine oder mehrere Klassen hinzugefügt, die das gewünschte Layout der Registrierungsmaske darstellen.

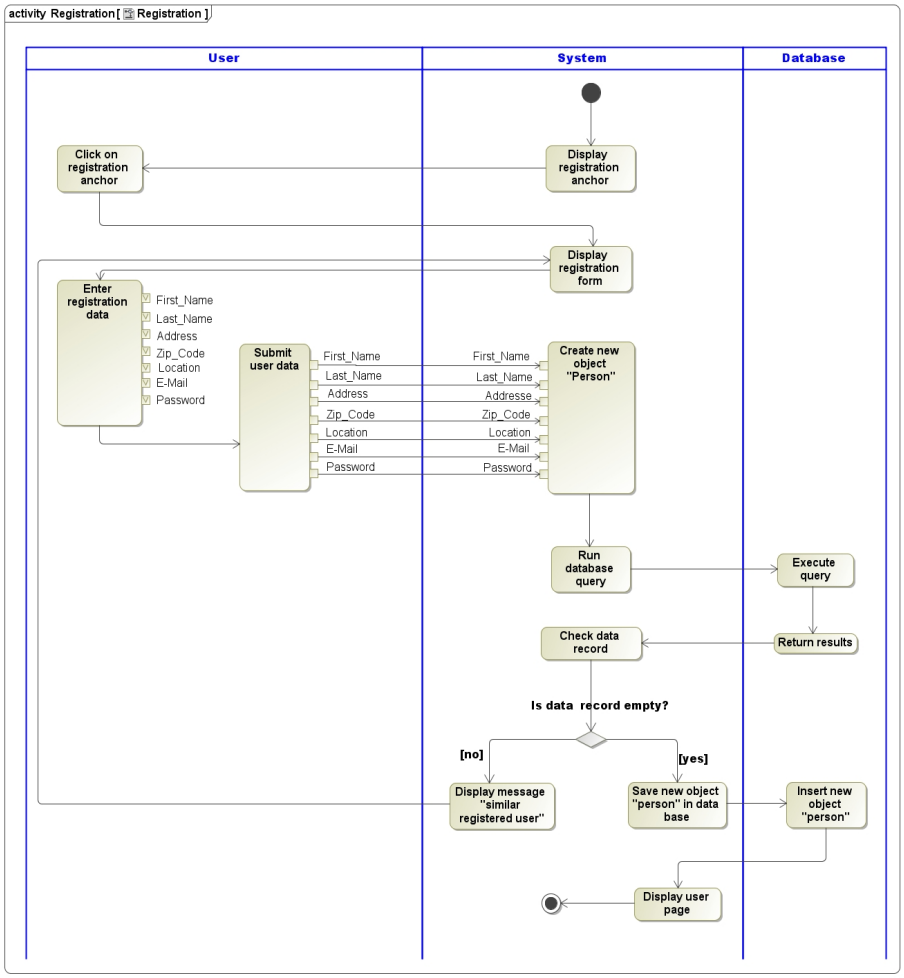

Abb. 10. Registrierung - Aktivitätsdiagramm

Reale Beispiele (optional): Als Alternative zum Beispiel von www.mysql.com auf der Abb. 9, zeigt die Abb. 11, dass die Registrierung auch schnell durchgeführt werden kann. Dabei erfolgt die Registrierung nur in einem Schritt, wo erst nur die gültige E-Mail-Adresse des Benutzers abgefragt wird. Sein Profil kann der Benutzer zum späteren Zeitpunkt vervollständigen. Dieses Feature kann mit der Hilfe von Entwurfsmuster „Benutzerprofil ändern“ (siehe Abschnitt 3.4) umgesetzt werden.

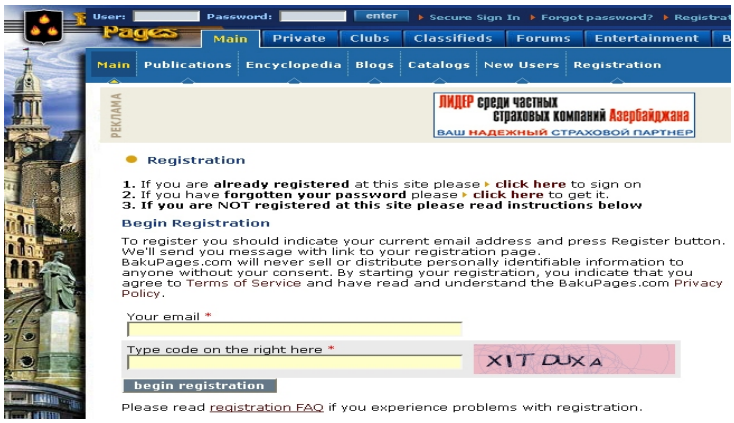

Abb. 11. Registrierung bei www.baku.ru (16.05.08)

3.2 Login

Beschreibung: Bevor die Web-Anwendung vollständig bzw. überhaupt benutzt werden kann, soll der Benutzer bei jedem Besuch der Web-Anwendung sich identifizieren, indem er sein persönlichen Benutzername und Passwort in einer dafür angebotenen Eingabemaske eingibt.

Zweck: Benutzerfreundlichkeit der Web-Anwendung kann erhöht werden, indem es zum Beispiel benutzerspezifische Daten in der Web-Anwendung gespeichert und nach der Anmeldung angezeigt werden. Sicherheit für den Betreiber wird gewährleistet, da immer nachvollzogen werden kann, von wem und wann die Web-Anwendung benutzt wurde. Dabei können Benutzer, die sich etwas zu Schulden kommen ließen, gesperrt werden. Im Rahmen dieser Überlegung werden bestimmte Handlungen nur den angemeldeten Benutzern erlaubt.

Auch bekannt als: Anmeldung.

Umsetzung: Auf der Benutzeroberfläche wird eine Eingabemaske für die Anmeldung angeboten. Ein Beispiel für das mögliche Layout ist auf der Abb. 12 vorgestellt.

Abb. 12. Ein Beispiellayout

Prozessbeschreibung:

1. Der Benutzer gibt seinen Benutzername und sein Passwort in die entsprechenden Eingabefelder ein.
2. Der Benutzer tätigt in der Eingabemaske den Aktionsknopf (auf der Abb. 12 „Log me in“ - Button).
3. Die eingegebenen Werte werden aus den entsprechenden Feldern gelesen.
4. Die Werte werden überprüft, dafür wird eine Datenbankanfrage gestartet. Dabei dürfen die Werte nicht leer sein und ein Datensatz mit dem gleichen Benutzernamen und Passwort soll zurückgeliefert werden.
5. Im Fall der erfolgreichen Anmeldung wird die nächste Seite der Web-Anwendung generiert, sonst wird eine Fehlermeldung angezeigt und der Benutzer wird aufgefordert das gesamte Prozedere zu wiederholen. Dabei ist es wichtig aus Sicherheitsgründen keinen Hinweis darüber zu geben, welches Feld genau falsch

ausgefüllt wurde. Denn im Fall eines Hackerangriffs würden solche Hinweise dem Hacker die Arbeit erleichtern.

Modellierung:

Im Prozessmodell wird das Pattern durch die Klasse „Login“ beschrieben. Außerdem wird im Prozessmodell das Aktivitätsdiagramm „Login“ (siehe Abb. 13) erstellt, das den gesamten Anmeldungsprozess beschreibt. Im Präsentationsmodell stellt die gleichnamige Klasse das Layout des Patterns auf der Seite dar.

Aktivitätsdiagramm auf der Abb. 13 beschreibt den allgemeinen Prozessverlauf. Dennoch die Entscheidung, wie es in der Wirklichkeit implementiert wird, wird dem Entwickler überlassen. Das Pattern „Login“, das im Rahmen dieser Diplomarbeit entwickelt wurde (siehe 5), basiert auf diesem Diagramm.

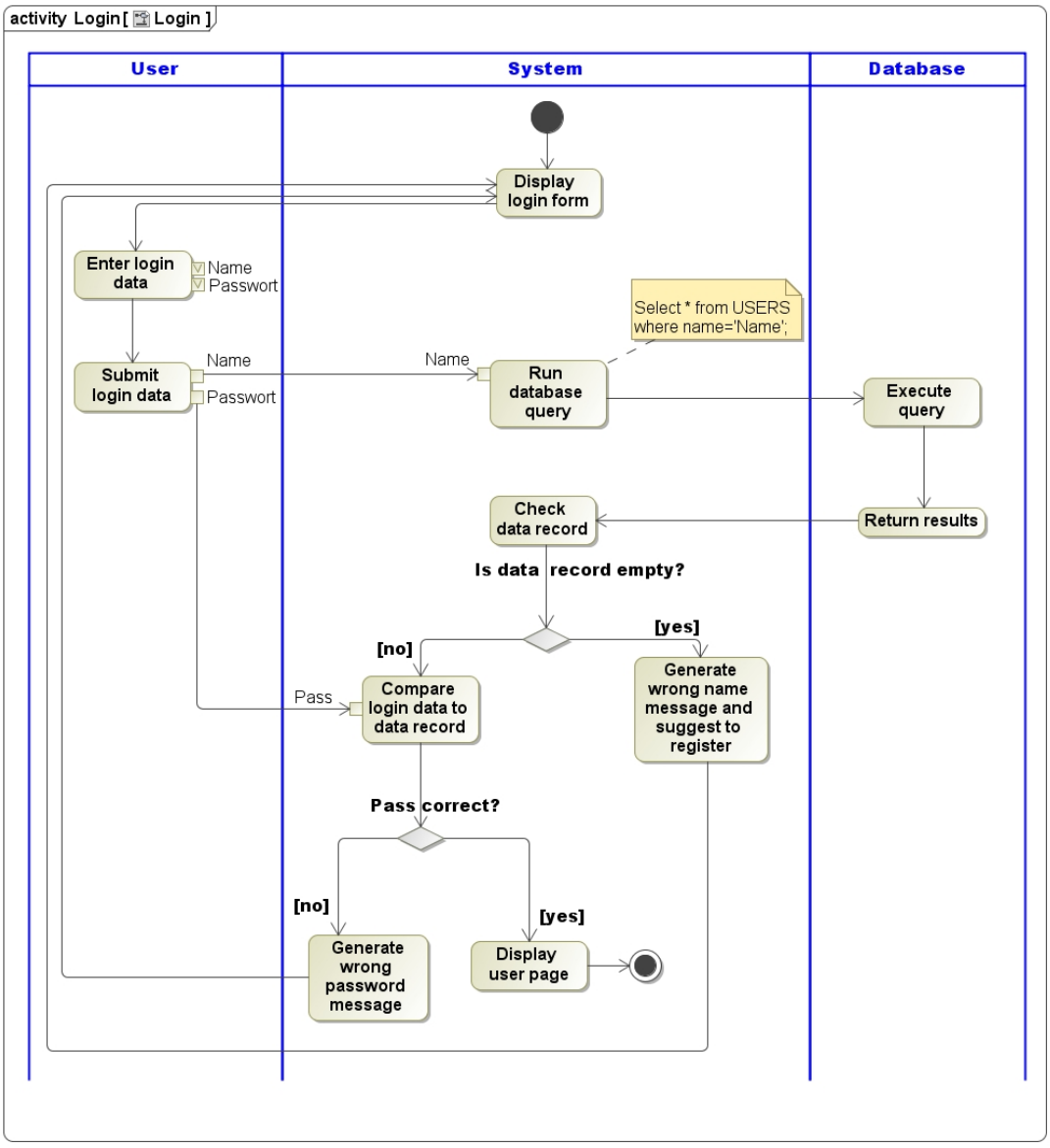

Abb. 13. Aktivitätsdiagramm „Login“

Reale Beispiele (optional): Auf der Webseite www.youtube.com bekommen Benutzer folgende Anmeldemaske (siehe Abb. 14), die sowohl ein schönes Layout als auch eine klare Übersichtlichkeit und Einfachheit aufweist.

Abb. 14. Anmeldemaske bei www.youtube.com (28.01.09)

3.3 Logout

Beschreibung: Beim Verlassen der Website soll der angemeldete Benutzer sich abmelden, indem er ein Abmeldebutton bzw. Abmeldelink tätigt.

Zweck: Der Sinn dieser Funktionalität hängt mit der Funktionalität Login (siehe Abschnitt 3.2) zusammen. Damit die genaueren Besuchszeiten jedes einzelnen Benutzers nachvollzogen werden können, soll der Benutzer sich beim Verlassen der Website abmelden können. Zur Sicherheit kann auch eine automatische Logout-Funktion implementiert werden, die nach einer definierten Zeitspanne startet, in der der Nutzer keine weitere Aktion getätigt hat.

Auch bekannt als: Abmeldung, Ausloggen.

Umsetzung: Auf der Benutzeroberfläche wird ein Link bzw. ein Button als Abmeldelink bzw. Abmeldebutton angeboten (siehe Abb. 15).

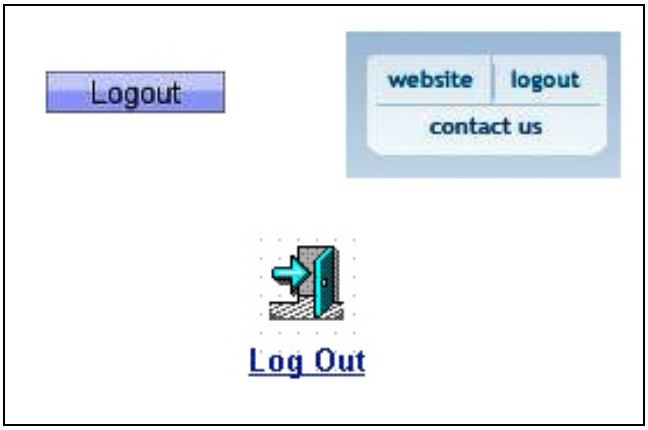

Abb. 15. Verschiedene Gestaltungsmöglichkeiten für Abmeldelink oder –button.

Prozessbeschreibung:

1. Der Benutzer tätigt den Abmeldelink.
2. Benutzungssession wird ausgelöst.
3. Benutzerspezifischen Daten werden in der Datenbank gespeichert (optional).
4. In der Datenbank wird der Abmeldungszeitpunkt eingetragen.
5. Die Abmeldungsseite mit der Abmeldungsbestätigung wird angezeigt.

Modellierung: Im Prozessmodell wird das Pattern durch die Klasse „Logout“ beschrieben. Außerdem wird im Prozessmodell das Aktivitätsdiagramm „Logout“ (siehe Abb. 16) erstellt, das den gesamten Abmeldeprozess beschreibt. Im Präsentationsmodell wird auf der entsprechenden Position das Abmeldebutton/Abmeldelink platziert.

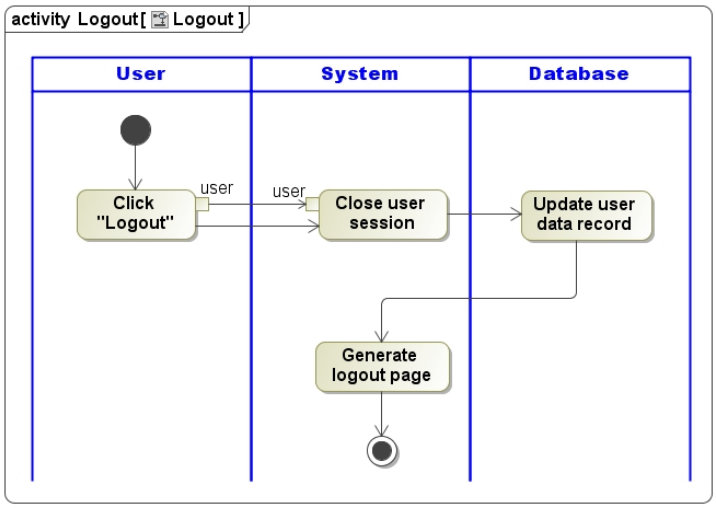

Abb. 16. Logout - Aktivitätsdiagramm

3.4 Benutzerprofil bearbeiten

Beschreibung: Ein registrierter Benutzer kann jeder Zeit seine bei der Registrierung angegebenen Daten bearbeiten bzw. vervollständigen. Dafür kann in der Form eines Buttons bzw. eines Links dem Benutzer die Möglichkeit angeboten werden, zum Registrierungsformular zurück zu kehren und die angegebenen Daten zu korrigieren.

Zweck: Diese Funktionalität bietet den Benutzern die Flexibilität seine Benutzerdaten zu ändern. Dies ist aus folgenden Perspektiven wichtig, dass zu einem solche Daten, wie z.B. Adresse, Telefon usw., sich tatsächlich mit der Zeit ändern und zu anderem manche Daten während der Registrierung noch nicht eingetragen wurden. Dennoch kann ein vollständigeres Benutzerprofil aus dem Betrachtungswinkel des Servicebetreibers sehr wichtig sein. Es ist z.B. von Interesse das durchschnittliche Benutzerprofil seiner Web-Anwendung zu erstellen, um den Service mehr an die Benutzergruppe anzupassen.

Auch bekannt als: User settings, modify user profil, modify account, modify user properties u.a.

Umsetzung: Auf der Benutzeroberfläche wird ein Link bzw. ein Button angeboten, der ermöglicht, zum Registrierungsformular zurück zu gelangen.

Prozessbeschreibung:

1. Der Benutzer tätigt dem Link, zum Benutzerprofil.
2. Registrierungsformular des entsprechenden Benutzers wird angezeigt.
3. Der Benutzer führt alle gewünschten Änderungen durch und tätigt den Submit-Button.
4. Neue Daten werden in der Datenbank gespeichert.
5. Eine Bestätigung der erfolgreichen Änderungen des Benutzerprofils wird dem Benutzer angezeigt.

Modellierung: Im Prozessmodell wird das Entwurfsmuster durch die Klasse „Modify userprofil“ beschrieben. Außerdem wird im Prozessmodell das Aktivitätsdiagramm „Modify userprofil“ (siehe Abb. 17) erstellt, das den gesamten Änderungsprozess des Benutzerprofils beschreibt. Im Präsentationsmodell werden auf den entsprechenden Stellen die „Benutzerprofil ändern“-Buttons/-Links positioniert.

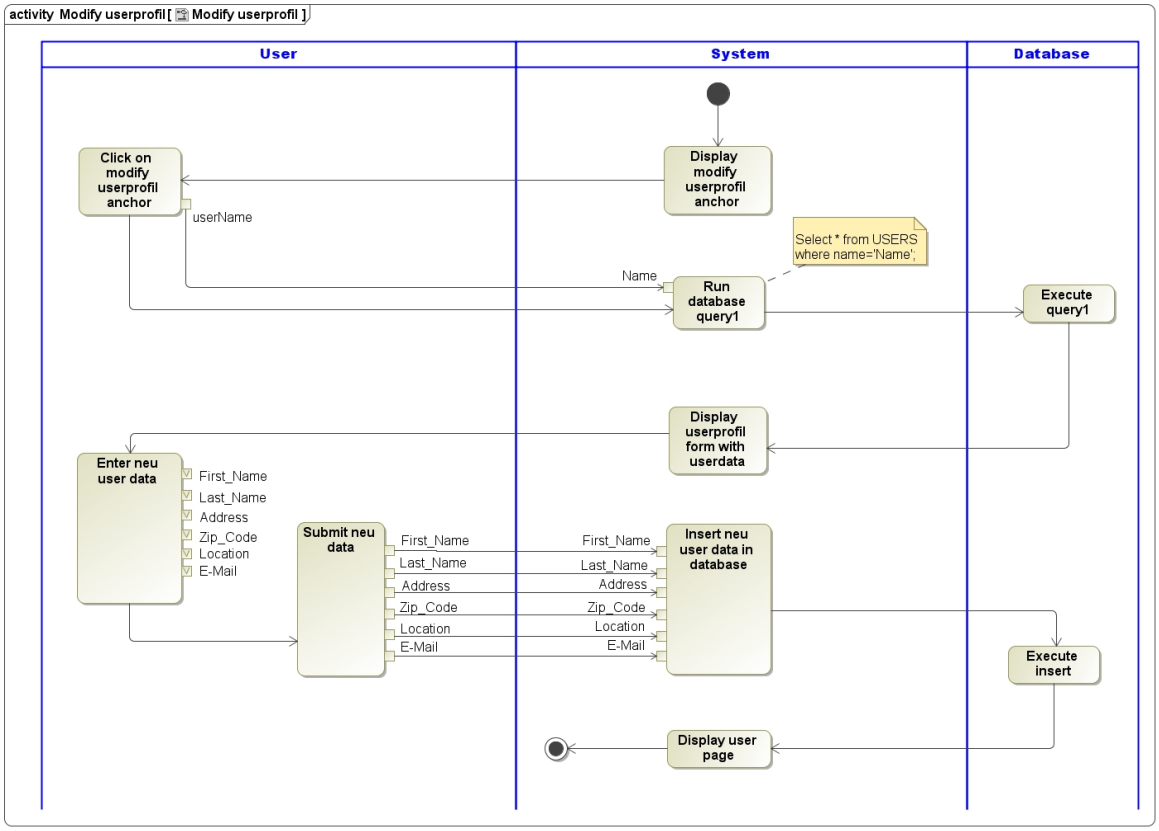

Abb. 17. Aktivitätsdiagramm für „Benutzerprofil ändern“

Reale Beispiele (optional): Auf der Webseite http://www.immobilienscout24.de haben Benutzer eine Möglichkeit, ihre Persönlichen Daten in einem übersichtlichen und einfachen Formular zu ändern und korrigieren (siehe Abb. 18).

Abb. 18. Einfacher Beispiel für Layout des „Benutzerprofil ändern“ –Entwurfsmuster (15.02.2009).

3.5 Reservierung

Beschreibung: Diese Funktionalität ist typisch für Reisebüroseiten oder für Seiten des elektronischen Handels. Dabei können die Benutzer ein Angebot für sich reservieren. Solch eine Reservierung kann sowohl als verbindliche Verpflichtung des Benutzers als auch als unverbindlichen Service zur Verfügung gestellt werden.

Zweck: Die Benutzer bekommen eine gewisse Freiheit beim Erwerb von bestimmten Handelsartikel. Denn die Reservierung versichert einerseits die Möglichkeit die reservierte Ware zu einem späteren Zeitpunkt zu erwerben, andererseits lässt es dem Benutzer die Alternative, in irgendeiner Weise von dem Erwerb zurückzutreten.

Auch bekannt als: Booking (engl.)

Umsetzung: Auf der Benutzeroberfläche soll neben der vorgestellten Ware die Reservierungsmöglichkeit anhand von einem Reservierungslink bzw. Reservierungsbutton angeboten werden.

Prozessbeschreibung:

1. Benutzer startet den Reservierungsprozess indem er den Reservierungslink tätigt.
2. Dem Benutzer werden die Reservierungskonditionen angezeigt.
3. Es wird überprüft ob der Benutzer angemeldet ist.
4. Im Fall, dass der Benutzer angemeldet ist, wird zum Punkt 7 übergangen, sonst zum Punkt 5.
5. Die Anmeldeseite mit der Aufforderung sich anzumelden wird generiert und angezeigt.
6. Anmeldeprozess wird durchgeführt.
7. Dem Benutzer wird die gewünschte Reservierung angezeigt.

8. In der Datenbank wird die entsprechende Ware als reserviert markiert.
9. Dem Benutzer wird Reservierungsbestätigung angezeigt.

Modellierung: Im Prozessmodell wird das Entwurfsmuster durch die Klasse „Reservierung“ beschrieben. Außerdem wird im Prozessmodell das Aktivitätsdiagramm „Reservierung“ (siehe Abb. 19) erstellt, das den gesamten Reservierungsprozess beschreibt. Im Präsentationsmodell werden auf den entsprechenden Positionen die Reservierungsbuttons/Reservierungslinks platziert.

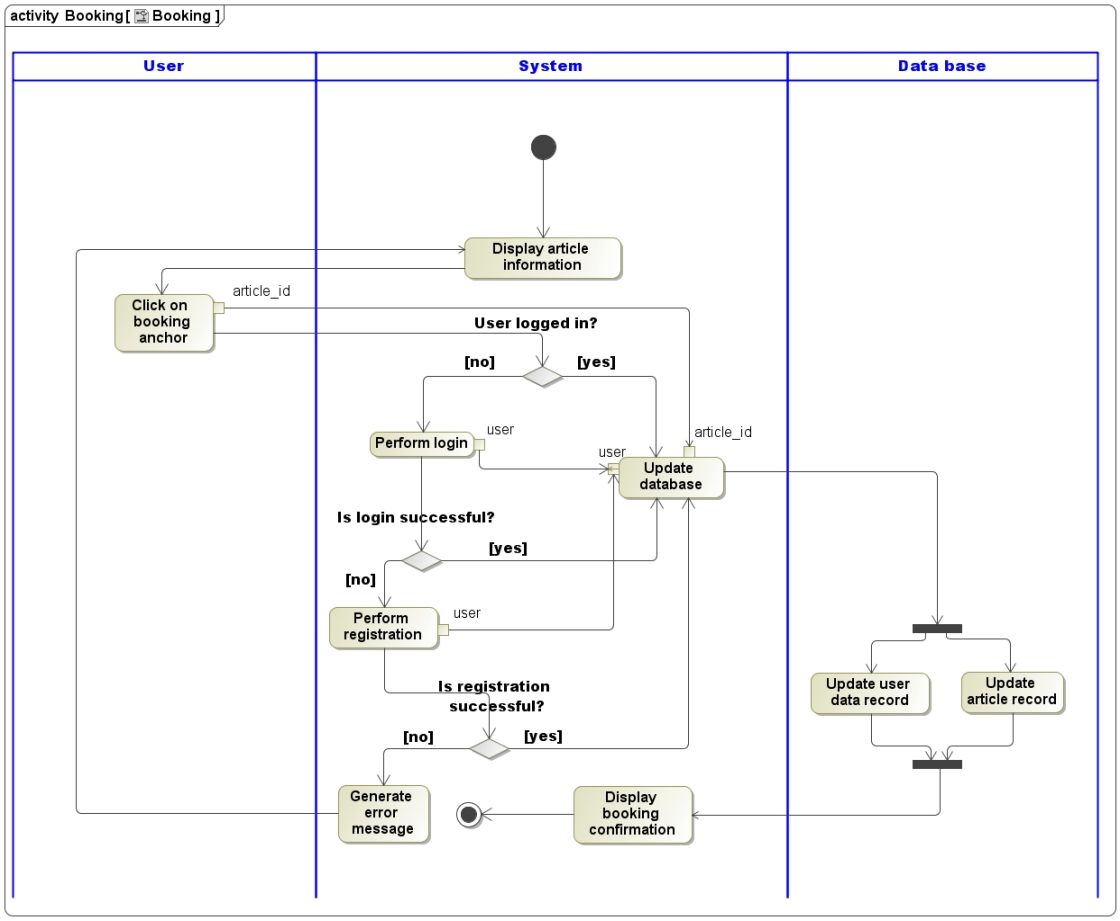

Abb. 19. Reservierung - Aktivitätsdiagramm

3.6 Einkaufwagen

Beschreibung: Zur Abwicklung des Einkaufsprozesses wird dem Benutzer eine Möglichkeit angeboten mehrere Produkte in so genannten virtuellen „Einkaufswagen“ zu sammeln. Dabei wird neben jedem Artikel auf der Seite die Möglichkeit angeboten diesen Artikel dem virtuellen Einkaufswagen hinzufügen. Dennoch ist dieses „Sammeln“ von Artikeln unverbindlich und kann entweder mit einem Kauf enden oder einfach von dem Benutzer abgebrochen werden.

Zweck: Diese Funktionalität vereinfacht den Kaufprozess von mehreren Artikeln. Dies spart Zeit und animiert den Benutzer zum weiteren Kauf. Der aktuelle Inhalt des Einkaufswagens kann jeder Zeit angezeigt werden. Der Käufer kann den Einkaufsprozess

auf beliebige Zeit erstrecken. Dabei wird der Benutzer an den realen Einkaufsprozess im Supermarkt erinnert, was ein Einkauf im Internet so intuitiv wie möglich gestaltet. Diese Vorgehensweise macht möglich, einen unerfahrenen Benutzer beim Einkauf im Internet zu unterstützen.

Auch bekannt als: Shopping Cart (englisch).

Umsetzung: Auf der Benutzeroberfläche wird neben jedem Artikel eine Möglichkeit angeboten, diesen Artikel dem Einkaufswagen hinzufügen. Auf der Abb. 20 ist ein reales Beispiel angezeigt, wie dies realisiert werden kann.

Abb. 20. Artikel dem Einkaufswagen hinzufügen bei http://www.kraeuterhaus.de (10.02.09)

Der aktuelle Zustand des Einkaufswagens kann jeder Zeit auf die Benutzeranforderung angezeigt werden. Dafür kann auf jeder Seite der Anwendung ein Link zu dem aktuellen Zustand des Einkaufswagens angezeigt werden (siehe Abb. 21).

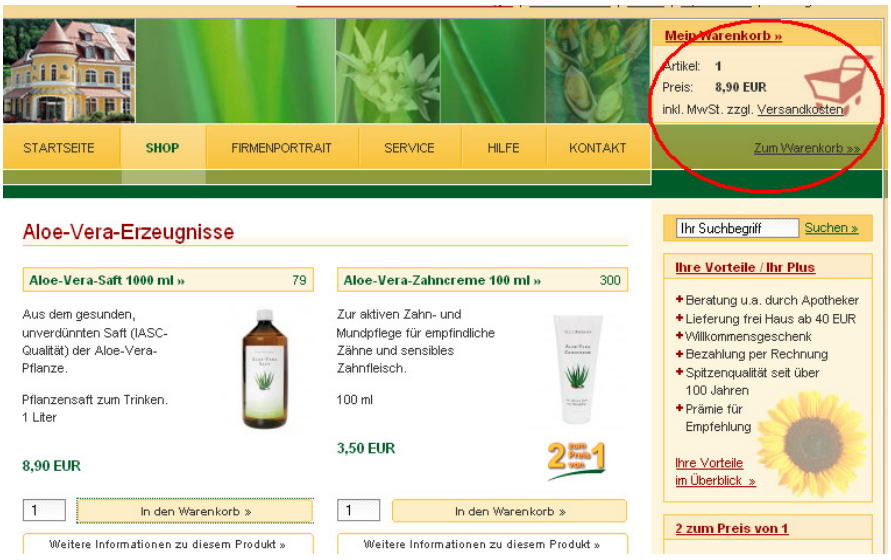

Abb. 21. Link zu dem aktuellen Zustand des Einkaufswagens bei http://www.kraeuterhaus.de (10.02.09)

Der Inhalt des Einkaufswagens schließt typischerweise die gesamte Liste der ausgewählten Artikeln, sowie deren kurzen Beschreibungen und Preise ein. Man kann die

Stückzahl für jeden Artikel ändern, sowie jeden Artikel komplett aus der Liste entfernen. Das Layout des Einkaufwagensinhalts ähnelt einer Rechnung. Dabei wird der Gesamtpreis für alle Artikel zusammen mit allen anderen Leistungen (wie z.B. Lieferung usw.) am Ende dieser „Rechnung“ angezeigt. Ein mögliches Layout des Einkaufswagensinhalts sieht man auf der Abb. 22.

Abb. 22. Der aktuelle Zustand des Einkaufswagens bei http://www.kraeuterhaus.de (10.02.09)

Prozessbeschreibung:

Der gesamte Prozess besteht aus zwei Unterprozessen und zwar aus dem Unterprozess1 – „Ein Artikel dem Einkaufswagen hinzufügen“ und aus dem Unterpozess2 – „Einkaufswageninhalt anzeigen“.

Unterpozess1:

1. Der Benutzer tätigt den „AddToShoppinCart“ - Button oder Link neben einem bestimmten Artikel.
2. Dieser Artikel wird dem virtuellen Einkaufswagen hinzugefügt.
3. Der Einkaufsprozess wird fortgesetzt.

Unterpozess2:

1. Der Benutzer tätigt den „ShowShoppingCart“ - Button oder Link.
2. Der aktuelle Inhalt des Einkaufswagens wird aufgelistet.
3. Im Fall, dass der Benutzer einige Änderungen auf der Artikelliste (sowie ein Artikel aus der Liste löschen, oder die Stückzahl zu einem Artikel ändern) vornimmt, wird zum Punkt 4 übergangen, sonst zum Punkt 7.
4. Änderung an der Artikelliste vornehmen.
5. Die Liste bezüglich dieser Änderung anpassen.
6. Gehe zum Punkt 3.
7. Im Fall, dass der Benutzer keine weiteren Änderungen auf der Artikelliste vornehmen will, wird dieser Prozess beendet.

Modellierung: Im Prozessmodell wird das Pattern durch die Klassen „AddToShoppingCart“ und „ShowSchoppingCart“ beschrieben. Außerdem werden im Prozessmodell die Aktivitätsdiagramme „AddToShoppingCart“ (siehe Abb. 23), die dem

Unterprozess1 entspricht, und „ShowShoppingCart“ (siehe Abb. 24) erstellt, die dem Unterprozess2 entspricht. Diese zwei Aktivitätsdiagramme beschreiben den gesamten Einkaufsprozess. Im Präsentationsmodell werden auf den entsprechenden Positionen die „AddToShoppingCart“ - Buttons/ - Links platziert und der Link – „ShowShoppingCart“ wird von überall in der Web-Anwendung erreichbar gemacht.

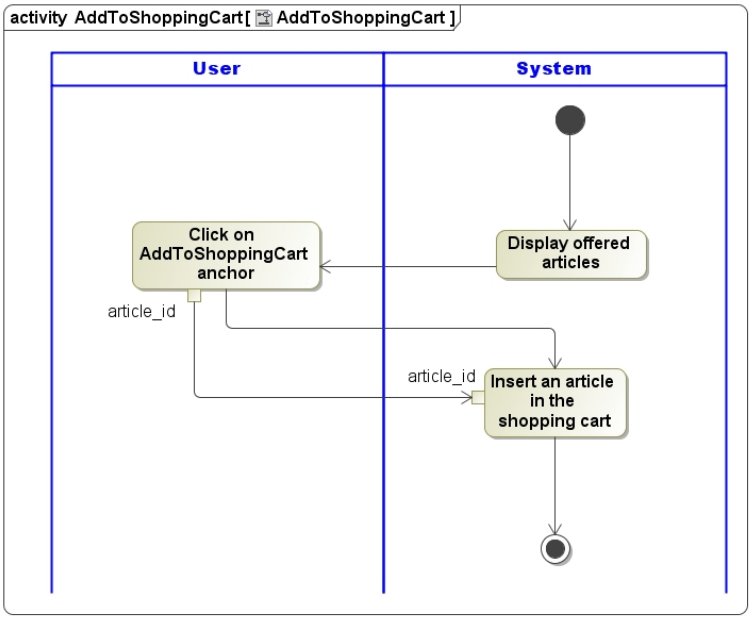

Abb. 23. AddToShoppingCart - Aktivitätsdiagramm

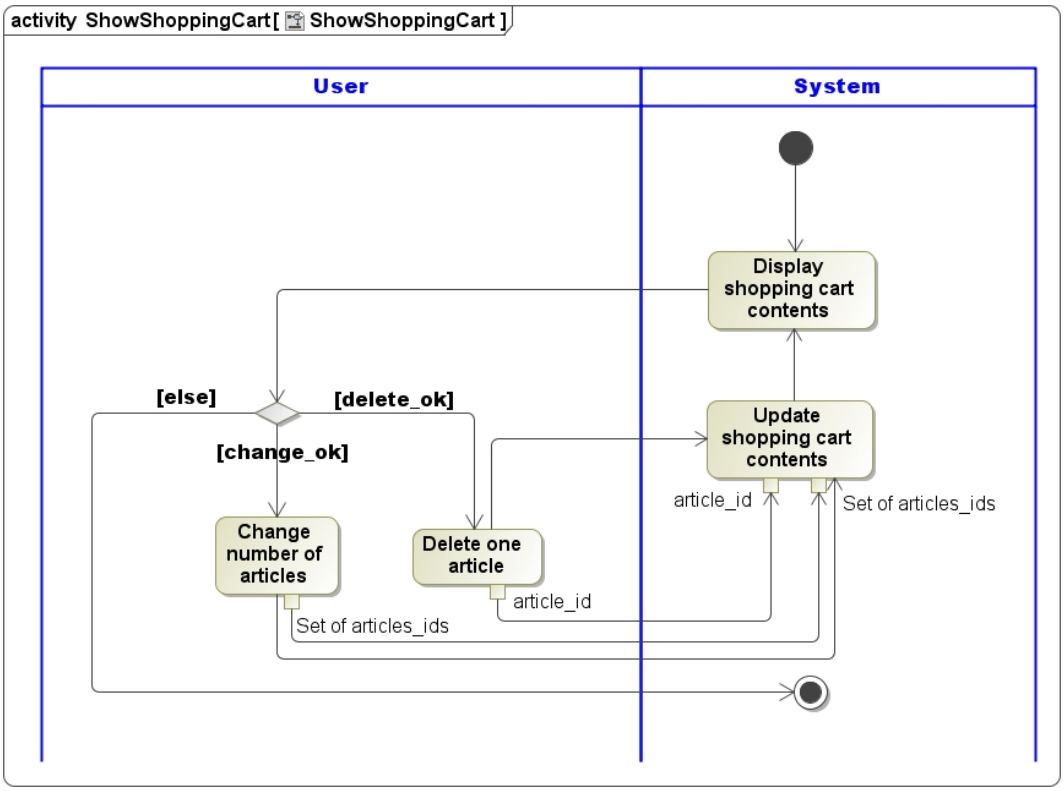

Abb. 24. Einkaufswageninhalt anzeigen - Aktivitätsdiagramm

Reale Beispiele (optional): Ein weiteres Beispiel für das Layout der Seite mit dem aktuellen Inhalt des Einkaufswagens ist auf der Abb. 25 angezeigt.

Abb. 25. Einkaufswagen bei www.amazon.de (19.07.08)

3.7 Bestellung

Beschreibung: Der krönende Abschluss für jeden Einkauf ist normalerweise der Kauf der ausgewählten Artikel. Innerhalb des Kaufprozesses wird eine verbindliche Bestellung der ausgewählten Waren durchgeführt. Diese Bestellung kann auch die Bezahlung im Voraus einschließen.

Zweck: Diese Funktionalität spielt eine Rolle des Vertrags zwischen dem Kunden und dem Servicebetreiber. Und soll sowohl den Betreiber als auch den Kunden in seinen Rechten unterstützen.

Auch bekannt als: Kaufprozess, Purchase.

Umsetzung: Als Abschluss des Einkaufens wird normalerweise der Kaufprozess durchgeführt. Dabei sind folgende sechs Schritte die wichtigsten bei dem Kaufprozess:

Prozessbeschreibung:

1. Kunde identifizieren
2. Lieferadresse und spezielle Optionen auswählen
3. Zahlungsmethode auswählen
4. Übersicht der kompletten Rechnung
5. Bestätigen und Auftrag erteilen
6. Bestätigung der erfolgreichen Bestellung per E-Mail an den Kunden verschicken.

Auf jedem der Schritte kann der Prozess durch den Kunden unterbrochen werden.

Modellierung: Im Prozessmodell wird das Entwurfsmuster durch die Klasse „Purchase“ beschrieben. Außerdem wird im Prozessmodell das Aktivitätsdiagramm „Purhase Process“ (siehe Abb. 26) erstellt, das den gesamten Kaufsprozess beschreibt.

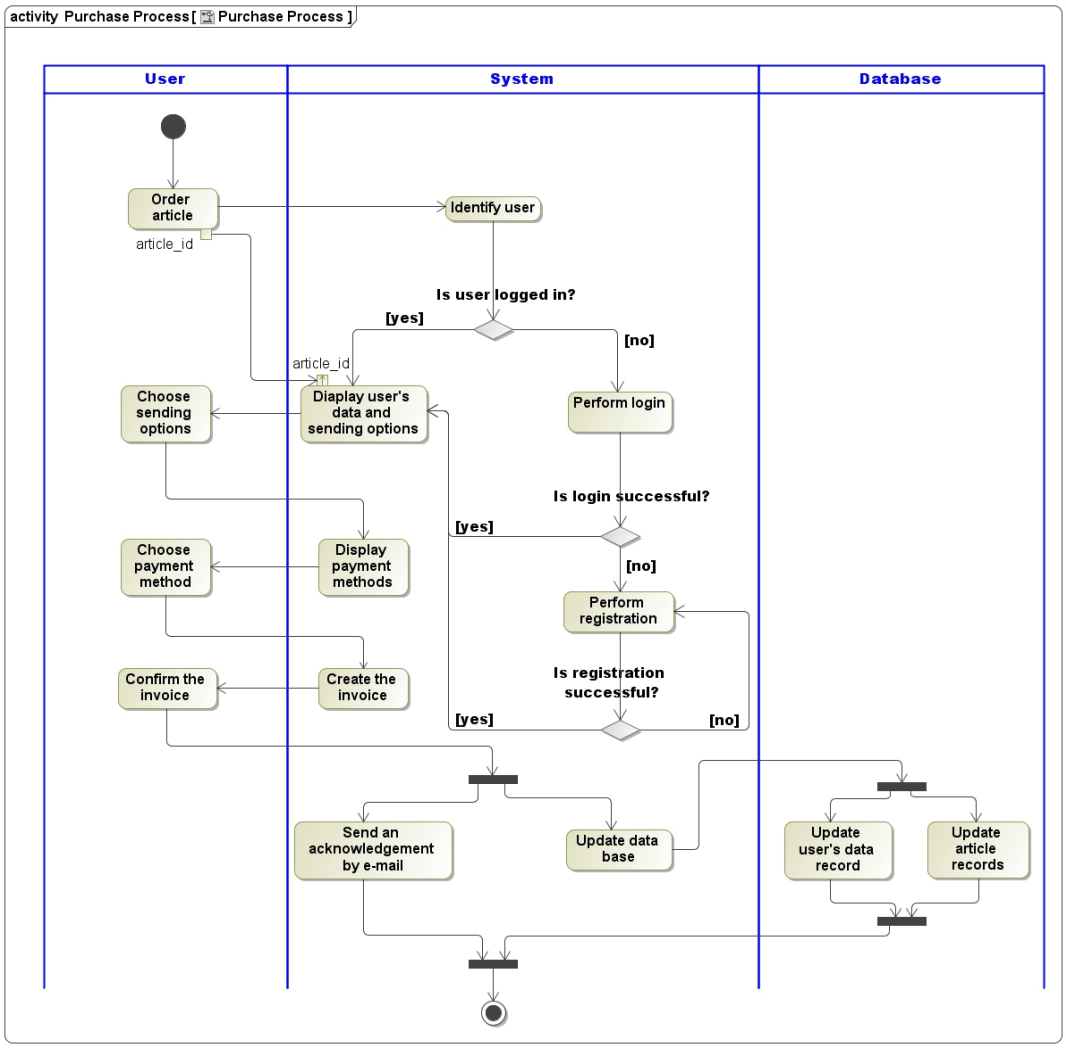

Abb. 26. Bestellung- Aktivitätsdiagramm

Im Präsentationsmodell kann sowohl neben jedem Artikel der Expresskaufbutton angeboten werden oder auf der Seite, die den aktuellen Warenkorbinhalt anzeigt, kann der Kaufbutton angezeigt werden. Die zweite Methode hat sich zurzeit bei den modernen Web-Anwendungen durchgesetzt (eine mögliche Realisierung sieht man auf der Abb. 27 sehen).

Abb. 27. Kaufbutton auf der Einkaufswageninhaltseite bei http://www.kraeuterhaus.de **(10.02.09)**

Reale Beispiele (optional): Ein weiteres Beispiel für den Kaufprozess auf der Seite vom „Otto Versand“ ist auf der folgendem Abb. 28 angezeigt. Hier werden von den Entwicklern auch die einzelnen Schritte des gesamten Kaufsprozess (links im Bild) repräsentiert. Das spricht für Benutzerfreundlichkeit der Web-Anwendung, da der Benutzer den aktuellen Stand des Prozessablaufs kennt.

Abb. 28. So wird auf „Otto“ –Online Shop zur Kasse gebeten (www.otto.de, 19.02.2009)

3.8 Suche

Beschreibung: Die Webapplikation kann nach einem oder mehreren Kriterien durchsucht werden. Im Fall dass es eine Kombination von mehreren Suchkriterien angeboten wird, spricht man von der erweiterten Suche. Ein typisches Beispiel für die einfache Suche stellt die Abb. 29 dar.

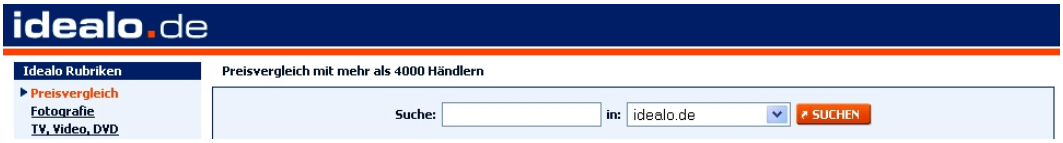

Abb. 29. Typisches Beispiel für die einfache Suche bei www.idealo.de (08.08.08)

Ein mögliches Beispiel der erweiterten Suche sieht man bei der „Google“-Webseite (siehe Abb. 30). Im diesen Fall können mehrere Suchkriterien, wie z.B. Sprache, Datum und Dateiformat, angegeben werden. Dadurch beschränken sich stark die möglichen Suchergebnisse entsprechen diesen Suchkriterien.

Abb. 30. Beispiel für die erweiterte Suche aus www.google.com (11.09.08).

Zweck: Damit die Benutzer einen besseren Überblick über die Webapplikation erhalten und schnell gesuchte Artikel oder die benötigte Information bekommen, wird die Suchfunktionalität verwendet. Dies macht das Nutzen der Webapplikation komfortabel und unterstützt besonderes den unerfahrenen Benutzern beim Einstieg.
Die Seiten des elektronischen Handels, Multinationale Seite oder Portalseite enthalten so viel Information, dass eine einfache Suchefunktion nicht immer ausreichend ist. In diesem Fall wird das Suche-Entwurfmuster auf weiteren Suchoptionen erweitert. Bei „Erweiterte Suche“ werden mehrere Suchkriterien vorgeschlagen, die Suchergebnisse stark nach den gesuchten Artikel-Typen (Artikel, Video, Audio-...), oder Artikel-Eigenschaften (Titel, Datum, Speicherstelle, Größe, Autor...) beschränken. Außerdem spielt auch die Ausgabesteuerung der Suchergebnisse eine wichtige Rolle. Solche Aspekte wie Sortieren oder Paginierungsgrößen müssen die Entwickler berücksichtigen.

Auch bekannt als: Search (englisch).

Umsetzung: Auf der Benutzeroberfläche wird eine Suchmaske angeboten. In dieser Maske werden alle gewünschten Suchkriterien mit den gewünschten Werten besetzt.

Prozessbeschreibung:

1. Der Benutzer setzt die Werte für die Suchkriterien ein.
2. Webapplikationsobjekte werden entsprechend der Suchkriterien durchsucht.
3. Die Suchergebnisse werden aufgelistet.

Modellierung: Im Prozessmodell wird das Pattern durch die Klasse „Search“ beschrieben. Die möglichen Suchkriterien können anhand von OCL-Ausdrücken auf der Modellebene angegeben werden. Außerdem wird im Prozessmodell das Aktivitätsdiagramm (siehe Abb. 31) „Search“ hinzugefügt, das den gesamten Suchprozess beschreibt. Bei „Erweiterte Suche“ können die Eingaben in mehreren Schritten angefordert werden. Dabei werden im Prozessablauf statt einen unübersichtlichen Suchformular mehrere kleinere Formulare eingebaut (ähnliche Vorgehensweise wie beim Wizard).

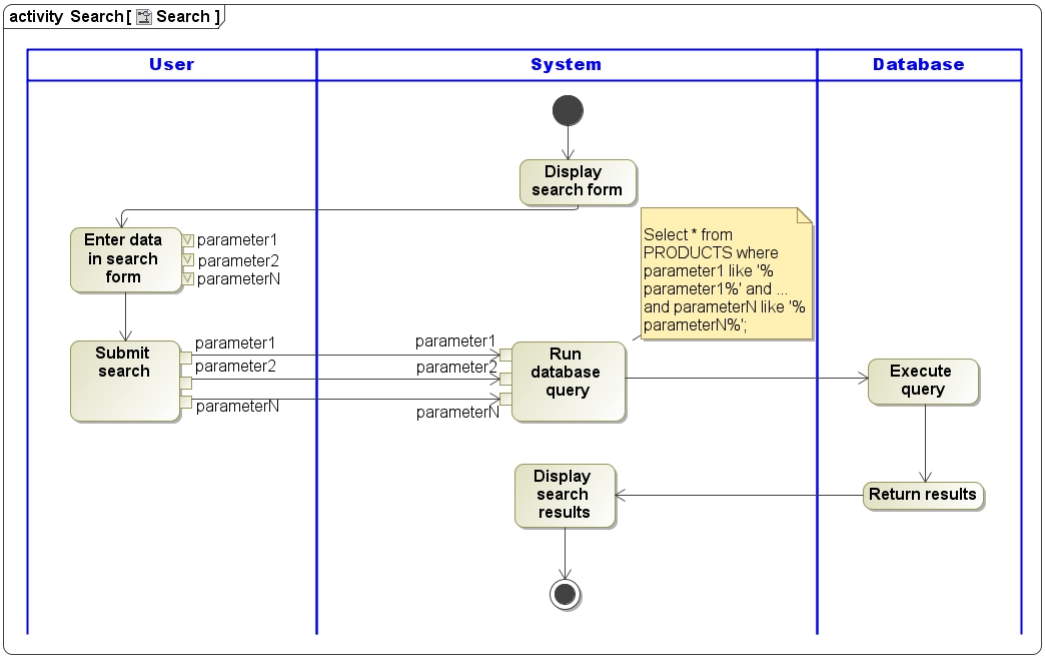

Abb. 31. Suchprozess- Aktivitätsdiagramm

3.9 Zusammenfassung

In diesem Kapitel wurden die am häufigsten benutzten Analyse-Patterns für die Web-Anwendungen zusammengefasst und schematisch beschrieben. In Rahmen dieser Diplomarbeit wurde versucht, für die Analyse-Patterns nicht nur eine grobe Beschreibung dessen Funktionalität und die Varianzen der Präsentationslayouts, sondern auch neben die Prozessabläufe den Objektfluss darzustellen. Ferner bieten diese Entwurfsmuster einige Vorteile für den Web-Anwendungsentwickler:

1. Granularität. Die richtige Abgrenzung und Problemaufteilung macht einzelne Entwurfsmuster nicht nur übersichtlicher, sondern steigt auch die Wiederverwendbarkeit.
2. Zeit- und Kostenreduzierung, da diese Entwurfsmuster bei vielen Web-Anwendungen benutzt werden und ähnliches Layout haben.
3. Leichte Erweiterbarkeit. Diese Analyse-Patterns können als Basis benutzt werden, da die sehr flexibel und auf weitere spezifische Anforderungen leicht anpassbar sind. So können bei „Registrierung“ oder „Benutzerprofil ändern“ in dem Prozessablauf die Prüfung mehreren Eingabedaten, wie z.B. Bankleitzahl oder Postleitzahl, eingebaut werden.
4. Vorteile bei Reorganisation von Web-Anwendung.
5. Der Entwickler kann selber auswählen, in welche Sprache diese Analyse-Patterns implementieren werden.
6. Der feste Katalog von Entwurfsmuster vereinfacht die Kommunikation in Entwicklerteams.
7. Erleichterung für Neueinsteiger mit qualitativ besserer Entwicklung, Wartung und Dokumentation.

In der Praxis spiegeln die Entwurfsmuster den Erfahrungsschatz unzähliger vorhergehender Projekte und Entwicklungen wieder. Diese Erfahrungen wurden von sinnvoller Verwendung von Designelementen in der Detailplanung bis zu effizienten Algorithmen in der Realisierung erzielt.

Sie sind jedoch kein Allheilmittel zur Schaffung der Benutzerfreundlichkeit. Dem Entwickler ist damit ein weiteres, effizientes Werkzeug an die Hand gegeben, mit welchem er ein besseres Verständnis, und damit eine bessere Lösung für gewisse Projektziele erarbeiten kann.

4 Lösungsansatz von Entwurfsmustern in der Modellierung

Diese Arbeit befasst sich auch mit der Einsatzmöglichkeit von Analyse-Pattern in der Praxis. Als Modellierungssprache wurden MDUWE, ATL als Transformationssprache und für die automatische Generierung von Web-Anwendungen das Transformationswerkzeug UWE4JSF ausgewählt, deren Generierungsprozess im nächsten Abschnitt kurz vorgestellt wird. In weiteren Abschnitten sind der gesamte Lösungsansatz von Entwurfsmustern in der Modellierung mit neuen Features beschrieben, wie Erweiterung der UWE-Metamodells mit Pattern-Mechanismus und Erstellung des neuen Profils, und ein Beispiel für die Entwurfsmustermodellierung gezeigt.

4.1 Kurzer Überblick über den Generierungsprozess von UWE4JSF

In der Diplomarbeit von Christian Kroiß [31] wurde ausführlich erklärt, wie mit MDUWE das plattformunabhängige Modell (Platform Independent Model, PIM) einer Web-Anwendung unter Verwendung des Transformationswerkzeugs UWE4JSF und durch Ergänzung von nicht generierten Anteilen eine vollständige JSF-Web-Anwendung erzeugt werden kann. In Abb. 32 wird ein Überblick über den Generierungsprozess von UWE4JSF gegeben.

Das eigentliche MDUWE-PIM zusammen mit dem konkreten Präsentationsmodell, die im Teildiagramm mit der Bezeichnung „Modellierung“ auf Abb. 32 dargestellt sind, bilden die Grundlage für die anschließende Generierung. Durch Modelltransformationen „UML2UWE“ wird ein kombiniertes MDUWE-Modell erzeugt, das nach der anschließenden Validierung und weitere Transformationen in ein plattformspezifisches Modell umgewandelt wird. Anschließend erfolgt eine Modell-zu-Text-Transformation, deren Resultat, zusammen mit einigen nicht generierten Anteilen, eine JSF-Anwendung bildet. Das Werkzeug UWE4JSF wird innerhalb der Entwicklungsumgebung Eclipse eingesetzt. Dies umfasst vor allem die Konfiguration des Web-Abwendungs-Projekts und der Transformationskette.

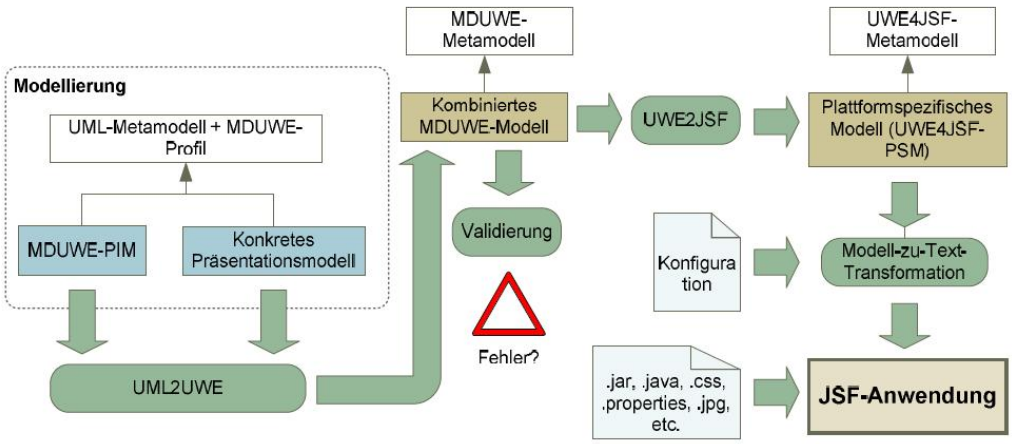

Abb. 32. Überblick über den Generierungsprozess von UWE4JSF

Ein zentrales Thema dieser Diplomarbeit ist die Einsetzung der Analyse-Pattern für die Modellierung eines Web-Systems mit MDUWE und ihre Integration durch weitere Transformationsschritten in der Transformationskette. Einen Überblick über die Integration des Entwurfsmusters im Werkzeug UWE4JSF vermittelt der nächste Abschnitt.

4.2 Integration der Entwurfsmuster im Transformationswerkzeug UWE4JSF

Durch den Einsatz der Model Driven UMLbased Web Engineering (MDUWE) und des Transformationswerkzeugs UWE4JSF in der Softwareentwicklung wird der Entwurf und die Generierung von Web-Systemen ermöglicht, die auf der Java Server Faces Plattform aufsetzen. Zusätzlich ermöglicht die Nutzung von Entwurfsmustern (Analyse-Pattern) die Wiederverwendung von erfolgreichen Entwürfen und verbessert die Flexibilität des Entwurfs.

Auf Abb. 33 wird ein Überblick über die Integration des Entwurfsmusters unter der Einsetzung des Transformationswerkzeugs UWE4JSF vorgestellt.

Mit MDUWE werden die plattformunabhängigen Modelle (Platform Independent Model, PIM) einer Web-Anwendung und verschieden Entwurfsmustern erstellt. Danach widmet sich dieser Teil der Frage, wie ausgehend von diesem PIM für die Web-Anwendung, unter Verwendung des Transformationswerkzeugs UWE4JSF durch PIM's von Entwurfsmustern ergänzt werden kann und in eine vollständige JSF-Web-Anwendung generiert werden kann.

Man erkennt zunächst im Teildiagramm mit der Bezeichnung „Modellierung der Web-Anwendung", dass das eigentliche MDUWE-PIM zusammen mit ihrem konkreten Präsentationsmodell die Grundlage für die anschließende Generierung bildet. Danach wird durch Modelltransformationen „UML2UWE" ein kombiniertes MDUWE-Modell erzeugt. Mit der gleichen Vorgehensweise werden einmalig auch die kombinierten MDUWE-Modelle für jedes Entwurfsmuster erstellt. In der späteren Version vom Transformationswerkzeug UWE4JSF können die kombinierten MDUWE-Modelle für mehrere Entwurfsmuster in einer Bibliothek gesammelt und den Benutzer zur Verfügung gestellt werden. Damit sind die Vorbereitungsschritten für die Einsetzung von Entwurfsmuster abgeschlossen.

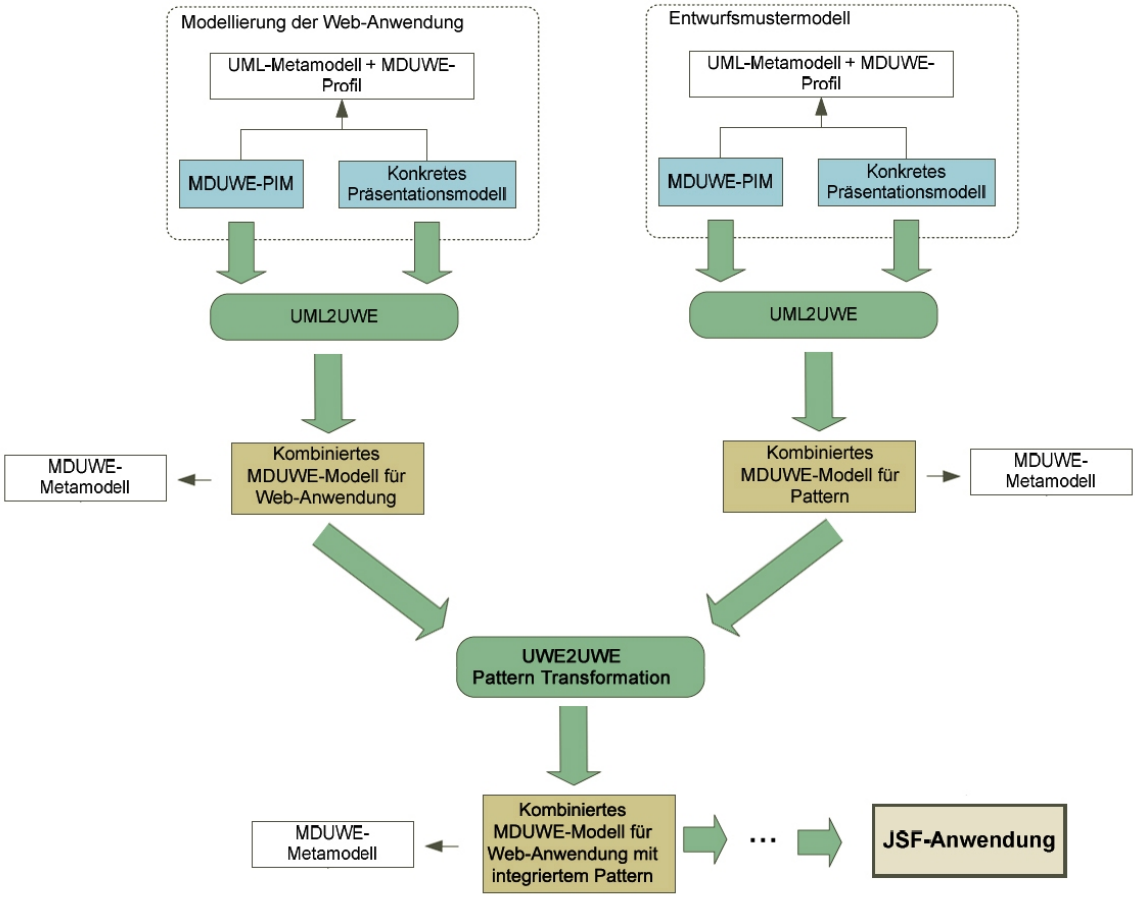

Abb. 33. Überblick über die Integration des Entwurfsmusters im Werkzeug UWE4JSF

Die Integration von Entwurfsmuster in der Web-Anwendung selber wird im Schritt „UWE2UWE Pattern Transformation“ geschehen, der aus eine oder mehrere Modell-zu-Modell Transformationen besteht (ausführlicher im Kapitel 5). Durch diese Modelltransformationen wird ein kombiniertes MDUWE-Modell für die Web-Anwendung mit integrierten Entwurfsmustern erzeugt, das anschließend validiert und weiter transformiert werden kann, um ein plattformspezifisches Modell zu erzeugen.

Um die Integration einen Entwurfsmuster zu realisieren, wurden in den früheren Entwicklungsversionen dieser Diplomarbeit die Namen für die Klassen und ihren Parameter in den Modellierungsschritt festgelegt. Im Beispiel auf die Abb. 34 sind diese Modellelemente umgekreist.

Dies erforderte der Schritt von der Integration – „UWE2UWE Pattern Transformation“, da bei der „Fusion“ von beiden kombinierten Modellen, für die Web-Anwendung und für Analyse-Pattern, bestimmte Klassen, wie die Prozessklasse „Login“ und die Präsentationsklasse „Login“ auf dem Abb. 34, und Geschäftsprozessen ersetzt, geändert und gelöscht werden. Bei der Modell-zu-Modell Transformation müssen die anzupassenden Modellelemente zuerst identifiziert werden und diese Modellelemente wurden in den früheren Entwicklungsversionen nach ihren Typ und Name gesucht wurden. Dadurch wurde

die Modellierung einer Web-Anwendung stark eingeschränkt und die Verwendung von den Entwurfsmustern könnte die Entwicklung der Web-Anwendung nur erschweren.

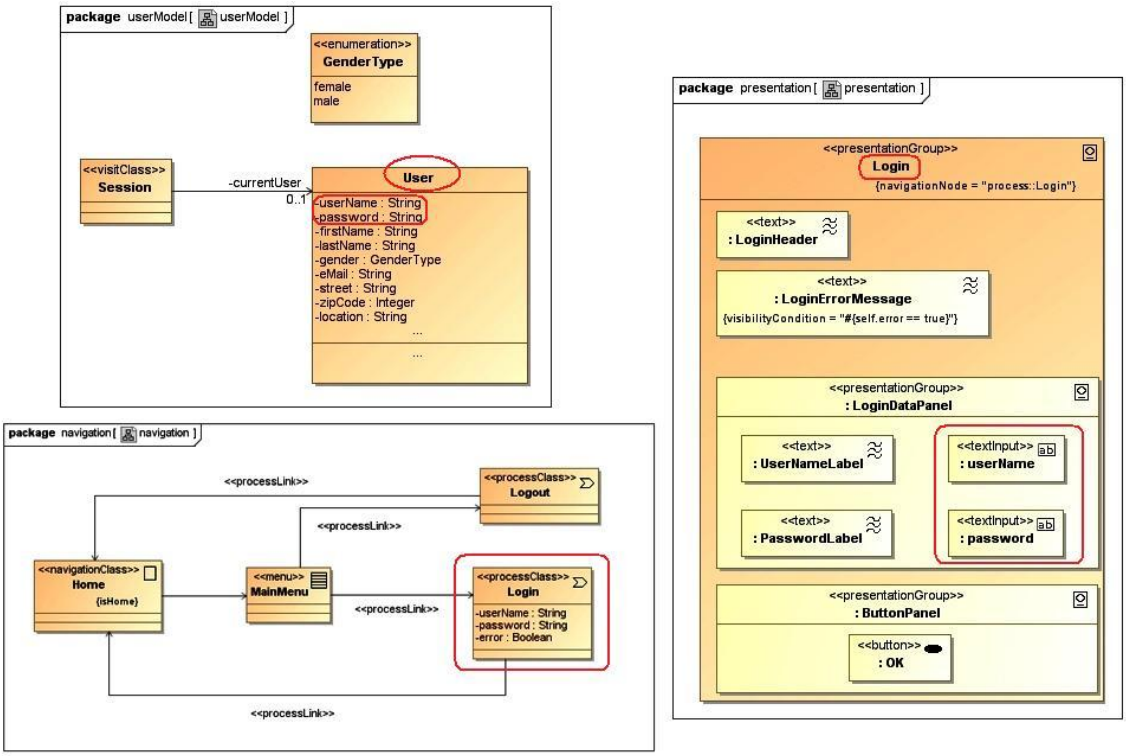

Abb. 34. Teilmodelle von Web-Anwendung mit Einsetzung des Entwurfsmusters „Login“ in der früheren Entwicklungsversionen.

Durch die Verwendung vom Pattern-Mechanismus (siehe Abschnitt 4.3), das in Rahmen diese Diplomarbeit entwickelt wurde, wurde der Gebrauch speziellen Stereotypen mit verschiedenen Parametern abhängig vom Transformationsalgorithmus möglich gemacht. Durch den Ansatz diesen Stereotypen wird das kombinierte MDUWE-Modell für die Web-Anwendung für Modell-zu-Modell Transformationen parametrisiert. Außerdem sind die Namensfreiheit für die Klassen und ihren Parameter und mehr Flexibilität im Modellierungsschritt gewährleistet.

Im nächsten Abschnitt wird die UWE-Metamodellerweiterung – das Pattern-Mechanismus dargestellt.

4.3 Beschreibung der Metamodelleserweiterung

Um die entwickelten Entwurfsmuster einzusetzen, ist eine UWE-Metamodellerweiterung nötig. Sonst wäre keine Parametrisierung möglich und der Modellierer müsste gezwungenermaßen an mehreren Stellen nur vorgeschlagene Parameter bei der Modellierung von Web-Anwendung einsetzen, wie z.B. der Name von der Prozessklasse musste unbedingt „Login“ sein und seinen Parameter sollten nur „*userName*“ und „*password*“ genannt werden. Dies erfüllt nicht den Sinn vom Entwurfsmuster.

Um den Analyse-Pattern mehr Flexibilität zu erteilen, wurde in Rahmen dieser Diplomarbeit das UWE-Metamodell auf weitere Klassen erweitert. Das Navigationsmodell wurde auf die

Klasse „NavigationPattern“, das Prozessmodell auf die Klasse „ProcessPattern“ und das Präsentationsmodell auf „PresentationPattern“ erweitert. Alle diese neuen Klassen spezialisieren die Klasse „UWEPattern“.

Pattern	UWE-Modell	Spezialisiert Klasse; Kurze Beschreibung
NavigationPattern	navigation	NavigationNode, UWEPattern;
ProcessPattern	navigation, process	ProcessClass, UWEPattern
PresentationPattern	presentation	Presentation Group, UWEPattern;

Außerdem jede UWEPattern-Klasse hat einen Parameter „patternName“ mit Typ String und kann mehrere (*) PatternParameter besitzen, die ihrerseits mit Parametern „name“ und „value“ belegt werden können (siehe Abb. 35).

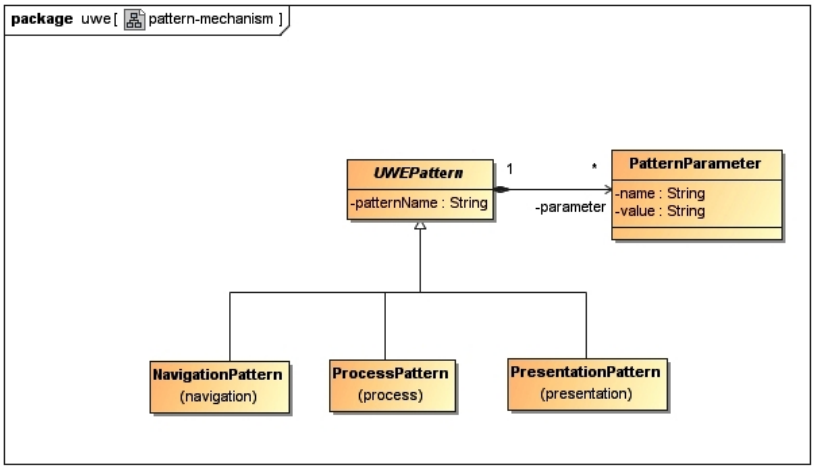

Abb. 35. Schema für das Pattern-Mechnismus.

Dieses Pattern-Mechanismus ermöglicht viel mehr Flexibilität für die Entwickler, der er nun an eine bestimmte Stelle die Parameter mit den entsprechenden Wertbelegungen modellieren muss. Die Vorgehensweise bei der Modellierung einer Web-Anwebdung wird im Abschnitt 4.5 anhand vom Beispiel „Login“ ausführlich erklärt.

Im folgenden Anschnitt 4.4 wird die Erweiterung von UWE-Profil beschrieben, die auf das Pattern-Mechanismus gebaut ist.

4.4 Erweiterung vom UWE-Profil

Um die Modellierung von der Web-Anwendung mit der Einsetzung von Entwurfsmuster für die Entwickler zu erleichtern, wurde im Rahmen dieser Diplomarbeit das UWE-Profil auf Patternprofil erweitert. Im Patternprofil werden die neuen Stereotype definiert, die für die Benutzung vom Analyse-Pattern eine wichtige Rolle spielen. Da es bei der Integration von den meisten Analyse-Pattern nicht nur um die Prozesslogik der entsprechenden Prozessklasse

handelt, sondern auch um die Änderungen bei den Layouts in dem Präsentationsmodell, werden für jedes Pattern mehrere Stereotypen gebraucht.

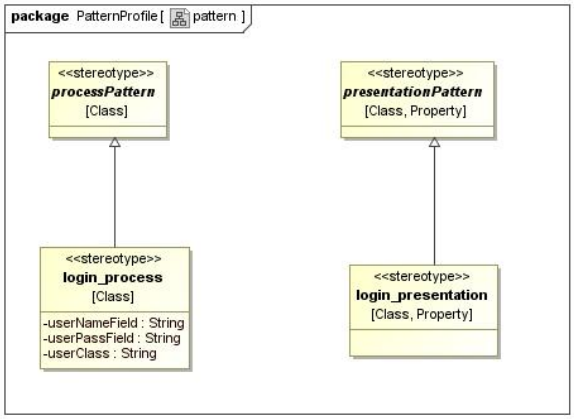

Abb. 36. Ein Beispiel zu Stereotypen für das Entwurfsmuster „Login“

Auf der Abb. 36 sind die Stereotypen für das Beispiel „Login“ vorgestellt. Das erste Stereotyp *<<login_process>>* liefert die benötigte Information für die ATL-Transformation (näheres dazu im Kapitel 5) über die anzupassenden Modellelemente, die aus der Belegung von Parametern „*userClass*“, „*userNameField*“ und „*userPassFiled*“ gelesen werden. Der andere Stereotyp *<<login_presentation>>* hat keine Parameter, da in diesem Fall fast standardisiertes Layout existiert und dieses Layout nach der Ausführung eine Kette von der ATL-Transformationen angepasst ins kombinierte MDUWE-Modell für die Web-Anwendung mit integriertem Entwurfsmuster übernommen wird.

Die Idee des Pattern-Mechanismus und das Patternprofil mit Stereotypen zu verwenden, ist bei der Implementierung der ATL-Transformationen entstanden, um den Entwickler der Web-Anwendung mehr Freiheit bei der Modellierung zu gewährleisten. Da bei der Modell-zu-Modell Transformation die anzupassenden Modellelemente zuerst identifiziert werden müssen, wird die Information über diese Modellelemente in den Eigenschaften von Klassen gespeichert. Auf diese Klassen werden spezielle Stereotypen angewandt, die für jedes Analyse-Pattern abhängig vom Transformationsalgorithmus erstellt werden. Durch die Anwendung von Stereotypen der Entwurfsmuster wird die für Modell-zu-Modell Transformation parametrisiert.

Unter Parametrisierung wird in dieser Diplomarbeit verstanden, den Gesamtumfang bei der Modellierung einer Web-Anwendung durch die Verwendung eines Analyse-Patterns zu reduzieren. Dabei werden bei der Modellierung von Prozessklassen und PresentationsGroups die entsprechen Stereotypen angewandt und deren Parameter und Eigenschaften dementsprechend gesetzt (ausführlicher im Abschnitt 4.5).

Deswegen sollten solche Stereotypen eine minimale erforderliche Menge von Parameter besitzen. Andernfalls wird die Benutzung von diesen Stereotypen unnötig kompliziert. Wie die Web-Anwendung mit der Einsetzung vom Entwurfsmuster in MDUWE modelliert wird, wird in dem nächsten Abschnitt 4.5 erklärt.

4.5 Modellierungsbeispiel für das Entwurfmuster „Login“

Das Entwurfmuster selber wird in MDUWE modelliert. Als Basis für den Prozessablauf und Präsentationsmodell können die Aktivitätsdiagramme und Modellierungsvorschläge aus

Kapitel 3 benutzt werden. So wurden im Rahmen dieser Arbeit zwei Entwurfsmuster „Login“ und „Registrierung“ modelliert. Die detaillierte Vorgehensweise bei der Modellierung in MDUWE ist ausführlich in der Diplomarbeit „*Modellbasierte Generierung von Web-Anwendungen mit UWE (UML-based Web Engineering)*“ von Christan Kroiß [31] erklärt.

Im Folgenden werden die erforderlichen Modellierungsschritte von Entwurfsmuster „Login“ dargestellt. Diese sind in zwei Entwicklungsteile gegliedert:

- Modellierung der Web-Anwendung;
- Modellierung des Entwurfsmusters.

Im ersten Teil „Modellierung der Web-Anwendung“ wird zunächst das Patternprofil mit speziellen Stereotypen für Verwendung des Entwurfsmusters (siehe Abb. 36) ausgearbeitet. Dieser Entwicklungsteilabschnitt spielt eine ganz wichtige Rolle für die weiteren Modellierungs- und Implementierungsschritten.

Für die Einsetzung vom jeweiligen Entwurfsmuster gibt es bestimmte Anzahl zu setzende Parameter in den entsprechenden Stereotyp. Dies ergibt sich aus der Überlegung, welche Informationen für die nach der Modellierung folgenden Modell-zu-Modell Transformationen notwendig sind. Zweifellos muss die Anzahl der Parameter minimal sein, sonst wäre der Entwurfsmuster unnötig kompliziert zu benutzen. Beim unteren Beispiel werden nur die Werte, wie Namen von der Klasse aus UserModell („*User*“) und seinen zwei Parameter (*uName* und *uPass*) für die ATL-Transformation gebraucht. Dementsprechend hat die Stereotype *<<login_process>>* nur drei Parameter: „*userClass*“, „*userNameField*“ und „*userPassField*“ (siehe Abb. 36). Diese müssen vom Entwickler bei der Modellierung der entsprechenden Prozessklasse „*Login*“ (siehe Abb. 37) mit dem Stereotyp *<<login_process>>* als Eigenschaften „*userClass*“ mit Wert „User“, „*userNameField*“ mit dem Wert „uName“ und „*userPassField*“ mit dem Wert „uPass“ gesetzt werden.

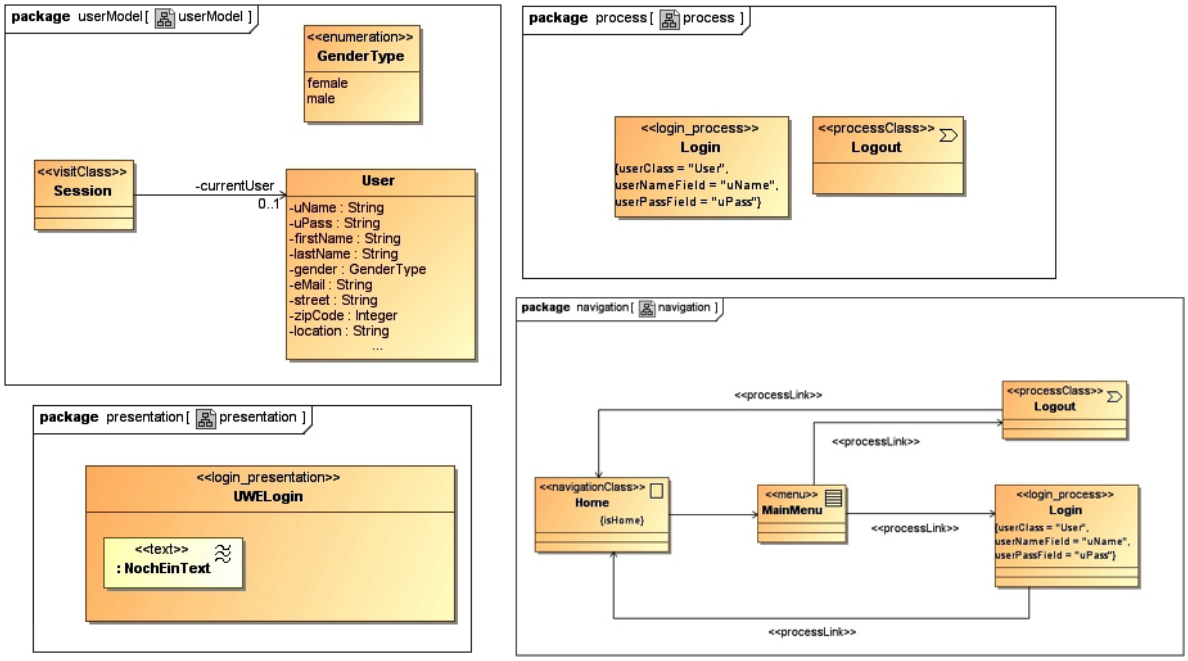

Abb. 37. Teilmodelle der Prozess-, Presentations- und Usermodellen von Web-Anwendung mit Einsetzung des Entwurfsmusters „Login“.

Wie auf Abb. 37 abgebildeten Prozessklasse, PresentationGroup und Klasse aus UserModell genannt werden (im Beispiel Prozessklasse „*Login*“, PresentationGroup „*UWELogin*“ und Klasse „*User*“), ist im weiteren Ansatz unbedeutend, da während der Modell-zu-Modell Transformation die Prozessklasse „*Login*“ und PresentationGroup „*UWELogin*“ nach ihren Stereotypen identifiziert werden.

Wie eine PresentationGroup mit dem Stereotyp *<<login_presentation>>* von der Web-Anwendung modelliert wird, hängt vom Design der Web-Anwendung und der Kreativität des Programmiererteams ab. Da es in diesem Beispiel in erste Linie um die Funktionalität geht, wird nur eine Klasse „*NochEinText*“ mit dem Stereotype *<<text>>* eingebaut ist. Diese Klasse wird unverändert in kombiniertes MDUWE-Modell für Web-Anwendung mit integriertem Pattern übernommen.

Damit ist der erste Teil der Modellierung abgeschlossen.

Im zweiten Teil „Modellierung des Entwurfsmusters“ wurde das gesamte Modell entwickelt, einschließlich des Prozessmodells mit dem Aktivitätsdiagramm und des Präsentationsmodells für das Pattern.

Das Prozessmodell beinhaltet die Prozessklasse „LoginProzess“ hat drei Parameter *error, userName* und *password*, die für die Prozesslogik essentiell sind, und Aktivitätsdiagramm „Login“.

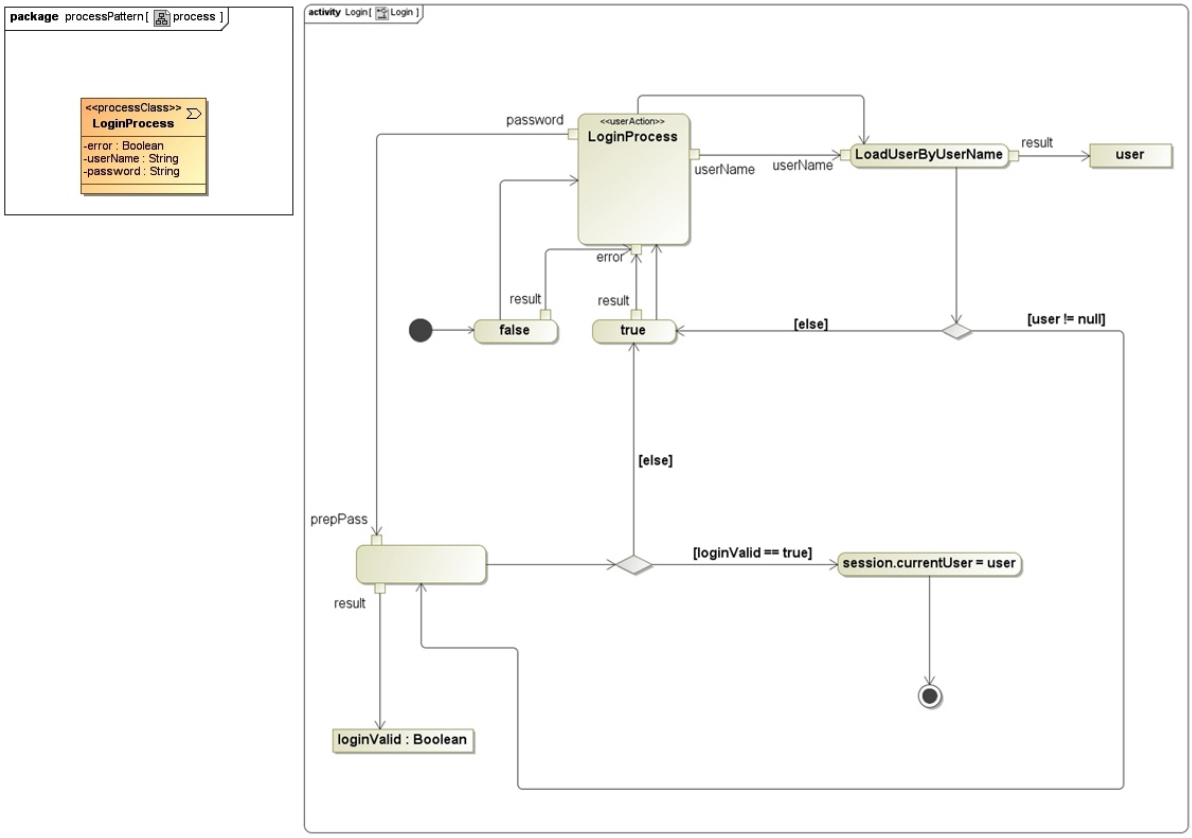

Abb. 38. Prozessklasse „LoginProcess“ und dazugehörige Aktivitätsdiagramm aus Prozessmodell des Entwurfsmuster „Login“

Das obere Aktivitätsdiagramm für das Patternmodell „Login“ (Abb. 38) zeigt den gesamten Anmeldungsprozess, der auf die genauere Prozessbeschreibung basiert, die im Abschnitt 3.2 dargestellt ist.

Im Präsentationsmodell stellt die Klasse „*LoginPresentation*“ das Layout des Patterns in Form eines Anmeldeformulars dar. „*LoginErrorMessage*“ wird bei der falschen Dateneingabe als Fehlermeldung eingezeigt und „*LoginHeader*“ ist für kleine Erläuterung zum Anmeldeformular gedacht. Ferner sind hier auch Engabefelder für Passwort und Benutzername neben ihren Labels und noch einen Submitbutton „ButtonPanel“ modelliert (siehe Abb. 39).

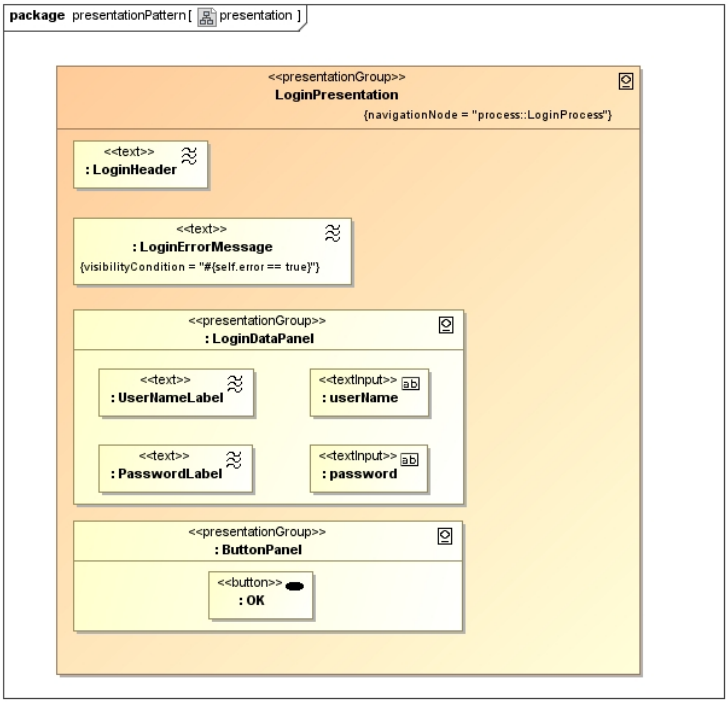

Abb. 39. Ausschnitt aus dem Präsentationsmodell für „Login“

Damit ist auch der andere Teil der Modellierung abgeschlossen.

Alle Klassen mit den Stereotypen *<<login_process>>* und *<<login_presentation>>* werden in dem Transformationsschritt „UWE2UWE Pattern Transformation“ durch spezielle Regeln (siehe Abschnitt 5.3) mit den entsprechenden Klassen aus Entwurfsmustermodell so geändert und zusammengefasst, dass die Parameter von der Prozessklasse „LoginProcess“, das Aktivitätsdiagramm „Login“ und das in der Klasse „LoginPresentation“ dargestellten Layout des Patterns in das kombiniertes MDUWE-Modell für Web-Anwendung mit integriertem Pattern übernommen werden.

Bei einem weiteren Beispiel „Registrierung“ wurde noch ein zusätzlicher Parameter „*userEMailField*“ für den Stereotyp *<<register_process>>* definiert. Damit werden die minimalen Anforderungen für den Einbau des Entwurfsmusters „Registrierung“ abgedeckt. Sollten im Usermodell für die „*userClass*“ weitere Parameter bei der Modellierung einer Web-Anwendung gedacht, werden diese während der Transformationenausführung aus die „*userClass*“ ermittelt und in den Prozessablauf und in Präsentationsmodell zur Laufzeit eingefügt (seihe Abschnitt 5.4).

4.6 Ergebnisse

In diesem Kapitel wurde der Modellierungsteil der Patternintegration in MDUWE für das Transformationswerkzeug UWE4JSF beschrieben. Das in Rahmen dieser Diplomarbeit entstanden Pattern-Mechanismus ermöglicht die notwendigen minimalen Anpassungen des Entwurfs vom Modell für die Web-Anwendung und gewährleistet mehr Freiheit bei der Modellierung. Ferner wurde das UWE-Profil um neues Patternprofil mit den speziellen Stereotypen erweitert. Diese Stereotypen mit verschiedenen Parametern sind abhängig vom Transformationsalgorithmus für jedes Entwurfsmuster definiert. Dadurch wird die Anwendung vom Entwurfsmuster parametrisiert. Anschließend wurde das Modellierungsbeispiel für den Entwurfmuster „Login“ dargestellt.

Der nächste Schritt für Integration des Entwurfsmuster ist eine Kette von Modell-zu-Modell Transformationen, die in der vorliegenden Arbeit als „UWE2UWE Pattern Transformation“ genannt ist und in ATL (Atlas Transformation Language) implementiert ist. Dieser Schritt wird im folgenden Kapitel 5 ausführlich erklärt.

5 Lösungsansatz für die Integration des Entwurfsmusters mittels UWE2UWE Pattern Transformationen

Dieser Kapitel befasst sich mit dem zweiten Teil der Integration von Entwurfsmuster – Model-zu-Modell Transformationskette, die in ATL implementiert ist. Dadurch werden das kombinierte MDUWE-Modell für die Web-Anwendung und kombinierten Modellen für Pattern in ein plattformunabhängiges kombiniertes MDUWE-Modell fusioniert, das weiter transformiert werden kann, um ein plattformspezifisches Modell zu erzeugen.

In den nächsten Abschnitten werden kurz der Aufbau einer ATL-Transformation, Tipps und Tricks für die Implementierung in ATL und die Beispiele für die Entwurfsmuster „Login“ und „Registrierung“ vorgestellt.

5.1 Aufbau ATL-Transformationen

Die Transformationen zwischen verschiedenen Modellen werden in der ATLAS Transformation Language (ATL) definiert, wie aus ein oder mehreren Quellmodellen ein Zielmodell erzeugt werden kann. In ATL werden die Transformationen selbst als Modelle angesehen, da deren abstrakte Syntax auch auf einem mit der ATL-Engine verankerten Metamodell basiert.

Das ATL–Programm wird in [15] als Modul bezeichnet.

Laut nach [16] gehören zum einen ATL - Modul (Mt):

• **ein Header–Bereich** mit der Name des Transformation-Moduls (Startwort ***module***) und der Angabe von Ziel- und Quellmetamodellen. Header-Bereich ist zwingend erforderlich, wobei der ATL–Dateiname und der Modulname übereinstimmt muss.

module module_name;
create output_models [**from**|**refines**] input_models;

• **ein Import–Bereich** dient für die Transformation notwendige Bibliotheken einzubinden (Startwort ***uses***).

uses extensionless_library_file_name;

• **Hilfsfunktionen** beschreiben globale Variablen und Funktionen und sind mit Java-Methoden vergleichbar. Die Hilfsfunktionen werden aus den Regeln aufgerufen, können auch sich selber aufrufen. Die Hilfsfunktionen weisen folgende Struktur auf:
Das Startwort ist ***helper*** und die Funktion endet mit einem Semikolon.
Das Schlüsselwort ***context*** ist optional und definiert die Megaklasse, auf die sich der Helper bezieht.
Der Funktionsname wird nach dem Schlüsselwort ***def :*** mit der Menge von Parametern geschrieben, wobei jedes Parameter von einem Paar (Parameter-Name, Parameter-Typ) identifiziert wird. Der obligatorische Rückgabetyp kommt gleich danach zwischen einem Doppelpunkt und einem Gleichheitszeichen.

helper context Integer **def**: max(x:Integer): Integer=...;

• **Regeln** bestimmen die Transformation zwischen Quell– und Zielmodellelementen und beginnen mit dem Schlüsselwort ***rule*** gefolgt von dem Namen der Regel. In dem Implementierungsteil werden in dem Abschnitt ***from*** ein Element aus Quellmodell definiert und in den ***to***-Abschnitt entsprechenden Element aus Zielmodell erzeugt. Optional können auch in den ***using***-Abschnitt die lokale Variable definiert werden. Außerdem in dem ***do***-Abschitt (auch optional) kann eine weitere Folge von der imperativen Anweisungen implementiert werden. Darüber hinaus können die Hilfsfunktionen und andere Regeln aufgerufen werden. Die Wertzuweisung erfolgt mit dem ← Operator

Nachfolgendes Beispiel demonstriert die Struktur eines ATL–Modules:

```
module modulname;
create OUT : outputmodell from IN : inputmodel;
uses libraryname;

helper context Metamodel!Element def : helpername() : return
                type= ...;

rule rulename {
  from in : Metamodel!Element
  using {
                        local variable: Metamodel!Element ..
                }
  to out : Metamodel!Element (
        attribute1 <- in.attribute1,
        attribute2 <- in.helpername() )
        do {
        statements
        }
}
```

5.2 Ein Paar Besonderheit bei der Programmierung in ATL

Da die Transformationen in ATL unidirektional sind (siehe [19]), kann während der Ausführung des ATL-Modules in den Quellmodellen nur navigiert werden, deren Elemente entweder unverändert bleiben oder komplett gelöscht. Währenddessen werden die Elemente in die Zielmodelle gleich bleibend oder modifiziert übernommen. Die bidirektionalen Transformationen können als Paare unidirektionaler Transformationsmodule - jeweils eine Transformationsspezifikation für jede Richtung - definiert werden.

Ein ATL-Modul wird aus den Regeln zusammengesetzt, die definieren, wie die Elemente aus Quellmodell verglichen und navigiert werden und die Elemente aus dem Zielmodell initialisiert und erzeugt werden. Wie es schon im Abschnitt 2.5 erwähnt wurde, ist ATL eine hybride Sprache, die Konzepte imperativer und deklarativer Programmierung vereint. Die verschiedene Programmierungsparadigmen sind durch zwei verschiedene Regeltypen unterstützt: Matched rules (deklarative Programmierung) und Called rules (imperative Programmierung).

Durch die Matched rules werden zu transformierende Elemente aus dem Quelmodell ausgesucht und in die entsprechenden Elemente in dem Zielmodell ungewandelt oder einfach kopiert.

```
rule Package {
   from s : UWE!"uwe::Package" (
                  if thisModule.inElements->includes(s)
                  then
                        s.oclIsTypeOf(UWE!"uwe::Package")
                  else false
                  endif)
   to t : UWE!"uwe::Package" mapsTo s (
         name <- s.name,
         .............
         profileApplication <- s.profileApplication)
}
```

Im oberen Code-Ausschnitt werden beispielsweise alle Package-Elementen ins Zeilmodell kopiert. Das Modul „UWEModelCopy.atl“ besteht ausschließlich aus Matched rules. Da bei der Ausführung dieses Moduls alle Elemente für die Integration eines Entwurfsmusters keine Rolle spielen, müssen diese aus Quellmodell ins Zielmodell kopiert werden.

Mit dem Ansatz der Called rules ist es möglich nicht nur einzige Elemente zu erzeugen, sondern auch ein ganz neues Modell zu erstellen.

Deswegen gibt es verschiedene Wege die Entwurfsmuster zu einem vom Modellierer entworfenes Modell mit der Hilfe von ATL-Transformationen zuzufügen. Die erste Möglichkeit ist eine schon (quasi) standardisierte Form, wie z.B. das Präsentationslayout für das Anmeldeformular im Entwurfsmuster „Login“, entsprechend der vorgegebenen Parameter modifizieren und hinzufügen. Dabei werden die Modellelemente aus dem kombinierten MDUWE-Modell für das Entwurfsmuster unbenannt und ins Zielmodell kopiert. Eine andere Möglichkeit ist, während der Transformationsausführung neue Elemente zu erzeugen und entsprechend zu referenzieren. Dies ist mit dem Einsatz der imperativen Programmierung möglich, wie im Entwurfsmuster „Registrierung“ (seihe Abschnitt 5.4). Das Präsentationslayout für das Registrierungsformular wird aus dem kombinierten MDUWE-Modell für das Entwurfsmuster „Registrierung“ übernommen und zur Laufzeit durch die

neuen Modellelementen vervollständigt. Folgendes Beispiel zeigt ein Ausschnitt aus der Called rule, die einen neuen Parameter von der Prozessklasse „Register“ erzeugt.

```
rule createNewProcessParameter (as:UWE!"uwe::Property"){
        using {
        s : UWE!"uwe::process::ProcessProperty" =
                thisModule.inPElements -> any(s|s.name = 'ParameterName');
        }
        to
                pp : UWE!"uwe::process::ProcessProperty" (
                        name <- as.name,
                        visibility <- s.visibility,
                                .....
                        processClass <- s.processClass)

        do {
                        pp;
                        }
}
```

Nachdem die Transformation entsprechend in der Run–Umgebung konfiguriert ist, kann diese ausgeführt werden. Zunächst wird eine neue ATL–Konfiguration erstellt. Diese ATL–Konfiguration werden anschließend die Projektname und das ATL–Modul zugewiesen. Abschließend erfolgt eine letzte Zuordnung der Modelle und der Metamodelle mit den vorhandenen Dateien. Dabei wird auch das Metamodellformat (UWE) festgelegt.

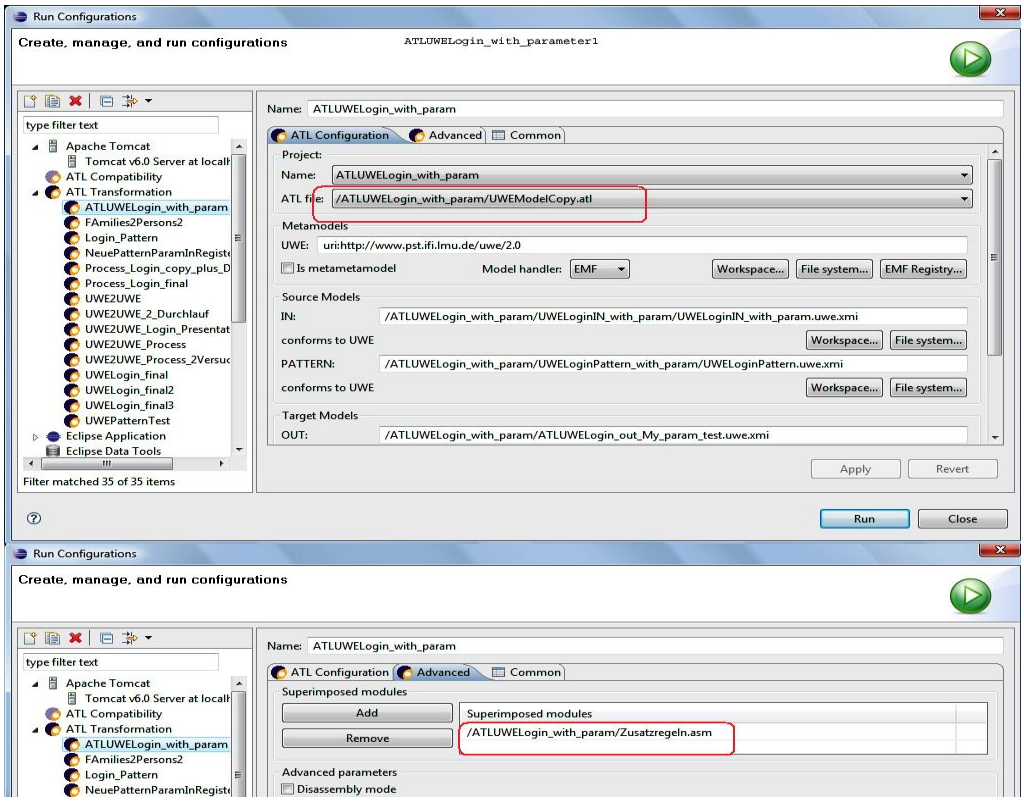

Abb. 40. ATL-Konfiguration für Beispiel „Login“

Ferner kann die Konfiguration eines ATL-Programm mehrere Module überlagert werden (auf die Abb. 40 eingekreist). Die Regeln im überlagerten Modul werden von gleichnamigen Regeln überschrieben. Das kann auch vorteilhaft sein, da die geringere Anpassungen für die größeren Module in einem neuen Modul implementiert werden können, was viel Zeit und unnötige Suche nach der zu änderten Regeln spart, besonders in Test- und Validierungsphasen der Entwicklung.

Die ersten früheren UWE2UWE Pattern Transformationen waren starr und unflexibel. Da alle zu änderten Elemente im Modell, wurden mit ihrem Namen angesprochen, wie der folgende Beispielcode zeigt.

```
rule ProcessClass {
   from s : UWE!"uwe::process::ProcessClass" (
        if thisModule.inElements->includes(s)
                and (s.name<>'LoginProcess' and s.name<>'Login')
                then
                        s.oclIsTypeOf(UWE!"uwe::process::ProcessClass")
                else false
                endif)
   to t : UWE!"uwe::process::ProcessClass" mapsTo s (
      name <- s.name,
          .....
      processActivity <- s.processActivity)
}
```

Die Transformation selber lieferte gute Ergebnissen, aber die Modellierungsteil wurde sehr stark eingeschränkt. Der Entwickler einer Web-Anwendung musste gezwungenermaßen nur festgelegten Namen für die mehreren Klassen und ihren Parameter benutzt. Mit der Entwicklung und Einführung von Pattern-Mechanismus (siehe Abschnitt 4.3) wurden die speziellen Stereotypen mit verschiedenen Parametern abhängig vom Transformationsalgorithmus gebrauch gemacht. Durch den Ansatz dieser Stereotypen wird die Modell-zu-Modell Transformationen parametrisiert, die Namensfreiheit für die Klassen und ihren Parameter und mehr Flexibilität im Modellierungsschritt gewährleistet. Die oben dargestellte Regel sieht jetzt anders aus. Die Prozessklasse „*Login*“ wird nach dem Parameter *patternName* ausgesucht, der dem eindeutigen Stereotype <<*login_process*>> entspricht:

```
rule ProcessPatternLogin {
        from a : UWE!"uwe::process::ProcessPattern"
                (a.patternName='login_process')
        using {
                s: UWE!"uwe::process::ProcessClass" = thisModule.inPElements ->
                        any(i| i.name='LoginProcess');
        }
        to t : UWE!"uwe::process::ProcessClass" (
                name <- a.name,
                visibility <- s.visibility,
                .....
                processActivity <- s.processActivity)

}
```

Die Transformationen wurden dadurch dynamischer und flexibler, wobei der Modellierer nur die minimal erforderlichen Parameter für entsprechenden Stereotyp setzen muss (siehe Abschnitte 4.4 und 4.5). Wie die Klassen selber genannt werden, sind in der letzte Versionen von ATL-Transformationen unbedeutend.

5.3 ATL-Transformation für das Entwurfsmuster „Login“

In diesem Abschnitt wird die Integration des Entwurfsmusters „Login“ in Praxis für den Transformationswerkzeug UWE4JSF (siehe Kapitel 2) erläutert. Dabei wird ein tieferer Einblick auf die spezifischen Regeln in der UWE2UWE Pattern Transformation gegeben. Die kombinierte MDUWE-Modelle für die Web-Anwendung „ATLUWELogin_with_param“ und das Entwurfsmuster „Login“ - „UWEPattern“ als Quellmodellen werden in das kombinierte MDUWE-Modell für die Web-Anwendung mit integriertem Pattern als Zielmodell „ATLUWELogin_with_param“ umgewandelt. Mit dem CD-Anhang wird den gesamten Eclipse-Projekt „ATLUWELogin_with_param“ abgegeben. Die Abb. 41 zeigt einen allgemeinen Anblick über den Transformationsablauf.

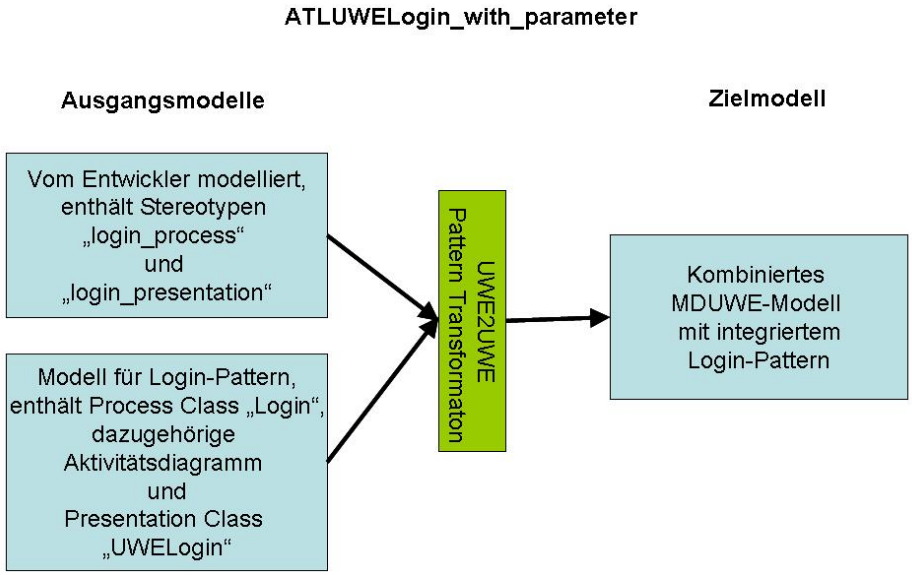

Abb. 41. :ATL-Transformation für „Login“

Für dieses Beispiel wurde die UWE2UWE Pattern Transformation nur ein ATL-Programm entwickelt, das aus zwei Modulen „UWEModelCopy.atl“ und „Zusatzregeln.atl“ besteht. Das Modul „UWEModelCopy.atl“ übernimmt das Kopieren aller Elemente von beiden Quellmodellen ins Zielmodell. Das Modul „Zusatzregeln.atl“ ist für die Integration des Patterns verantwortlich. Hier bringt die Aufteilung der Transformation in zweier Modulen den Vorteil, dass die Regeln für die Umwandlung und den Einbau nur wesentlich wenigeren Elementen im ATL-Modul „Zusatzregeln.atl“ die gleichnamigen Regeln im Modul „UWEModelCopy.atl“ überschreiben. Damit wird viel Zeit für die Implementierung und das Testen gespart.

Wie das folgende Codestück zeigt, können die Helpers (Hilfsfunktionen) sehr hilfreich sein, indem diese die globalen Variablen definieren. Zur Laufzeit werden die Helpers einmal ausgeführt und hier definierten Mengen von Elementen können direkt in verschiedenen

Regeln angesprochen werden. Andernfalls wird der gleiche Befehl in mehreren Regeln ausgeführt und die Effizient des ATL-Programmes kann darunter stark leiden. Die erste Hilfsfunktion „`inPElements`“ liefert die Menge aller Elementen aus dem Quellmodell „UWEPattern“ und die andere „`patternParameter`“ die Menge von Parameter der Klasse „Login“ mit dem Stereotyp <<*login_process*>> (siehe Abb. 37) aus dem kombinierten MDUWE-Modell für die Web-Anwendung „ATLUWELogin_with_param“.

```
helper def : inPElements : Sequence(UWE!"uwe::Element") =
                                UWE!"uwe::Element".allInstancesFrom('PATTERN');
helper def : patternParameter: Sequence(UWE!"uwe::PatternParameter") =
                                UWE!"uwe::PatternParameter".allInstancesFrom('IN');
```

Da bei der Transformation keine neuen Modellelemente erzeugt werden, beinhalten die beiden Module nur Matched rules (deklarative Programmierung). Im Anschluss werden die entscheidende für dieses Beispiel Regeln aus dem Modul „Zusatzregeln.atl“ erklärt.

```
rule Model {
   from s : UWE!"uwe::Model" (
                if thisModule.inElements->includes(s)
                and (s.name<>'UWEPattern')
                then
                        s.oclIsTypeOf(UWE!"uwe::Model")
                else false endif)
   to t : UWE!"uwe::Model" mapsTo s (
         name <- s.name,
         .....)
}
```

Die obere Regel kopiert alle Elemente „UWE!uwe::Model“ nur aus Ausgangsmodell IN (kombiniertes MDUWE-Modell für Web-Anwendung), Modell „*UWELogin*“ aus kombiniertes MDUWE-Modell für Pattern wird gelöscht. Ähnliche Regeln wurden auch für die Elemente „UWE!uwe::presentation::PresentationModel“ und „UWE!uwe::process::ProcessModel“ definiert. Mit diesen Regeln wird erzielt, dass die Struktur des Quellmodells IN wird ins Zielmodell OUT (kombiniertes MDUWE-Modell für Web-Anwendung mit integriertem Pattern) übernommen wird.

Im weiteren Codestück wird die Regel gezeigt, die Prozessklasse „*Login*“ mit dem Stereotyp <<*login_process*>> aus dem kombinierten MDUWE-Modell für die Web-Anwendung (siehe Abb. 37) in eine normale Prozessklasse umwandelt. Dabei werden die Parameter und die für den Prozessablauf entscheidende Eigenschaften von der Prozessklasse „*LoginProcess*“ und das Referenz auf das Aktivitätsdiagramm (siehe Abb. 38) aus kombiniertem MDUWE-Modell für Entwurfsmuster ins Zielmodell OUT übernommen. Andere Eigenschaften, wie `name`, `visibility`, `superClass`, und eigene Klassenparameter bleiben unverändert.

Entsprechende Regel wurde auch für die Klasse „*LoginPresentation*“ (siehe Abb. 39) und Klasse „*UWELogin*“ mit dem Stereotyp <<*login_presentation*>> aus Präsentationsmodellen beiden kombinierten MDUWE-Modellen definiert.

Diese Regeln spielen bei der Transformation eine der wichtigsten Rollen. Aus zwei Klassen verschiedenen Modellen werden expliziert nach bestimmten Eigenschaften und Parameter gesucht und diese werden ins Zielmodell übernommen. Mit anderen Wörtern: diese zwei

Klassen und ihre Bestandteile werden in einen Topf reingeschmissen, zusammengekocht und am Ende bekommt man nur eine korrekt definierte Klasse.

```
rule ProcessPatternLogin {
        from a : UWE!"uwe::process::ProcessPattern"
                (a.patternName='login_process')
        using {
                s: UWE!"uwe::process::ProcessClass" = thisModule.inPElements ->
                        any(i| i.name='LoginProcess');
        }
        to t : UWE!"uwe::process::ProcessClass" (
                name <- a.name,
                visibility <- a.visibility,
                .....
                ownedAttribute <- a.ownedAttribute->union(s.ownedAttribute),
                superClass <- a.superClass,
                processActivity <- s.processActivity)

}
```

Die nächste Regel wandelt den Patternparameter „*userNameField*“ aus der Prozessklasse „*Login*“ mit dem Stereotyp <<*login_process*>> aus dem kombinierten MDUWE-Modell für die Web-Anwendung (siehe Abb. 37) in den zutreffenden Parameter der Prozessklasse „*Login*“ zu. Während der Ausführung dieser Regel wird der Wert des Patternparameters „*userNameField*“ ausgelesen und regelrecht an entsprechenden Parameter als Name zugewiesen. Außerdem wird diesem Parameter den richtigen Typ vergeben.

```
rule PatternParameter2ownedAttribute1{
        from s: UWE!"uwe::process::ProcessProperty"
                        (thisModule.inPElements->includes(s) and  s.name='userName')
        using {
                as : UWE!"uwe::PatternParameter" =
                        thisModule.patternParameter -> any(i| i.name='userNameField');
        }
        to t: UWE!"uwe::process::ProcessProperty" (
                name <- as.value,
                .....
                type <- thisModule.inPrimElements-> any(i|i.name=s.type.name),
                .....
                processClass <- s.processClass
                )
}
```

Da bei der Entwurfsmustermodellierung die Einzelheiten über die Eigenschaften der Klasse aus Usermodell von der Web-Anwendung (in diesem Beispiel die Klasse „*User*“) noch nicht bekannt sind, müssen in den Prozessverlauf einige Zentrale Puffer-Knoten (Central Buffer Node) den richtigen Typ zugewiesen werden. Hier wird der Wert des Patternparameters „*userClass*“ ausgelesen und an entsprechenden Zentrale Puffer-Knoten zugeteilt werden. Dies ist in folgender Regel implementiert.

```
rule CentralBufferNodeUser {
```

```
    from s : UWE!"uwe::CentralBufferNode" (
                    if thisModule.inPElements->includes(s) and s.name='user'
                    then
                            s.oclIsTypeOf(UWE!"uwe::CentralBufferNode")
                    else false
                    endif)
    to t : UWE!"uwe::CentralBufferNode" (
            name <- s.name,
            ..........................................................
            owner <- s.owner,
            outgoing <- s.outgoing,
            incoming <- s.incoming,
             type <- thisModule.inElements-> any(i|i.name= (thisModule.patternParameter ->
                    any(i| i.name='userClass')).value and i.oclIsKindOf(UWE!"uwe::Class")),
            ..........................................................)
}
```

Schließlich werden die Patternparameter von der Prozessklasse mit Stereotype *<<login_process>>* aus IN gelöscht. Durch die zusätzliche Modell-zu-Modell Transformationen können weitere Entwurfsmuster ins kombinierte MDUWE-Modell integriert werden. Deswegen werden die Patternparameter von anderen Entwurfsmustern unverändert ins Zielmodell kopiert. Dies wird in der nachstehenden Regel definiert.

```
rule PatternParameter {
    from s : UWE!"uwe::PatternParameter" (
                    thisModule.patternParameter->includes(s)
                    and s.name<>'userNameField' and s.name<>'userPassField'
                    and s.name<>'userClass'
    )
    to t : UWE!"uwe::PatternParameter" mapsTo s (
            name <- s.name.debug('rule PatternParameter'),
            value <- s.value)
}
```

Das Modul „Zusatzregeln.atl“ enthält mehrere Regeln, die einen ähnlichen Aufbau haben und für die Umwandlung von den restlichen Modellelementen zuständig sind. Im Anhang A.A werden alle Regel aus diesem Modul aufgelistet. Ferner kann der gesamte Code für dieses Modul und „UWEModelCopy.atl“ im Projekt „ATLUWELogin_with_param“ auf CD-Anhang angeschaut werden.

5.4 ATL-Transformation für das Entwurfsmuster „Registrierung“

Der Beschreibung von dem Transformationsteil für das Entwurfsmuster „Registrierung“ widmet sich dieser Abschnitt. Anders als beim Beispiel „Login“ wurde hier das ATL-Programm mit dem Ansatz beiden Programmierungsparadigmen (deklarative und imperative) implementiert. Mit der Modellierung von der Prozessklasse mit dem Stereotyp <<register_process>> (siehe Abb. 42) werden die minimalen Anforderungen für den Einbau des Entwurfsmusters „Registrierung“ mit den Parametern „userRegNameField“ für die

Benutzername, „userRegPassField“ für das Passwort und „userRegEMailField“ für die E-Mailadresse abgedeckt. Aber im Usermodell für die „userClass“ können weitere Parameter, z.B. für den Nachnamen und die Postadresse, bei der Modellierung einer Web-Anwendung definiert werden (siehe Abb. 44). Diese Parameter werden während der Transformationenausführung aus die „userClass“ ermittelt und in den Prozessablauf und in Präsentationsmodell zur Laufzeit eingefügt, was nur mit der imperativen Programmierungsparadigma in ATL möglich ist.

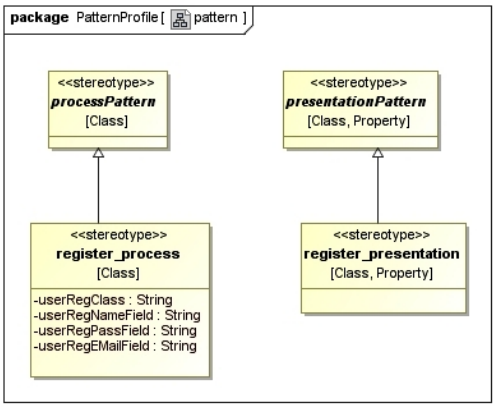

Abb. 42. Patternprofil für „Registrierung“

Die UWE2UWE Pattern Transformation für Integration dieses Entwurfsmuster ist in zwei ein nach einander auszuführenden ATL-Transformationen „ATLUWERegister_with_Parameter_1_Durchlauf“ und „ATLUWERegister_with_Parameter_Final_Durchlauf“ aufgeteilt. Die Abb. 43 stellt einen allgemeinen Überblick über den Transformationsablauf dar.

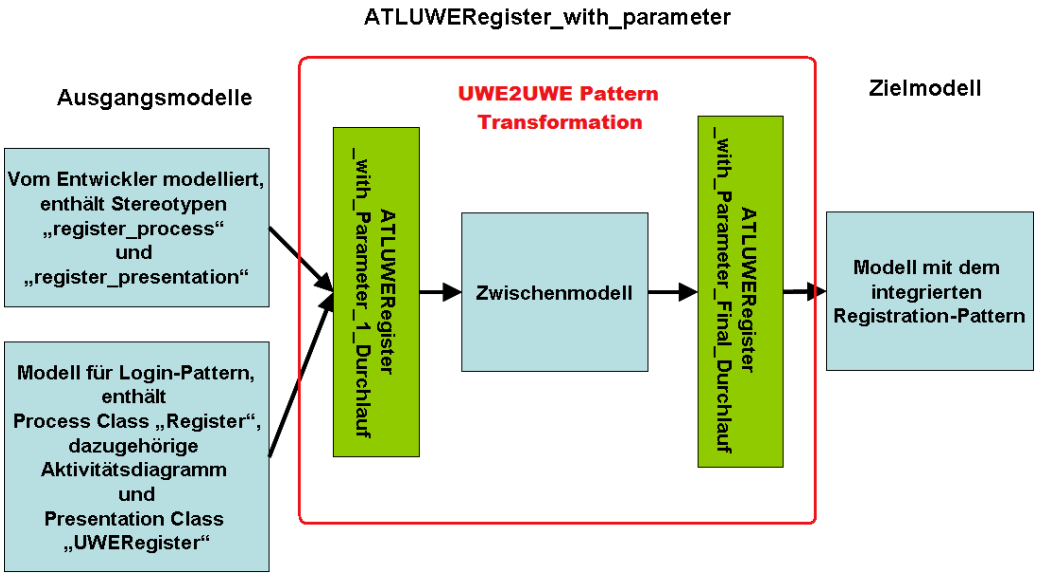

Abb. 43. ATL-Transformation für „Registrierung“

Das kombinierte MDUWE-Zwischenmodell „1_Durchlauf_Modellenbindung.uwe.xmi“ ist für die erste ATL-Transformation das Zielmodell und für die zweite das Quellmodell. Nach der Ausführung der beiden Transformationen werden die kombinierten MDUWE-Modelle für die Web-Anwendung „ATLUWERegister_with_param“ und das Entwurfsmuster „Registrierung“ - „UWEPattern“ als Quellmodellen ins das kombinierte MDUWE-Modell für die Web-Anwendung mit integriertem Pattern als Zielmodell

„ATLUWERegister_with_param“ umgewandelt. Auf dem CD-Anhang wird den Eclipse-Projekt „ATLUWERegister_with_param_ohne_Login“ vollständig gespeichert.

Jede von den beiden ATL-Transformationen bestehen aus zwei Modulen: „UWEModelCopy.atl“ und jeweils „Zusatzregeln_Register.atl“ und „Zusatzregeln_Register_final.atl“. Wie im vorherigen Beispiel übernimmt das Modul „UWEModelCopy.atl“ in jeder ATL-Transformation das Kopieren aller Elemente von den Quellmodellen ins Zielmodell. Für Änderungen, Erzeugen und Löschen von den Modellelementen wurden die Module „Zusatzregeln_Register.atl“ und „Zusatzregeln_Register_final.atl“ entwickelt. Da Machted rules (deklaratives Programmierungsparadigma) haben in diesem Beispiel die identische Aufbaustruktur und den gleichen Lösungsansatz wie in dem oben beschriebenen Beispiel für das Entwurfsmuster „Login“, werden in diesem Abschnitt nur einzelne Called rules (imperatives Programmierungsparadigma) und die Hilfsfunktionen Helpers erklärt.

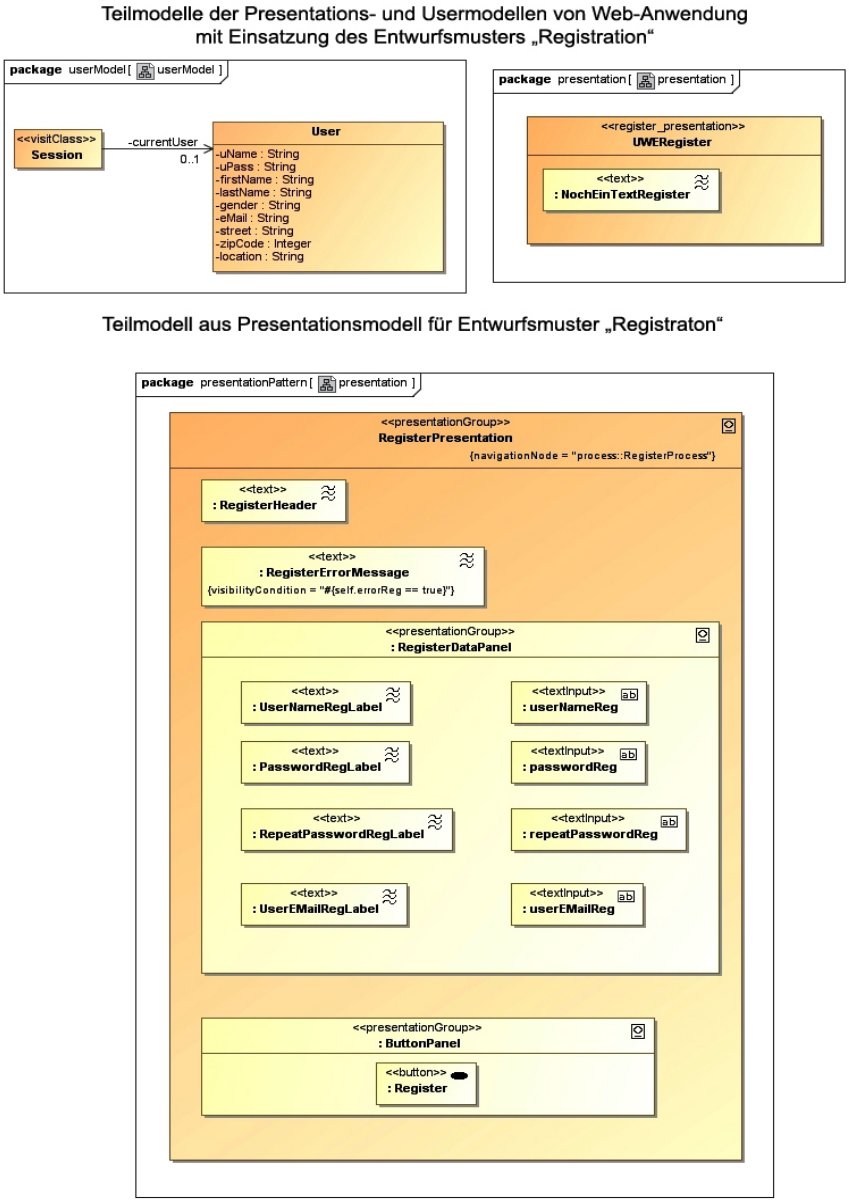

Abb. 44. Teilmodellen aus den beiden aufbereiteten Modellen

Wie schon früher erwähnt wurde, können vom Entwickler einer Web-Anwendung für die Klasse „User“ aus UserModell mehrere Parameter vorgesehen werden (siehe Abb. 44), um die weiteren Daten als Benutzername, Passwort und E-Mailadresse vom Benutzer zu definieren. Dennoch erfüllen diese drei Parameter die minimalen Anforderungen für die Integration des „Registrierung“ - Entwurfsmusters. Deswegen wurde das Basismodell dieses Entwurfsmusters dementsprechend nur auf Verwendung dieser drei Parameter aufgebaut. Um die Prozessklasse, Prozessablauf und Präsentationslayout während der Transformationen zu erweitern, wurden in ATL-Modulen „Zusatzregeln_Register.atl“ und „Zusatzregeln_Register_final.atl“ Called rules verwendet. Durch die Anwendung von der Called rules werden zur Laufzeit neue Modellelemente erzeugt und entsprechend des Geschäftsprozess „Registrierung“ im Zielmodell OUT (kombiniertes MDUWE-Modell für die Web-Anwendung mit integriertem Pattern) eingebaut.

Wie auf Abb. 44 gezeigt ist, besitzt die Klasse „User“ mehr Parameter als in Präsentationslayout vom Entwurfsmuster vormodelliert wurde. Für weitere Parameter „firstName“, „lastName“, „gender“, „street“, „zipCode“ und „location“ müssen in der PresentationGroup „RegisterDataPanel“ in der Präsentationsmodell von OUT jeweils gleichnamige "TextInput“-Elemente eingebaut werden. Das geschieht durch die Anwendung von dem folgenden Helper „addPresentationRegisterPanelTextInput()“ und der Regel „createNewPresentationRegisterDataPanelTextInput (b)“.

Helper „addPresentationRegisterPanelTextInput()“ iteriert über die Menge von einzubauenden Elementen und für jedes Element ruft die Regel „createNewPresentationRegisterDataPanelTextInput (b)“ auf.

```
helper def : addPresentationRegisterPanelTextInput() :
                Sequence(UWE!"uwe::presentation::TextInput") =
        let tis: Sequence(UWE!"uwe::Property") =
                                thisModule.inUserClassPropertyOhnePP.debug('tis')
        in
            tis->collect(ti | thisModule.createNewPresentationRegisterDataPanelTextInput(ti));
```

Mit der folgenden Regel wird ein neues "TextInput"-Element für die PresentationGroup "RegisterDataPanel" erzeugt. Das "TextInput"-Element entspricht einem Parameter von der Klasse „User“ aus UserModell, der nicht im Profil "process_register" definiert ist.

```
rule createNewPresentationRegisterDataPanelTextInput (b:UWE!"uwe::Property"){
        using {
                s : UWE!"uwe::presentation::TextInput" =
                        thisModule.inPTextInput -> any(s|s.name = 'repeatPasswordReg');
        }
        to ti : UWE!"uwe::presentation::TextInput" (
                name <- b.name,
                valueExpression <- b.name,
                qualifiedName <- s.qualifiedName,
                .....
                )
        do {
                        ti.debug('TextInput: ');
        }
}
```

Als Ergebnis werden für jeden Parameter der Klasse „User“ die Eingabenfelder im Registrierungsformular in den PresentationGroup „RegisterDataPanel“ im OUT definiert. (siehe Abb. 45)

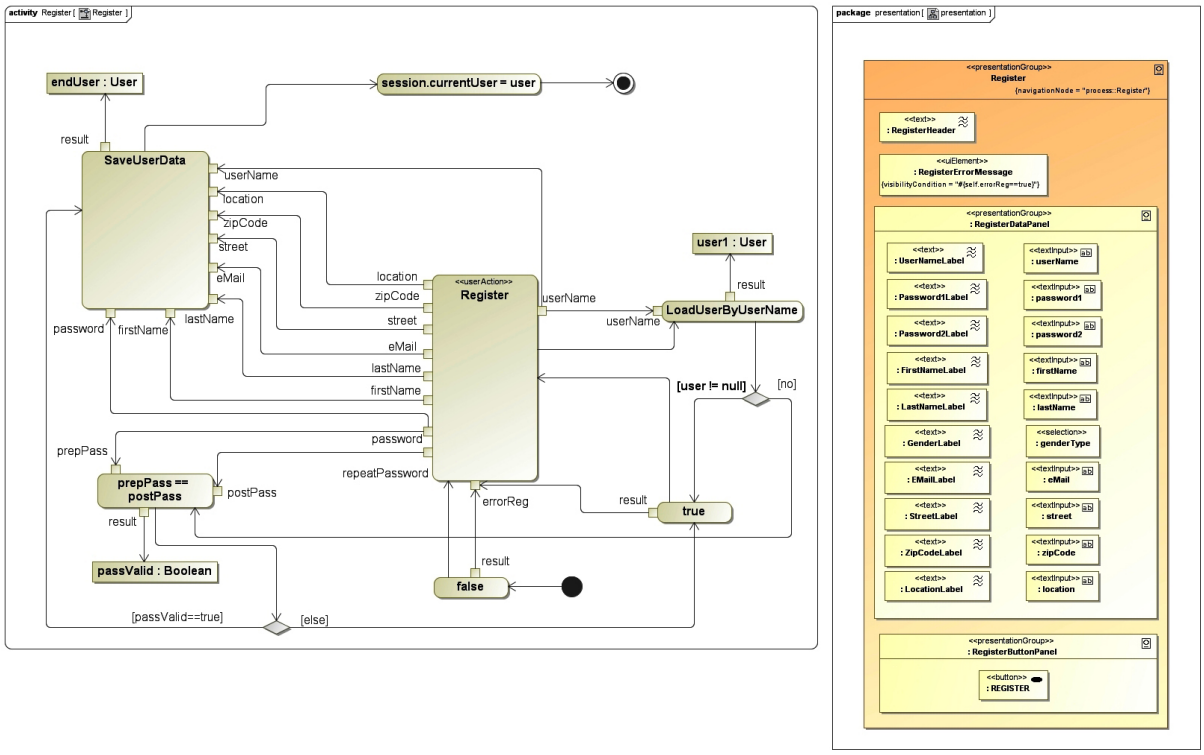

Abb. 45. Teilmodelle von Zielmodell OUT in UML

Ähnlich werden die Regeln für das Erzeugen und den Einbau von den "Text"-Elementen in den PresentationGroup "RegisterDataPanel", fehlenden Parameter in der Prozessklasse „Register“, Input Pins in <<userAction>> „Register“ und Output Pins im Systemaktion „SaveUserData“ implementiert. Die Abb. 45 stellt das vollständige Aktivitätsdiagramm und Präsentationslayout von OUT in UML dar.

Die ATL-Transformationen sind unidirektionale Transformationen und können nur write-only Zielmodelle erzeugen, d.h. in den Quellmodellen kann während der Ausführung der Transformation nur navigiert, aber nichts geändert werden und während in Zielmodellen nicht navigiert werden kann, aber die Änderungen vorgenommen werden können. Deswegen wurde die zweite ATL-Transformation implementiert, das Modul „Zusatzregeln_Register_final.atl“ beinhaltet. Die Hauptaufgabe dieses Modul besteht darin, dass die neue gleich genannte Input Pins in <<userAction>> „Register“ und Output Pins Systemaktion „SaveUserData“ zu verbinden (siehe Abb. 45). Dies ist durch die Ausführung der anschließenden Regeln möglich.

Folgende Regel überschreibt die gleiche Regel für Elemente "Activity" im Modul "UWEModelCopy.atl". Alle Activity-Elementen werden in das Zielmodell übernommen. Ferner werden die neuen "ObjectFlow"-Elemente hinzugefügt.

```
rule Activity {
   from s : UWE!"uwe::Activity" (thisModule.inElements->includes(s))
   to t : UWE!"uwe::Activity" mapsTo s (
         name <- s.name,
         .....................................................
```

```
        edge <- s.edge -> union(thisModule.addObjectFlows)
}
```

Helper „addObjectFlows()“ iteriert über die Menge von einzubauenden Elementen und für jedes Element ruft die Regel „createObjectFlow(n: UWE!"uwe::OutputPin")“ auf.

```
helper def: addObjectFlows() : Sequence(UWE!"uwe::ObjectFlow") =
        let ops: Sequence(UWE!"uwe::OutputPin") =
                thisModule.inOutPins.debug('ops')
        in
          ops->collect(op | thisModule.createObjectFlow(op));
```

Mit der folgenden Regel wird ein neues "ObjectFlow"-Element für die Aktivität "Register" erzeugt. Das neue "ObjectFlow"-Element verbindet den Paar von gleichnamigen Output Pin(<<userAction>> Register) und Input Pin(<<CallBehaviorAction>> SaveUserRegData) zusammen.

```
rule createObjectFlow(n: UWE!"uwe::OutputPin") {
   using{
        k : UWE!"uwe::InputPin" = (thisModule.inInPins ->
                                   any(ipel|ipel.name = n.name));
   }
   to t : UWE!"uwe::ObjectFlow"(
        name <- '',
        source <- n.debug('OutputPin'),
        target <- k.debug('InputPin'),
        weight <- thisModule.createLiteralInteger()
                )
        do{
                t.debug('verbunden');
        }

}
```

Nach der Ausführung von der UWE2UWE Pattern Transformation für das Entwurfsmuster „Registrierung“ wird das vollständige kombinierte MDUWE-Modell für die Web-Anwendung mit integriertem Pattern erzeugt, das nach der anschließenden Validierung und weitere Transformationen in ein plattformspezifisches Modell umgewandelt. Mittels weiteren Transformationen im Werkzeug UWE4JSF kann eine JSF-Anwendung gebildet werden.

5.5 Ergebnisse

In diesem Kapitel wurde der Integrationsschritt für die Entwurfsmuster in der Web-Anwendung mittels Modell-zu-Modell Transformationen für das Transformationswerkzeug UWE4JSF vorgestellt. Eine Kette von Modell-zu-Modell Transformationen, die in der vorliegenden Arbeit als „UWE2UWE Pattern Transformation“ genannt ist, ist in ATL (Atlas Transformation Language) implementiert. ATL ist eine hybride Sprache, die Konzepte imperativer und deklarativer Programmierung vereint. Mittels deklarativen Sprachkonstrukten können die Modellelemente aus Quellmodellen ins Zielmodell einfach kopiert, gelöscht oder modifiziert übernommen. Mit der Hilfe von imperativen

Sprachkonstrukten können zur Laufzeit der Transformation neuer Elemente erzeugt und entsprechend den Regeln referenziert werden, wie im Beispiel von der Integration des Entwurfsmuster „Registrierung“.

ATL ist eine Transformationssprache, deren Regeln unidirektional sind und keine Kopplung der Modelle unterstützt. Die Modellelemente aus den Zielmodellen können geändert werden, aber es kann nicht während der Transformationsausführung in den Zielmodellen navigiert werden. Aus diesem Grund kann „UWE2UWE Pattern Transformation“ aus mehreren ein nach einander Teiltransformationen bestehen.

Die beiden Beispiele für die Integration der Entwurfsmuster „Login“ und „Registrierung“ in der Web-Anwendung illustrieren nicht die Implementierung der Modell-zu-Modell Transformationen, sondern zeigen auch, dass die Entwicklung von speziellen Komponentenbibliotheken für den Einsatz von Analyse-Pattern definitiv möglich. Dadurch lässt sich die Modellierung für Fälle, in denen wiederkehrende Muster auftreten, stark vereinfachen.

6 Fazit und Ausblick

Das Erkennen und Einteilen in Muster und Strukturen liegt in der Natur des Menschen, wodurch die Auffassung und die Verarbeitung von Informationen vereinfacht werden. Weiterhin erleichtert es die Ordnung innerhalb bestimmter Gebiete, welche durch verschiedene Sprachen, differenzierte Sichtweisen und unterschiedliche Hintergründe sehr weitläufig sein können.

Entwurfsmuster bilden die Möglichkeit, Entwicklungszeiten zu verkürzen und die Qualität von Softwarelösungen zu steigen, indem sie etablierte Verfahren als wieder verwendbare Vorlagen verfügbar machen. Sie führen eine übergreifende Sprache zwischen den unterschiedlichen Beteiligten des Erstellungsprozesses von Software ein und erlauben eine gemeinsame Nutzung von Wissen über die Grenzen einer spezifischen Problemlösung oder Zielplattform hinaus.

Diese Arbeit befasst sich unter anderen mit der Darstellung und Beschreibung verschiedenen Analyse-Pattern, die für Web-Anwendungen relevant sind. Für die Prüfung deren Einsatzmöglichkeit in der Praxis wurden MDUWE als Modellierungssprache, ATL als Transformationssprache ausgewählt. Ferner wurde in Rahmen dieser Diplomarbeit ein Pattern-Mechanismus definiert und für die Modellierung des Pattern-Profils angewendet. Dadurch wurden die betroffenen Klassen in Context-, Prozess- und Präsentationsmodellen parametrisiert und während der Modell-zu-Modell Transformationen das Patternmodell in das Quellmodell korrekt integriert. Die Einsetzung der Implementierung durch eine Transformationsspezifikation in ATL ermöglicht eine einfache Anpassung der Transformationsregeln und der durch sie erzeugten Modelle, an die jeweiligen Bedürfnisse der Benutzer.

Die vorliegende Arbeit zeigt, dass sich der Ansatz des Entwurfsmusters unter die Kombination aus MDUWE und ATL als durchaus einsatzfähig erwiesen hat. Die Beispiele im Kapitel 5.3 und 5.4 demonstrieren nicht nur die Implementierung der vorgestellten Transformationen, sondern zeigen mit den vorliegenden Modellen auch den Gebrauch den Analyse-Pattern im UWE. Auf diese Weise dient die Arbeit zugleich als Vorlage für die Entwicklung weitere Analyse-Pattern für die Modellierung einer Web-Anwendung als auch zur Inspiration, sich mit den Spezifika der MDUWE-Sprache auseinander zu setzen, um den Nutzen von Entwurfsmustern weiter zu erhöhen und ihre Umsetzung zu vereinfachen.

Die Wahl von Eclipse als Basisplattform ist zweifellos auch für die zukünftigen Projekte zutreffend, da sich diese in vielen Bereichen immer mehr zum De-facto-Standard entwickelt. Besonders trifft es gerade für die Themen Modellierung und modellgetriebene Softwareentwicklung zu. Leider ist die Performance der in ATL Modelltransformationen notwendigerweise verbesserungswürdig. Die ATL-Transformationen für größere Projekte können in der Praxis definitiv problematisch sein, wenn vor allem bei der häufigen Änderungen im Präsentationsmodell, die sofort überprüft werden müssen. Zurzeit befindet sich eine optimierte Version der virtuellen Maschine von ATL in der Entwicklung, die effizienter sein wird.

Ein konkreter Ansatzpunkt für tatsächliche Weiterentwicklungen ist dagegen die Entwicklung von weiteren Analyse-Patterns für MDUWE. In Abschnitten 4.5 und Kapitel 5 wurde die Vorgehensweise für die Modellierung des Patterns und die ATL-Transformationen für seine Integration anhand Beispiele vorgestellt.

Andere Möglichkeiten zur Erweiterung und Verbesserung ergeben sich durch den modularen Aufbau von MDUWE und UWE4JSF. Das umfasst zum einen die Entwicklung von speziellen Komponentenbibliotheken für den Einsatz im konkreten Präsentationsmodell. Auf diese Art könnte das konkrete Präsentationsmodell durch vorgefertigte Elementkonfigurationen ergänzt werden, die sich in modernen UML-Tools wie MagicDraw zu Modulen zusammenfassen lassen. Dadurch lässt sich die Modellierung für die Fälle, in denen wiederkehrende Muster auftreten, stark vereinfachen.

Referenzen

[1] Ch. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, *A Pattern Language*, Oxford University Press, New York 1977.

[2] A Pattern Library for Interaction Design (http://www.welie.com/patterns/index.php)

[3] K. Eilebrecht, G. Starke. *Patterns kompakt. Entwurfsmuster für effektive Software – Entwicklung,* Spektrum-Akademischer Vlg, September 2006

[4] Designing Interfaces: Patterns for Effective Interaction Design (http://designinginterfaces.com/)

[5] WebPatterns.org (http://webpatterns.org/)

[6] Yahoo! Design Pattern Library (http://developer.yahoo.com/ypatterns/index.php)

[7] M. Mahemoff. *Ajax Design Patterns: Creating Web 2.0 Sites with Programming and Usability Patterns*, Beijing u. a., O"Reilly, 2006

[8] UI-Patterns.com (http://ui-patterns.com/)

[9] R. S. Pressman, *Software Engineering: A Practitioner's Approach,* McGraw-Hill, 2004

[10] E. Gamma, J. Vlissides, R. Helm, R. Johnson. *Entwurfmuster: Elemente wiederverwendbarer objektorientierter Software,* Addison-Wesley, 2004

[11] G. Kappel, B. Pröll, S. Reich, W. Retschitzegger. *Web Engineering- systematische Entwicklung von Webanwendungen,* Addison-Wesley, 2006

[12] M. Weiss. *More Patterns for Web Applications,* Carleton University, Ottawa, Canada. (http://www.scs.carleton.ca/~weiss/papers/europlop05-a2.pdf)

[13] A. Knapp, N. Koch, F. Moser, G. Zhang. *ArgoUWE: A Case Tool for Web Applications*, First Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE´03), 2003.

[14] Atlas Transformation Language (ATL), http://www.eclipse.org/m2m/atl/

[15] ATLAS group: ATL: ATLAS Transformation Language. User Manual 0.6, 2005

[16] ATLAS group: ATL: ATLAS Transformation Language. Specification of the ATL VM 0.1, 2005

[17] A. Kraus, A. Knapp, N. Koch, *Model-Driven Generation of Web Applications in UWE*, In Proc. MDWE 2007 - 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS, Vol 261, 2007.

[18] A. Kraus, *Model Driven Software Engineering for Web Applications,* Dissertation, Ludwig-Maximilians-Universität München), 2007.

[19] S. Buckl, *Modell-basierte Transformationen von Informationsmodellen zum Management von Anwendungslandschaften* , Diplomarbeit, Technische Universität München, 2005

[20] J. Miller, J. Mukerji, *MDA Guide Version 1.0.1*, OMG, 2003

[21] E. B. Fernandez, *Building systems using analysis patterns*, In Proceedings of the third international Workshop on Software Architecture, Seiten 37-40, Orlando, FL, 1998. ACM.

[22] M. Hahsler, *Analyse Patterns im Softwareentwicklungsprozeß*, Dissertation, Abteilung für Informationswirtschaft Wirtschaftsuniversität Wien, 2001

[23] P. Roßbach, T. Stahl, W. Neuhaus, *Grundlegende Konzepte und Einordnung der Model Driven Architecture (MDA)*, Artikel in Jaxenter, 2003

[24] D. Gebauer, B. Scherrer, M. Kühl, K. Müller-Glaser, *Verfahren zurAufstellung formaler Metamodelle,* Artikel, Forschungszentrum Informatik FZI, Elektronische Systeme und Mikrosysteme (ESM), 2006

[25] OMG's MetaObject Facility, http://www.omg.org/mof.

[26] OMG, XML Metadata Interchange (XMI), formal/2007-12-01

[27] Eclipse, www.eclipse.org

[28] Object Management Group – Unified Modeling Language (UML), http://www.uml.org

[29] UWE – UML-based Web Engineering, http://www.pst.ifi.lmu.de/projekte/uwe

[30] N. Koch, A. Knapp, G. Zhang, and H. Baumeister, *UML-based Web Engineering: An Approach based on Standards* (book chapter). In *Web Engineering: Modelling and Implementing Web Applications*. Gustavo Rossi, Oscar Pastor, Daniel Schwabe and Luis Olsina (Eds.), chapter 7, 157-191, ©Springer, HCI, 2008.

[31] Ch. Kroiß, *Modellbasierte Generierung von Web-Anwendungen mit UWE (UML-based Web Engineering),* Diplomarbeit, LMU, 2008

[32] MagicDraw, http://www.magicdraw.com/

[33] Object-Graph Notation Language, http://www.ognl.org.

[34] JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces

[35] N. Koch und Ch. Kroiß, *UWE Metamodel and Profile,* User Guide and Reference, Version 1.0 - February 2008

7 Anhang.

A. Transformationsregeln für das Entwurfsmuster “Login”.

In folgende Tabelle werden alle Regeln aus dem Modul „Zusatzregeln.atl“ dargestellt. Mit der Hilfe von diesem ATL-Modul wird der Entwurfsmuster „Login“ in dem ursprünglichen Modell nach der Transformation nicht nur eingebaut, sondern auch entsprechend eingegebenen Werten von Parametern modifiziert.

Der ATL-Modul „UWEModelCopy.atl“ ist für die Übernahme allen Elementen aus den Quellmodellen verantwortlich, die nicht geändert werden müssen.

ATLUWELogin_with_parameter	ATL-Transformation für die Analyse-Pattern „Login“
Kurze Beschreibung	Diese Transformation generiert aus der erste Quellmodell, von Entwickler selber in UWE mit den Stereotypen <<login_process>> und „login_presentation“ und dem Modellierungsvorschlag entsprechend eingegeben Parametern modelliert, und der zweite Modell, Login-Patternmodell, die Zielmodell mit dem Presentation Class in der Präsentationsteilmodell und der Process Class und Aktivitäten in dem Prozessteilmodell. Diese Aktivitäten beschreiben den Ablauf des Prozess „Login“.
Metamodell	Das Ausgangs- und Zielmodelle haben das gleiche Metamodell: UWE-Metamodell (http://www.pst.ifi.lmu.de/uwe/2.0).
ATL-Modul: UWEModelCopy.atl	Kopiert alle Elemente aus beiden Ausgangsmodellen.
ATL-Modul: Zusatzregeln.atl	Ändert, kopiert, löscht und fasst diejenigen Elemente aus der Quellmodellen zusammen, die bei der Anwendung des Login-Patterns zu verwandeln sind.

Tranformationsregeln aus dem ATL-Modul „Zusatzregeln.atl“		
Regelname		Auswirkung
Model	☞	Alle Modell-Elemente werden nur aus Ausgangsmodell (IN) kopiert, Modell „UWELogin“ aus Login-Pattern wird gelöscht.
Processmodel	☞	Kopiert Prozessmodell nur aus IN; Prozessmodell aus PATTERN wird gelöscht
PresentationModel	☞	Kopiert Präsentationsmodell nur aus IN; Präsentationsmodell aus PATTERN wird gelöscht
Activity InitialNode ControlFlow ObjectFlow DecisionNode InputPin CallBehaviorAction ActivityFinalNode LiteralInteger OpaqueExpression Text Button	☞	Diese Regeln für Elemente mit den Typen „Activity“, „InitialNode“, „ControlFlow“, „ObjectFlow“, „DecisionNode“, „InputPin“, „CallBehaviorAction“, „ActivityFinalNode“, „LiteralInteger“, „OpaqueExpression“, „Text“ und „Button“ überschreiben die gleichen Regeln im Modul „UWEModelCopy.atl“, da alle Typen von Elementen aus Patternmodell modifiziert werden müssen. Im diesen Fall werden die oben genannten Elementtypen neu referenziert (PrimitiveType-Elemente werden nur aus IN-Modell übernommen). Alle Elemente werden in das Zielmodell übernommen.
OutputPin	☞	Alle andere Elemente „OutputPin“ aus den beiden Quellmodellen, außer 2 OutputPins mit Namen „password“ und „userName“ in den „Login“-Activity aus PATTERN werden in das Zielmodell übernommen.
OutputPinProcessPatternLogin1 OutputPinProcessPatternLogin2	☞	Diese Regel weist den Elementen „Outpin“ aus PATTERN in den „Login“-Activity, der aus <<userAction>> „LoginProzess“ ausgeht, die Werte der Parametern „userNameField“ und „userPassField“ vom Stereotyp <<login_process>> aus dem IN zu. Modifizierte Elemente werden in das Zielmodell übernommen.
TextInput	☞	Alle Elemente „TextInput“ aus den beiden Quellmodellen, außer 2 TextInputs mit Namen „password“ und „userName“ in den „LoginDataPanel“-PresentationGroup aus dem Präsentationsteilmodell des PATTERNs werden in das Zielmodell übernommen.
TextInputLoginUserName	☞	Diese Regel weist den Elementen

		„userName“ in den „LoginDataPanel“-PresentationGroup aus dem Präsentationsteilmodell des PATTERNs die Werte des Parameters „userPassField“ und „userNameField“ vom Stereotyp „login_prozess“ aus IN zu. Modefizierte Elemente werden in das Zielmodell übernommen.
PrimitiveType	☞	Diese Regel löscht redundante Elemente „PrimitiveType“ aus PATTERN.
ProcessProperty	☞	Mit dieser Regel alle Elemente des Types „ProcessProperty“ aus IN werden in OUT (Zielmodell) übernommen.
ProcessPropertyPType	☞	Allen Parametern aus Prozessteilmodell aus PATTERN mit Primitiven Typen, außer „userName“ und „password“, werden Primitive Typen aus IN zugewiesen
CentralBufferNode	☞	Alle Elemente „CentralBufferNode“ aus den beiden Quellmodellen, außer 2 CentralBufferNode mit Namen „loginValid“ und „user“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.
CentralBufferNodeLoginValid	☞	Diese Regel weist den Element „CentralBufferNode“ mit Namen „loginValid“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs den PrimitiveType „Boolean“ aus IN zu, da alle PrimitiveTypes aus PATTERN gelöscht sind. Modifiziertes Element wird in das Zielmodell übernommen.
CentralBufferNodeUser	☞	Diese Regel weist den Element „CentralBufferNode“ mit Namen „user“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs den Type des Classes aus UserModell aus IN zu, indem den Wert des Parameters „userClassField“ vom Stereotyp <<login_process>> aus IN mit der OCL-Ausdruck gelesen wird. Modifiziertes Element wird in das Zielmodell übernommen.
UserAction	☞	Alle Elemente „UserAction“ aus den beiden Quellmodellen, außer UserAction mit Namen „LoginProzess“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.

UserActionLoginProcess	☞	Mit dieser Regel wird den Element „UserAction“ mit Namen „LoginProzess“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs umbenannt und wird genaso heisen, wie Stereotyp <<login_process>> aus Prozessteilmodell aus IN. Modifiziertes Element wird in das Zielmodell übernommen.
OpaqueAction	☞	Alle Elemente „OpaqueAction“ aus den beiden Quellmodellen, außer UserAction mit Namen „passKontrol“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.
OpaqueActionLogin	☞	Mit dieser Regel wird den Element „OpaqueAction“ mit Namen „passKontrol“ in den „Login“-Activity aus dem Prozessteilmodell des PATTERNs umbenannt und neuen angepassten Body-Ausdrück zugewiesen. Modifiziertes Element wird in das Zielmodell übernommen.
PatternParameter2-ownedAttribute1 PatternParameter2-ownedAttribute2	☞	Patternparameter „userNameField“ und „userPassField“ aus IN werden in ownedAttribute für ProcessClass in OUT umgewandeln
ProcessPattern	☞	Alle Prozess Pattern aus beiden Modellen, außer Prozess Pattern mit dem Stereotyp <<login_process>> aus Prozessteilmodell aus IN, werden in OUT übernommen.
ProcessPatternLogin	☞	Prozess Pattern mit dem Stereotyp <<login_process>> aus Prozessteilmodell aus IN wird in Prozess Class in OUT umgewandelt.
PresentationGroup	☞	Alle Presentation Groups aus beiden Modellen, außer Presentation Group mit dem Namen „LoginPresentation“ aus Präsentationsteilmodell aus PATTERN, werden in OUT übernommen.
PresentationPattern	☞	Alle Presentation Pattern aus IN, außer Presentation Pattern mit dem Namen „login_presentation“ aus Präsentationsteilmodell aus IN, werden in OUT übernommen.
PresentationPatternUWELogin	☞	Presentation Pattern mit dem Namen „login_presentation“ aus Präsentationsteilmodell aus IN wird in neue PresentationGroup umgewandelt. Dabei werden

		alle Elementen aus PresentationGroup „LoginPresentation“ aus Präsentationsteilmodell aus PATTERN und aus Presentation Pattern mit dem Namen „login_presentation“ aus Präsentationsteilmodell aus IN in neue PresentationGroup in OUT zusammengefasst. PresentationGroup „LoginPresentation“ aus Präsentationsteilmodell aus PATTERN wird ins OUT nicht übernommen.
PatternParameter	☞ ☞	Patternparameter von <<login_process>> aus IN werden damit gelöscht.

B. Transformationsregeln für das Entwurfsmuster “Registrierung”.

Für die Integration des Analyse-Pattern „Registrierung“ besteht ATL-Transformationen aus zwei Teiltransformationen, die eine nach einander ausgeführt werden (siehe Kapitel 5.4): „ATLUWERegister_ohne_Login_with_Parameter_1_Durchlauf“ und „ATLUWERegister_ohne_Login_with_Parameter_Final_Durchlauf“.

In folgende Tabelle werden alle Regeln aus dem Modul „Zusatzregeln_Register.atl“ dargestellt. Mit der Hilfe von diesem ATL-Modul wird der Entwurfsmuster „Registartion“ in dem ursprünglichen Modell nach der Transformation „ATLUWERegister_ohne_Login_with_Parameter_1_Durchlauf“ nicht nur eingebaut, sondern auch entsprechend Parameterisierungsansatz modifiziert.

Der ATL-Modul „UWEModelCopy.atl“ ist für die Übernahme allen Elementen aus den Quellmodellen verantwortlich, die nicht geändert werden müssen.

ATLUWERegister_ohne_Login_ with_Parameter_1_Durchlauf	Erste ATL-Transformation für die Analyse-Pattern „Registrierung“
Kurze Beschreibung	Diese Transformation generiert aus der erste Quellmodell, von Entwickler selber in UWE mit den Stereotypen „register_process“ und „register_presentation“ und dem Modellierungsvorschlag entsprechend eingegeben Parametern modelliert, und der zweite Modell, Regiatration-Patternmodell, die Zielmodell mit dem Presentation Class in der Präsentationsteilmodell und der Process Class und Aktivitäten in dem Prozessteilmodell. Diese Aktivitäten beschreiben den Ablauf des Prozess „Registrierung“.
Metamodell	Das Ausgangs- und Zielmodelle haben das gleiche Metamodell: UWE-Metamodell (http://www.pst.ifi.lmu.de/uwe/2.0).

ATL-Modul: UWEModelCopy.atl	Kopiert alle Elemente aus beiden Ausgangsmodellen.
ATL-Modul: Zusatzregeln_Register.atl	Ändert, kopiert, löscht und fasst diejenigen Elemente aus der Quellmodellen zusammen, die bei der Anwendung des Register-Patterns zu verwandeln sind.

Tranformationsregeln aus dem ATL-Modul „Zusatzregeln_Register.atl“		
Regelname		Auswirkung
Model	☞	Alle Modell-Elemente werden nur aus Ausgangsmodell (IN) kopiert, Modell „UWEPattern“ aus Registrierung-Pattern wird gelöscht.
Processmodel	☞	Kopiert Prozessmodell nur aus IN; Prozessmodell aus PATTERN wird gelöscht
PresentationModel	☞	Kopiert Präsentationsmodell nur aus IN; Präsentationsmodell aus PATTERN wird gelöscht
Activity InitialNode ControlFlow ObjectFlow DecisionNode InputPin CallBehaviorAction ActivityFinalNode LiteralInteger OpaqueExpression Text Button	☞	Diese Regeln für Elemente mit den Typen „Activity“, „InitialNode“, „ControlFlow“, „ObjectFlow“, „DecisionNode“, „InputPin“, „CallBehaviorAction“, „ActivityFinalNode“, „LiteralInteger“, „OpaqueExpression“, „Text“ und „Button“ überschreiben die gleichen Regeln im Modul „UWEModelCopy.atl“, da alle Typen von Elementen aus Patternmodell modifiziert werden müssen. Im diesen Fall werden die oben genannten Elementtypen neu referenziert (PrimitiveType-Elemente werden nur aus IN-Modell übernommen) Alle Elemente werden in das Zielmodell übernommen.
OutputPinPattern	☞	Alle andere Elemente „OutputPin“ aus den beiden Quellmodellen, außer 3 OutputPins mit Namen "passwordReg", "userEMailReg" und "userNameReg" in den „Register“-Activity aus PATTERN werden in das Zielmodell übernommen.
OutputPinProcessRegPattern1 OutputPinProcessRegPattern2 OutputPinProcessRegPattern3	☞	Diese Regel weist den Elementen „Outpin“ aus PATTERN in den „Register“-Activity, der aus <<userAction>> „Register“ ausgeht, die Werte der Parametern „userNameField“, „userEMailReg“ und „userPassField“ vom Stereotyp „register_prozess“ aus dem IN zu. Modifizierte Elemente werden in das Zielmodell übernommen.
TextInput	☞	Alle Elemente „TextInput“ aus den beiden Quellmodellen, außer 3 TextInputs mit Namen

		"passwordReg", "userEMailReg" und "userNameReg" in den "RegiaterDataPanel"-PresentationGroup aus dem Präsentationsteilmodell des PATTERNs werden in das Zielmodell übernommen.
TextInputUserNameReg TextInputUserRegPass TextInputUserEMailReg	☞	Diese Regel weist den Elementen „TextInput“ mit mit Namen "passwordReg", "userEMailReg" und "userNameReg" in den "RegiaterDataPanel"-PresentationGroup aus dem Präsentationsteilmodell des PATTERNs die Werte des Parameters „userRegPassField“, „userRegNameField“ und „userRegEMailField“ vom Stereotyp „register_prozess“ aus IN zu. Modifizierte Elemente werden in das Zielmodell übernommen.
PrimitiveType	☞	Diese Regel löscht redundante Elemente „PrimitiveType“ aus PATTERN.
ProcessProperty	☞	Mit dieser Regel alle Elemente mit dem Type „ProcessProperty“ aus IN werden in OUT (Zielmodell) übernommen.
ProcessPropertyPType	☞	Allen Parametern aus Prozessteilmodell aus PATTERN mit Primitiven Typen, außer außer "userNameReg", 'userEMailReg' und "passwordReg", werden Primitive Typen aus IN zugewiesen
CentralBufferNode	☞	Alle Elemente „CentralBufferNode“ aus den beiden Quellmodellen, außer 3 CentralBufferNode mit Namen "passValid", "endUser" und "preUser" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.
CentralBufferNode-RegPassValid	☞	Diese Regel weist den Element „CentralBufferNode“ mit Namen "passValid" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs den PrimitiveType "Boolean" aus IN zu, da alle PrimitiveTypes aus PATTERN gelöscht sind. Modifiziertes Element wird in das Zielmodell übernommen.
CentralBufferNodePreUser	☞	Diese Regel weist den Element "CentralBufferNode" mit Namen "preUser" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs den Type des Classes aus UserModell aus IN zu, indem den Wert des Parameters "userRegClass" vom

		Stereotyp "register_prozess" aus IN mit der OCL-Ausdruck gelesen wird. Modifiziertes Element wird in das Zielmodell übernommen.
CentralBufferNodeEndUser	☞	Diese Regel weist den Element "CentralBufferNode" mit Namen "endUser" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs den Type des Classes aus UserModell aus IN zu, indem den Wert des Parameters "userRegClass" vom Stereotyp "register_prozess" aus IN mit der OCL-Ausdruck gelesen wird. Modifiziertes Element wird in das Zielmodell übernommen.
UserAction	☞	Alle Elemente "UserAction" aus den beiden Quellmodellen, außer UserAction mit Namen "RegisterProzess" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.
UserActionRegProcess	☞	Mit dieser Regel wird den Element "UserAction" mit Namen "RegisterProzess" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs umbenannt und wird genaso heisen, wie Stereotyp "register_process" aus Prozessteilmodell aus IN. Modifiziertes Element wird in das Zielmodell übernommen.
OpaqueAction	☞	Alle Elemente "OpaqueAction" aus den beiden Quellmodellen, außer UserAction mit Namen "RegisterPassControl" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs werden in das Zielmodell übernommen.
OpaqueAction-RegisterPassControl	☞	Mit dieser Regel wird den Element "OpaqueAction" mit Namen "RegisterPassKontrol" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs umbenannt und neuen angepassten Body-Ausdrück zugewiesen. Modifiziertes Element wird in das Zielmodell übernommen.
OpaqueActionRegister-SessionCurrentUser	☞	Mit dieser Regel wird den Element "OpaqueAction" mit Namen "sessionCurrentUser" in den "Register"-Activity aus dem Prozessteilmodell des PATTERNs umbenannt und neuen angepassten Body-Ausdrück zugewiesen. Modifiziertes Element wird in das Zielmodell übernommen.
RegPatternParameter2-ownedAttribute1	☞	Patternparameter "userRegNameField", "userRegPassField" und

RegPatternParameter2-ownedAttribute2 RegPatternParameter2-ownedAttribute3		„userRegEMilField“ aus IN werden in ownedAttribute für ProcessClass in OUT umgewandelt
ProcessPattern	☞	Alle Prozess Pattern aus beiden Modellen, außer Prozess Pattern mit dem Stereotyp „register_process“ aus Prozessteilmodell aus IN, werden in OUT übernommen.
ProcessPatternRegister2-ProcessPattern	☞	Prozess Pattern mit dem Stereotyp "register_process" aus Prozessteilmodell aus IN wird als Prozess Pattern mit umgewandelten Parameter in OUT übernommen.
PresentationGroup	☞	Alle Presentation Groups aus beiden Modellen, außer Presentation Groups mit dem Namen Namen "RegisterPresentation" und "RegisterDataPanel" aus Präsentationsteilmodell aus PATTERN, werden in OUT übernommen.
PresentationGroup-RegisterDataPanel	☞	Zu Presentation Group mit dem Namen "RegisterDataPanel" aus Präsentationsteilmodell aus PATTERN, werden neue Elementen hinzugefügt und in OUT übernommen.
PresentationPattern	☞	Alle Presentation Pattern aus IN, außer Presentation Pattern mit dem Namen "register_presentation" aus Präsentationsteilmodell aus IN, werden in OUT übernommen.
PresentationPattern-UWERegister	☞	Presentation Pattern mit dem Namen "register_presentation" aus Präsentationsteilmodell aus IN wird in neue PresentationGroup umgewandelt. Dabei werden alle Elementen aus PresentationGroup "RegisterPresentation" aus Präsentationsteilmodell aus PATTERN und aus Presentation Pattern mit dem Namen "register_presentation" aus Präsentationsteilmodell aus IN in neue PresentationGroup in OUT zusammengefasst. PresentationGroup "RegisterPresentation" aus Präsentationsteilmodell aus PATTERN wird ins OUT nicht übernommen.
addProcessParameters()	☞	Alle neu erzeugte Parameter werden zu den ProcessClass „Register“ hinzugefügt.
createNewProcessParameter (as:UWE!"uwe::Property")	☞	Mit dieser Regel wird ein neuer Parameter für die ProcessClass "Register" erzeugt. Der neue

	Parameter entspricht einem Parameter des Klasses aus UserModell, der nicht im Profil "process_register" definiert ist.
addPresentationRegister-PanelTextInput()	☞ Alle neu erzeugte "TextInput"-Elemente werden zu den PresentationGroup "Register" hinzugefügt.
createNewPresentationRegister-DataPanelTextInput (b:UWE!"uwe::Property")	☞ Mit dieser Regel wird ein neuen "TextInput"-Element für die PresentationGroup "Register" erzeugt. Das "TextInput"-Element entspricht einem Parameter des Klasses aus UserModell, der nicht im Profil "process_register" definiert ist.
addPresentationRegister-PanelText()	☞ Alle neu erzeugte "Text"-Elemente werden zu den PresentationGroup "Register" hinzugefügt.
createNewPresentation-RegisterDataPanelText (as:UWE!"uwe::Property")	☞ Mit dieser Regel wird ein neuen "Text"-Element für die PresentationGroup "Register" erzeugt. Das "Text"-Element entspricht einem Parameter des Klasses aus UserModell, der nicht im Profil "process_register" definiert ist.
addRegisterUserAction-OutputPin()	☞ Alle neu erzeugte "OutpinPin"-Elemente werden zu den <<userAction>> "Register" hinzugefügt.
createNewOutputPin (as:UWE!"uwe::Property")	☞ Mit dieser Regel wird ein neuen "OutpinPin"-Element für die <<userAction>> "Register" erzeugt. Das neue "OutpinPin"-Element entspricht einem Parameter des Klasses aus UserModell, der nicht im Profil "process_register" definiert ist.
addRegisterSaveUserData-InputPin()	☞ Alle neu erzeugte "InpinPin"-Elemente werden zu den <<CallBehaviorAction>> "SaveUserData" hinzugefügt.
createNewInputPin (as:UWE!"uwe::Property")	☞ Mit dieser Regel wird ein neuen "InpinPin"-Element für die <<CallBehaviorAction>> "SaveUserData" erzeugt. Das neue "InpinPin"-Element entspricht einem Parameter des Klasses aus UserModell, der nicht im Profil "process_register" definiert ist.

In folgende Tabelle werden alle Regeln aus dem Modul „Zusatzregeln_Register_final.atl“ dargestellt. Mit der Hilfe von diesem ATL-Modul wird der Entwurfsmuster „Registartion“ in dem Zwischenmodell nach der Transformation „ATLUWERegister_ohne_Login_with_Parameter_Final_Durchlauf“ das Zielmodell vervollständigt.

Der ATL-Modul „UWEModelCopy.atl“ ist für die Übernahme allen Elementen aus den Quellmodellen verantwortlich, die nicht geändert werden müssen.

ATLUWERegister_ohne_Login_with_Parameter_Final_Durchlauf	Final ATL-Transformation für die Analyse-Pattern „Registrierung“
Kurze Beschreibung	Diese Transformation erzeugt und referenziert die fehlenden Elementen in Zwischenmodell und vervollständigt das Zielmodell.
Metamodell	Das Ausgangs- und Zielmodelle haben das gleiche Metamodell: UWE-Metamodell (http://www.pst.ifi.lmu.de/uwe/2.0).
ATL-Modul: UWEModelCopy.atl	Kopiert alle Elemente aus beiden Ausgangsmodellen.
ATL-Modul: Zusatzregeln_Register_Final.atl	Kopiert und referenziert neue erzeugte Elemente (Inputs- und Outputpins) aus der Zwischenmodell, die noch zu verbinden sind.

Tranformationsregeln aus dem ATL-Modul „Zusatzregeln_Register_Fianl.atl“	
Regelname	Auswirkung
Activity	☞ Diese Regel für Elemente mit den Typen „Activity“ überschreibt die gleiche Regel im Modul „UWEModelCopy.atl“, da bestimmte ObjectsFlow – Elementen werden erzeugt und referenziert. Alle Elemente werden in das Zielmodell übernommen.
ProcessPattern	☞ Alle Prozess Pattern aus beiden Modellen, außer Prozess Pattern mit dem Stereotyp „register_process“ aus Prozessteilmodell aus IN, werden in OUT übernommen.
ProcessPatternRegister2ProcessClass	☞ Prozess Pattern mit dem Stereotyp <<login_process>> aus Prozessteilmodell aus IN wird in Prozess Class in OUT umgewandelt.
PatternParameter	☞ Patternparameter von „register_process“ aus IN werden damit gelöscht.
addObjectFlows()	☞ Mit dieser Regel werden neu erzeugte Elementen mit dem Type "ObjectFlow" den Paar OutputPin(<<userAction>>, Register) und InputPin (<<CallBehaviorAction>>, SaveUserRegData) zusammengebunden.
createObjectFlow	☞ Mit dieser Regel wird ein neuen

(n:UWE!"uwe::OutputPin")		"ObjectFlow"-Element für die Aktivität "Register" erzeugt. Das neue "ObjectFlow"-Element verbindet den Paar OutputPin(<<userAction>>, Register) und InputPin(<<CallBehaviorAction>>, SaveUserRegData) zusammen.
createLiteralInteger()	☞	Mit der Regel wird für den neuen "ObjectFlow"-Element den Eingenschaft "weight" erzeugt.