Institut für Informatik
Ludwig-Maximilians-Universität München


Diplomarbeit

# Modellbasierte Anforderungsanalyse für die Entwicklung von adaptiven RIAs

**Sergej Kozuruba**

Aufgabensteller: Prof. Dr. Martin Wirsing

Betreuer: Dr. Nora Koch

Abgabedatum: 30. September 2010

## Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quelle nund Hilfsmittel angegeben, wowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

München, den 30. September 2010

.....

# Zusammenfassung

UML-based Web Engineering ist ein Ansatz zur Modellierung und Entwicklung von Webanwendungen mit dem Ziel den gesamten Entwicklungsprozess abzudecken. Jedoch fehlten bisher Elemente zur Modellierung der frühen Phase der Anforderungsanalyse, wodurch bereits zu diesem Zeitpunkt ein detailierteres Modell einer Webanwendung festgehalten werden könnte. Zudem würde eine detailreichere Anforderungsanalyse es ermöglichen die nachfolgenden Modelle durch Modelltransformationen zumindest teilweise daraus automatisch zu generieren.

Daher soll zunächst innerhalb dieser Diplomarbeit eine Webanwendung eines sozialen Netzwerks, welche den Benutzern das Abspeichern und den Austausch von Seitenfavoriten ermöglicht mit dem Entwicklungstool MagicUWE erstellt und anschließend implementiert werden. Diese Anwendung soll als Beispiel dienen, um anhand der dafür erstellten Modelle die Erweiterungen zu erarbeiten.

Sowohl die Anwendungsfalldiagramme, als auch die dazugehörigen Aktivitätsdiagramme der Anforderungsanalyse sollen um geeignete Stereotypen erweitert werden, um die meisten Informationen aus dem Inhalts-, Navigations-, Prozess- und Präsentationsmodell einer Webanwendung bereits zu dieser Phase modellieren zu können. Außerdem soll die Erweiterung im Besonderen die Modellierung von Adaptivität und von Rich Internet Applications während der Anforderungsanalyse ermöglichen.

Schließlich soll das Plugin MagicUWE des für die Modellierung verwendeten Entwicklungstools MagicDraw erweitert werden, um die Modellierung der neuen Stereotypen damit zu ermöglichen und die neu entwickelten Modelltransformationen anwenden zu können. Diese Modelltransformationen sollen zum einen aus den erweiterten Modellen der Anforderungsanalyse das automatische Erzeugen des Inhalts, der Navigationstruktur, der Prozesse und des Präsentationsmodells ermöglichen. Außerdem soll man mit diesen Transformationen auch die Diagramme der vier Modelle um einzelne aus der Anforderungsanalyse gewonnene Elemente erweitert können.

# Inhaltsverzeichnis

# 1 Einleitung

In den letzten paar Jahrzehnten hat das Internet sich von einer kaum beachteten Informationsplattform zur Quelle von unzähligen Webseiten und Webanwendungen entwickelt. Durch die immer höhere Verfügbarkeit und Akzeptanz als auch durch die leistungsfähigere Soft- und Hardware stehen diese Webanwendungen heute den klassischen Desktopanwendungen in kaum etwas nach. Diese Komplexität spiegelt sich jedoch auch bei der Entwicklung solcher Anwendungen wieder, weshalb für diese seit einiger Zeit Methoden entwickelt werden, um diesem Umstand Herr zu werden.

Einer dieser Ansätze ist das UML-base Web Engineering (UWE). Wie der Name schon sagt, basiert dieser auf der Unified Modelling Language(UML), wodurch sich einige Vorteile für die Entwicklung ergeben. Dabei wird der Entwicklungsprozess in UWE nach dem Prinzip der Trennung der Verantwortungsbereiche in unterschiedliche Diagramme unterteilt und dank des modellgetriebenen Ansatzes findet die Transformation von Modellen zwischen den Entwicklungsphasen besondere Beachtung.

Da UWE wegen der sich schnell verändernden Technik auf dem Gebiet der Webanwendungen Raum für Erweiterungen bietet und wahrscheinlich immer bieten wird, beschäftigt sich diese Arbeit ebenfalls damit UWE in einem kleinen Teilgebiet zu erweitern. Dieses erste Kapitel beschäftigt sich dabei die Motivation und die daraus resultierende Erweiterung zu skizzieren. Zudem liefert der letzte Abschnitt dieses Kapitels eine kurze Übersicht über dieses Dokument.

## 1.1 Problemstellung

Auf dem derzeitigen Entwicklungsstand verfügt UWE über viele gut durchdachte Möglichkeiten eine Webanwendung zu modellieren. Die vier Modelle (Inhalt, Navigation, Prozesse und Präsentation) erlauben es den Entwicklern detailiert verschiedene Sichten auf die Anwendung darzustellen. Sogar Eigenschaften von Rich Internet Applications lassen sich

in das Präsentationsmodell einbinden und auch wenn die Modellierung von Adaptivität noch nicht in ein Modellierungstool integriert wurde, so lässt sich diese ebenfalls in UWE darstellen.

Im ersten Schritt der Erstellung einer Webanwendung gibt es jedoch zur Zeit noch Mängel. So verfügt UWE über keine speziellen Möglichkeiten um alle notwendigen Informationen bereits in der Anforderungsanalyse festzuhalten. Da hier zur Modellierung nur gewöhnliches UML mit seinen Anwendungsfall- und Aktivitätsdiagrammen bereitsteht, wird der Entwurf der Webanwendung in dieser Phase ungenau oder sehr kompliziert. Fehler oder Ungenauigkeiten, die sich in dieser Phase der Entwicklung einschleichen führen jedoch bei der späteren Modellierung oft zu größeren Problemen, wie zum Beispiel einer falscher Umsetzung der Anforderungen.

Außerdem wird bei UWE auf eine automatische Gewinnung von Modellen durch Modelltransformationen Wert gelegt. Dies verlangt nach einer Möglichkeit bereits aus der Anforderungsanalyse die Modelle der Entwurfsphase zu generieren. Für solche Transformationen benötigt man natürlich Regeln, wie aus den Elementen der Anforderungsanalyse die nachfolgenden Modelle und ihre Inhalte erzeugt werden. Jedoch können Transformationsregeln nur die bereits vorhandenen Informationen umsetzen, aber nichts Neues mehr hinzufügen. Daher liefern diese nur dann ein sinnvolles Ergebnis, wenn auch die Anforderungsanalyse ausreichend genug modelliert wurde, was wiederum eine Anpassung der zu dieser Phase zur Verfügung stehenden Modellierungselemente erfordert.

## 1.2 Lösungsansatz

Im Rahmen dieser Arbeit wurde nun versucht die vorhin beschriebenen Mängel zu beseitigen. Dazu wurde anhand einer Webanwendung sowohl das Profil von UWE um neue Stereotypen erweitert, als auch die Funktionen von MagicUWE mit neuen Optionen zur Modelltransformation angereichert.

Die dafür verwendete Webanwendung bekam den Namen Philoponella, welche ein soziales Netzwerk zum Speichern und zum Austausch von Informationen zu Internetadressen (auch Favoriten genannt) ermöglicht. Zusätzlich verfügt Philoponella über adaptive Elemente und über Eigenschaften von Webanwendungen, welche als Rich Internet Applications bezeichnet werden. Anhand der für ihren Entwurf verwendeten Modelle konnten dann die weiteren Schritte leichter entwickelt und getestet werden. Außerdem eignet sich die Anwendung auch dafür die Veränderungen, welche innerhalb dieser Arbeit an UWE vorgenommen wurden und ihren Einsatz besser zu demonstrieren.

Basierend auf den Modellen der Beispielanwendung wurde daraufhin das Profil von UWE erweitert, um die innerhalb der Einleitung beschriebenen Probleme bei der Modellierung

der Anforderungsanalyse zu beheben. Dabei wurde dort, wo es von Nutzen war für die verschiedenen Elemente der Anforderungsanalyse neue Stereotypen eingeführt, was auf der Ebene der groben Anforderungsanalyse die Anwendungsfälle betrifft, während es in den Aktivitätsdiagrammen neue Stereotypen für Aktionen und Ein- und Ausgabepins ergibt.

In einem weiteren Schritt wurde es durch die verbesserte Anforderungsanalyse ermöglicht eine Reihe von Transformationsregeln entwickelt und als ein Plugin des Entwicklungstools MagicDraw integriert, durch welche es automatisch und ohne Zutun eines Entwicklers möglich ist aus dem anfänglichen Modell alle weiteren Modelle abzuleiten. Diese ermöglichen es einem sowohl zu Beginn der Entwicklung die ersten Modelle komplett zu erzeugen, als auch bereits vorhandene Modelle durch Informationen aus der Anforderungsanalyse zu ergänzen.

## 1.3 Aufbau der Arbeit

Nachdem in diesem Kapitel eine knappe Einleitung zum Thema dieser Arbeit geliefert wurde, gliedert sich der Rest wie folgt: In nächsten Kapitel folgt eine Definition und Einführung von Begriffen und Konzepten, welche für diese Arbeit von Bedeutung sind. Kapitel 3 widmet sich daraufhin der Beschreibung der Beispielanwendung anhand der bei ihrer Entwicklung entstandenen Modelle. Entgegengesetzt zu der normalen Vorgehensweise bei der Entwicklung einer Anwendung wird erst darauffolgend die Anforderungsanalyse von Philoponella zusammen mit den neu eingeführten Erweiterungen des UWE-Profils in Kapitel 4 beschrieben. Danach widmet sich Kapitel 5 der Beschreibung der möglichen Modelltransformationen und der automatischen Erzeugung der Modelle des Inhalts, der Navigation, der Prozesse und der Präsentation. Abschließend wird in Kapitel 6 die Umsetzung der Modelle der Beispielanwendung mit dem Google Web Toolkit kurz skizziert und im letzten Kapitel findet sich schließlich eine Zusammenfassung der Ergebnisse und ein kleiner Ausblick auf weitere Entwicklungsmöglichkeiten auf diesem Gebiet.

# 2 Grundlagen

Dieses Kapitel soll zunächst die für diese Arbeit zentralen Begriffe wie Web Engineering, UML-based Web Engineering, Rich Internet Applications sowie Adaptivität klären und diese zu charakterisieren. Womit es die Grundlagen zum Verständnis der nachfolgenden Kapitel bereitstellt.

## 2.1 Web Engineering

Das World Wide Web wurde erschaffen um das konkrete Problem des Verbreitung von Informationen zu lösen [6], jedoch eröffnete es eine neue Weg zur Kommunikation sowie eine eigene, spezielle Art von Anwendungen. So entwickelte sich das ursprünglich nur als reines Informationsmedium konzipiertes Web im Laufe der Jahre zu einer deutlich heterogenen Menge von einfachen statischen Textseiten bis hin zu komplexen, vollwertigen Informationssystemen. Damit entstand der Begriff der Web-Anwendung, welcher laut dem Buch *Web Engineering*[11] wie folgt definiert ist:

> *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, den Web-Browser, verwendet werden.*

Die stetig wachsende Komplexität führt wiederum dazu, dass die einst noch ausreichente Entwicklung in einer spontanen Art und Weise und auf individuellen Entwicklungspraktiken beruhend heute nicht mehr angemessen ist und zu umfangreichen Qualitätsmängeln bis hin zu gravierenden Problemen führen kann. Als Antwort auf diese chaotische Entwicklung von Webseiten und Applikationen sowie als eine Anerkennung der Unterschiede zwischen dieser und konventioneller Softwareentwicklung sowie anderer Disziplinen wurde das Teilgebiet des Web

Engineering ins Leben gerufen. Auch für diesen Begriff liefert das Buch *Web Engineering*[11] eine umfassende Definition :

> *(1) Web Engineering ist die Anwendung systematischer und qualifizierbarer Ansätze um Anforderungsbescchreibung, Entwurf, Implementierung, Test, Betrieb und Wartung qualitativ hochwertiger Web-Anwendungen kosteneffektiv durchführen zu können.*
>
> *(2) Web Engineering bedeutet auch die wissenschaftliche Disziplin, die sich mit der Erforschung dieser Ansätze beschäftigt.*

Die wichtigsten Grundprinzipien des Web Engineerings sind dabei klar definierte Ziele und Anforderungen, systematischer, in sorgfältig gestaltete Phasen gegliederte Entwicklung und die stetige Überwachung des gesamten Entwicklungsprozesses. Um diese Prinzipien zu gewährleisten wurden diverse Modellierungsansätze geschaffen. Diese basieren auf klassischen Methoden oder erweitern eine bestehender Modellierungssprache und lassen sich nach *Modeling Web Applications*[37] in vier Kategorien einteilen. Obwohl sich diese Arbeit ausschließlich mit dem UML-based Web Engineering, kurz UWE, beschäftigt, sind diese hier dennoch kurz erwähnt:

– **Datenorientiert:** Diese beruhen auf dem ER-Modell und konzentrieren sich auf die Modellierung von datenbanklastigen Webanwendungen. Vertreter hierfür sind das Relationship Management Methodology (RMM) und das Web Modeling Language (WebML).

– **Hypertextorientiert:** Der Fokus liegt hier auf dem Hypertext-Charakter der Anwendungen. Als Beispiele können hier das Hypertext Design Model (HDM) und das Web Site Design Method (WSDM) genannt werden.

– **Objektorientiert:** Für diese Ansätze bildet hauptsächlich UML oder OMT die Grundlage. Neben dem bereits erwähnten UML-based Web Engineering (UWE) wären hier das Object-Oriented Hypermedia Design Method (OOHDM), Object-Oriented Web Solutions (OOWS) und Object-Oriented Hypermedia (OO-H) zu erwähnen.

– **Softwareorientiert:** Hier kommen Techniken zur Anwendung, welche sich sehr stark klassischer Softwareentwicklung folgen, wofür das Web Application Extension (WAE) als Beispiel genannt werden kann.

## 2.2 UML-based Web Engineering (UWE)

Wie bereits erwähnt ist UWE ein objektorientierter Ansatz zur Modellierung und Entwicklung von Webapplikationen, welcher gegen Ende der neunziger Jahre entstanden ist, mit dem Ziel den gesamten Entwicklungsprozess einer Webanwendung beginnend mit der Anforderungsanalyse und durch alle Designmodelle hindurch abzudecken. Dabei liegt der Fokus auf graphischer Modellierung und systematischen und zumindest teilweise automatisiertem Design.

Weiterhin zeichnet sich UWE durch das Befolgen verschiedener, von der Object Management Group (OMG) formulierter Standarts aus. Nebem dem Prinzip der Model Driven Architecture (MDA) sind hier auch das Austauschformat für Modelle, XML Metadata Interchange (XMI), MetaObject Facility (MOF) als Standard für Metamodelldefinitionen, die Transformationssprache Query View Transformation (QVT) und natürlich die Extensible Markup Language (XML) als Auszeichnungssprache erwähnenswert [33].

Die offensichtlichste Eigenschaft von UWE ist aber seine Übereinstimmung mit einer der wichtigsten Modellierungssprachen, der Unified Modelling Language (UML), da zur Modellierung eine leichtgewichtige Erweiterung dieser verwendet wird. Der Grund hierfür ist neben der breiten Akzeptanz dieser Sprache in der Softwareentwicklung auch die vielseitigen Möglichkeiten, welche von UML zu seiner Erweiterung bereitgestellt werden [17]. Dazu ist das Metamodell von UWE als eine konservative Erweiterung von UML, ein so genanntes UML-Profil, welcher sich aus einer Hierarchie von Stereotypen und einer Menge von Bedingungen, sogenannten Constraints, definiert. Dies bedeutet im Fall von UWE, dass es so weit wie es möglich ist vom De-facto-Standard Gebrauch macht, was zumindest teilweise bei Anwendungsfall-, Aktivitäts- und Klassendiagrammen der Fall ist. Wo es hingegen UML an speziellen Elementen zur Modellierung von Webanwendungen mangelt, bereichert UWE dieses durch Definition von Stereotypen.

Dieses Vorgehen bietet mehere Vorteile. Zum einen haben UML-Diagramme einen hohen Bekanntheitsgrad und es ist für Entwickler, welche damit Erfahrung haben ein Leichtes ein solches, auf einem UML-Profil basierendes Modell zu verstehen, wie es bei UWE der Fall ist. Außerdem können zur Modellierung in UWE eigentlich alle CASE-Tools herangezogen werden, welche UML und ihre Erweiterungen unterstützen. Dazu ist es bereits ausreichend die Stereotypen für die Konzepten von UWE zu definieren. Einige Tools bieten auch das Einbinden von vordefinierten UML-Profilen und für einige wenige existieren zudem auch Plugins. Im Rahmen dieser Arbeit wird zum Beispiel das Plugin MagicUWE, eine Erweiterung für das CASE-Tool MagicDraw verwendet. Diese bietet neben dem einfachen Einsatz von UWE-Stereotypen auch die Möglichkeit den Entwickler durch Modelltransformationen zu unterstützen.

## 2.3 Der Entwicklungsprozess in UWE

Da sich kaum etwas so schnell wie das Web entwickelt, sollten die Modelle in jeder Entwicklungsphase einfach an relevante Neuerungen anpassbar sein. Um dieser Anforderung gerecht zu werden folgt UWE dem strikten Prinzip die Verantwortungsbereiche so gut es geht zu trennen (Separation of Concerns). Wie auch in anderen Web Engineering Ansätzen werden eigene Modelle in unterschiedlichen Phasen des Entwicklungsprozesses erstellt und dienen dazu verschiedene Bereiche (Inhalt, Navigationsstruktur und Präsentation) der Webapplikation darzustellen. Zudem wird die Modellierung ausreichend abstrakt gehalten, um eine Plattformunabhängigkeit möglichst lange in der Entwicklung zu gewährleisten.

Der grobe Entwicklungsprozess einer Webanwendung mit UWE sieht damit so aus, dass zunächst in der Anforderungsanalysephase die Anforderungen gesammelt werden. Diese bilden dann die Grundlage für die weiteren Modellierungsschritte. Darauf folgend beginnt die eigentliche Modellierung mit der Spezifikation des Inhalts, welche gleich einer klassischen Anwendung die einzelnen Konzepte bzw. Klassen und ihre Beziehungen untereinander festhält. Danach wird aus den bisher modellierten Informationen ein unabhängiges Modell der Navigationsstruktur erstellt. Dieses sogennante Navigationsmodell stellt die möglichen Navigationspfade innerhalb der Anwendung dar. Da der Benutzer einer Webanwendung in der Regel nicht nur verschiede Seiten betrachten, sondern auch an bestimmten Punkten Daten eingeben oder modifizieren will, werden diese wegen ihrer Besonderheit im Navigationsmodell gesondert dargestellt. Diese Abläufe werden als Prozesse bezeichnet und werden durch das Prozessmodell festgehalten. Schließlich folgt die Festlegung der Benutzeroberfläche im Präsentationsmodell.

Der modellgetriebene Ansatz bestärkt aber nicht nur den Einsatz von Modellen bei der Entwicklung, sondern legt auch Wert auf Transformationen innerhalb aller Entwicklungsphasen. Transformationen werden als notwendiger Bestandteil der MDA angesehen und werten die Modelle von reiner Dokumentation zum primären Werkzeug des Entwicklungsprozesses [23]. UWE befolgt diesen Ansatz indem es schrittweise die Anwendungsspezifikation durch automatisierte Transformationen und manuelle Verfeinerung aufbaut. Zu Beginn steht das Computational Independent Model (CIM). Dieses beschreibt unabhängig von allen Implementierungsdetails die Anforderungen des Systems und ist bei UWE mit dem Anforderungsmodell gleichzusetzen. Daraus entsteht dann das Platform Independent Model (PIM), welches die Software im Detail spezifiziert, aber noch keine Informationen zur Realisiertung enthält. Im Detail beteutet dies, dass zunächst das Inhaltsmodell und das Prozessmodell erzeugt werden, daraus entsteht dann das Navigationsmodell und wiederrum daraus das Präsentationsmodell. Diese funktionalen Modelle, welche unterschiedliche Sichten auf die Applikation darstellen, werden durch weitere Transformationen innerhalb des PIM

sowie durch manuelle Eingriffe der Modellierer verfeinert, bis diese zu einem einzigen großen Modell zusammengefasst werden. Neben der Validierung wird dieses benutzt um daraus dann durch Einbeziehen architektonischer Modellierungsmerkmale ein sogenanntes Integrationsmodell entstehen zu lassen, welches zu guter Letzt in das Platform Specific Model (PSM) umgewandelt wird. Das PSM enthält nun Spezifikationen zu eingesetzten Technologien und kann zu guter Letzt in Programmcode umgesetzt werden [42].

All dies basiert auf dem Konzept der Metamodellierung, wobei ein Metamodell eine genaue Definition der Elemente einer Modellierungssprache, ihre Beziehungen unter einander und die Regeln um ein syntaktisch korrektes Modell zu erstellen darstellt [23]. Wie schon erwähnt ist das Metamodell von UWE eine konservative Erweiterung des UML Metamodells, das als ein UML-Profil darstellbar ist. Weiterhin ist es nach dem Prinzip der 'Separation of Concerns' (Trennung der Verantwortungsbereiche) in Pakete unterteilt, wobei für jedes der oben erwähnten Modelle jeweils ein Paket existiert (Requirements, Content, Navigation, Process und Navigation). Eine detaillierte Betrachtung der einzelnen Pakete erfolgt in den folgenden Abschnitten, während das Bild 2.1 die Paketsruktur zeigt.

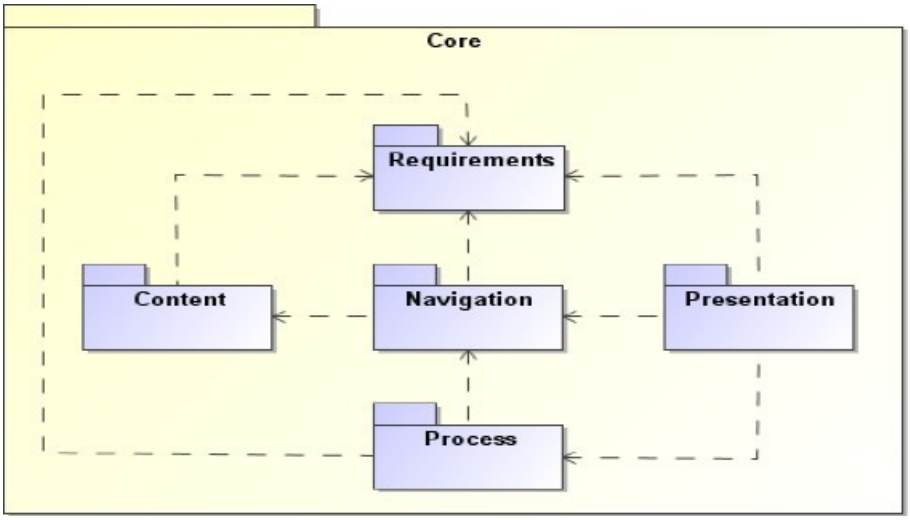


Abbildung 2.1: Paketstruktur des UWE-Modells

### 2.3.1 Anforderungsanalyse

Der erste Schritt in der Entwicklung ist das Sammeln und Festlegen der Anforderungen an die Webapplikation, welche in UWE im Anforderungsmodell festgehalten werden. Für das Web Engineering bedeutet dies die Identifikation des notwendigen Inhalts, das Festhalten der benötigten Navigation und der Geschäftsprozesse und die Definition der Interaktion für verschiedene Benutzergruppen [8].

In UWE kann dies in zwei unterschiedlichen Detailstufen festgehalten werden. Zunächst

dient das Use-Case-Diagramm dazu einen Überblick über alle Funktionalitäten der Software zu liefern. Darüber hinaus bieten Aktivitätsdiagramme die Möglichkeit einzelne Anwendungsfälle detailiert zu beschreiben.

### 2.3.2 Das Inhaltsmodell

Wie bereits kurz skizziert dient das Inhaltsmodell der Beschreibung der verschiedenen Konzepte des Systems, welche zumindest teilweise aus dem Anforderungsmodell gewonnen werden. Da es hier keine Unterschiede zwischen einer Webanwendung und konventioneller Software gibt, wird auch ein unverändertes UML-Klassendiagramm zur Darstellung verwendet und die unterschiedlichen Entitäten werden als Klassen mit allen dafür notwendigen Beziehungen wie zum Beispiel Assoziation oder Generalisierung, allen benötigten Attributen und eventuell auch Methoden dargestellt.

Allerdings ist eine Besonderheit zumindest von modernen Webapplikationen ist, dass immer mehr Wert auf ein personalisiertes Erscheinungsbild und Verhalten gelegt wird. Es hat sich als wertvoll erwiesen, diese für die Anpassung an einen Benutzer oder die Umgebung notwendigen Daten in einem gesonderten Modell zu halten. Dieses wird je nach Schwerpunkt entweder als User Model oder als Context Model bezeichnet. Eine genauere Betrachtung erfolgt im Abschnitt zur Adaptivität.

### 2.3.3 Die Navigationsstruktur

Aufbauend auf dem Inhaltsmodell und den Anforderungen entsteht das Navigationsmodell. Eine vollständig automatisierte Generierung ist hier jedoch nicht möglich, da das grobe Modell erst auf Grundlage der als für die Navigation relevant markierten Klassen aus dem Inhaltsmodell entstehen kann. Außerdem kann so aus einem einzelnen Inhaltsmodell unterschiedliche Navigationsmodelle für verschiedene Sichten erzeugt werden. Dieses repräsentiert eine statische Sicht der Navigation zwischen den Inhaltspräsentationen und Einstiegs- und Austrittspunkte von Webprozessen und wird als ein gerichteter Graph dargestellt, dessen Knoten navigierbare Punkte, an denen der Benutzer Informationen und Funktionalitäten erhält. Anders als das Inhaltsmodell gibt es hier deutliche Unterschiede zu bereits bestehenden UML-Diagrammen. Dies führt dazu, dass das Metamodell zur Navigationsstruktur mit diversen zusätzlichen Stereotypen angereichert wird, welche in Abbildung 2.2 zusammengefasst sind.

Die Basis des Metamodells stellen die beiden abstrakten Klassen *«navigationNode»* und *«link»* dar. Dabei stellt ***«navigationNode»*** zunächst einen beliebigen Navigationsknoten dar. Wie bereits erwähnt wird jedes mal wenn dieser erreicht wird dem Benutzer Informationen oder Funktionalitäten präsentiert, jedoch ist zu beachten, dass eine solche Klasse nicht zwingend

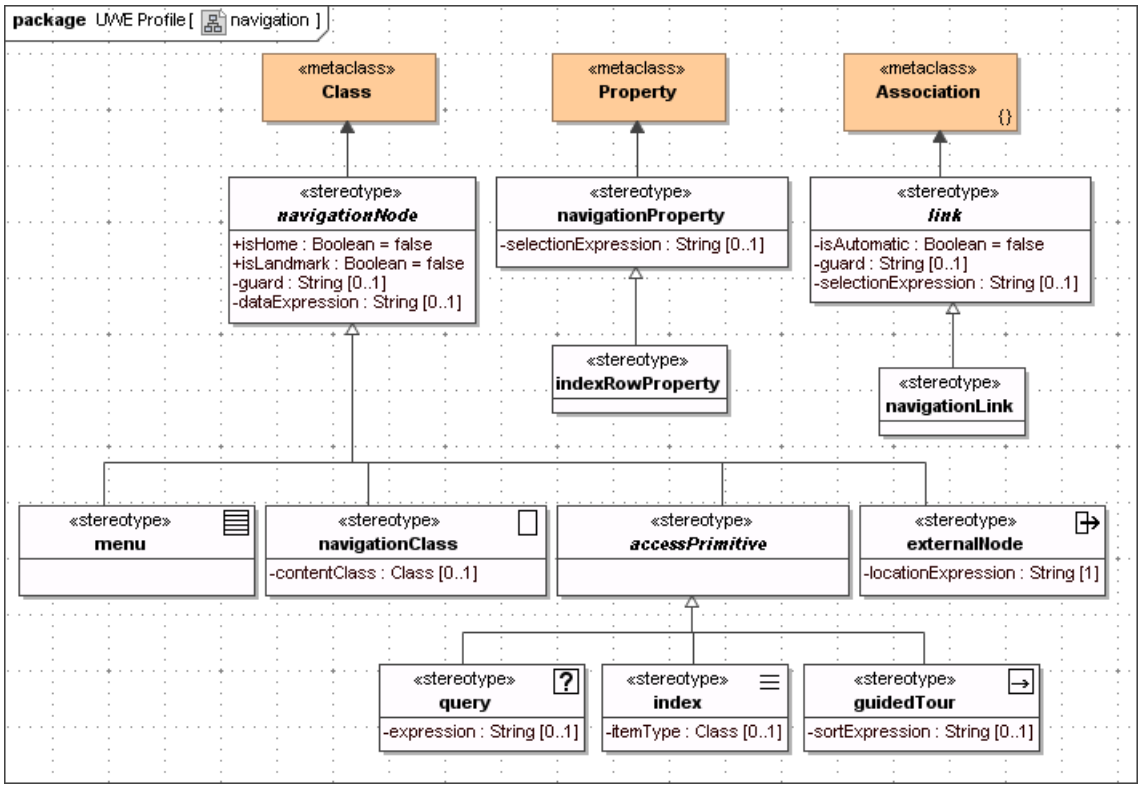


Abbildung 2.2: Metamodell der Navigationsstruktur von UWE

eine Seite der Webanwendung darstellt. ***«Link»*** hingegen dient dazu zwei Klassen des Typs *«navigationNode»* zu verbinden, aber ähnlich wie bei *«navigationNode»* muss dieser nicht unbedingt eine Transition zwischen zwei Seiten repräsentieren.

*«navigationNode»* kann in mehrere konkrete Klassen des Metamodells unterteilt werden. Zunächst gibt es die Klasse ***«navigationClass»***, diese wird mit jeweils einer Klasse des Inhaltsmodells assoziiert und stellt einen navigierbaren Knoten in der Hypertextstruktur zur Darstellung dieses dar.

Weiterhin existiert die Klasse ***«menu»***, welche den Zweck hat mehrere ausgehende Navigationspfade zu behandeln. Damit ist jedoch nicht zwingend ein Menu in Sinne einer Benutzeroberfäche gemeint.

Schließlich werden noch drei Zugriffsfunktionen in der abstrakten Klasse *«accessPrimitive»* zusammengefasst. Diese verbindet ihr Zweck Instanzen von zu Navigationsklassen zugehörigen Inhaltsklassen zu selektieren und es wird unterschieden zwischen ***«index»***, ***«query»*** und ***«guidedTour»***. Die Klasse *«index»* erlaubt es dem Benutzer aus einer vorher ermittelten Menge von Instanzen eine auszuwählen, die dann zum Inhalt der nachfolgenden Navigationsklasse wird. *«query»* hingegen bekommt keine Instanzenmenge von ihrem Vorgänger übermittelt, sondern dient dazu Inhalt aus einer anderen, Anfragen unterstützender

Datenquelle wie zum Beispiel einer Datenbank zu erhalten. *«guidedTour»* schließlich dient dazu sequentiellen Zugriff auf Instanzen von Navigationsklassen zu ermöglichen.

Die Unterteilung der Klasse Link ist deutlich einfacher. Innerhalb des Navigationspakets gibt es nur die Unterklasse ***«navigationLink»***, welche benutzt wird um alle vorhin beschriebenen Arten von Knoten zu verbinden. Es ist aber noch zu beachten, dass es zwei Klassen gibt, welche im Navigationsmodell eingesetzt werden, jedoch aus dem Paket für Prozesse stammen. Diese heißen *«processClass»* und *«processLink»* und werden im Abschnitt über Prozessmodellierung behandelt [25].

Um eine erste Skizze der Navigationsstruktur aus dem Inhaltsmodell der Anwendung zu erhalten liefert UWE einige methodische Richtlinien. Für die Navigation relevant erscheinende Klassen aus dem Inhaltsmodell werden zunächst in diesem markiert und werden zusammen mit ihren Assoziationen als Navigationsklassen und Navigationslinks übernommen. Für jede Klasse die mehr als eine von ihr ausgehende Assoziation besitzt werden dann Menüs (*«menu»*) hinzugefügt. Schließlich werden ein Index (*«index»*), eine Führung (*«guidedTour»*) oder eine Anfrage (*«query»*) zwischen alle Navigationsklassen eingefügt, die durch eine Assoziation verbunden sind und der Zielknoten über eine Multiplizität größer als eins verfügt. Dabei werden die Eigenschaften der entsprechenden Klasse des Inhaltsmodells über welche der Index oder die Anfrage laufen als Navigationsattribute zur Klasse im Navigationsmodell beigefügt [17].

### 2.3.4 Prozessmodellierung

Die Struktur des Prozessmodells ist deutlich einfacher und hängt stark mir dem Navigationsmodell zusammen. Zusammengefasst wird dies in der Abbildung 2.3.

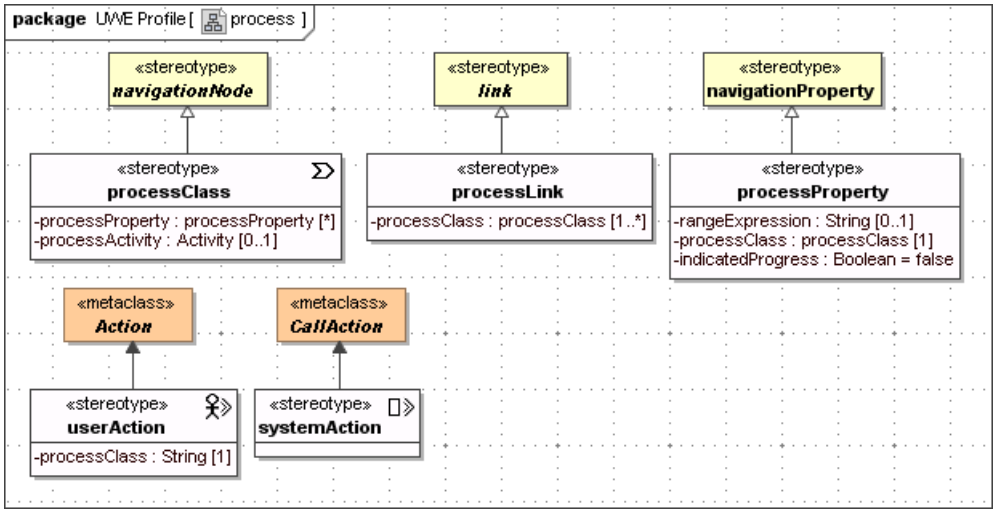


Abbildung 2.3: Metamodell der Prozesse von UWE

Wie oben bereits angedeutet gibt es zur Integration von Geschäftsprozessen, welche von Benutzerinteraktionen abhängen, die Klasse ***«processClass»***. Diese werden aus den Anwendungsfällen gewonnen, welche nicht nur zur reinen Navigation dienen und ist die Unterklasse der Klasse *«navigationNode»* aus dem Navigationspaket. Diese Prozesse werden im Navigationsmodell an entsprechenden Stellen platziert und zeigen dort wie ein Prozess durch den Benutzer erreicht werden kann sowie die Fortsetzung der Navigation nachdem dieser ausgeführt worden ist. Mit anderen Navigationsklassen werden diese aber nicht durch die Klasse *«navigationLink»* verbunden, sondern durch eine andere Unterklasse von *«link»*, der Klasse ***«processLink»***.

Nachdem die Ein- und Ausstiegspunkte der Prozesse im Navigationsmodell festgelegt wurden, werden die eigentlichen Prozesse nun im Prozessmodell im Detail beschrieben. Da die Prozesse hierarchische Vererbungsstruktur ähnlich dem Klassendiagramm des Inhaltsmodells aufweisen können, besteht eine Möglichkeit das Prozessmodell zu verfeinern darin, ein sogenanntes Prozessstrukturdiagramm, welches genau diese Struktur darstellt anzulegen. Da Prozesse in den meisten Fällen ein Benutzerinterface zur Eingabe und Darstellung von Daten benötigen, werden diese im Präsentationsmodell durch entsprechende Klassen dargestellt. Sollte jedoch an mehreren Stellen eines Prozesses eine Dateneingabe durch den Benutzer notwendig sein, so wird dies durch das Erzeugen von jeweils einem Prozess pro Eingabeschritt und die Verbindung dieser mit dem Hauptprozess aus dem Navigationsmodell durch eine Assoziation gelöst.

Weiterhin wird für jeden Prozess, der im Detail beschrieben werden soll ein Aktivitätsdiagramm festgelegt, welches den Ablauf von diesem Prozess beschreibt. Dieses wird als Prozessflußdiagramm bezeichnet und unterscheidet sich im Wesentlichen nur durch eine Verfeinerung der Aktionsknoten durch ein Paar UWE-spezifische Stereotypen von einem normalen UML-Diagramm. Eine Aktion zur Darstellung von Eingabe von prozessrelevanten Daten durch den Benutzer wird durch die Klasse ***«userAction»*** repräsentiert. Jede *«userAction»*-Klasse wird dabei mit einer *«processClass»* assoziiert um festzulegen, welche Daten bearbeitet werden und welche Präsentationsklasse dargestellt wird. Alle Aktivitäten an denen der Benutzer nicht beteiligt ist stellt hingegen die Klasse ***«systemAction»*** dar.

### 2.3.5 Das Präsentationsmodell

Das letzte Modell, das hier vorgestellt wird ist das in Abbildung 2.4 auf Seite 19 gezeigte Präsentationsmodell. Dieses dient dazu eine abstrakte Darstellung der Benutzerschnittstelle zu modellieren und ist das umfangreichste von allen. Es basiert zum einen auf dem Navigationsmodell und zumindest Teilweise auch auf den Prozessen aus dem Prozessmodell. Dabei werden konkrete Aspekte der Benutzeroberfläche wie zum Beispiel verschiedene Stile oder die genaue Platzierung der einzelnen Elemente innerhalb einzelner Fenster der Webappli-

kation weggelassen und nur die Grundstrukturen der Benutzeroberfläche, wie Textpassagen, Bilder oder Formulare werden zur Darstellung der einzelnen Navigationsknoten benutzt. Der Vorteil einer solchen Darstellung durch das Präsentationsmodell ist die Unabhängigkeit von den derzeit aktuellen Technologien zur Implementierung von Webanwendungen, was einem erlaubt sich Gedankten darüber zu machen welche Funktionalität an welcher Punkt der Benutzeroberfläche benötigt wird, bevor man sich auf die Details der Implementierung konzentrieren muss.

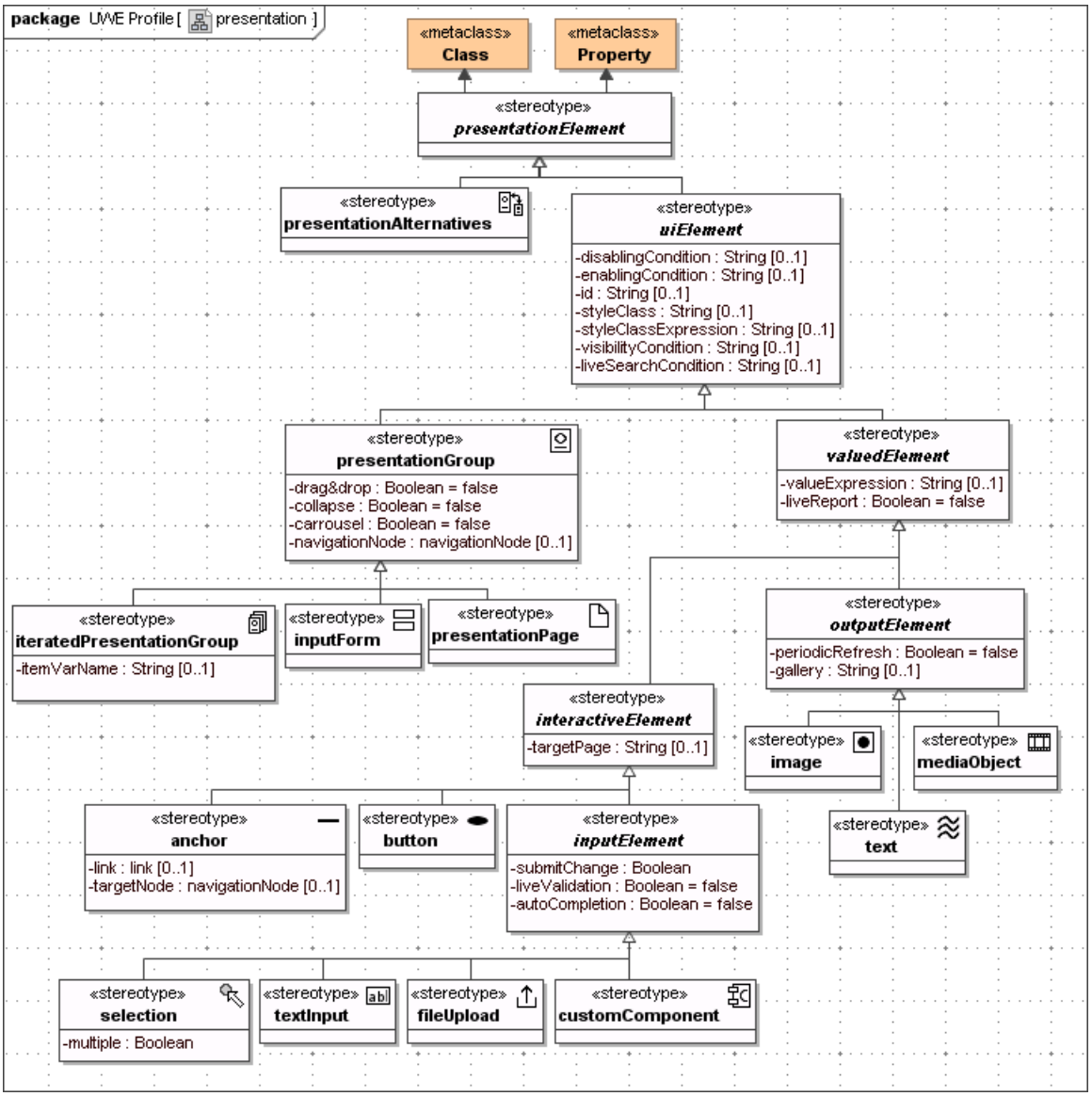


Abbildung 2.4: Metamodell der Präsentation von UWE

Die Basis des Präsentationsmodells stellt die Klasse ***«presentationGroup»*** dar. Diese definiert die Zusammenstellung verschiedener Präsentationselemente, welche die Darstellung

des Inhalts von Navigationsknoten wie Navigationsklassen, Menüs oder Prozessen repräsentieren. Das bedeutet, dass sobald der mit der Präsentationsklasse assoziierte Navigationsknoten erreicht wird der komplette Inhalt dieser Klasse angezeigt wird. Dabei können die einzelnen Präsentationsklassen andere Darstellungselemente enthalten, welche Unterklassen der abstrakten Klasse *«uiElement»* sind. Da dies auch auf *«presentationGroup»* zutrifft führt es zu einer baumartigen Darstellungsstruktur. Gehören dabei zwei Präsentationsklassen zu einem solchen Baum, so bedeutet dies, dass diese zusammen dargestellt werden und die zugehörigen Navigationsknoten automatisch durchlaufen werden. Ist dies jedoch nicht der Fall, so muss das Navigieren der Verbindung durch eine Benutzeraktion ausgelöst werden.

Man kann *«presentationGroup»* noch weiter unterscheiden, da diese drei Unterklassen besitzt. Die erste Klasse ist dabei ***«presentationPage»***. Ihre Besonderheit ist, dass diese Klasse nicht als Inhalt einer anderen Präsentationsklasse auftrete darf. Es bedeutet auch, dass *«presentationPage»* immer die Wurzel der vorhin erwähnten Darstellungsbäume definiert. Außerdem muss sie im Gegensatz zu einer *«presentationGroup»* nicht mit einem Navigationsknoten assoziiert werden, solange diese zumindest ein Element enthält, auf welches dies zutrifft.

Die zweite Unterklasse dient zur Darstellung von Alternativen und wird mit ***«iteratedPresentationGroup»*** bezeichnet. Dazu enthält sie eine Menge von *«presentationGroup»*-Elementen, deren Inhalt je nach dem aktuellen Stand der Navigation alternativ an der gleichen Stelle der Darstellungsseite präsentiert wird. Um dies zu erreichen soll einfach jedes Mal wenn ein mit einer der Alternativen assoziierter Navigationsknoten erreicht wird, der Inhalt der aktuell dargestellten Klasse mit der neu erreichten ersetzt werden. Für den Fall, dass keine Alternative explizit ausgewählt wurde, kann zusätzlich eine *«presentationGroup»* als Standardausgabe vorgegeben werden.

Als letztes gibt es noch den Stereotyp ***«inputForm»***, welcher die gleichen Eigenschaften wie *«presentationGroup»* besitzt. Dieser dient jedoch im Unterschied zur einfachen Gruppe der Darstellung eines Formulars, das bei Webseiten eine Sonderstellung einnimmt, da es das Absenden von eingegebenen Daten erleichtert.

Zusätzlich zu einer *«presentationGroup»* kann ein *«uiElement»* auch eine Unterklasse der abstrakten Klasse *«valuedElement»* sein. Unter ihr werden nämlich alle Darstellungselemente zusammengefasst, die für die Ausgabe und Bearbeitung des Inhalts zur Verfügung stehen. Diese werden in die jeweils entsprechende *«presentationGroup»* eingefügt und bilden sozusagen die Blätter der verschiedenen Darstellungsbäume. Alle Unterklassen von *«valuedElement»* können dabei in drei Kategorien eingeteilt werden.

Die erste Gruppe der Unterklassen bilden die statischen Darstellungselemente, welche alle von der abstrakten Klasse *«outputElement»* erben. Alternativ können sie auch benutzt werden, um Werte für Anfrageparameter zu liefern. Hier unterscheidet man drei Elemente.

***«text»*** dient dazu statische Textpassagen festzulegen, ***«image»*** steht für die Darstellung statischer Bilder und ***«mediaObject»*** schließlich für andere Multimediadaten.

Die zweite Gruppe besteht aus Unterklassen, welche eine Transition im Navigationsmodell auslösen können. Es gibt in dieser Kategorie zwei verschiedene Klassen, die beide direkte Unterklassen der abstrakten Klasse *«interactiveElement»*. Die erste Klasse bildet hier ***«button»***. Diese muss jedoch nicht zwingend einen Knopf innerhalb der Webanwendung darstellen, sondern stellt nur ein beliebiges Element dar wodurch der Benutzer irgendwelche Aktionen, wie zum Beispiel eine Anfrage oder das Absenden von Daten einleiten kann. Ergänzt wird die Gruppe durch eine Klasse ***«anchor»***. Wie auch *«button»* definiert dieser kein konkretes Element der Darstellung, sondern dient dazu eine Möglichkeit für den Benutzer zu einem anderen Punkt des Systems zu navigieren zu definieren.

Es gibt noch eine dritte Unterklasse von *«interactiveElement»*, welche ebenfalls abstrakt ist. Diese heißt *«inputElement»*und fasst Elemente zur Behandlung von Benutzereingaben zusammen. Dazu zählt zunächst die Klasse ***«textInput»***. Sie repräsentiert alle denkbaren Arten von Texteingabefeldern. Hinzu kommt die Klasse ***«selection»***, welche dem Benutzer erlaubt Angaben zu machen, indem er eine oder mehrere bereitgestellter Auswahlmöglichkeiten selektiert. Diese Funktionalität kann in einer konkreten Webanwendung auf verschiedene Arten realisiert werden. Mehrere Radiobuttons, Checkboxen oder auch eine Combobox sind hier denkbar. Dazu kommt noch die Klasse ***«fileUpload»***. Sie stellt die Möglichkeit des Benutzers dar ein oder mehrere Dateien zum Server hochzuladen.

Als letztes wäre in diesem Modell noch ***«presentationAlternatives»*** zu erwähnen. Diese dient dazu Elemente zu repräsentieren, von denen jeweils nur einer gleichzeitig angezeigt werden darf, was dadurch geschieht dass solche Elemente in einem Objekt des Typs *«presentationAlternatives»* zusammengefasst werden.

Erwähnenswert wäre hier noch dass die Beziehung zwischen den Präsentationsklassen und den Elementen der Benutzerschnittstelle die einer Komposition ist. Für die Darstellung der Präsentationsmodelle bedeutet dies, dass die Komposition dargestellt wird, indem der Inhalt einer *«presentationGroup»* innerhalb dieser dargestellt wird, falls diese durch das CASE-Tool ermöglicht wird.

## 2.4 Rich Internet Applications

Der Begriff Rich Internet Application (RIA) entstand zu Beginn dieses Jahrtausends und beschrieb damals die neu aufkommenden Webanwendungen, die durch eine reichhaltigere Benutzeroberfläche und eine verbesserte Benutzeraktion sich von bisherigen, traditionellen Webapplikationen unterscheiden. Dabei bedeutet traditionell bei Webanwendungen, dass der

Client alle Benutzereingaben als Anfragen an den Server weiterleitet, dieser bearbeitet die Anfrage und sendet eine Antwort, worauf der Client die Seite neu lädt. Vor allem die durch die synchrone Kommunikation entstehende Wartezeiten und das ständige Neuladen der Seiten wirken sich sehr negativ auf die Benutzerfreundlichkeit aus.

Durch die wachsende Verwendung von JavaScript und die Einführung von AJAX mit seinen asynchronen Kommunikationsmöglichkeiten konnte die Interaktion mit einer Webanwendung deutlich verbessert werden. 'Rich' bezieht sich somit auf die dadurch entstehenden Mehrwert eines Clients, welcher nun nicht nur für das Weiterleiten der Benutzereingaben an den Server verantwortlich ist, sondern auch selbst mehr oder weniger große Teile der Berechnungen durchführt, nur für den Client relevante Daten selber verwaltet und die Kommunikation mit dem Server verringert, indem er nur noch neue Daten meist im Hintergrund von Server anfordert. Für den Benutzer macht sich eine RIA vor allem durch die Bereitstellung von vielen im klassischen HTML nicht verfügbaren Elementen eines Benutzerinterface bemerkbar [4]. Damit lassen sich Rich Internet Applications auch als ein Hybrid ansehen, um die Vorteile von Webanwendungen und Desktopapplikationen zu verbinden. Ein Vergleich der drei Arten von Anwendungen ist in der aus *Modellierung und Generierung von Web 2.0 Anwendungen* [30] entnommenen Tabelle 2.1 zusammengefasst.

| **Eigenschaft Anwendung** | **klassische Webanwendung** | **Desktopanwendung** | **Rich Internet Application** |
|---|---|---|---|
| Installation | nein | ja | nein |
| Datenhaltung | zentral | meist verteilt | zentral |
| Erreichbarkeit | überall | an einen Ort gebunden | überall |
| Aktualisierung | sehr einfach und billig | meist teuer | sehr einfach und billig |
| Bedienelemente | keine oder sehr beschränkt | ja | ja |
| Tastaturbedienung | keine oder sehr beschränkt | ja | ja |
| Reaktionszeiten | meist lange Wartezeiten | sehr schnell | kurze bis keine Wartezeiten |
| Darstellungsänderung | Seite wird komplett geladen | dyn. Benutzeroberfläche | fast dyn. Benutzeroberfläche |

Tabelle 2.1: Vergleich der drei Arten von Anwendungen

Zum Abschluss dieses Abschnitts ist wohl noch eine eingehende Definition einer RIA angebracht, auch wenn diese nicht leicht zu finden ist. Die Definition der Online-Community *SearchSOA.com*[41] ist zwar etwas lang, jedoch dafür sehr ausführlich. Ihre sinngemäße Übersetzung lautet dabei wie folgt:

> *Eine Rich Internet Application (RIA) ist eine Webanwendung, welche entwickelt wurde, um die selben Eigenschaften und Funktionen zu liefern, die normalerweise mit Desktopanwendungen in Verbindung gebracht werden. Für gewöhnlich teilen RIAs die Berechnungen über das Internet/Netzwerk auf, indem sie die Benutzeroberfläche und damit verbundene Aktivitäten und Fähigkeiten auf der Clientseite und die Manipulation und Bearbeitung der Daten auf die Seite der Applikationsservers platzieren.*

*Eine RIA läuft dabei normalerweise innerhalb eines Webbrowsers und benötigt für gewöhnlich keine Softwareinstallation um auf der Clientseite zu funktionieren. Allerdings können einige RIAs nur innerhalb eines oder mehrerer spezieller Browser korrekt funktionieren. Aus Sicherheitsgründen läuft der Clientteil der meisten RIAs innerhalb einer isolierten Teils eines Desktopclients, welcher Sandbox genannt wird. Die Sandbox begrenzt die Sichtbarkeit und den Zugriff auf die Dateistruktur und das Betriebssystem des Clients für den Applikationsserver auf der anderen Seite der Verbindung.*

*Dieser Ansatz erlaubt dem Clientsystem lokale Aktivitäten, Berechnungen, Umformatierungen und Ähnliches zu handhaben und dabei die Menge und die Frequenz des Client-Server-Verkehrs zu verringern. Dies gilt besonders im Vergleich zu Client-Server-Implementierungen mit einem sogenannten Thin Client.*

*Eine charakteristische Eigenschaft einer RIA (im Gegensatz zu anderen webbasierenden Applikationen) ist die Clientengine, welche zwischen dem Benutzer und dem Applikationsserver vermittelt. Diese wird beim Start einer RIA geladen und kann während nachfolgender Operationen durch zusätzliche Downloads erweitert werden, in welchen die Engine als eine Browsererweiterung agiert um das Benutzerinterface und die Serverkommunikation bearbeiten zu können.*

## 2.5 Modellierung von RIAs in UWE

Der UWE-Ansatz zur Modellierung von Rich Internet Applications geht davon aus, dass hier wie auch bei traditionellen Webanwendungen typische Problemstellungen auftreten und sieht vor spezielle Eigenschaften der RIAs basierend auf Lösungsmustern so genannten Patterns zu modellieren. Der Vorteil an diesem Konzept ist die Wiederverwendbarkeit, die entweder manuell oder innerhalb eines automatisierten Entwicklungsprozesses stattfinden kann.

Dabei beschreiben drei Bestandteile die Funktionsweise eines RIA-Merkmals. Zunächst wird das Verhalten der RIA durch ein Ereignis ausgelöst, welches entweder durch den Benutzer oder durch das System zustande kommt. Als Reaktion darauf wird eine systeminterne Operation ausgeführt, die das Verhalten des RIA-Merkmals umsetzt. Abschließend kommt noch die Präsentation des Ergebnises der Operation an den Benutzer.

Allgemein müssen bei diesem Konzept die RIA-Patterns erstmal erstellt und zur Wiederverwendung bei der Modellierung in einer Bibliothek gespeichert werden. Weiterhin benötigt man eine Eventsprache zur Beschreibung der Benutzer- oder Systemereignisse, welche für die jeweiligen RIA-Elemente von Interesse sind. Um anzuzeigen, welche Muster an welcher

Stelle Anwendung finden braucht man schließlich noch neue Modellelemente, welche in die bestehenden Modelle der Webanwendungen integriert werden.

Im Rahmen von UWE werden für die einzelnen Lösungsmuster UML-Zustandsautomaten angegeben. Dies erfolgt einmalig und auf einem möglichst hohen Abstraktionsniveau und wird nach Abschluss in einen frei erweiterbaren Pattern-Katalog aufgenommen. Zur Integration der so entstandenen Lösungsmuster in die bisherigen UWE-Modelle muss die Modellierungssprache erweitert werden. Um das Risiko durch Erweiterungen ihren intuitiven Charakter zu verlieren zu minimieren, ist eines der Hauptanliegen von UWE minimalistisch und so bündig wie möglich zu sein. Dies wird durch Ergänzen der bestehenden Modellelemente durch sogenannte Tags erreicht, wobei eine solche Markierung eine Definition eines Metaattributes im Metamodell voraussetzt.

Zu diesem Zeitpunkt bestehenden Patterns bzw. dazugehörige Metaattribute werden im diesem Absatz und in der Tabelle 2.2 kurz skizziert. Die Klasse ***«uiElement»*** bekommt das Metaattribut *liveSearchCondition*, womit eine Anpassung der Suche innerhalb aller Unterklassen gekennzeichnet werden kann. Für ***«presentationGroup»*** gibt es gleich drei Metaattribute: *drag&drop* zeigt die Möglichkeit die Anordnung der Präsentationselemente zu verändern, bei *collapse* können interessante oder unwichtige Elemente hinzugefügt oder ausgeblendet werden und durch *carrousel* bedeutet die Darstellung in Form eines so genannten Karussells. In der abstrakten Klasse ***«inputElement»*** zusammengefasste Eingabeelemente können durch *liveValidation* mit der clientseitigen, parallel zur Eingabe erfolgenden Auswertung der Eingabedaten und durch *autoCompletion* mit während der Eingabe erfolgenden Anzeige von Vervollständigungsmöglichkeiten. Ausgabeelemente, die Teil der abstrakten Klasse ***«outputElement»*** sind, können mit *periodicRefresh* als in Echtzeit aktualisiert und mit *gallery* als zu einer als Gallerie bekannten Anzeigeform erweiterbar markiert werden. ***«valuedElement»*** besitzt zudem noch die Eigenschaft *liveReport*. Dadurch sollen die Objekte diese Typs so gekennzeichnet werden, dass der Benutzer zusätzliche Informationen angezeigt bekommt, sobald dieses Element von diesem Betrachtet wird, also zum Beispiel wenn der Mauszeiger sich über diesem Element befindet. Außerdem wurde die Metaklasse ***«selection»*** mit der Eigenschaft *multiple* versehen, um die Möglichkeit zur Mehrfachauswahl darstellen zu können [30].

## 2.6 Adaptivität

Ein immer wichtiger werdender Begriff bei Webanwendungen ist die Adaptivität. Ein kurze Definition dazu liefert *E-Learning: Das Wörterbuch*[3]:

| Stereotyp der Präsentation | Tags | Funktion |
|---|---|---|
| *«uiElement»* | *liveSearchCondition* | Direkte Anpassung der Suche |
| *«presentationGroup»* | *drag&drop* | Verschieben der Darstellungselemente |
| | *collapse* | Animiertes Verändern der Oberfläche |
| | *carrousel* | Präsentation der Liste als Karussell |
| *«inputElement»* | *liveValidation* | Direkte Auswertung der Eingaben |
| | *autoCompletion* | Vervollständigung der Eingabe |
| *«outputElement»* | *periodicRefresh* | Aktualisierung in Echtzeit |
| | *gallery* | Präsentation der Liste als Galerie |
| *«valuedElement»* | *liveReport* | Anzeige von Informationen unter dem Mauszeiger |
| *«selection»* | *multiple* | Gleichzeitige Selektion mehrerer Elemente |

Tabelle 2.2: RIA-Eigenschaften des UWE-Modells

> *Adaptivität ist die Eigenschaft eines Systems, sich an eine veränderte Umwelt bzw. neue Bedingungen und Anforderungen selbst anzupassen. Bei Informations- und Kommunikationstechnologien und Anwendungssystemen bedeutet Adaptivität u. a. die Möglichkeit der Personalisierung und damit der Orientierung an Aufgaben und Bedürfnissen des Benutzers.*

Indem sie sich über die Charakteristika, also die Vorlieben, Interessen, Aufgaben, Ziele und das Wissen des Benutzers oder den Kontext bewusst sind können adaptive Webanwendungen passendere und somit auch bessere Seiten liefern. Insgesamt braucht es für ein adaptives Websystem neben dem eigentlichen Websystem noch das Benutzermodell (User model), welches die vorhin erwähnten Eigenschaften beschreibt und einen Mechanismus, der eine Anpassung anhand dieser bereitstellt. Durch die ständige Anpassung des Inhalts und der Funktionalität an die Fähigkeiten und Interessen des Benutzers verbessern so die adaptiven Anwendungen die Interaktion der Benutzer mit dem Computer, die Zeit und Genauigkeit der Bearbeitung, die Lernprozesse des Benutzers und auch seine Zufriedenheit bei der Bedienung. Eine adaptive und flexible Anwendung kann dadurch eine größere Gruppe an Benutzern ansprechen und wird somit für mehr Benutzer interessant.

An dieser Stelle wäre noch der Unterschied zwischen zwei ähnlichen Begriffen erwähnenswert. So gibt es adaptive und adaptierbare Anwendungen. Adaptierbare Systeme können angepasst werden. Dies geschieht meißt durch den Benutzer, indem ihm die Möglichkeit gegen wird durch verstellen unterschiedlicher Parameter das System seinen Vorstellungen anzupassen. Adaptive Systeme hingegen führen diese Anpassungen autonom durch. Dazu benutzt das System die im Benutzermodell gespeicherten Informationen um sich an den jeweiligen Benutzer anzupassen. Da dieser Vorgang iterativ ist spricht man auch vom Adaptation Lifecycle (Lebenszyklus der Adaptivität), welcher in Abbildung 2.5 dargestellt ist.

Innerhalb dieses Zyklus unterscheidet man vier Phasen: *presentation* (Präsentation), *interaction* (Interaktion), *user observation* (Benutzerbeobachtung) und *adjustment* (Angleichung). Die Phasen Präsentation und Interaktion sind hier identisch zu den Vorgängen innerhalb

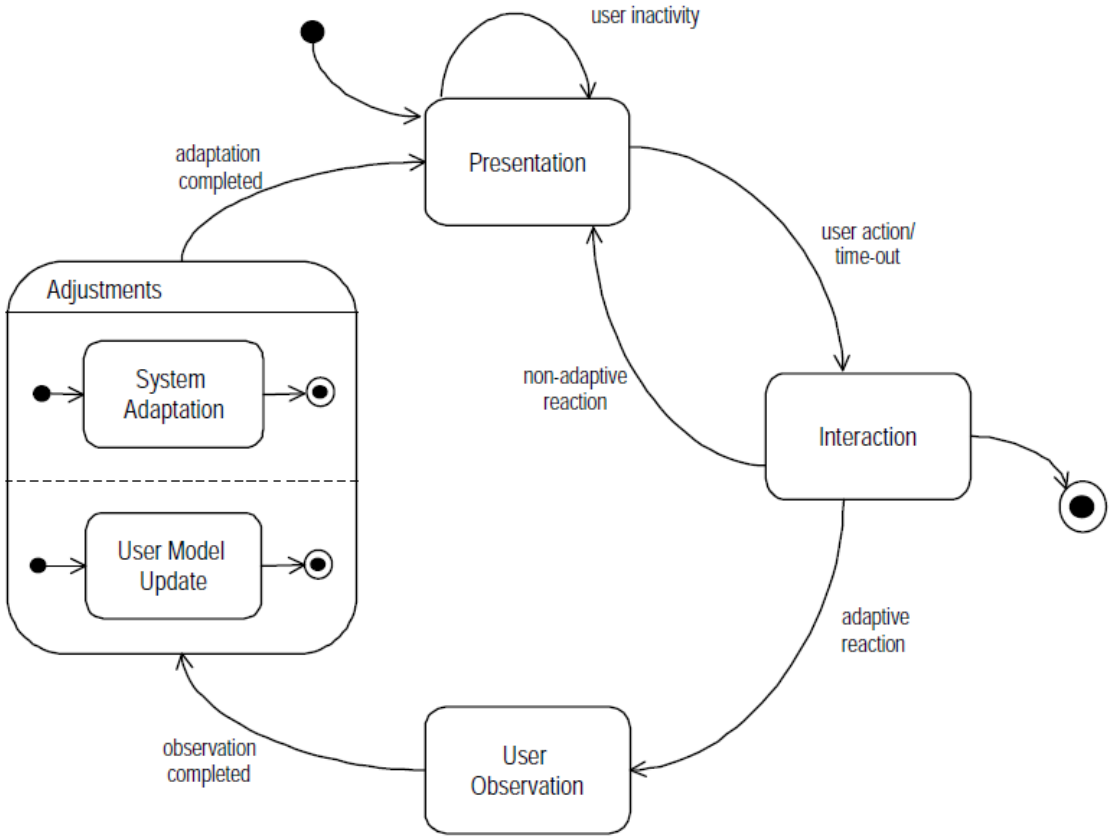


Abbildung 2.5: Lebenszyklus der Adaptivität

eines nicht adaptiven Systems, wo sich die Darstellung des derzeitigen Zustands und die Abfrage der Benutzereingaben stehts abwechseln. Wenn aber ein adaptives System bei den Eingaben feststellt, dass es sich um für Anpassungen interessante Daten handelt, kann es sich dafür entscheiden den Kreislauf zu vergrößern indem die zusätzlichen beiden Phasen ausgeführt werden. In der Phase der Benutzerbeobachtung wird dabei aus den gesammelten Interaktionsdaten die relevanten ausgewählt und bei Bedarf für die nächste Phase vorbereitet. Die Angleichung des Systems geschieht dann durch zwei parallele Vorgänge. Zum Einen wird das Benutzermodell mit den neuen Daten aktualisiert. Außerdem passt das System seine Erscheinung dem aktuellen Stand des Benutzermodells an [15].

Insgesamt gibt es drei Möglichkeiten, wie das System auch ohne explizite Einstellungen von der Benutzerseite eigenständig Information über diesen sammeln kann. Zum Einen geschieht dies durch Festlegung von verschiedenen Benutzerstereotypen während des Entwicklungszeit. Während der Laufzeit kann außerdem der Benutzer befragt werden, wodurch zum Beispiel das Benutzermodell initialisiert wird. Am wichtigsten ist aber dafür die Beobachtung des Benutzerverhaltens innerhalb des oben erwähnten Kreislaufs. Das dabei beobachtete Verhalten setzt sich zusammen aus der Navigation über die Linkstrukturen, dem Auswählen

verschiedener Menüpunkte und dem Ausfüllen von Formularen. Auch das Fehlen jeglicher Aktivität durch den Benutzer über einen bestimmten Zeitraum kann als eine Art von Verhalten aufgefasst werden.

Nachdem das System über ein Benutzermodell verfügt, existieren diverse Möglichkeiten wie es sein Erscheinungsbild für den Benutzer ändern kann. Diese können durch die drei wichtigsten Teile einer Webanwendung(Inhalt, Navigationsstruktur und Präsentation) unterschieden werden.

Die Anpassung auf der Ebene des Inhalts verfolgt das Ziel die Benutzerfreundlichkeit für eine breite Gruppe von Benutzern mit unterschiedlichem Hintergrund zu verbessern. Dies kann entweder durch verstecken irrelevanten Inhalts, wie zum Beispiel das Kollabieren von unnötigen Textabschnitten bzw. die Expansion relevanter Textteile oder durch Austausch ausgewählter Elemente, wie das Darstellen einer aus einer Gruppe von Alternativen ausgewählten Seite.

Auf der Ebene der Navigationsstruktur wird durch die Adaptation versucht den Benutzer davor zu bewahren überflüssige oder für ihn irrelevante Navigationspfade zu auszuwählen. Hier gibt es zahlreiche Methoden dies zu erreichen. Als erstes gibt es den Ansatz der globalen Führung, welcher versucht für den Benutzer den kürzesten Weg zu interessanten Informationen zu liefern. Dies kann zum Beispiel durch die Sortierung von Links nach der Relevanz bewirkt werden. Betrachtet man hingegen nur den nächsten Schritt so spricht man von einer lokalen Führung des Benutzers, beispielsweise indem man ihm nur eine Möglichkeit zur Navigation präsentiert. Weiterhin versucht die Methode der globalen Orientierung den Benutzer zu unterstützen, indem es ihn über die vorhandenen Navigationsstrukturen in Kenntnis setzt. Eine Möglichkeit dafür ist das Markieren der verschiedenen Links durch unterschiedliche Farben, Begleittext oder Ähnliches. Auch hier gibt es die lokale Methode. Diese lokale Orientierung hilft dem Benutzer indem es ihm die sinnvollen Navigationsmöglichkeiten von diesem Punkt aus anzeigt oder es kann auch die irrelevanten Pfade dem Benutzer unzugänglich machen, indem es die entsprechenden Links entfernt. Zuletzt gibt es noch den Ansatz der personalisierten Sicht, wobei für jeden Benutzer eine personalisierte Sicht erzeugt wird und diese dann ständig aktualisiert wird, indem weitere passende Links dafür gesammelt werden.

Die dritte Ebene ist die der Präsentation. Hierbei wird das Layout an die Vorlieben und Bedürfnisse des Benutzers angepasst. In einer multilingualen Umgebung ist eine wichtige Methode die Auswahl einer Sprache, welcher der Benutzer auch verstehen kann. Dies kann durch die beim Inhalt erwähnten Techniken erreicht werden. Außerdem gibt es noch den Ansatz verschiedener Layout-Varianten, welche alle denkbaren Darstellungsalternativen des Layouts wie Farben, Schriftsätze, Textausrichtung und so weiter beinhalten können. Ein besondere Möglichkeit ist hier der Einsatz von verschiedenen Styleguides.

## 2.7 Adaptivität in UWE

Da Adaptivität eine eigene Ebene der Modellierung von Webanwendungen darstellt ist es nicht sonderlich wünschenswert diese im ursprünglichen Modell für die in Frage kommenden Elemente einzusetzen, da dies zu Redundanzen und einer höheren Fehleranfälligkeit führen kann [2]. Um diese Probleme zu vermeiden schlägt UWE den Einsatz von Aspektorientierter Modellierung vor und definiert verschiedene Arten von Aspekten für die statische oder auch für die Adaptivität zur Laufzeit. Dies ist ein Ansatz der Modellierung der ein neuartiges Konstrukt einsetzt: Den Aspekt.

So ein Aspekt setzt sich aus zwei Teilen zusammen. Der erste Teil ist die Auswahl bestimmter Elemente auf welche dieser Aspekt angewandt wird und wird als *«Pointcut»* bezeichnet. Im zweiten Teil, welcher als *«Advice»* bezeichnet wird sind dann die neu hinzukommenden Eigenschaften für die ausgewählten Elemente festgelegt. Somit entsteht das vollständige Modell indem das ursprüngliche Modell mit den Aspekten zusammengeführt wird. Dieser Prozess wird als *«Weaving»* bezeichnet.

Die Integration von Adaptivität in die Navigationsstrukturen von UWE wird durch Hinzufügen von der neu hinzukommenden Metaklasse *«NavigationAnnotation»* an einen Link erreicht. Diese Annotationen repräsentieren die Anpassungen des Benutzermodells durch das beobachtete Verhalten. Hierbei beschreibt ein Aspekt im *«Pointcut»* welche Links für diese Adaptation ausgewählt werden sollen, während im *«Advice»* die Annotation festgelegt wird.

Die Adaptivität zur Laufzeit unterscheidet sich von der statischen dadurch, dass die Information für das Zusammenführen der Aspekte mit dem Modell erst zur Laufzeit vollständig bereitstehen. Hier unterscheidet man drei Aspekttypen: Aspekte zur Linkannotation (*«link annotation aspect»*), Aspekte zur Linküberquerung (*«link traversal aspect»*) und Aspekte zur Linktransformation (*«link transformation aspect»*).

Die ersten beiden Sorten haben die selbe Struktur. Ihr Pointcut ist ein Navigationsdiagramm und ihr advice eine OCL-Beschränkung. Der eigentliche Unterschied liegt in der Art dieser Beschränkung. Für Aspekte zur Linkannotation ist diese eine Invariante und muss für alle durch den Pointcut ausgewählten Links halten, womit die Anpassung von Linkannotationen an Benutzereigenschaften modelliert wird. Bei Aspekten zu Linküberquerung, welche die Anpassung des Benutzermodells ermöglichen, ist dies hingegen eine Nachbedingung und muss immer halten, wenn ein ausgewählter Link durchschritten wird.

Schließlich gibt es noch die Aspekte zur Linktransformation, die die adaptive Linkerzeugung und Linkzerstörung modellieren sollen. Diese bestehen in beiden Teilen aus einem Navigationsdiagramm, wobei hier der Pointcut die Knotenpunkte anzeigt, zwischen denen die Transformation stattfinden soll und das Advice den Zustand nach der Transformation

darstellt.

Nachdem nun die Modellierung von Adaptivität in UWE vorgestellt wurde, gibt es noch eine wichtige Information, welche unbedingt erwähnt werden sollte: Zu dem Zeitpunkt der Entstehung dieser Arbeit ist dieser Teil der Modellierung in UWE noch nicht in einem Modellierungswerkzeug integriert.

# 3 Modellierung der Beispielanwendung

Obwohl die Anforderungsanalyse im Entwicklungszyklus vor der eigentlichen Modellierung kommt, folgt diese Arbeit einem gegensätzlichen Ansatz, indem zuerst die Beispielanwendung mit allen dazugehörigen Modellen dargestellt und erst im nächsten Kapitel die Anforderungsanalyse im Detail betrachtet wird. Dies erscheint deswegen sinnvoller, da in Rahmen dieser Arbeit die Erweiterung des UWE-Modells sich auf die Anforderungsanalyse beschränkt. Die Erweiterungen hingegen werden hauptsächlich deswegen eingeführt, um die Generierung der Modelle aus diesen zu erleichtern und zumindest teilweise zu automatisieren. Daher sind die Änderungen viel leichter nachzuvollziehen, wenn die daraus resultierenden Modelle bereits bekannt sind.

## 3.1 Überblick über Philoponella

Durch die heutige Fülle an Inhalten im World Wide Web ist es die übliche und fast schon notwendige Praxis sich die wichtigen Internetadressen als Favoriten zu speichern. Jeder moderner und sogar bereit viele längst veraltete Browser haben dazu die entsprechenden Funktionen bereitgestellt, was jedoch ein Problem mit sich bringt: Die Favoriten sind immer an ein einzelnes System gebunden. Dies bedeutet, dass man entweder nicht von jedem Arbeitsplatz diese zur Hand hat, oder diese manuell abgleichen muss. Die Favoriten dort zu halten wo sie auch gebraucht werden, nämlich an einer Adresse im Netz beseitigt dieses Problem. Außerdem eröffnet dies die Möglichkeit die Suche nach interessanten Adressen zu vereinfachen, indem man die einzelnen Listen der Benutzer mit ihren persönlichen Favoriten zu einem Netzwerk zusammenschließt. Die innerhalb dieser Arbeit erstellte Beispielanwendung mit dem Titel Philoponella soll genau dies ermöglichen. Dabei ist die Anwendung so konzipiert, dass sie ausreichende Möglichkeiten bietet, die neuen Konzepte zu entwickeln und zu testen.

Da eine komplexe Software den Rahmen hier sprengen würde, beschränkt sich Philoponella zunächst nur auf die wichtigsten Funktionen eines solchen Systems. Zunächst werden von Philoponella alle Links zusammen mit den dazugehörenden Informationen verwaltet und jeder Benutzer hat die Möglichkeit die gespeicherten Informationen zu durchsuchen und deren Informationen zu betrachten. Um diese Suche für die Benutzer zu erleichtern sollen dabei die Linkinformationsobjekte in Kategorien und Unterkategorien geordnet sein. Jedem Besucher wird weiterhin die Möglichkeit bereitgestellt sich beim System zu registrieren, was ihm mehrere neue Optionen eröffnet. Als erstes ergibt sich für registrierte Benutzer dadurch die Möglichkeit selbst die Liste der Informationsseiten zu erweitern, wobei die einzelnen Einträge neben den eigentlichen Adressen noch weitere Kurzinformationen wie Kommentare, Bewertungen und Vorschaubilder enthalten. Auch die Kategorie für den Link kann gewählt werden und bei Bedarf ist es registrierten Benutzern erlaubt eine neue Kategorie zu erstellen.

Außerdem besitzt jeder registrierte Benutzer einen persönliche Bereich. Hier finden sich natürlich erstmal seine persönlichen Einstellungen und Daten. Weiterhin liegt hier auch der Hauptzweck dieser Anwendung, nämlich das anlegen persönlicher Favoritenlisten, welche damit persistent und über das Internet verfügbar gespeichert werden. Um auch dem Communitycharakter der Anwendung gerecht zu werden, fehlt natürlich noch die Möglichkeit mit den Benutzern zu interagieren. Da jedoch die vielen Möglichkeiten, die sich daraus ergeben das System zu sehr aufblähen würden, beschränkt sich Philoponella nur aufs notwendigste. Die Benutzer können hier zunächst einmal den für andere sichtbaren Bereich anderer Benutzerprofile betrachten, was auch beinhaltet, dass sie den freigegebenen Teil der Favoriten der anderen einsehen können. Zudem können sie auch die Benutzer, ähnlich den Favoriten in einer Freundesliste, verwalten, und diesen Nachrichten zukommen lassen. Außerdem sind für die Einträge in die Listen der Freunde und Favoriten vier verschiedene Zustände vorgesehen: Für alle sichtbar, für Freunde des Benutzers sichtbar, nur sichtbar für einen selbst und ignoriere den Eintrag. Dabei führt der letzte Zustand bei einem Benutzer dazu, dass seine Kommentare und Nachrichten für den ihn ignorierenden Benutzer nicht sichtbar sind und bei einer Seiteninformation, dass diese nicht in der Suchliste auftaucht.

Daneben soll sich diese Anwendung zusätzlich durch zwei besondere Merkmale auszeichnen, auf welche bei der Erweiterung des UWE-Profils und darin vor allem bei der Anforderungsanalyse besonders geachtet werden soll:

Zum Einem soll diese in die Kategorie der Rich Internet Applications fallen. Dies bedeutet vor Allem eine zeitgemäße Benutzeroberfläche mit entsprechenden Elementen der Benutzeroberfläche, welche kaum von einer Desktopanwendung zu unterscheiden ist. Erreicht wird dies durch die Umstrukturierung des Document Object Models (DOM) durch JavaScript und eine asynchrone Kommunikation mittels XMLHttpRequest-Objekten, welche es ermöglichen nur benötigte Daten anstelle einer kompletten Seite anzufordern.

Das zweite Merkmal der Anwendung soll die bereits vorgestellte Adaptivität sein. Also soll das System durch Beobachtung und persönliche Einstellungen eine Anpassung der Präsentation, des Inhalts und des Verhaltens an den jeweiligen Benutzer anpassen. Im Fall von Philoponella bedeutet dies vor allem eine Sortierung der Listen und Präsentation passender Vorschläge.

Schließlich soll Philoponella durch die verschiedenen vorher vorgestellten Möglichkeiten zur Benutzerinteraktion in die Kategorie der sozialen Netzwerke fallen. Wobei sich wie schon angedeutet die Optionen auf die Grundelemente einer solchen Anwendung wie Nachrichten und Freundeslisten beschränken.

Nach diesem groben Überblick wird Philoponella anhand des Modells nun im Detail beschrieben. Wie bereits gesagt besteht die Modellierung in UWE aus vier Teilen: dem Inhalt, der Navigation, der Präsentation und zusätzlich noch aus den Prozessdetails. Diese werden in den folgenden vier Abschnitten für Philoponella ausführlich dargestellt.

## 3.2 Der Inhalt von Philoponella

Das Modell des Inhalts ist in zwei Bereiche unterteilt. Den ersten Teil bildet dabei das eigentliche Inhaltsmodell. In ihm enthalten sind nur die Klassen, welche primär für die Repräsentation der normalen Anwendungsdaten dienen. Auf der anderen Seite steht das Benutzermodell, welches die restliche Klassen enthält. Diese repräsentieren dabei neben der für dem Benutzer relevanten Anwendungsdaten auch solche, die für das adaptive Verhalten zuständig sind. Allerdings ist eine solche Trennung nicht hundertprozentig eindeutig, da viele Anwendungsdaten auch für die Berechnung der Anpassungen herangezogen werden und viele Daten die hauptsächlich für die Adaptivität nützlich sind trotzdem auch für den Benutzer ein wenig Relevanz besitzen. Zur besseren Übersicht wird hier das Inhaltsmodell in Abbildung 3.1 auf Seite 33 präsentiert.

### 3.2.1 Das Inhaltsmodell

Wie bereits erwähnt ist diese Hälfte der Inhaltsdaten primär für die Benutzer vom Nutzen. Das zentrale Objekt von Philoponella bildet die Adressen der diversen Internetseiten, welche oft auch als Links bezeichnet werden. Daher erscheint es logisch, dass die wichtigste Klasse (zumindest in dieser Hälfte) zur Speicherung der Links und den zugehörigen Informationen dient. Sie trägt den Namen **LinkInfo** und hält die Adresse im Attribut adress fest. Da es in Philoponella nicht möglich sein soll, dass zwei Einträge zur selben Adresse angelegt werden, dient dieses Attribut auch zur eindeutigen Identifikation eine Objekts. Da jedoch nur die

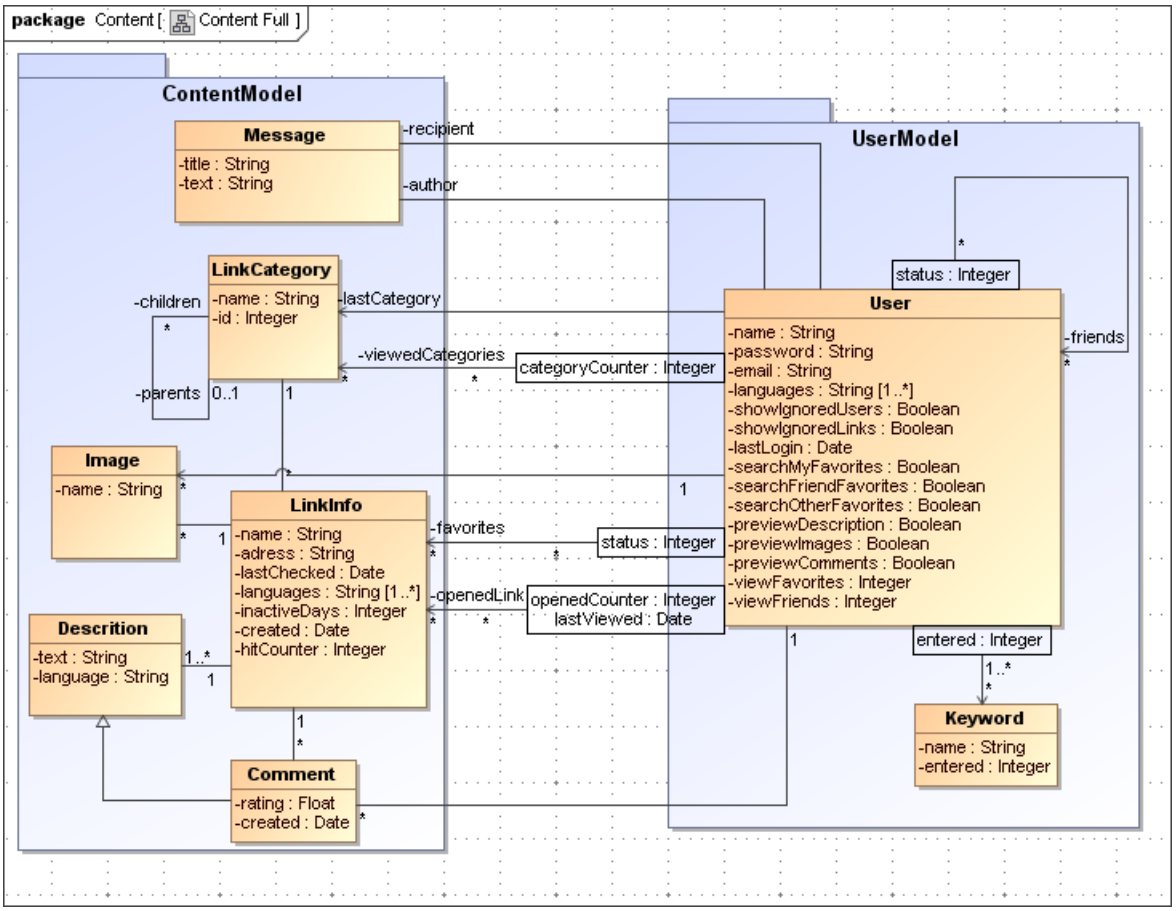


Abbildung 3.1: Philoponellas Inhaltsmodell

Adressen für den Benutzer kaum aussagekräftig genug wären, um diese auch sinnvoll benutzen zu können, werden zu den einzelnen Seiten auch weitere Informationen gespeichert. Dazu gehört natürlich zunächst der Name der Seite, welcher im Attribut name gespeichert wird. Die Identifikationsnummer dient dabei zur eindeutigen Bestimmung einer LinkInfo, was vor allem bei den Anfragen des Clients an den Server von nutzen ist. Hier wäre zwar auch die Erzwingung von eindeutigen Namen denkbar, dies würde jedoch zu einer zu starken Einschränkung führen. Denn wenn es auch recht selten ist, so lassen sich immer wieder gleichnamige Seiten im Internet finden. Da außerdem die Seiten nicht auf eine bestimmte Sprache gebunden sind und verschiedene Benutzer über verschiedene Sprachkenntnisse verfügen, ist es auch sinnvoll die Sprachen, in welchen die Seite angeboten wird zu Speichern, was durch das Attribut language erledigt wird. Schließlich gibt es noch zwei Daten, welche sowohl für den Nutzer als auch für Systemanpassungen von Interesse sein können. Das Attribut created hält dabei das Erstellungsdatum des Objekts fest. Der Status der Erreichbarkeit bzw. die bisherige Dauer der Inaktivität einer Seite wird hingegen vom Attribut inactiveDays gespeichert und lastChecked hält das dazugehörige letzte Datum der Überprüfung fest. Zu Letzt speichert hitCounter

Es gibt noch weitere Daten, welche über eine Seite gespeichert werden, jedoch über eine eigene Klasse verfügen. Dazu gehört zunächst **Description**, die Klasse zur Darstellung eine Beschreibung einer Seite. Neben dem in text festgehaltenen Beschreibungstext enthält dieser noch in language die Sprache, in welcher diese Beschreibung verfasst ist. Eine Beschreibung macht natürlich nur Sinn, wenn sie genau zu einer LinkInfo gehört. Dagegen kann eine LinkInfo durchaus mehrere Beschreibungen enthalten, solange diese in verschiedene Sprachen erstellt wurden.

Zusätzlich zu einer Beschreibung kann eine Seite auch durch Vorschaubilder näher beschrieben werden. Dafür ist die Klasse **Image** zuständig, welche über eine eindeutige Identifikationsnummer name die richtige Zuordnung der Dateien sicherstellt. Da die Vorschaubilder einer Seite von verschiedenen Benutzern erstellt werden dürfen, wird jedes Objekt vom Typ Image sowohl einer LinkInfo als auch einem User[1] zugeordnet.

Schließlich ist es für Benutzer möglich eine Seite zu kommentieren. Dies wird durch die Klasse **Comment** repräsentiert. Wie bereits Image steht auch diese sowohl mit einem Objekt vom Typ User[1] als auch einem Objekt mit dem Typ LinkInfo. Da ein Kommentar auch ein Text in einer bestimmten Sprache darstellt, ist dieser eine Unterklasse von Description. Zusätzlich dazu ist aber bei der Klasse Comment auch noch das im Attribut created festgehaltene Erstellungsdatum von Interesse. Außerdem wird dem Ersteller die Möglichkeit gegeben eine Bewertung der Seite anzugeben, welches in rating abgelegt wird.

Auch wenn diese Klassen ausreichend Möglichkeiten bieten eine Seite zu beschreiben, gibt es im Zusammenhang mit LinkInfo noch eine weitere Klasse. Diese soll wie am Anfang des Kapitels beschrieben durch bessere Strukturierung der Seiten innerhalb von Kategorien die Suche erleichtern. Die Klasse trägt den Namen **Category** und jedes Objekt vom Typ LinkInfo muss einem solchen Objekt zugeordnet sein. Eine Kategorie besitzt dabei zunächst einen Namen (name), aber wie auch bei Seitennamen reicht dieser nicht zur eindeutigen Identifikation aus. Es soll nämlich durchaus möglich sein mehrere Kategorien gleich zu benennen, was vor allem bei Unterkategorien innerhalb unterschiedlicher Kategorien möglich sein soll. Daher braucht diese Klasse eine Attribut zur eindeutigen Identifikation, welches in Form einer Nummer in id gespeichert wird. Außerdem wird der Baumcharakter der Kategorien durch eine Verbindung zu weiteren Kategorien dargestellt. Dies bedeutet, dass ein Objekt vom Typ Category über höchstens ein Elternobjekt und beliebig viele Kinderobjekte desselben Typs verfügt.

Bis jetzt dienen alle Klassen zur Repräsentation und Organisation der Daten einer Internetseite, aber die Grundlage eine Webanwendung, in der die Benutzer ein soziales Netzwerk errichten können, ist die Möglichkeit Nachrichten mit anderen Benutzern auszutauschen. Daher ist die letzte Klasse innerhalb dieser Hälfte des Inhaltsmodells **Message**.

[1] wird innerhalb des nächsten Abschnitts besprochen

Um diese Mechanik nicht unnötig zu verkomplizieren beschränkt sich diese Klasse auf einfache Nachrichten, welche nur einen Titel und einen Text in Form der beiden Attribute title und text enthalten. Allerdings braucht jede Nachricht sowohl einen Absender als auch einen Empfänger, was durch zwei Verbindungen zur Klasse User[1] dargestellt wird.

### 3.2.2 Das Benutzermodell

Die zweite Hälfte des Inhalts bildet das Benutzermodell, dessen Hauptteil die Klasse **User** bildet. Diese enthält alle Daten zur Darstellung eines Benutzers und erlaubt dem System sich an diesen anzupassen. Zunächst besitzt auch ein Benutzerobjekt in name einen Namen, welcher allerdings im Gegensatz zu einer Seite oder Kategorie den Benutzer eindeutig auszeichnet, da mehrere Benutzer mit dem gleichen Namen nur zu Verwirrung führen würden. Um die Benutzerdaten vor unerlaubten Zugriffen zu schützen besitzt jeder Benutzer zudem ein Passwort in Form einer in password gespeicherten Zeichenkette. Außerdem wird jedem Benutzer eine Emailadresse im Attribut email zugeordnet, was nützlich ist, um den Benutzer zu kontaktieren. Schließlich werden auch innerhalb von User die vom Benutzer beherrschten Sprachen in dem Attribut languages gespeichert. Zusätzlich zu diesen Daten wird im Attribut lastLogin für jeden Benutzer festgehalten wann er sich das letzte Mal eingeloggt hat.

Neben den Daten über den Benutzer enthält User auch die von diesem vorgenommenen Einstellungen. Dazu gehören zunächst die beiden Optionen zur Darstellung der ignorierten Benutzer durch showIgnoredUsers und Seiten durch showIgnoredLinks. Diese geben an, ob der Benutzer will, dass andere diese Listen sehen können. Damit der Benutzer einstellen kann, welche Favoriten er durchsuchen will besitzt User die drei boolschen Attribute: searchMyFavorites falls auch eigene Favoriten angezeigt werden sollen, searchFriendFavorites für die Favoriten der Freunde und searchOtherFavorites für die Anzeige aller Linkinformationen, die nicht in eine der beiden anderen Kategorien fallen. Eine weitere Einstellung des Benutzers ist die Darstellung der Details der Linkinformationen. Dabei kann der Benutzer durch ebenfalls drei boolsche Attribute auswählen, ob durch die Auswahl von previewDescription die Beschreibungen, durch previewComments die Kommentare und durch previewImages die Vorschaubilder angezeigt werden sollen. Zuletzt werden noch die beiden Einstellungen zur Anzeige der Listen der Favoriten in viewFavorites und Freunde in viewFriends gespeichert. Diese geben an, welche Teile der entsprechenden Liste beim Ansehen der eigenen Seite angezeigt werden sollen.

Außer der bisher besprochenen Attribute besitzt die Klasse User viele Verbindungen zu anderen Klassen. Als erstes wäre da die Verbindung zu sich selbst zu erwähnen. Diese repräsentiert die Freundesliste eines Benutzers und ist mit einem Kennzeichner für den Status des Freundes ausgestattet. Für die Kommunikation gibt es zwei Verbindungen zu der Klasse Message, da ein Benutzer hier der Absender und der andere der Empfänger ist. Bei den Kategorien wird zu jedem Objekt festgehalten welche zuletzt betrachtet und ob bzw. wie oft

welche Kategorie geöffnet wurde. Ähnlich verhält es sich auch bei der Klasse LinkInfo. Auch hier wird für die Linkinformationen gespeichert, welche wie oft betrachtet wurden. Jedoch wird hier anstatt nur das letzten Objekt festzuhalten, für jeden einzelnen das Datum des letzten Zugriffs archiviert. Eine weitere Verbindung zu LinkInfo stellt die Liste der Favoriten dar, wobei auch hier der Kennzeichner den Status von diesem angibt. Auch zu Comment und Image gibt es eine Verbindung, welche den Ersteller dieser angibt.

Zu guter Letzt gibt es noch eine Verbindung zu der noch nicht besprochenen Klasse **Keyword**, wodurch die vom Benutzer eingegeben Suchbegriffe gespeichert werden. Diese Klasse hat dazu das Attribut name, während das Attribut entered angibt, wie oft der Begriff insgesamt eingegeben wurde. Der Kennzeichner der Verbindung mit dem gleichen Namen gibt hingegen nur an wie oft dieser Begriff von diesem Benutzer benutzt worden ist.

## 3.3 Philoponellas Navigationsstruktur

Die Navigationsstruktur der Webanwendung wird in UWE durch das Navigationsmodell dargestellt. Dieses stellt die Übergänge zwischen den Verschiedenen Navigationspunkten und von welchen aus man welche Prozesse aufrufen kann. Dabei sind die einzelnen Navigationsknoten nicht zwingend mit einer Webseite gleichzusetzen, mehrere Knoten können durchaus innerhalb einer Seite dargestellt werden und es ist auch durchaus denkbar, dass ein Knoten durch mehrere Seiten dargestellt werden muss. Auch wenn die Aufteilung des Navigationsmodells nicht ganz so einfach und klar ist wie beim Inhalt, so bieten dafür die einzelnen Navigationsknoten eine gute Möglichkeit. Hier gibt es in Philoponella neben dem Standardknoten für den Start noch drei weitere. Sie ergeben sich zum einen aus den beiden Hauptklassen LinkInfo und User und außerdem aus der Möglichkeit für diverse persönliche Einstellungen.

### 3.3.1 Startpunkt

Die Struktur um den Basisnavigationsknoten ist recht einfach. Neben der Navigationsklasse **Home** und dem dazugehörigen Menü MainMenu gibt es hier nur zwei weitere Knoten. Zunächst bietet Philoponella jedem Besucher die Möglichkeit die Liste der gespeicherten Linkinformationen zu durchstöbern. Um dies darzustellen ist die Listenklasse LinkIndex vom Menü aus erreichbar. Aber da dies recht umständlich ist, um nach bestimmten Seiten zu suchen, können die Besucher auch eine richtige Suche durch Eingabe von Suchbegriffen einleiten. Dies wird durch die ebenfalls vom Menü aus erreichbare Anfrageklasse Search repräsentiert, welche weiter zum LinkIndex führt, da das Ergebnis einer Suche ebenfalls eine Liste ist, auch wenn diese vielleicht stark eingeschränkt ist.

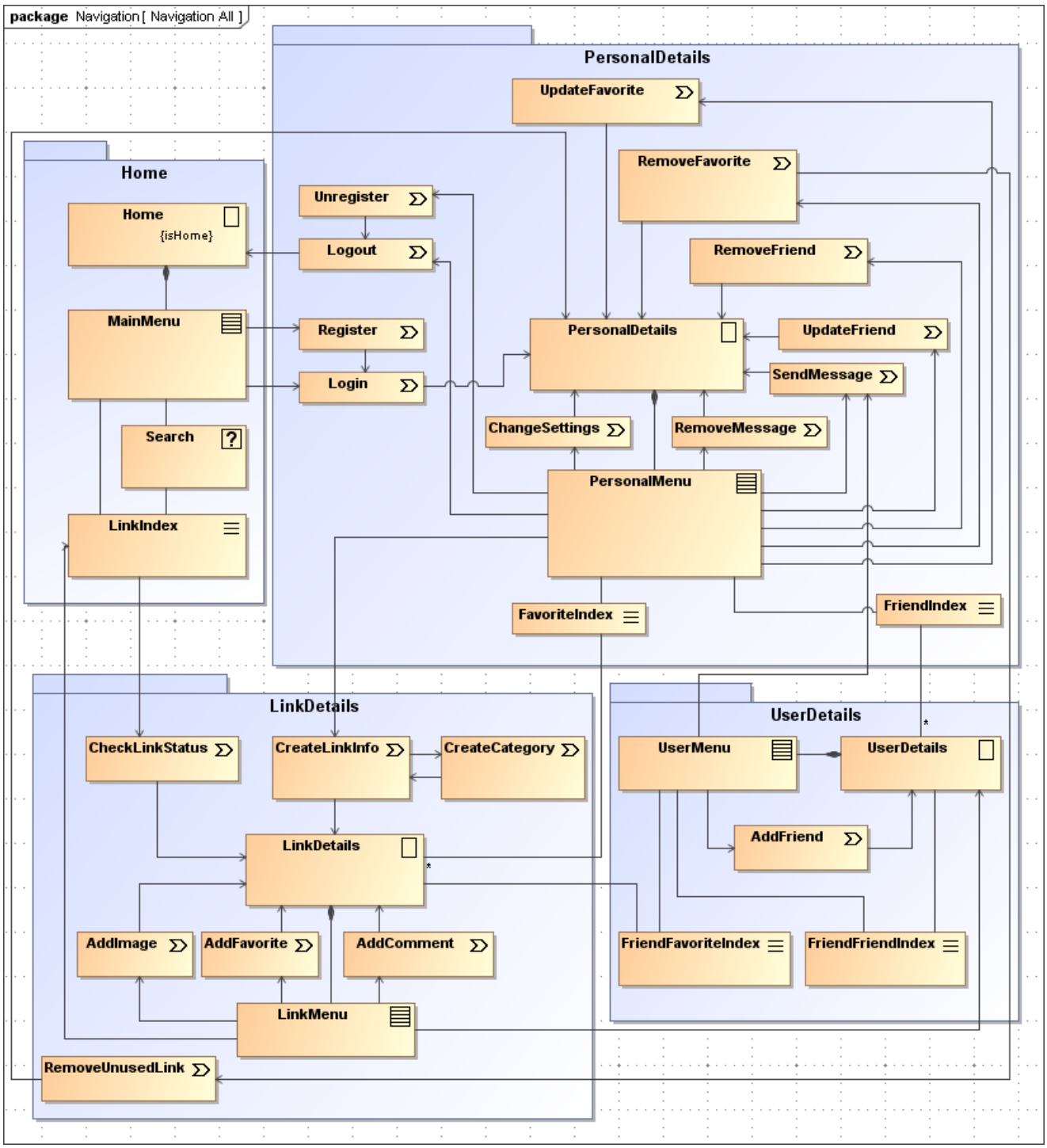


Abbildung 3.2: Philoponellas Navigationsstruktur

Alle weiteren Funktionalitäten von Philoponella liegen in den anderen drei Bereichen der Navigationsstruktur. Dabei ist der persönliche Bereich direkt vom Hauptmenü aus erreichbar, indem sich der Besucher entweder einloggt oder registriert, was jeweils eine Verbindung zu den beiden Prozessklassen Login und Register von MainMenu aus modelliert wird. Weiterhin kann auch der Besucher die in der Liste aufgeführten Seiten im Detail betrachten indem er die Detailansicht für diese öffnet, welcher das Zentrum des Abschnitts für Linkinformationen bildet. Damit die Detailansicht immer auf dem neuesten Stand ist, soll das System bei jedem Aufruf die Verfügbarkeit der Seite überprüfen. Dies wird durch die Verbindung von LinkIndex zu der Prozessklasse CheckLinkStatus dargestellt.

### 3.3.2 Persönliche Optionen

Der umfangreichste Teil des Navigationsmodells enthält die Prozesse und Navigationsstrukturen, welche einem registrierten Benutzer zur Verfügung stehen. In seinem Zentrum steht die Navigationsklasse **PersonalDetails** und das passende Menü PersonalMenu. Erreichbar ist dieser Bereich vom Startpunkt aus durch die beiden Prozessklassen Login und Register. Dies modelliert den Umstand, dass ein Besucher sich selbstverständlich erst als ein bestimmter Benutzer ausweisen muss, bevor er zu seinen persönlichen Optionen gelangen kann oder sich registrieren muss, falls er dies noch nicht erledigt hat. Natürlich gibt es da auch den umgekehrten Weg, d.h. ein Benutzer kann sich wieder abmelden oder sich sogar ganz aus dem System entfernen. Repräsentiert wird dies durch die beiden vom Menü aus erreichbaren Prozesse Logout und Unregister. Bei beiden Richtung ist außerdem zu beachten, dass die Registrierung und ihr Gegenstück den entsprechenden Anmeldevorgang beinhaltet, was dadurch wiederspiegelt, dass Register als direkten Nachfolger die Klasse Login hat und bei Unregister ist dies Logout. Außerdem sollte jeder registrierte Benutzer die Möglichkeit bekommen seine Einstellungen innerhalb der Anwendung zu ändern, ohne den Umweg über das Abmelden und Neuregistrieren zu gehen. Für diese Möglichkeit steht die Prozessklasse ChangeSettings.

Eine der wichtigsten Funktionalitäten von Philoponella ist die Möglichkeit für den Benutzer eigene Favoritenlisten und Seiteninformationen zu erstellen. Daher ist vom Menü aus eine Listenklasse namens FavoriteIndex erreichbar, welche wiederum zur Navigationsklasse des Linkinformationenabschnitts führt. Außerdem gehören dazu auch die beiden Prozesse RemoveFavorite und UpdateFavorite, welche die Verwaltung der Favoritenliste, also das Ändern des Favoritentyps bzw. das Entfernen aus der Liste, darstellen. Direkt vom PersonalMenu aus erreicht man außerdem den Bereich der Linkinformationen über die Verbindung zu dem Prozess CreateLinkInfo, welcher das Erstellen eines neuen Informationsobjekts zu einer Seite darstellt.

Die zweite Liste im persönlichen Bereich bildet die Freundesliste. Analog zu den Favoriten

ist auch hier eine Listenklasse vom Menü aus erreichbar. Sie trägt den Namen FriendIndex und führt zur Navigationsklasse des letzten Bereichs, welcher sich um die Details anderer Benutzer dreht. Ebenfalls gehören hierzu zwei Prozessklassen. RemoveFriend modelliert dabei das Entfernen einen Freundes aus der Liste, während UpdateFriend eine Statusänderung des Freundes repräsentiert.

Ein weiter hierzu gehörender Prozess ist SendMessage, welcher für das Senden einer Nachricht an einen anderen Benutzer steht. Dieser kann sowohl von persönlichen Menü aus als auch von der Navigationsklasse für andere Benutzer erreicht werden, wobei der zweite Fall die Möglichkeit eine Nachricht an den User direkt beim Betrachten seiner Detailansicht an ihn abzuschicken anzeigt. Damit die Liste der Nachrichten übersichtlich bleibt. gibt es schließliche die Option erhaltene Nachrichten zu löschen, was RemoveMessage darstellt.

### 3.3.3 Linkinformationen

Dieser Bereich der Navigationsstruktur dreht sich um die Linkinformationen. Wie gewohnt ist hierin zunächst eine Navigationsklasse mit dem Namen **LinkDetails** und ein Menü mit dem Namen LinkMenu enthalten. Von Start aus ist dieser Bereich über die Verbindung zu dem Prozess CheckLinkStatus erreichbar, welcher seinerseits zu LinkDetails führt. Dabei modelliert CheckLinkStatus die Überprüfung der Verfügbarkeit der Seite bei jedem Aufruf der Seiteninformationen.

Weiterhin kann man auch vom persönlichen Abschnitt des Navigationsmodells diesen erreichen. Dies geschieht entweder direkt vom FavoriteIndex aus oder über die Verbindung zu der Prozessklasse CreateLinkInfo. Die zweite Option stellt die Möglichkeit eines Benutzers eigene Seiteninformationen anzulegen. Zu diesem Vorgang gehört bei Bedarf auch das Erstellen einer neuen Kategorie für die neue Seiteninformation, was durch den Prozess CreateCategory modelliert wird.

Innerhalb dieses Navigationsstrukturteils gibt es noch drei Prozessklassen, welche keine Verbindung zu anderen Abschnitten bilden, d.h. sie sind von LinkMenu erreichbar und führen nur zu LinkDetails. Als erstes kann der Benutzer beim Betrachten der Linkinformationen diese Seite zu seinen Favoriten hinzufügen. Diesen Vorgang wird durch den Prozess AddFavorite repräsentiert. Darüber hinaus kann ein Benutzer die einzelnen Linkinformationen durch hinzufügen von Kommentaren und Bildern erweitern, was durch die beiden Prozessklassen AddComment und AddImage verkörpert wird.

Eine besondere Prozessklasse innerhalb der Linkinformatitionsteils ist RemoveUnusedLink, welche von dem Prozess RemoveFavorite des Abschnitts für persönliche Optionen aus erreichbar ist. Die Idee dahinter ist, dass die Linkinformationen, welche von keinem Benutzer mehr gebraucht werden und somit für keinen mehr ein Favorit sind, gelöscht werden.

### 3.3.4 Details anderer Benutzer

Der letzte Abschnitt der Navigation ist der Bereich über die Details anderer Benutzer. Hier lautet die Navigationsklasse **UserDetails** und wie immer gibt es das dazu passende Menü UserMenu. Da dieser Bereich nur dazu dient dem Benutzer das Betrachten anderer Benutzerprofile bzw. des öffentlichen Teil eines Benutzerprofils zu ermöglichen, gibt es hier nur eine Prozessklasse. Diese trägt die Bezeichnung AddFriend und dient dazu das Hinzufügen von Benutzern zur eigenen Freundesliste zu modellieren. Zuletzt ist es einem Benutzer erlaubt den öffentlichen Teil der Freundes- und Favoritenliste eines anderen Benutzers einzusehen, was durch zwei Listenklassen repräsentiert wird, welche die amüsanten Namen FriendFavoriteList und FriendFriendList tragen.

## 3.4 Die Prozesse von Philoponella

Nachdem im Navigationsmodell die für Philoponella notwendigen Prozesse aufgezählt wurden, werden diese in diesem Abschnitt näher beschrieben. Dazu verwendet man in UWE zwei Arten von Diagrammen: Die Prozessstruktur beschreibt die Beziehungen zwischen den Prozessen und die Aktivitätsdiagramme beschreiben die Abläufe innerhalb der jeweiligen Prozesse. Hier könnte man wegen der Ähnlichkeit der Struktur die gleiche Aufteilung in die drei Gruppen (Persönliche Optionen, Linkinformationen und Details anderer Benutzer) wählen wie bei der Navigation, jedoch bietet es sich hier mehr die Prozesse nach Ähnlichkeit zu sortieren. Dies führt zu einer Unterteilung in vier Gruppen: Benutzerauthentifizierung, Listenverwaltung, Benutzerprozesse und Informationsveränderung.

### 3.4.1 Benutzerauthentifizierung

Die erste Gruppe der Prozesse behandelt die Benutzerauthentifizierung, also die richtige Zuweisung und Verwaltung der einzelnen Benutzer. Da alle diese Prozesse mit dem An- und Abmelden eines registrierten Benutzers zusammenhängen liegen diese alle in Modell in der Gruppe der privaten Ansicht(PersonalDetails).

Dazu gehört zunächst die Registrierung, welche durch den Prozess **Register** beschrieben und in Abbildung 3.4 dargestellt wird. Hierzu gibt in Philoponella der Benutzer einen Namen, seine Emailadresse, sein Passwort und die Sprachen, die er beherrscht ein. Zunächst muss das System die eingegebenen Daten überprüfen. Dabei dürfen der Benutzername und die Emailadresse nicht schon in Philoponella gespeichert sein und das Passwort muss eine Mindestlänge enthalten. Erst bei korrekter Eingabe kann der Benutzer diese bestätigen, worauf der Benutzer gespeichert und eingeloggt wird. Obwohl innerhalb dieses Prozesses die

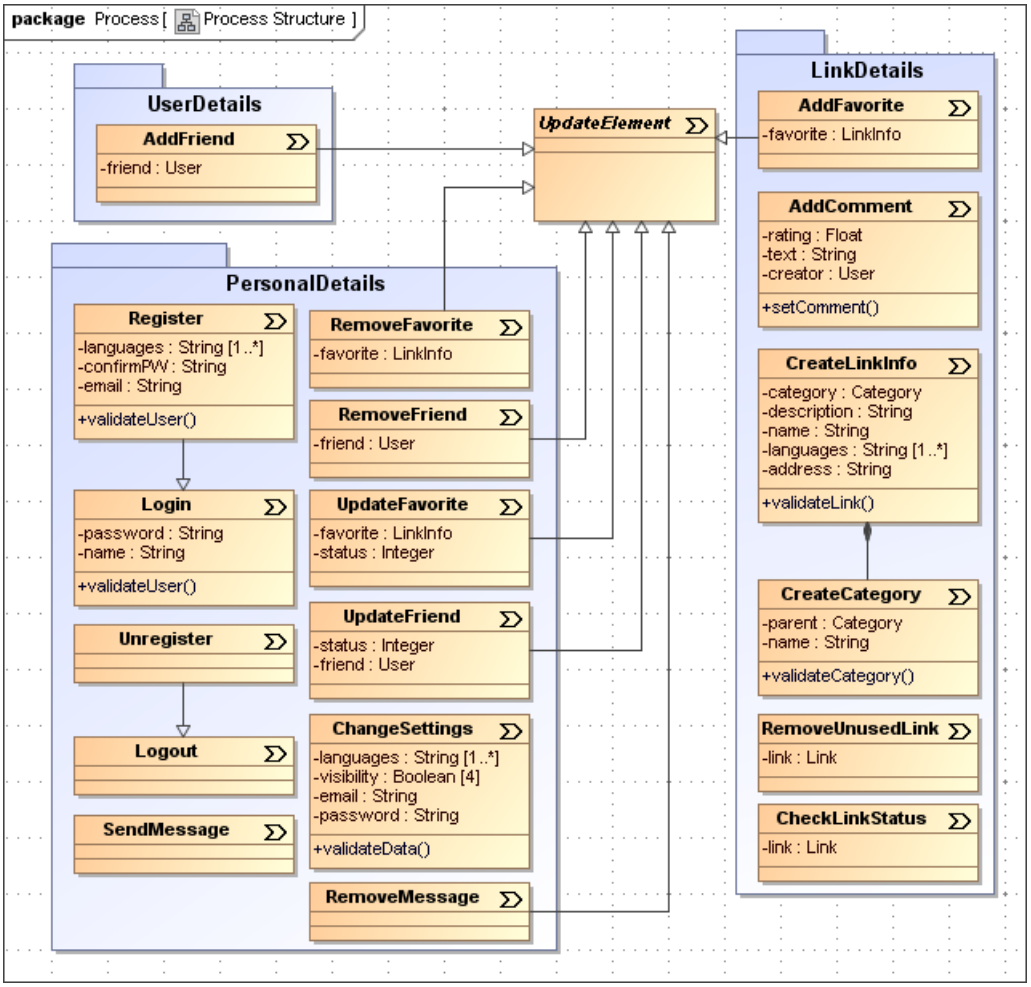


Abbildung 3.3: Philoponellas Prozessstruktur

Anmeldung des Benutzers eine Aktion darstellt, wird diese nicht durch eine komplexe Aktion dargestellt, welche durch das Aktivitätsdiagramm des entsprechenden Prozesses beschrieben wird. Denn der als nächstes beschriebene Prozess beschreibt vor allem die zur Anmeldung nötigen Benutzereingaben, während sie hier bereits durch die Registrierung vorliegen und hier daher nur eine Systemaktion benötigt wird. Zuletzt wechselt Philoponella die Ansicht entweder bei erfolgreicher Registrierung zu den privaten Details oder zurück zur Startansicht bei Abbruch.

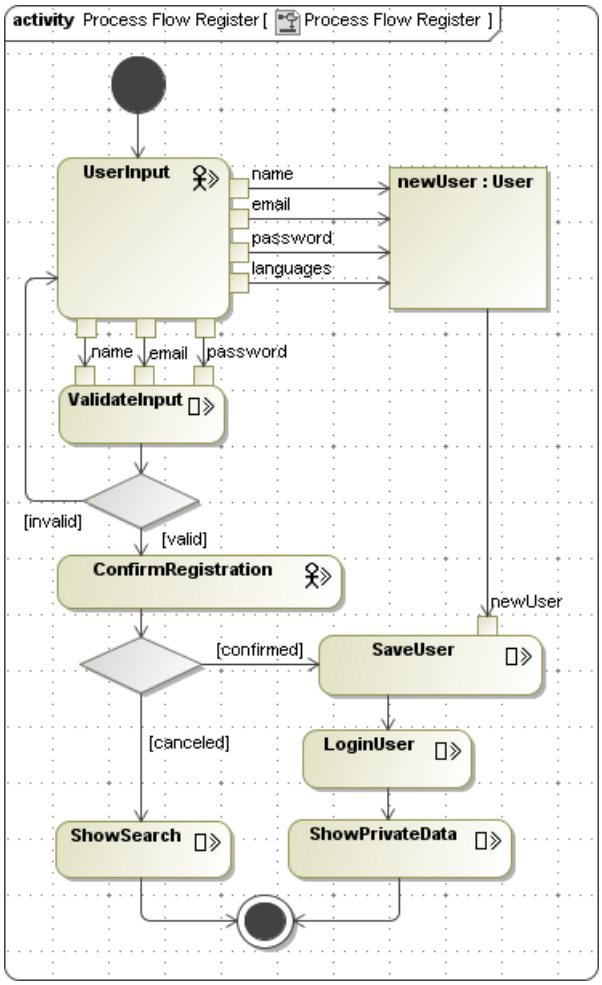


Abbildung 3.4: Das Prozessflussdiagramm zu Philoponellas Prozess Register

Ist der Benutzer bereits registriert kann er sich wie in **Login** dargestellt auch direkt einloggen. Dieser ist mit Register durch eine Aggregation verbunden, da eine Registrierung immer auch das Einloggen beinhaltet, obwohl wie bereits beschrieben dieses nicht direkt im Aktivitätsdiagramm enthalten ist. Dazu muss er einfach seinen Namen und das Passwort eingeben, woraufhin Philoponella die Daten überprüft. Bei Erfolg wird der Benutzer eingeloggt und die Ansicht wechselt zu den privaten Benutzerdetails, während bei einem Fehlschlag der

ganze Vorgang abgebrochen wird.

Die beiden übrigen Vorgänge sind sehr einfach aufgebaut. Beim durch **Logout** beschriebenen Abmelden wird der Benutzer nachdem er die Aktion ausgewählt hat einfach vom System ausgeloggt, während im Prozess **Unregister** die komplette Entfernung des Benutzers aus dem System modelliert wird, wobei dieser nach dem ausloggen einfach noch zusätzlich aus dem System entfernt werden muss. Wie bei ihren Gegenstücken sind auch diese beiden Prozesse durch eine Aggregation verbunden, da ein Benutzer vor dem Löschen aus dem System immer auch ausgeloggt sein muss. Dies spiegelt sich auch in den beiden dazugehörigen Aktivitätsdiagrammen. Bei Logout enthält dieses gerade einmal eine den Vorgang darstellende Systemaktion, ebenso verhält es sich in Unregister, nur dass der Abmeldevorgang durch eine komplexe Aktion ebenfalls darin enthalten ist.

### 3.4.2 Listenverwaltung

Bei der Listenverwaltung gibt es sechs sehr ähnliche Prozesse zur Veränderung der Favoriten- oder der Freundesliste. Hinzu kommt noch ein Prozess für die Nachrichtenliste. Man kann sie nicht in einen Bereich der Anwendung einordnen, da sie sowohl bei der Ansicht von Benutzerdetails als auch bei den Linkinfomationen zur Anwendung kommen. Dabei kann man jeder der beiden Listen ein Element hinzufügen, es entfernen oder den Status des Elements ändern. Für die Favoritenliste wird dies durch die Prozesse AddFavorite, RemoveFavorite und UpdateFavorite beschrieben, während bei der Freundesliste dafür die Prozesse AddFriend, RemoveFriend und UpdateFriend zuständig sind. Außerdem wird durch RemoveMessage der Umstand modelliert, dass der Benutzer auch Elemente aus der Nachrichtenliste entfernen kann.

Da diese jedoch beinahe identisch sind, sind sie alle als eine Unterklasse des Prozesses **UpdateElement** modelliert und dieser beschreibt den allgemeinen Vorgang der allen zu Grunde liegt. Auch gibt es nur von diesem hier eine Abbildung und zwar das Bild 3.5. Dabei wählt der Benutzer zunächst ein Element(LinkInfo oder User) aus. In Anschluss holt sich das System die bisherige Elementliste(favorites oder friends) des Benutzers zusammen mit dem gerade angegebenen Element, verändert sie entsprechend und speichert die aktualisierte Liste wieder im Benutzerobjekt ab. Dies wird im Aktivitätsdiagramm durch eine Systemaktion dargestellt, welche neben mit der Vorgängeraktion verbundenen Eingabepin für das ausgewählte Element sowohl einen Eingabepin wie auch einen Ausgabepin verbunden mit dem selben Objektknoten vom Typ User enthält. Nachdem dies geschehen ist passt Philoponella die Ansicht entsprechend an.

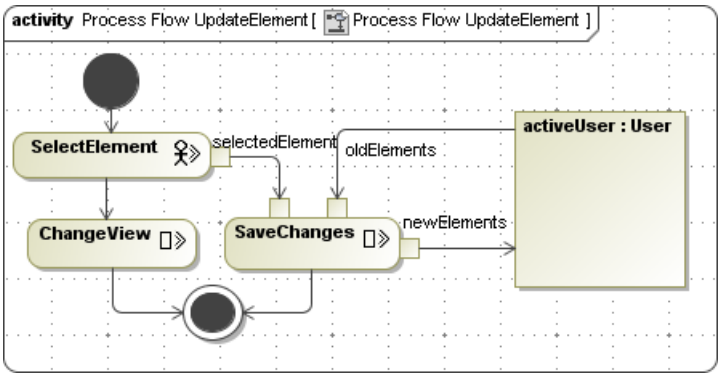


Abbildung 3.5: Das Prozessflussdiagramm zu Philoponellas Prozess UpdateElement

### 3.4.3 Benutzerprozesse

Um den Bereich der Prozesse für persönliche Details vollständig zu machen fehlen jetzt nur noch zwei Benutzeraktivitäten.

Zunächst kann der aktive Benutzer eine Nachricht an andere verschicken, was durch den Prozess **SendMessage** beschrieben wird. Hierbei wird zunächst das Formular vom Absender ausgefüllt, indem er den Empfängernamen(recipient), den Betreff der Nachricht(title) und die Nachricht selbst(text) eingibt. Die Anwendung überprüft danach, ob ein Benutzer mit solchem Namen existiert und speichert bei Erfolg die Nachricht für diesen ab. Der ganze Vorgang wird im Prozessflussdiagramm in Abbildung 3.6 präsentiert.

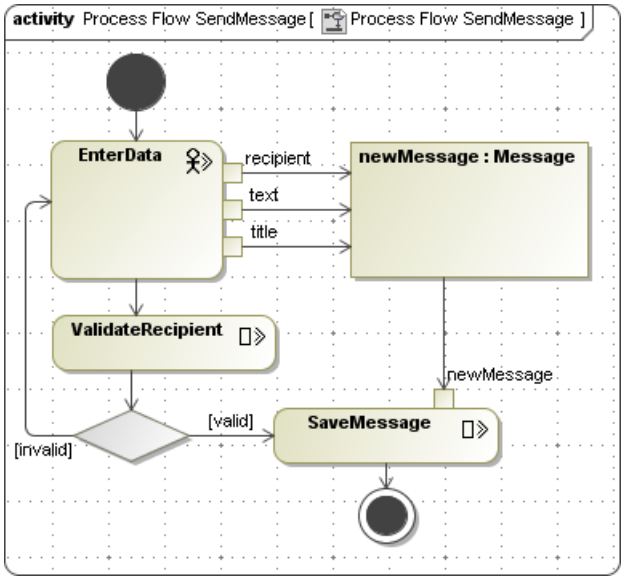


Abbildung 3.6: Das Prozessflussdiagramm zu Philoponellas Prozess SendMessage

Außerdem kann der Benutzer natürlich auch seine persönlichen Details verändern. Dies

beschreibt der Prozess **ChangeSettings**, wobei hier zu Beginn der Benutzer seine Daten eingibt. Sie umfassen die Emailadresse, das Passwort, die Spracheinstellungen und die Auswahl, ob die ignorierten Links und Benutzer angezeigt werden sollen. Wiederrum überprüft die Anwendung, ob die Adresse und das Passwort in Ordnung sind und lässt den Benutzer erst bei Erfolg den Abschluss des Vorgangs bestätigen. Geschieht dies, so speichert Philoponella die neuen Daten ab und der Dialog wird nach Abschluss dieser Aktion wie auch bei einem Abbruch geschlossen.

### 3.4.4 Informationsveränderung

Innerhalb der letzten Gruppe beschäftigen sich alle Prozesse mit dem Hinzufügen, dem Entfernen und dem Verändern der Seiteninformationen, wobei allerdings die Prozesse für die drei Listen bereits separat vorgestellt wurden. Alle diese Prozesse befinden sich im Bereich der Anwendung für die Detailansicht für Seiten(LinkDetails).

Hier gibt es zunächst die beiden einfachen, systeminternen Prozesse CheckLinkStatus und RemoveUnusedLinks. **CheckLinkStatus** modelliert dabei die Überprüfung der Verfügbarkeit einer Seite, bevor für diese die Details angezeigt werden. Dazu holt sich Philoponella zunächst die Adresse der entsprechenden Seite und speichert den überprüften Verbindungsstatus in connectionStatus. Danach lädt es das Attribut inactiveDays aus dem Seitenobjekt und speichert die veränderten Daten wieder.

Bei **RemoveUnusedLinks** werden hingegen die ungenutzten Seiteninformationen aus dem System entfernt. Es beginnt mit der Auswahl einer LinkInfo, was bei Philoponella der gerade aus einer Favoritenliste entfernte Eintrag ist. Zu diesem wird geprüft, ob diese Seite noch bei einem anderen Benutzer in der Favoritenliste eingetragen ist. Falls das der Fall ist, bricht der Vorgang hier ab. Ist diese Seite jedoch für keinen Benutzer mehr ein Favorit, so wird sie als überflüssig behandelt und aus dem System entfern.

Der nächste Prozess ist **AddComment**, der Repräsentativ für diese Gruppe in der Abbildung 3.7 gezeigt wird. Er benötigt nun wieder die Interaktion mit einem Benutzer, da er das Hinzufügen eines Kommentars an eine Linkinformation modelliert. Dazu wird zunächst der alte Kommentar dieses Benutzers zu diesem Informationsobjekt geladen und dem Benutzer präsentiert, was durch ein Objektknoten modelliert wird, welcher die durch Eingabepins an der darauf folgenden Aktion repräsentierten Daten: Kommentartext, Bewertung und Sprache beinhaltet. Diese Aktion stell nun das Aktualisieren der Daten durch den Benutzer dar und die ebenfalls zum Stereotyp *«userAction»* gehörende Nachfolgeaktion repräsentiert die darauf folgende Bestätigung zum Überschreiben der Daten. Zwar könnte man hier eventuell diese beiden Stereotypen zusammenfassen, jedoch wird so die Notwendigkeit einer Überschreibebestätigung besser hervorgehoben. Schließlich werden auch in diesem Prozess die

Daten gespeichert und der Dialog geschlossen.

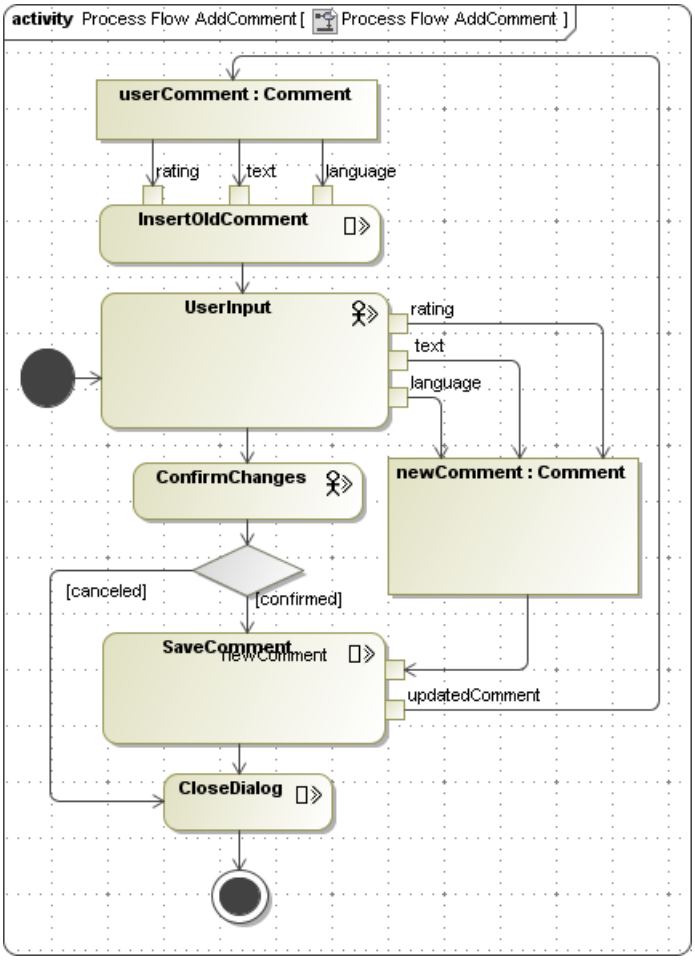


Abbildung 3.7: Das Prozessflussdiagramm zu Philoponellas Prozess AddComment

Als letztes bleiben noch das Erstellen der Seiteninformationsobjekte und der Kategorien, welche durch CreateLinkInfo und CreateCategory beschrieben werden. Dabei kann eine Kategorie bei Bedarf während des Erstellens einer neuen Information erzeugt werden, was durch die Kompositionsbeziehung der beiden Prozessklassen modelliert wird. **CreateCategory** beginnt dabei mit der Auswahl der Elternkategorie und der Eingabe eines Namens durch den Benutzer. Daraufhin überprüft Philoponella, ob es schon eine Kategorie mit dem gleiche Namen innerhalb der Elternkategorie gibt. Ist dies nicht der Fall, dann wird die neue Kategorie gespeichert und die Anwendung kehrt zum Erstellen einer neuen Seiteninformation zurück. Bei Abbruch wird selbstverständlich der Dialog ohne einen Speichervorgang beendet.

Zuletzt wird der Vorgang zur Erzeugung einer neuen Seiteninformation durch den Prozess **CreateLinkInfo** modelliert. Auch dieses beginnt mit einer Benutzereingabe, welche den Namen, die Adresse, die Sprachen und die Beschreibungen einer Seite beinhaltet. Außerdem muss dafür eine Kategorie eingegeben werden und falls der Benutzer keine passende findet, kann er wie

in CreateCategory beschrieben eine neue Kategorie erzeugen. Im Aktivitätsmodell wird es wie üblich durch eine entsprechende komplexe Aktion dargestellt, wobei diese allerdings sowohl von der Aktion UserInput erreichbar ist aber auch eine Verbindung in Gegenrichtung darstellt. Da keine zwei gleichen Adressen in Philoponella vorkommen dürfen, wird nach der Eingabe die Adresse überprüft und nur wenn sie noch nicht gespeichert ist kann der Benutzer den Vorgang bestätigen. Falls er dies getan hat, speichert das System die neue Information und öffnet die dazu passende Ansicht, ansonsten kehrt es zum Ausgangspunkt zurück.

## 3.5 Philoponellas Präsentationsmodell

Das Präsentationsmodell von UWE stellt die Ausgabe der Webanwendung dar. Ohne auf die Layoutdetails einzugehen enthält dieses alle in der Webseite vorkommenden Elemente einer Benutzeroberfläche. Da das dazugehörende Diagramm sehr umfangreich ist, wurde dieses auf die Abbildungen 3.8 und 3.9 auf den Seiten 48 und 49 aufgeteilt.

Die Grundlage bilden hier die Klassen des Stereotyps *«presentationPage»* und in Philoponella gibt es davon nur eine. Sie trägt den Namen **Home** und ist der Ausgangspunkt der ganzen Anwendung.

Diese Hauptseite enthält vier Darstellungsgruppen, von denen drei die verschiedenen Sichten der Anwendung bilden und da immer nur jeweils eine davon zu sehen ist, sind sie in einer Gruppe zur alternativen Darstellung genannt MainAlternatives zusammengelegt. Diese sind die Ansicht zum Durchstöbern und Suchen von Webseiten namens LinkSearch, der Seite für sowohl eigene als auch fremde Benutzerprofile mit dem Namen UserDetails und LinkDetails, welches die Ansicht der Informationsseite zu einer Internetadresse modelliert. Als vierte Präsentationsgruppe ist noch NavigationBar zu erwähnen, welche den Teil der Navigationsleiste modelliert, der immer zu sehen sein soll.

Außerdem sind in dieser Alternativengruppe alle von Philoponella benötigten Formulare enthalten. Davon gibt er hier fünf: Registration zur Registrierung neuer Benutzer, CreateLinkInfo zum Erstellen neuer Seiteninformationen, CreateCategory zum Erstellen einer neuen Kategorie, AddComment, um zu einer Linkinformation einen Kommentar abzugeben und SendMessage, um eine Nachricht an einen Benutzer zu schicken.

### 3.5.1 Navigationsleiste

Der Name NavigationBar oder Navigationsleiste ist für die erste und einfachste Präsentationsgruppe vielleicht etwas irreführend. Zum einen muss diese in der Webanwendung nicht zwingend als Leiste implementiert werden und zum anderen stellt diese nur einen

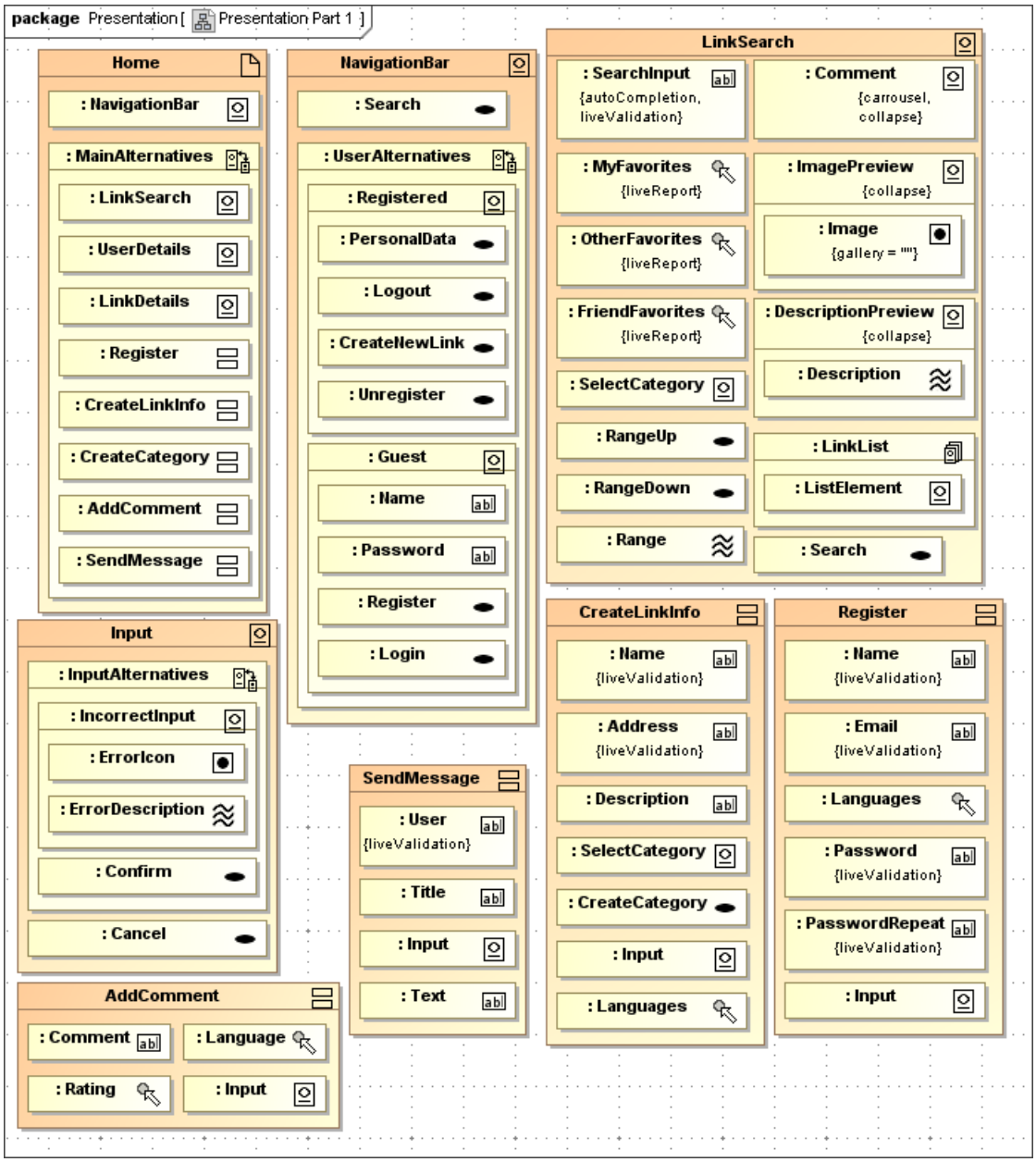


Abbildung 3.8: Erster Teil von Philoponellas Präsentationsmodell

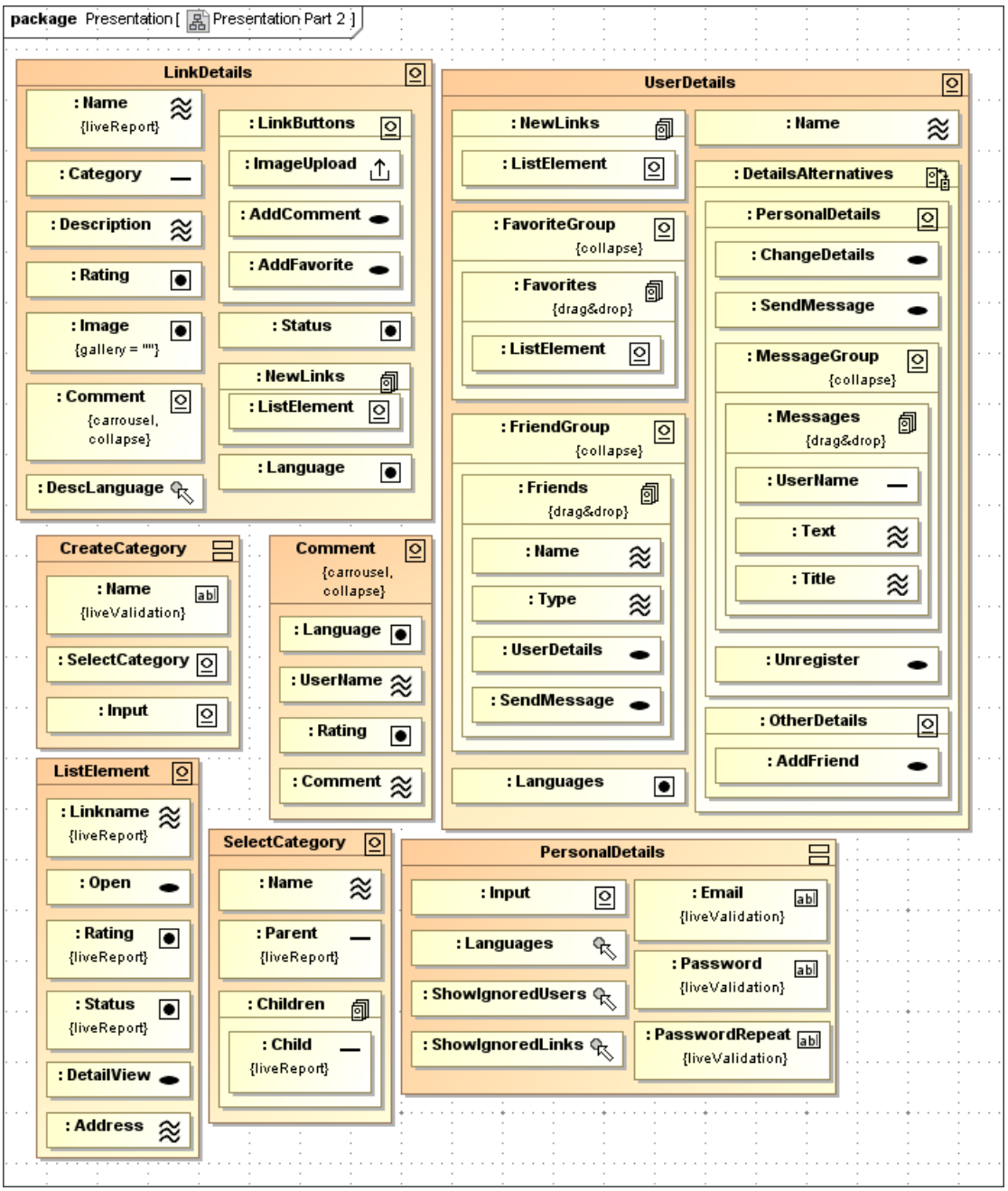


Abbildung 3.9: Zweiter Teil von Philoponellas Präsentationsmodell

Ausschnitt der Navigationsmöglichkeiten dar, die für die An- und Abmeldung zuständig sind, die Möglichkeit zu der Standardansicht (Liste der Webseiten) zurückzukehren und neue Einträge zu erstellen. Für das letztere reicht der Knopf Search aus. Die An- und Abmeldung erfordert da schon zwei Präsentationsgruppen: Guest und Registered. Da sich diese beiden Gruppen gegenseitig ausschließen sind sie in der Alternativgruppe UserAlternatives zusammengefasst.

Bevor der Besucher der Webseite angemeldet ist soll er hier entweder sich einloggen können oder falls er noch nicht registriert ist dies nachholen. Dafür stellt Guest die Elemente bereit. Für die Anmeldung gibt es zwei Textfelder Name und Password zur Eingabe des Namens und des Passworts sowie einen Knopf mit dem Titel Login. Die Registrierung ist etwas komplexer und kann nicht mehr innerhalb der Navigationsleiste erledigt werden. Daher gibt es einen Knopf mit der Bezeichnung Register, welcher das Registrationsformular Registration zur Anzeige bringen soll.

Im Formular Registration befinden sich dann alle Elemente zur Eingabe der für die Registrierung notwendigen Daten. Dies umfasst zunächst einmal die Eingabe des Namens im Feld Name, der Emailadresse im Feld Email und des Passworts in den Feldern Password und PasswordRepeat. Zwei Felder für das Passwort sind hier sinnvoll, damit der Benutzer beim Eingeben sich nicht so leicht vertippen kann. Zuletzt kann man bei der Registrierung noch die gesprochenen Sprachen im Auswahlelement Languages festlegen.

Außerdem darf hier die Präsentationsgruppe Input nicht fehlen. Diese kommt bei allen Formularen in diesem Modell zur Anwendung und hat den Zweck dem Benutzer die Eingabefehler zu melden, was durch die beiden im der Gruppe IncorrectInput zusammengefassten Elemente ErrorIcon und ErrorDescription geschiet oder ihm zu erlauben das Formular mit dem Knopf Confirm abzusenden oder mit Cancel zu verwerfen. Da die Anzeige der Gruppe zur Fehlermeldung und der Bestätigungsknopf nicht gleichzeitig dargestellt werden sollten, sind diese nochmals in der Alternativengruppe InputAlternatives zusammengefasst.

Nachdem sich der Benutzer angemeldet hat ändern sich seine Optionen hin zum Betrachten der eigenen Profilseite oder dem Abmelden. Die dafür benötigten Elemente werden durch Registered bereitgestellt, wobei für beide jeweils ein Knopf ausreichend ist.

Schließlich hat ein angemeldeter Benutzer die Möglichkeit eine neue Informationsseite anzulegen, was durch den Knopf CreateNewLink eingeleitet werden soll. Dies führt zum Formular CreateLinkInfo, welches genau diesen Zweck erfüllt. Zunächst enthält es die bereits besprochene Gruppe Input und die Gruppe SelectCategory zur Auswahl einer Kategorie. Diese enthält zunächst einmal für die Identifikation der aktuellen Kategorie den Text Name, dann den Anchor Parent zur Auswahl der Elternkategorie und das iterative Element Children, welches wiederum den Anchor Child zur Auswahl einer Kindkategorie enthält. Sowohl Parent als auch Children verfügen dabei über das Merkmal *liveReport*. Als Eingaben benötigt man hier einen Namen, die

Seitenadresse und eine Beschreibung, welche durch die Textfelder Name, Address und Description dargestellt wird und die durch das Auswahlobjekt Languages dargestellten Sprachen.

Außerdem wird die Möglichkeit des Benutzers bei Bedarf einen neue Kategorie anzulegen durch den Knopf CreateCategorie modelliert. Dieser Knopf führt seinerseits zu dem für diesen Prozess zuständigem Formular CreateCategory. Wie auch CreateLinkInfo enthält es Input und SelectCategory und darüber hinaus nur das Textfeld Name, das zur Eingabe des Kategorienamens dient. In beiden Formularen tragen die Elemente Name und beim ersten auch noch Address das RIA-Merkmal *liveValidation*, was die Überprüfung dieser Daten auf Gültigkeit modellieren soll.

### 3.5.2 Linkliste

Zur Darstellung der Seitenadressen durch eine Liste, der Präsentation ausgewählter Details einer Seite und die Suche nach bestimmten Seiten innerhalb dieser wird durch die Gruppe LinkSearch erledigt.

Zur Einstellung der Suchkriterien verfügt diese Gruppe über vier Möglichkeiten und über einen Knopf mit dem Namen Search, welcher zum Abschicken der Suchparameter dient. Zuerst kann der Benutzer einen Suchbegriff eingeben. Hierfür enthält LinkSearch das Feld SearchInput, welches über die beiden RIA-Merkmale *autoSuggestion* verfügt.

Zusätzlich kann der Benutzer den Typ der zu suchenden Adressen einschränken, indem er über die drei Auswahlelemente MyFavorites, FriendFavorites und OtherFavorites die eigenen Favoriten, die Favoriten seiner Freunde oder alle anderen Adressen von der Suche ausschließt bzw. sie wieder einbindet. Alle drei verfügen dabei über die RIA-Eigenschaft *liveReport*, wodurch modelliert werden soll, dass der Benutzer die Anzahl der Linkinformationen hinter diesem Element bei Bedarf angezeigt kriegt.

Für die dritte Möglichkeit gibt es die beiden Knopfelemente RangeUp und RangeDown. Diese ermöglichen den ausgegebenen Bereich der Liste nach oben oder unten zu verschieben. Außerdem gehört hierzu auch das Textelement Range, welches den gerade aktuellen Bereich angibt.

Als letztes gibt es für den Benutzer die Option die Kategorie für die Suche auszuwählen, wozu LinkSearch die bereits besprochene Gruppe SelectCategory enthält.

Weiterhin beinhaltet die Gruppe eine iterative Gruppe LinkList, welcher die Liste der Linkinformationen darstellt und die Gruppe LinkElement beinhaltet. Dazu enthält LinkElement den Text Linkname, welcher den Namen der Seite liefert und das Merkmal *liveReport* enthält. Über diese RIA-Eigenschaft werden dem Benutzer Details über den Entstehungszeitpunkt angezeigt. Die beiden Bildelemente Rating zur graphischen Ausgabe der Gesamtwertung und

Status zur Präsentation des letzten Verbindungsstatus der Seite, welche ebenfalls die gleiche RIA-Eigenschaft zur näheren Beschreibung der angezeigten Information besitzen. Für die Interaktion enthält diese Gruppe die beiden Knöpfe Detailview und Open, welche es dem Benutzer ermöglichen sollen entweder die Detailansicht für diese Seite zu öffnen oder gleich auf diese Adresse weitergeleitet zu werden.

Als letztes sind hierin drei Gruppen enthalten, welche zur Präsentation bestimmter Details einer Seite dienen und mit dem RIA-Merkmal *collapse* ausgestattet sind, da der Benutzer sich aussuchen kann, welche davon er sehen will. Die erste Gruppe repräsentiert die Beschreibung und heißt daher DescriptionPreview. Sie enthält einzig und allein das Textelement Description. Als zweites kommt die Gruppe ImagePreview, welche das Bildelement Image enthält. Dieses repräsentiert die Vorschaubilder für die Seite und trägt das RIA-Merkmal *gallery*, da es dem Benutzer möglich sein soll alle Vorschaubilder bei Bedarf in einer Galerieansicht zu betrachten. Die letzte Gruppe ist Comment und verfügt zusätzlich zu *collapse* auch über das Merkmal *carrousel*, da man durch die einzelnen Kommentare bequem navigieren können soll. Es enthält zunächst das Textelement Comment, welches den eigentlichen Kommentar enthält. Dazu kommt noch der Name des Verfassers in UserName und die beiden graphischen Darstellungen der verwendeten Sprache durch Language und der Wertung durch Rating.

### 3.5.3 Benutzerseite

Die umfangreichste Gruppe ist UserDetails, was unter anderem damit zusammenhängt, dass diese Gruppe sowohl für die Repräsentation der Profildetails des gerade aktiven Benutzers, als auch für die Details der anderen Benutzer zuständig ist. Zunächst enthält diese das Textelement Name und das Bild Language, welche den Namen des Benutzers und seine gesprochenen Sprachen repräsentieren. Außerdem hat diese auch die iterative Gruppe NewLinks, welche den Abschnitt der Ansichtsseite repräsentiert worin die Anwendung dem Benutzer neue Seiteninformationen anzeigt.

Darüber hinaus enthält diese Ansicht zwei Gruppen zur Darstellung der Listen, die für alle Benutzer zumindest teilweise sichtbar sind: FavoriteGroup und FriendGroup. Beide haben dabei das RIA-Merkmal *collapse*, da der Benutzer sie nach Bedarf öffnen und schließen kann. FavoriteGroup enthält die iterative Gruppe Favorites, welche wiederum die bereits besprochene Gruppe ListElement enthält. Genauso enthält FriendGroup die iterative Gruppe Friends und diese enthält analog mit Name die Repräsentation des Benutzers durch den Namen. Außerdem haben sowohl Friends als auch Favorites das RIA-Merkmal *drag&drop*, da man die einzelnen Einträge dieser Gruppen in verschiedenen Kategorien ablegen können soll.

Als letztes gibt es die Alternativengruppe DetailAlternatives, welche dazu dient die beiden unterschiedlichen Gruppen für die eigene und fremde Profilseite anzuzeigen, da diese logischer

Weise nicht gleichzeitig sichtbar sein sollen. Die einfachere Gruppe ist OtherDetails, welche zur Darstellung der Anzeigeelemente auf fremden Profilseiten dient. Sie hat als ein einziges Element den Knopf AddFriend, welcher dem aktiven Benutzer das Hinzufügen dieses Benutzers als Freund ermöglicht.

Etwas komplexer ist die Ansicht auf der eigenen Seite, was sich in der Gruppe PersonalDetails widerspiegelt. Hier werden dem Benutzer vier Aktionen ermöglicht, was durch vier Knopfelemente repräsentiert wird. Als erstes kann sich der Benutzer sein Profil aus dem System löschen lassen, dies repräsentiert Unregister.

Weiterhin wird durch SendMessage die Erstellung einer Nachricht ermöglicht, indem diese zum Offnen des gleichnamigen Formulars führt. Dieses enthält wiederum die Textfelder für Titel und Inhalt Title und Text sowie ein Textfeld für den Namen des Empfängers User. Das letztere ist dabei mit dem RIA-Merkmal *liveValidation* ausgestattet, damit der Benutzer sofort eine Benachrichtigung bekommt, wenn er einen falschen Empfänger eingetragen hat. Schließlich enthält dieses auch die schon früher beschriebene Gruppe Input.

Zuletzt wird die Änderung der persönlichen Daten durch den Knopf ChangeDetails eingeleitet. Auch hier gibt es ein entsprechendes Formular mit der Bezeichnung PersonalDetails. Dieses entspricht ziemlich genau dem Formular für die Registrierung, nur das es kein Feld für den Namen enthält, da dem Benutzer nicht erlaubt werden soll seinen Benutzernamen zu ändern. Zusätzlich enthält es die beiden Auswahlfelder ShowIgnoredUsers und ShowIgnoredLinks, welche die Möglichkeit repräsentieren die Liste der ignorierten Benutzer oder Seiten für andere sichtbar zu machen oder dies zu unterbinden.

Außerdem enthält PersonalDetails eine weitere Gruppe mit dem Merkmal *collapse*. Diese ist MessageGroup und kann nur beim Betrachten der eigenen Seite geöffnet werden. Ansonsten ist diese aber analog zu den beiden zu Beginn dieses Abschnitts besprochenen Gruppen aufgebaut. Sie enthält ebenfalls eine iterative Gruppe namens Messages mit dem Merkmal *drag&drop*. Allerdings enthält diese Gruppe gleich die drei Textelemente welche den Titel, den Inhalt und den Verfasser der Nachricht darstellen.

### 3.5.4 Detailsicht

Der vierte und letzte Abschnitt des Repräsentationsmodells bildet die Gruppe LinkDetails. Diese enthält sämtliche Details einer Seite auf einen Blick und bietet außerdem einem registrierten Benutzer die Möglichkeit mehrere Aktionen auszuführen. Zunächst enthält dieses die Textelemente Name und Description, welche den Namen und die ausgewählte Beschreibung repräsentieren. Da es mehrere Beschreibungen in verschiedenen Sprachen geben kann, beinhaltet es außerdem zur Auswahl der Beschreibung das Element DescLanguage. Weiterhin gibt es hierin vier Bilder: Rating, Status, Language und Image. Dadurch soll graphisch

die Gesamtbewertung, der Verbindungsstatus, die Sprachen und die Vorschaubilder der Seite dargestellt werden. Letztes Element hat dabei analog zum in der Gruppe LinkSearch enthaltenen das RIA-Merkmal *gallery*. Auch die Gruppe Comment entspricht genau der gleichnamigen Gruppe aus LinkSearch. Schließlich wird die Kategorie der Seite durch das Element Category modelliert.

Für registrierte Benutzer stehen hier auch drei Aktionen zur Verfügung. Durch den Knopf AddFavorite wird die Möglichkeit repräsentiert diese Seite in die Liste der Favoriten aufzunehmen. Ebenso stellt das Element ImageUpload das Hinzufügen eines Bildes zu dieser Linkinformation dar.

Darüber hinaus kann der Benutzer auch einen Kommentar zu dieser Seite verfassen, was durch den Knopf AddComment dargestellt wird. Zu diesem gehört das gleichnamige Formular, welches neben der üblichen Gruppe Input dem Benutzer die Eingabe des Kommentars in einer bestimmten Sprache und die zusätzliche Abgabe einer Wertung ermöglichen soll. Für die Eingabe des Kommentars soll dabei das Textfeld Comment dienen, während die Auswahlelemente Rating und Language zur Festlegung der Sprache und der Wertung dienen.

## 3.6 Adaptivität in Philoponella

Wie bereits erwähnt ist die Modellierung der Adaptivität in UWE noch nicht implementiert. Daher werden diese Vorgänge für Philoponella nicht in einem Modell festgehalten, sondern nur hier beschrieben. Dabei gibt es fünf Fälle, wo sich Philoponella dem aktuellen Benutzer anzupassen versucht. Diese sind außerdem alle nochmal auf Seite **??** in der Tabelle 3.1 aufgelistet.

### 3.6.1 Suchfeldvorschläge

Es ist mittlerweile zu einem Quasi-Standard geworden, dass die Suchfelder dem Benutzer Vorschläge für den Suchbegriff anbieten. Idealer Weise sind diese Vorschläge so gut es nur geht an die Bedürfnisse des Benutzers angepasst. Diese Anpassungsart fällt in einen Bereich der adaptiven Navigationsstruktur, welcher als globale Führung bezeichnet wird. Ziel hierbei ist es den Navigationspfad bis zum Ziel für den Benutzer abzukürzen, indem man ihm geeignete Links anbietet oder unpassende vorenthält.

In Philoponella findet es jedes Mal Anwendung, wenn der Benutzer den Suchbegriff im Feld ändert. Dann vergleicht das System den bisher eingegebenen Text mit allen bisher von diesem Benutzer eingegebenen Suchbegriffen und erstellt eine Liste mit denjenigen, welche mit dem bisher Eingetippten zusammenpassen. Dann werden diese nach Anzahl der Aufrufe

sortiert. Falls die Liste zu kurz ist kann das System auch die Suchbegriffe aller User heranziehen und auch hier die passenden Einträge in einer Liste speichern. Diese wird ebenfalls anhand der Aufrufe sortiert und an die bisherige Liste angehängt. Wenn diese finale Liste zu lang ist, kann sie auf eine vernünftige Länge beschnitten und dem Benutzer als Vorschläge zur Auswahl angezeigt werden.

### 3.6.2 Änderung der Linkliste

Eine weitere von den Suchmaschinen sehr bekannte Methode zur Anpassung ergab sich durch die oft gigantischen Mengen an Einträgen, die ein Benutzer als Antwort auf seine Suchanfrage bekommt. Hierbei wird die Liste der Einträge sortiert und eventuell auch unterschiedlich hervorgehoben, was zwei Techniken der Navigationsanpassung darstellt.

Jede Liste aus Seiteninformation wird in Philoponella so vor der Ausgabe verändert. Dabei bestimmt das System zunächst alle Linkinformationen innerhalb der Parameter und entfernt zusätzlich noch Seiten, welche nicht in einer vom Benutzer verstandenen Sprache angeboten werden. Danach werden diese anhand der Aufrufe durch den Benutzer sortiert, bei Gleichstand entscheidet die Anzahl der Aufrufe aller Freunde des Benutzers und falls noch immer die Reihenfolge nicht vollständig feststeht, werden alle Aufrufe einbezogen. Danach werden die unterschiedlichen Einträge zusätzlich noch auf drei verschiedene Weisen markiert. Hier wird zwischen nie besuchten Seiten, seit dem letzten Besuch veränderte Linkinformationen und bekannten, unveränderten Einträgen unterschieden.

### 3.6.3 Kategorienliste

Eine weitere kleinere Anpassung ist die Sortierung der Liste der Kategorien bei der Auswahl einer davon. Dies läuft ziemlich ähnlich zur vorher beschriebenen Methode, nur das hier keine Kategorien hervorgehoben werden.

Dazu speichert Philoponella jeden Aufruf einer Kategorie für den aktuellen Benutzer. Soll hingegen eine Kategorienliste angezeigt werden, wie dies zum Beispiel bei der Liste der Kinder einer Kategorie während der Suche nach Seiteninformationen der Fall ist, so werden diese nach der Anzahl der Aufrufe sortiert. Außerdem werden dem Benutzer die am häufigsten betrachteten Kategorien unabhängig von der aktuell gewählten Kategorie in einer separaten Auswahlliste präsentiert.

### 3.6.4 Anpassen der Ansicht für Informationsdetails

Die hier folgende Anpassung ist eine Kombination aus den im Vorgängerkapitel beschriebenen Techniken und einer Methode, welche in den Bereich der adaptiven Präsentation

einzuordnen ist und bei multilingualen Websystemen Anwendung findet. Bei diesen ist es meist notwendig die ausgegebenen Texte an die vom Benutzer beherrschten Sprachen anzugleichen.

Dieser Vorgang wird bei Philoponella eingeleitet, sobald die Anfrage zu einer Detailansicht einer Seiteninformation abgeschickt wird. Zunächst werden die Kommentare in einer Sprache, welcher der Benutzer nicht beherrscht entfernt. Danach werden die übrigen Kommentare sortiert, so dass noch nicht betrachtete Kommentare zu Beginn der Liste stehen. Danach werden die Kommentare anhand der Freundesliste hervorgehoben, um die Kommentare der Freunde besser erkennen zu können. Außerdem werden die Beschreibungen nach einer Version in einer für diesen Benutzer passenden Sprache gesucht und bei Erfolg ausgewählt.

### 3.6.5 Anpassung der Benutzeransicht

Auch hier kommen die vorher beschriebenen Techniken der Navigationsanpassung zum Zuge und werden daher nur kurz behandelt.

Beim Aufrufen einer Benutzeransicht in Philoponella werden die beiden Listen (Freunde und Favoriten) anhand der Zugriffshäufigkeit sortiert. Da die Elemente der Favoritenliste wiederum Linkinformationen sind, werden diese wie bei Liste in der Suchansicht markiert. Die Elemente der Freundesliste sind hingegen Benutzer, weswegen hier ein anderes Kriterium zur Anwendung kommt. Die einzelnen Freunde werden anhand ihrer Aktivität bzw. des Zeitpunkts ihres letzten Besuchs ausgezeichnet, wobei gerade eingeloggte Benutzer zusätzlich gekennzeichnet werden.

### 3.6.6 Einstellungen der Benutzeroberfläche

Eine weitere Art der Anpassung ist heute so stark bei Webanwendungen verbreitet, so dass sie nicht mehr als solche wahrgenommen wird. Die Rede ist von Anpassung des Inhalts anhand bisher vom diesem Benutzer vorgenommenen Einstellungen, was durch das einklappen der für den Benutzer als uninteressant eingestuften Daten erreicht wird. Diese Technik des adaptiven Inhalts wird als Stretchtext bezeichnet.

In Philoponella wird neben der Einstellung der Suchoptionen auf den Stand des letzten Besuchs beim erneuten Login, auch bei jedem Aufruf der Detailinformationen, wie der Seitendetails und der Benutzerdetails die auf- und zuklappbaren Präsentationselemente auf die von diesem Benutzer bevorzugte Position eingestellt. Dies bedeutet vor allem bei den Seiteninformationen die Kommentare, die Beschreibung und die Bildvorschau und bei den Benutzerdetails die Liste der Favoriten und die Freundesliste.

### 3.6.7 Neuheiten

Eine letzte, kleinere Form der Adaptivität, welche hier der Vollständigkeit halber beschrieben wird ist eine weitere Sortierung und Hervorhebung der Links. Es ist nämlich auch seit längerem üblich die neuen Inhalte einer Webanwendung zu präsentieren und diese dabei nach Relevanz für den Benutzer zu sortieren.

Dies geschieht innerhalb von Philoponella bei der Präsentation der neu hinzugekommenen Seiteninformationen, welche in der Seitendetail- und Benutzerdetailansicht präsentiert werden. Bei der Linkdetailansicht werden die Neuheiten anhand der Ähnlichkeit zur Kategorie der gerade betrachteten Seite sortiert, während beim Betrachten von Benutzerdetails diese anhand der Ähnlichkeit zur der Liste der Favoriten angeordnet werden.

| **Angepasstes Element** | **Anpassungsarten** | **Modell** |
|---|---|---|
| Suchfeldvorschläge | Sortieren | Inhalt |
| Liste der Suchergebnisse | Sortieren, Hervorheben, Entfernen | Navigation |
| Liste der Seitenkategorien | Sortieren, Einfügen | Navigation |
| Details der Seiteninformationen | Sortieren, Entfernen | Inhalt |
| Favoriten- & Freundesliste | Sortieren, Hervorheben | Navigation |
| Benutzeroberflächeneinstellungen | Wiederherstellen | Präsentation |
| Liste der neuen Seiteninformationen | Einfügen, Hervorheben | Navigation |

Tabelle 3.1: Auflistung der Adaptionen in Philoponella

# 4 Erweiterung des UWE-Profils

Nachdem alle Grundlagen abgehandelt wurden ist es nun endlich an der Zeit den Hauptzweck der Arbeit zu besprechen. Dieser ist die Erweiterung der Anforderungsanalyse, damit mehr Informationen für die Modellierung daraus gewonnen werden können. Zu der Erweiterung gehören dabei zunächst die im nachfolgenden Abschnitt beschriebenen Anwendungsfälle. Aber da diese die Anforderung einer Anwendung nicht genau genug beschreiben können, kommen noch die veränderten Elemente der Aktivitätsdiagramme hinzu. Zusätzlich zu der Erweiterung wird in diesem Kapitel außerdem die in UWE modellierte Anforderungsanalyse von Philoponella nachgereicht. Dies vervollständigt nicht nur die Modellierung von Philoponella, sondern zeigt dadurch auch den Einsatz der neuen Elemente anhand eines kompletten Beispiels.

## 4.1 Anforderungsmodellierung

Die grobe Form der Anforderungsanalyse bilden das Anwendungsfalldiagramm. Da die Anwendungsfälle dazu dienen verschiedene Vorgänge der Webanwendung darzustellen, gibt es zwei Möglichkeiten diese zu erweitern: Zunächst können hier die für die Adaption notwendigen Vorgänge aufgelistet und hervorgehoben werden. Weiterhin wird durch die weiteren Anwendungsfälle die grobe Navigationsstruktur der Anwendung beschrieben, jedoch fehlt eine Unterscheidung dieser Vorgänge zwischen Prozessen und reinen Navigationsvorgängen. Wie in UWE üblich findet dabei die Erweiterung der Anwendungsfälle statt, indem für diese neue, passende Stereotypen eingeführt werden. Außerdem wird wie bereits erwähnt im zweiten Teil dieses Abschnitts das Anwendungsfalldiagramm von Philoponella beschrieben.

### 4.1.1 Neue Stereotypen

Die Erweiterung von UWE besteht zunächst aus vier neuen Stereotypen für die Anwendungsfälle, welche in der Tabelle 4.1 aufgelistet sind. Diese orientieren sich zunächst an den Aktivitäten, die ein Benutzer innerhalb einer Webanwendung ausführen kann. Außerdem ist die Erweiterung des Metamodells um diese Stereotypen in einem Diagramm in Abbildung 4.1 auf Seite **??** angezeigt.

| Name | Symbol | Zweck |
|---|---|---|
| *«navigation»* |  | reine Präsentation an den Benutzer |
| *«process»* |  | Benutzereingaben verändern auch den Inhalt des Systems |
| *«adaptation»* |  | Anpassung des Systems an den Benutzer |
| *«observation»* |  | Beobachtung des Benutzers |

Tabelle 4.1: Neue Stereotypen für die Anwendungsfälle

Zunächst gibt es einen klassische Vorgang aus den Anfängen des Internets, welcher das reine Betrachten von Inhalten ist und keine Änderungen des Systeminhalts nach sich zieht. Oft bilden solche Aktivitäten die die zentralen Punkte des Anwendungsfalldiagramms und dienen als Anlaufpunkt für viele andere Aktivitäten. Diese werden dargestellt durch den bereits früher schon vorgeschlagenen Stereotyp ***«navigation»***. Der Name ist sinnvoll wegen der Ähnlichkeit zu dem Stereotyp *«navigationClass»* aus dem Navigationsdiagramm. Ebenfalls aus diesem Grund wird dieser Stereotyp durch das gleiche Symbol dargestellt, welches ein stehendes Rechteck ist. Die Natur dieses Stereotyps legt nahe, dass die so markierten Anwendungsfälle in der Regel nur sinnvoll in Verbindung mit einem menschlichen Akteur sind, da ein Websystem das reine Auslesen von Daten normalerweise nur durchführt, wenn es zur Präsentation für einen Benutzer benötigt wird.

Der zweite Anwendungsfall dient zur Darstellung von Vorgängen, bei denen der Benutzer auch Daten im System verändert. Auch dieser hat Ähnlichkeit zu einem Stereotyp aus der Modellierung der Navigationsstruktur. Dort werden Vorgänge, welche das Verändern von Daten beinhalten durch den Stereotyp *«processClass»* dargestellt, daher wird dieser Stereotyp für einen Anwendungsfall entsprechend als ***«process»*** bezeichnet. Analog zum ersten Anwendungsfall wurde hier ein breiter Pfeil als das Symbol zur Darstellung gewählt. Außer vielleicht bei statischen Seiten oder sehr einfachen Webanwendungen ist dieser Stereotyp der wohl am häufigsten vorkommender und kann er sowohl auf der Benutzerseite, als auch auf der Seite des Systems vorkommen.

Zusätzlich zu diesen beiden Anwendungsfällen werden noch zwei weitere Spezialfälle eingeführt und da ein wichtiger Punkt dieser Arbeit die Modellierung der Adaptivität während der Anforderungsanalyse ist, ist genau dies der Zweck der beiden folgenden Stereotypen.

Als erstes gibt es dabei den Stereotyp ***«adaptation»***, welcher die Vorgänge repräsentiert bei denen das System seinen Inhalt, seine Navigation oder seine Präsentation ändert. Das repräsentierende Symbol für diese Fälle bildet dabei ein fünfzackiger Stern. Da diese Vorgänge dazu dienen die Ausgabe der Webanwendung an den Benutzer anzupassen, kommen diese Anwendungsfälle immer nur als ein Teil der Anwendungsfälle, welche eine Darstellung von Daten modellieren. Dies bedeutet, dass jedes Vorkommen eines Anwendungsfalls vom Typ *«adaptation»* zu einem Anwendungsfall auf der Benutzerseite durch die Beziehung *«include»* verbunden werden muss. Hierbei sind zunächst natürlich die reinen Präsentationsfälle betroffen, aber auch die Anpassung der Prozessvorgänge ist nicht unmöglich.

Ergänzt wird *«adaptation»* durch sein Gegenstück: den Stereotyp ***«observation»***. Damit das System nämlich auch für verschiedene Benutzer brauchbare Anpassungen vornehmen kann braucht es zu jedem Benutzer die passenden Daten. Diese können entweder aus den bereits im System vorhandenen Daten gewonnen werden oder muss aus dem Verhalten des Benutzers herausgelesen werden. Zur Modellierung des zweiten Falls dient dieser Stereotyp. Dabei können diese Vorgänge vom einfachen Abspeichern von gewählten Navigationsübergängen bis zu komplizierten Berechnungen reichen. In Unterschied zum Stereotyp *«adaptation»*, welcher die Aktivitäten zwischen dem Auslesen und dem Anzeigen der Daten modelliert, finden die Berechnungen eines Vorgangs von Typ *«observation»* statt, bevor die Daten im System abgespeichert werden. Schließlich wäre noch das diesen Stereotyp repräsentierende Symbol zu erwähnen, welches eine stilisierte Darstellung eines Auges ist.

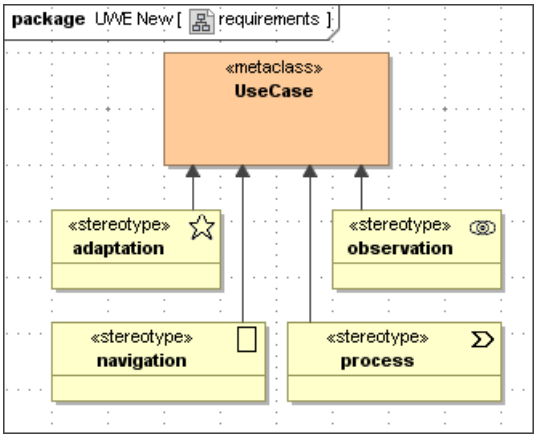


Abbildung 4.1: Die Erweiterung von UWE um die Anwendungsfallstereotypen

### 4.1.2 Überblick über Philoponellas Anwendungsfalldiagramm

Das Anwendungsfalldiagramm von Philoponella kann sehr klar in drei Bereiche geteilt werden. Die ersten beiden Teile bilden die Vorgänge an denen ein Benutzer beteiligt ist, was durch die beiden Akteure Guest und User modelliert wird. Für die normalen Besucher der

Seite, welche sich nicht registrieren wollen oder einfach noch nicht geschafft haben sich zu registrieren oder einzuloggen, gibt es den Akteur Guest. Hingegen für eingeloggte und somit auch registrierte Benutzer der Seite ist der Akteur User zuständig. Den letzten Bereich der Anwendungsfälle bildet die Anwendung selbst, d.h. in diesem Abschnitt werden die Vorgänge modelliert, an denen kein Benutzer beteiligt ist. modelliert wird dies durch den Akteur System. Das komplette Diagramm ist in Abbildung 4.2 auf Seite 62 dargestellt.

### 4.1.3 Anwendungsfälle für nicht registrierte Besucher

Der erste Teil der Anwendungsfälle ist auch der, den die Besucher als erstes zu sehen bekommen, denn er beinhaltet die Anwendungsfälle für nicht registrierte Besucher. Daher werden hier nur die Anwendungsfälle betrachtet, die durch eine Assoziation mit Guest verbunden sind.

Auch ein nicht angemeldeter Besucher kann den Großteil des Inhalts des Systems ansehen. Hierzu gehören zu aller erst die Listen der Seiteninformationen, welche ein Besucher durchsuchen kann. Diesen Vorgang modelliert der Anwendungsfall Browse LinkInfos, welcher den Stereotyp *«navigation»* besitzt, da beim Durchsuchen vielleicht die ausgegeben Daten verändert werden, dies aber keine Auswirkungen auf den Inhalt des Systems hat. Eine Aufgabe von Philoponella ist es das Durchsuchen der längeren Liste an Seiteninformationen zu erleichtern. Dazu müssen die Suchparameter angepasst werden, was durch Change BrowseSettings dargestellt wird. Meist tritt dieser Anwendungsfall als Teil von Browse LinkInfos auf, weswegen Change BrowseSettings diesen auch erweitert. Obwohl hierbei der Benutzer verschiedene Eingaben tätigen kann, ändert er dabei nur die ausgegeben Daten, nicht jedoch den Inhalt des Systems. Daher ist auch dieser Fall vom Typ *«navigation»*.

Außerdem gibt es in diesem Teil noch zwei weitere Anwendungsfälle, welche auch den Stereotyp *«navigation»* besitzen. Als erstes kann ein Besucher die in der Liste angezeigten Seiteninformationen auch im Detail betrachten, was durch den Anwendungsfall View LinkDetails modelliert wird. Ebenso kann ein Besucher auch die Details eines Benutzers ansehen. Dies stellt View UserDetails dar.

Schließlich braucht ein Besucher die Möglichkeit sich beim System anzumelden. Dazu kann er sich entweder registrieren, was mit dem Anwendungsfall Register dargestellt wird, oder sich anmelden, falls er sich bereits registriert hat, wozu der Fall Login existiert. Diese beiden Vorgänge sind vom Stereotyp *«process»*. Für Register ist dies offensichtlich, da ein Benutzer beim registrieren in die Datenbank des Systems eingetragen wird. Aber auch der Fall Login verdient diesen Typ, obwohl bei diesem keine Daten auf Dauer verändert werden, denn

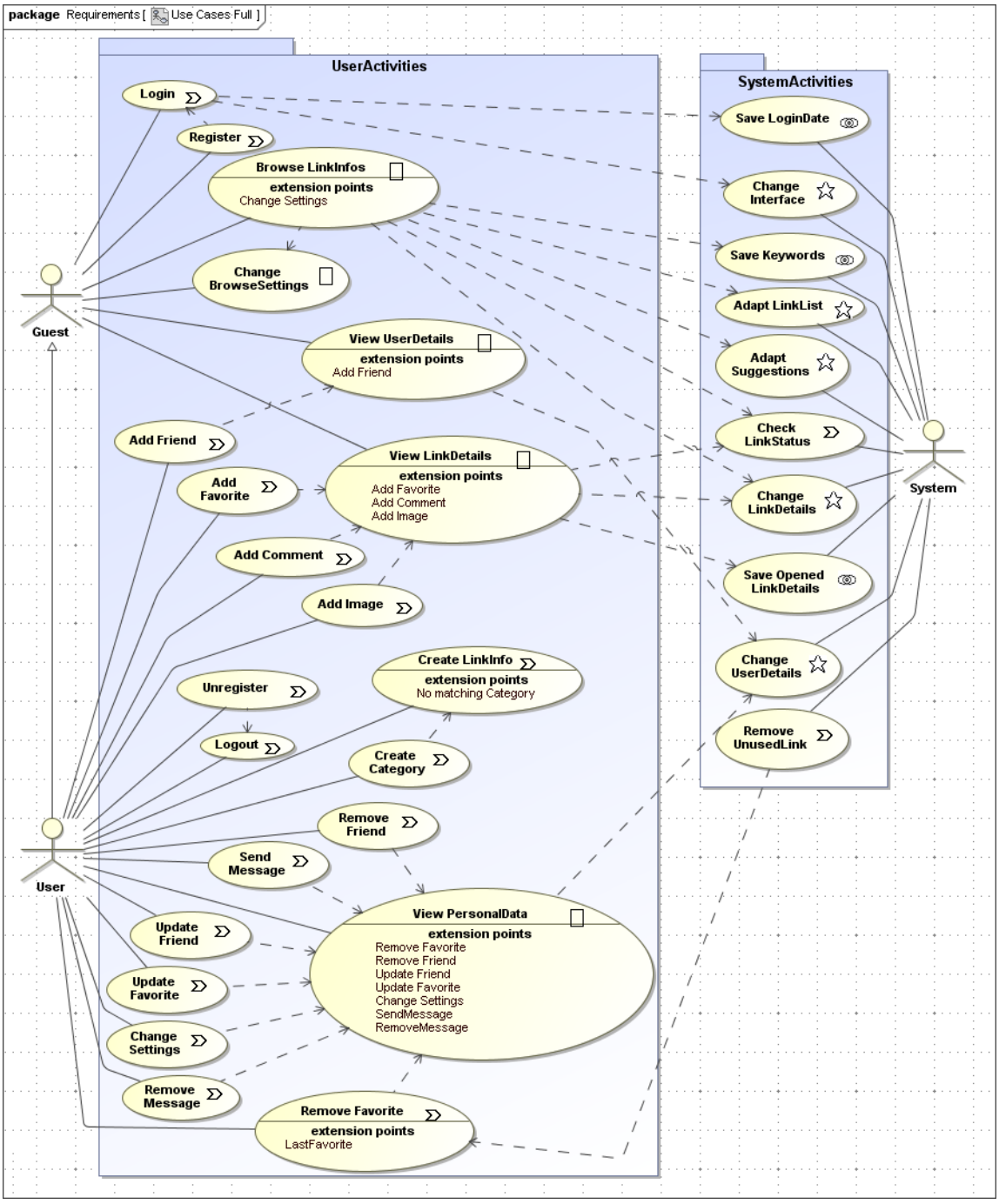


Abbildung 4.2: Das Anwendungsfalldiagramm von Philoponella

zwischen dem An- und Abmelden eines Benutzers muss das System die Sitzungsinformationen, welcher Art sie auch sein mögen speichern.

### 4.1.4 Anwendungsfälle für angemeldete Benutzer

Der Hauptteil der Anwendung fällt in den Bereich für angemeldete Benutzer, welcher fast ausschließlich die Prozesse der Anwendung modelliert. Alle diese Anwendungsfälle werden durch eine Assoziation mit dem Akteur User verbunden, welcher eine Generalisierung von Guest darstellt.

Zuerst aber kann ein angemeldeter Benutzer natürlich seine Detailinformationen betrachten. Dieser Vorgang wird durch View PersonalData modeliert, welcher als einziger in diesem Bereich vom Typ *«navigation»* ist. Beim Betrachten der persönlichen Informationen kann man aber auch verschiedene Veränderungen vornehmen, was durch verschiedene Fälle vom Stereotyp *«process»* repräsentiert wird. Als aller erstes gehören dazu die persönlichen Einstellungen des Benutzers, welche er selbstverständlich ändern kann, was Change Settings dargestellt.

Weiterhin gehören zu den persönlichen Daten eines Benutzer die an ihn gerichteten Nachrichten, welche er beim Betrachten dieser auch beantworten kann. Dafür gibt es hier den Fall Send Message. Ist eine Nachricht angekommen, dann muss der Benutzer auch die Möglichkeit haben diese zu löschen, damit er die Übersicht über seine Nachrichten nicht verliert. Auch dafür gibt es einen Anwendungsfall, welcher die Bezeichnung Remove Message trägt.

Zuletzt gibt es noch vier Anwendungsfälle, welche mit der Liste der Freunde und der Favoritenliste zusammenhängen. Diese kann der Benutzer nicht nur betrachten, sondern auch die einzelnen Elemente dieser bearbeiten oder entfernen. Für die Freundesliste wird dies durch die beiden Anwendungsfälle Update Friend und Remove Friend dargestellt. Analog sind dies bei der Liste der Favoriten die Fälle Update Favorite und Delete Favorite.

Da sich ein Benutzer auch abmelden oder ganz aus dem System entfernen kann, gibt es die beiden eigenständigen Anwendungsfälle Logout und Unregister, welche aus den gleichen Gründen wie auch ihre Gegenstücke dem Stereotyp *«process»* angehören.

Die weitere Möglichkeit des Benutzers neue Einträge zu den Linkinformationen zu erstellen modeliert der Anwendungsfall Create LinkInfo und falls bei diesem Vorgang nicht die passende Kategorie gefunden wird, hat der Benutzer die zusätzliche Möglichkeit auch eine neue Kategorie einzutragen. Dies repräsentiert der Fall Create Category, welcher Create LinkInfo aus diesem Grund erweitert.

Beim Betrachten von Seiten- oder Benutzerdetails hat ein angemeldeter Benutzer

zusätzlich zu einem einfachen Besucher die Option diese als Favoriten bzw. Freund zu speichern. Dazu erweitert Add Favorite den Anwendungsfall View LinkDetails und der Fall Add Friend erweitert entsprechend View UserDetails.

Schließlich kann ein Benutzer auch Linkinformationen durch das Hinzufügen eines eigenen Kommentars oder Vorschaubildes erweitern. Da es beim Betrachten der Details geschieht, sind die dazugehörigen Anwendungsfälle mit View LinkDetails durch eine Erweiternbeziehung verbunden, wobei ersteres der Anwendungsfall Add Comment modelliert und den zweiten Vorgang Add Image.

### 4.1.5 Systeminterne Anwendungsfälle

Der letzter Teil der Anwendungsfalldiagramms bilden die Anwendungsfälle verbunden mit dem Akteur System. Hier gibt es neben zwei gewöhnlichen Vorgängen, noch mehrere Fälle, welche die Adaptivität vom Philoponella beschreiben.

Bei den Anwendungsfällen vom Typ *«process»* gibt es zunächst Check LinkStatus. Dieser modelliert die Tatsache, dass das System bei jedem Aufruf einer Seiteninformation den Status der entsprechenden Adresse überprüft. Daher sind die Anwendungsfälle View LinkDetails und Browse LinkInfos durch eine Beziehung vom Typ Einschließen mit diesem verbunden. Hinzu kommt der Anwendungsfall Remove UnusedLink, welcher den Vorgang repräsentiert, dass jedes Mal wenn ein Benutzer einen Link aus seiner Favoritenliste entfernt, das System überprüft, ob dieser noch von einem anderen Benutzer gebraucht wird und diesen löscht, wenn dies nicht der Fall ist. Daher ist dieser Fall mit RemoveFavorite verbunden.

Weiterhin hat das System drei Anwendungsfälle vom Typ *«observation»*, welche bei Philoponella recht einfach sind, da sie nur Informationen für spätere Benutzung abspeichern. Dazu gehört zunächst das durch Save LoginDate modellierte Speichern der Anmeldedaten, welches logischerweise vom Anwendungsfall Login eingeschlossen wird. Außerdem werden alle Suchbegriffe von Philoponella gesammelt, was der von Browse LinkInfos eingeschlossene Anwendungsfall Save Keywords repräsentiert. Zuletzt gibt es den Fall Save Opened LinkDetails, welcher den Umstand darstellt, dass Philoponella sich alle geöffneten Linkinformationen und die dazugehörigen Kategorien merkt. Da dies beim Betrachten der Linkinformationen geschieht, ist dieser Anwendungsfall auch mit View LinkDetails durch eine Einschlussbeziehung verbunden.

Als letztes werden hier noch die Anwendungsfälle zur Modellierung der Adaption in Philoponella vorgestellt. Diese gehören alle zum Stereotyp *«adaptation»*. Hier gibt es Adapt Suggestions, welcher die Anpassung der Suchfeldvorschläge darstellt und mit Browse LinkInfos verbunden wird. Auch Adapt LinkList wird von Browse LinkInfos eingeschlossen, da dieser Anwendungsfall die Anpassungen der Listen beim Durchsuchen der Seiteninformationen

darstellt. Ebenfalls damit und auch mit View LinkDetails verbunden ist der Anwendungsfall Change LinkDetails. Damit soll die Anpassung der Darstellung der Linkdetails modeliert werden. Analog dazu gibt es den Fall Change UserDetails, welcher die Anpassung der Präsentation der Benutzerdetails darstellt. Da dies sowohl bei persönlichen Daten, als auch beim Betrachten anderer Benutzer geschieht, beinhaltet diesen sowohl View UserDetails als auch View PersonalData. Zuletzt gibt es noch den Fall Change Interface als Teil des Anmeldevorgangs, wodurch das Anpassen der Einstellungen der Benutzeroberfläche repräsentiert wird.

## 4.2 Detaillierte Anforderungsmodellierung

Da die Anwendungsfälle nur einen groben Überblick über eine Anwendung geben können, kann man diese in UML im Detail beschreiben, indem man für sie Aktivitätsdiagramme erstellt. Ebenso beschreiben die vorgestellten Anwendungsfälle bisher nur eine grobe Struktur einer Webanwendung und man kann nur sehr wenig Informationen daraus für die Modellierung gewinnen. Damit bereits in der Phase der Anforderungsanalyse möglichst viele relevante Parameter angegeben werden können, braucht man auch hier Aktivitätsdiagramme, wobei diese ein wenig modifiziert bzw. um Elemente erweitert werden müssen. Die Veränderungen am Modell von UWE erfolgt dabei zum einen durch Einführung neuer Stereotypen für die Aktionen als auch durch Stereotypen für Ein- oder Ausgabepins. Abbildung 4.3 stellt diese schon einmal dar, bevor ihr Zweck in den nachfolgenden beiden Abschnitten beschrieben wird.

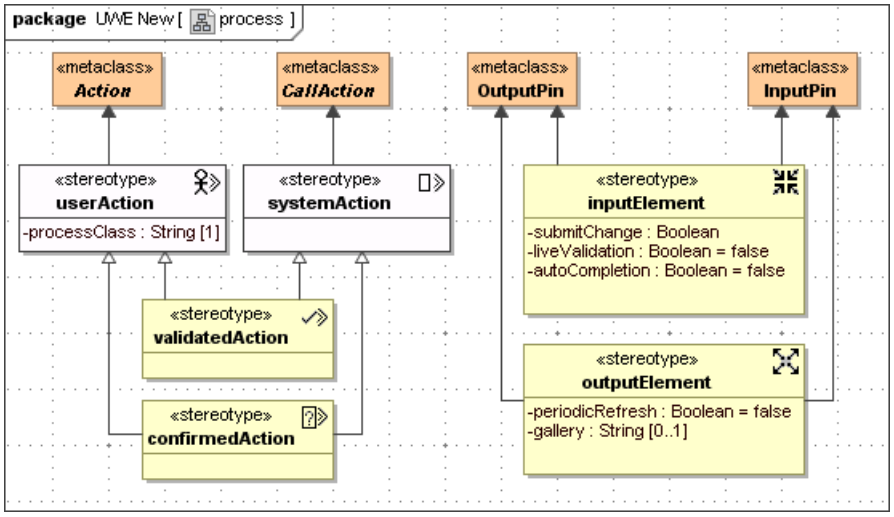


Abbildung 4.3: Die Erweiterung von UWE um Stereotypen der Aktivitätsdiagramme

### 4.2.1 Erweiterte Aktionen

Der Hauptbestandteil eines Aktivitätsdiagramms sind die Aktionen. Damit lassen sich sowohl die verschiedenen Ablaufschritte, als auch deren Abfolge, welche durch die Verbindungen untereinander dargestellt wird. Es ist schwer die beiden bereits in UWE vorhandenen Stereotypen *«userAction»* und *«systemAction»* zu erweitern, ohne dass die Menge der Stereotypen unübersichtlich und damit nicht mehr so leicht verständlich wird. Nach einer gründlichen Beobachtung der Aktivitätsdiagramme aus Philoponella ist jedoch zweimal das häufig gemeinsame Vorkommen von jeweils zwei Aktionen in UWE auffällig. Daher werden hier zwei neue Stereotypen eingeführt, um die jeweils zusammen vorkommenden Aktionen zu ersetzen und dadurch das Aktivitätsdiagramm zu verkleinern. Bevor diese ausführlich erklärt werden, sind diese in der Tabelle 4.3 festgehalten.

| Name | Symbol | Zweck |
|---|---|---|
| *«validatedAction»* |  | Benutzereingaben werden vom System überprüft |
| *«confirmedAction»* |  | Systemaktion muss erst vom Benutzer bestätigt werden |

Tabelle 4.2: Neue Stereotypen für die Aktionen der Aktivitätsdiagramme

Als erstes ist auffällig, dass oftmals auf eine Benutzeraktion also eine Aktion vom Stereotyp *«userAction»*, welche eigentlich immer die Eingabe von Daten durch den Benutzer darstellt eine Aktion vom Typ *«systemAction»* folgt, welche die Validierung der eingegeben Daten durch das System repräsentiert. Liegen dabei nach der Eingabe korrekte Benutzerdaten vor, dann wird die Aktivität durch eine nachfolgende Aktion ausgeführt. Können die Daten hingegen allerdings so nicht vollständig vom System akzeptiert werden, dann wartet die Anwendung auf die Korrektur bzw. die Eingabe besserer Daten durch den Benutzer. Dies wird durch einen Übergang zurück zu der anfänglichen Benutzeraktion dargestellt, was in den meisten Fällen auch die Ausgabe einer Fehlermeldung oder Warnung mit sich führt. Ein Beispiel hierfür wäre die Eingabe der Benutzerdaten bei der Registrierung, wobei der Benutzername oder das Passwort bestimmte Kriterien erfüllen müssen.

Dies alles lässt sich in einer Aktion zusammenfassen, wofür der Stereotyp ***«validatedAction»*** eingeführt wird. *«validatedAction»* bildet dabei eine Art Hybrid zwischen einer Benutzeraktion und einer systeminternen Aktion, da sie Aktionen beider Stereotypen einschließt, wobei die Eingabe des Benutzers im Vordergrund steht und das System nur passend auf diese zu reagieren versucht. Als Symbol für *«validatedAction»* wurde die für die Stereotypen der Aktionen beiden Pfeilspitzen gewählt, welche aber in Abgrenzung zu den anderen beiden Typen auf ein Häkchen ähnlich dem einer Checkbox folgen.

Eine Aktion des Stereotyps *«validatedAction»* kann man auch als eine komplexe Aktion ansehen, welche die vorhin beschriebenen Vorgänge einschließt. Um dies zu verdeutlichen ist in

Abbildung 4.5 ein Aktivitätsdiagramm dargestellt, durch welches man eine *«validatedAction»* ersetzen könnte.

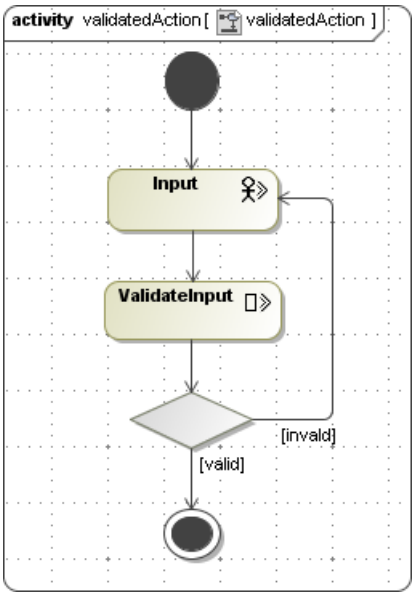


Abbildung 4.4: Durch den Stereotyp *«validatedAction»* vereinfachte Aktivität

Die zweite Auffälligkeit ist der ersten sehr ähnlich, geht aber gewissermaßen einen umgekehrten Weg. Zunächst folgt auch hier eine Aktion des Stereotyps *«systemAction»* auf eine Aktion des Typs *«userAction»*. Dennoch ist dieser Vorgang ein gänzlich anderer, da die systeminterne Aktion hier die Verarbeitung von irgendwelchen Daten im System oder zumindest den Anfang einer solchen Verarbeitung repräsentiert. Die Benutzeraktion hingegen beinhalten nicht die Eingabe von Daten, sondern sie repräsentiert die Bestätigung der Systemaktion durch den Benutzer. Also wird die Systemaktion nur ausgeführt, wenn der Benutzer dies auch bestätigt, was durch eine zusätzliche Verbindung der Benutzeraktion mit einer nachfolgenden Systemaktion, welche nicht von dieser Bestätigung abhängig ist oder dem Ende der Aktivität, falls eine solche Aktion nicht existiert repräsentiert wird. Hierfür ist ein gutes Beispiel das Überschreiben oder Löschen wichtiger Daten.

Und auch dieser Vorgang lässt sich durch einen Stereotyp zusammenfassen, welcher die Bezeichnung ***«confirmedAction»*** trägt. Wie bereits *«validatedAction»* ist *«confirmedAction»* ebenfalls ein Hybrid zwischen einer systeminternen Aktion und einer Benutzeraktion. Nur steht hier die Systemaktion im Vordergrund, da vom Benutzer nicht mehr als eine Bestätigung bzw. ein Abbruch erwartet wird. Um dies zu demonstrieren ist das Symbol für *«confirmedAction»* wie das Symbol des Stereotyps *«systemAction»*, nur ist das stehende Rechteck bei *«confirmedAction»* nicht leer, sondern enthält ein Fragezeichen.

Die Verwendung dieses Stereotyps kann ebenfalls als eine komplexe Aktion verstanden werden und ist zum besseren Verständnis in der Abbildung **??** dargestellt. Außerdem soll

durch die beiden Aktivitätsenden repräsentiert werden, dass von *«confirmedAction»* je nach Ausgang der Benutzeraktion verschiedene Aktionen erreicht werden können.

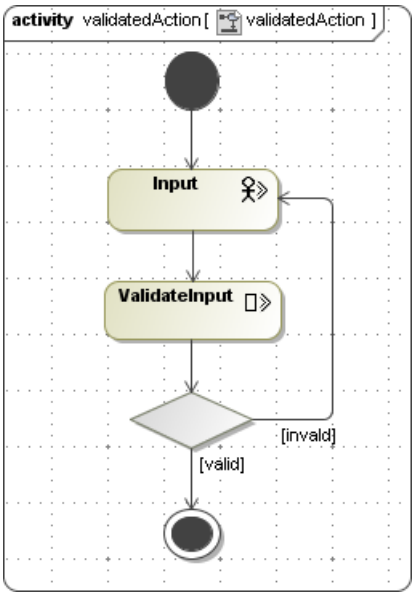


Abbildung 4.5: Durch den Stereotyp *«confirmedAction»* vereinfachte Aktivität

### 4.2.2 Verwendung der Pins

Ein weiteres Element der Aktivitätsdiagramme sind Eingabe- und Ausgabepins. Diese Objekte sind immer ein Teil einer Aktion und sollen die ausgehenden und einfließenden Daten für diese Aktion repräsentieren. Auch wenn diese nicht neu für UWE sind, sollen ihre Verwendung bei der Anforderungsanalyse hier besprochen werden.

Zunächst einmal können beide Pinarten benannt werden. Dies kann man nutzen, um eine Art Auflistung der für diese Aktion benötigten und der dadurch gewonnen Daten an eben diese Aktion anzuhängen. Vor allem für die Anforderungsanalyse ist dies von Nutzen, da durch die Pins die einzelnen Datenfragmente dargestellt werden können, ohne die Inhaltsstruktur der Anwendung mitskizzieren zu müssen. Dadurch wird es ermöglicht, dass man bereits in dieser Phase für die Aktionen benötigten Attribute darstellen kann, soweit sie hier bereits bekannt ist. Außerdem bedeutet dies, dass man in dieser Phase sich nur um die einzelnen Informationen Gedanken machen muss und die Gruppierung der Daten zu Klassen und Modellierung ihrer Beziehungen untereinander auf den entsprechenden Teil der Modellierungsphase verlegen kann. Von besonders großem Nutzen ist dies für diejenigen am Projekt beteiligten Personen, die nichts mit der Modellierung oder Implementierung der Anwendung zu tun haben und daher vielleicht über unzureichende Kenntnisse verfügen, um eine solche Inhaltsstruktur richtig zu verstehen.

Da außerdem die Eingabepins mit den Ausgabepins direkt verbunden werden können, stellt dies eine einfache Möglichkeit den Datenfluss zwischen den unterschiedlichen Aktivitäten darzustellen.

Will man dennoch Objektklassen in der Anforderungsanalyse bereits benennen oder mit bestimmten Aktionen und den darin enthaltenen Daten in Verbindung bringen, so kann man Objektknoten in das Diagramm einfügen und mit den entsprechenden Pins verbinden. Da es zu diesem Zeitpunkt noch keine Definition der Objektklassen gibt, bietet es sich an die Klasse des Objekts in dem Namen des Objektknotens zu notieren und den Typ zu diesem Zeitpunkt wegzulassen.

Nachdem die grundlegende Struktur eines Aktivitätsdiagramms der Anforderungsanalysephase beschrieben ist, fällt einen vielleicht auf, dass die Pins dazu benutzt werden können die noch nicht innerhalb der Anforderungsanalyse darstellbaren Präsentationselemente darzustellen. Daher gibt es in dieser Erweiterung noch zwei neue Stereotypen für Pins, welche in Anlehnung an die Elemente des Präsentationsmodels die Namen *«inputElement»* und *«outputElement»* tragen und in der Tabelle 4.3 festgehalten sind. Zwar sind die Pins dieser Stereotypen in der Regel Ausgabepins, jedoch braucht man auch manchmal diese Typen für Eingabepins. Daher lässt es sich nicht vermeiden, diese beiden Stereotypen sowohl für Eingabe- als auch für Ausgabepins zu definieren und man darf dabei nicht den Stereotypen mit der Art des Pins verwechseln.

| Name | Symbol | Zweck |
|---|---|---|
| *«inputElement»* |  | Darstellung von Präsentationselementen zur Eingabe von Daten |
| *«outputElement»* |  | Darstellung von Präsentationselementen zur Ausgabe von Daten |

Tabelle 4.3: Neue Stereotypen für die Aktionen der Aktivitätsdiagramme

Die Art des Pins kann entweder Eingabepin oder Ausgabepin bzw. Inputpin oder Outputpin sein und gibt die Möglichkeit, wie die Objektflussrichtung bei diesen sein darf. Hingegen dienen die beiden Stereotypen der Beschreibung von ganz anderen Anwendungseigenschaften, welche hier nun beschrieben werden sollen. Ein in der Anforderungsanalyse noch sehr schwach erfasster Bereich des Modells ist nämlich die Präsentation und die beiden neuen Stereotypen sollen an dieser Stelle die Möglichkeiten zur Beschreibung verbessern.

Dabei soll der Stereotyp ***«outputElement»*** die Elemente des Präsentationsmodells bzw. auch der Benutzeroberfläche der fertigen Anwendung repräsentieren, welche allein zur Darstellung der Informationen dienen. Ebenfalls solche Elemente in Form von generalisierten Stereotypen vereint der abstrakte Stereotyp *«outputElement»* aus dem Präsentationsmodel, woraus sich auch die Namensgleichheit dieser beiden Stereotypen ergibt.

Mit dem zweiten Stereotyp verhält es sich ähnlich. Dieser trägt die selbe Bezeichnung wie

ein anderer abstrakter Stereotyp aus dem Präsentationsmodel, nämlich ***«inputElement»*** und dient ebenfalls der Darstellung der gleichen Elemente wie der durch die generalisierten Stereotypen von *«inputElement»* im Präsentationsmodel. Dies sind Elemente der Benutzeroberfläche, die zur Eingabe von Daten durch den Benutzer dient.

Obwohl die Pins nur sehr kleine Elemente eines Aktivitätsdiagramms sind, kann man trotzdem auch für ihre Stereotypen Symbole definieren. Symbolisch dargestellt werden diese Stereotypen dabei durch vier kleine, in die Mitte zeigende Pfeile für *«inputElement»* und durch vier kleine, nach Außen zeigende Pfeile für *«outputElement»*.

Ein weiterer, noch nicht behandelter Punkt ist die Modellierung der Eigenschaften von Rich Internet Applications. Diese werden in UWE durch spezielle Objekteigenschaften sogenannte Tags dargestellt. Sie hängen außerdem von dem Stereotyp ab, da diese Eigenschaften nur bei bestimmten Benutzeroberflächenelementen benutzt werden können. Die beiden Stereotypen *«outputElement»* und *«inputElement»* der Pins enthalten die gleichen Eigenschaften wie ihre abstrakten Namensvettern aus dem Präsentationsmodel. Auch können diese Eigenschaften wie bei konkreten Elementen der Präsentation mit Werten belegt werden, um bereits in der Anforderungsanalyse gewollte RIA-Eigenschaften zu spezifizieren.

### 4.2.3 Aktivitätsdiagramme von Philoponella

Da die Aktivitätsdiagramme der Anforderungsanalyse zu jeweils einem Anwendungsfall gehören, lassen sie sich ebenfalls nach den drei Akteuren aufteilen. Jedoch werden hier die Aktivitätsdiagramme geteilt nach dem Stereotyp des Anwendungsfalls beschrieben. Der Vorteil liegt hierbei darin, dass sich die Diagramme eines Stereotyps zumindest teilweise ähnlich sind.

### 4.2.4 Philoponellas Aktivitätsdiagramme zur Adaption

In diesem Abschnitt gibt es die die Aktivitätsdiagramme zu den Anwendungsfällen, welche die Adaptivität modelieren sollen. Dies sind alles natürlich systeminterne Vorgänge ohne Benuterinteraktion.

Als erstes gibt es die Aktivität **Adapt Suggestions** zur Beschreibung der Suchfeldvervollständigung. Es beginnt mit dem Aussortieren der nicht passenden Suchbegriffe durch die Aktivität SelectMatched. Dazu muss man zunächst den derzeitigen Suchbegriff haben, was durch den Pin keyword dargestellt wird und die durch userKeywords repräsentierte Liste der bisher eingegeben Suchbegriffe dieses Benutzers. Die so gewonnene Liste wird daraufhin sortiert und bei Bedarf durch passende Suchbegriffe ergänzt. Der mit dem Aktivitätsende verbundene Pin finalKeywords zeigt dabei schließlich die nach dieser Anpassung fertiggestellte Liste der Suchbegriffe.

Ähnlich dazu läuft auch die Anpassung der Liste der Seiteninformationen in **Adapt LinkList**. Auch hier wird eine Liste repräsentiert durch links genommen und in der Aktion Sort LinkInfos sortiert. Dazu benötigt das System allerdings nur bereits vorhandene Daten, weshalb hier auch nur Pins ohne Stereotypen vorkommen. Danach werden bestimmte Elemente der Liste hervorgehoben, was durch HighlightLinkInfos dargestellt wird. Zu beachten ist besonders, dass beide Aktionen jeweils zwei gleiche Informationen benötigen, was durch die gleich benannten Pins an beiden Aktionen friends und favorites modelliert wird.

Eine weitere Aktivität der Adaption ist **Change UserDetails**, welche die Anpassung der Benutzeransicht beschreibt. Da diese ein gutes Beispiel für eine adaptive Aktivität darstellt, ist ihr Diagramm auch in Abbildung 4.6 gezeigt. Zunächst werden die Freundes- und die Favoritenliste analog zu bisher sortiert, jedoch sind alle Informationen diesmal alles Listen. Dies kann man zum besseren Verständnis durch eine entsprechende Multiplizitätangabe bei den Pins markieren, allerdings sind diese Informationen nicht im Diagramm sichtbar, weshalb sich der Nutzen hier in Grenzen hält. Als letztes wird bei dieser Anpassung modelliert durch die Aktion FindSimilar eine Liste neuer, für den jeweiligen Benutzer interessante Linkinformationen erstellt. Hier produzieren alle Aktionen eine Ausgabe, was die drei mit dem Aktivitätsende verbundenen Pins sortedFavorites, sortedFriends und newLinks repräsentieren.

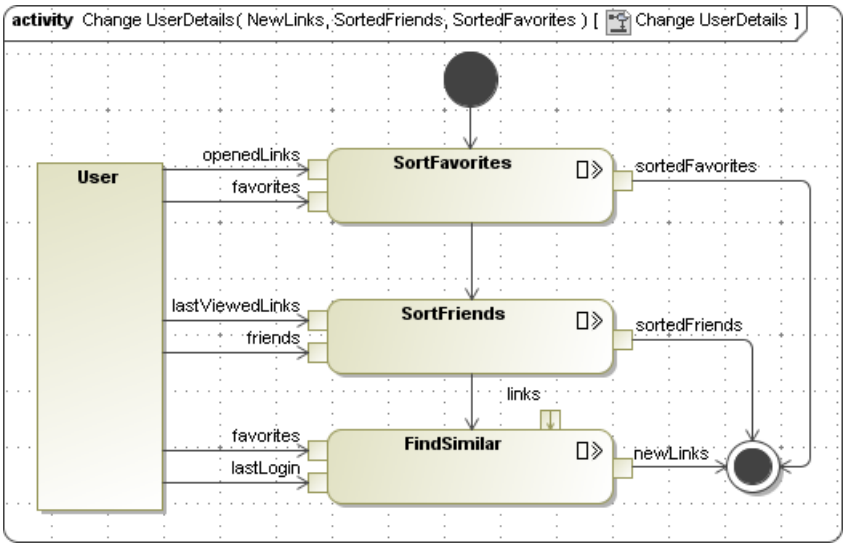


Abbildung 4.6: Das Aktivitätsdiagramm zu Philoponellas Anwendungsfall Change UserDetails

Die Aktivität **Change LinkDetails** zur Darstellung der Adaption der Seitendetails verläuft auch sehr ähnlich. Zunächst wird die passende Beschreibung ausgewählt, dann die unpassenden Kommentare entfernt und die übriggebliebenen sortiert. Der Pin friends bei SortComments modelliert dabei die Art der Sortierung, nämlich die Bevorzugung der Kommentare von Freunden des Benutzers. Zum Schluss werden auch hier wie bei Change UserDetails eine Liste interessanter Linkinformationen generiert.

Eine Besonderheit stellt die letzte Aktivität zur Anpassung dar, da sie keine Ausgabe, d.h. keine mit dem Aktivitätsende verbundene Pins besitzt. **Change Interface** modelliert dabei das Verändern der Benutzeroberfläche an den Benutzer beim anmelden. Dabei wird zunächst die als letztes ausgewählte Kategorie eingestellt, was SetCategory beschreibt, danach werden Details beim Betrachten der Seiteninformationen und der Benutzerinformationen angepasst. Hier haben alle Eingabepins keinen Stereotyp, da sie Daten repräsentieren, welche im System bereits gespeichert sind, aber alle Ausgabepins sind vom Typ *«outputElement»*, weil diese Aktivität keine angepassten Daten liefert, sondern die Einstellungen der Benutzeroberfläche verändert. Da sich diese Anpassung durch besondere Form der Modellierung auszeichnet wird ihr Aktivitätsdiagramm in Abbildung 4.7 gesondert präsentiert.

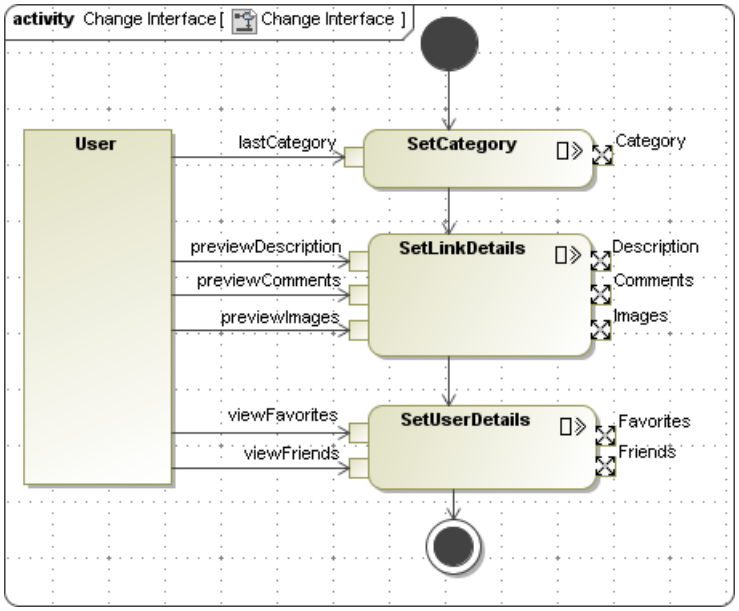


Abbildung 4.7: Das Aktivitätsdiagramm zu Philoponellas Anwendungsfall Change Interface

Für die drei Anwendungsfälle des Stereotyps *«observation»* gibt es keine Aktivitätsdiagramme. Diese beschränken sich aufs einfache Abspeichern der Daten und benötigen nicht zwingend ein Aktivitätsdiagramm, da diese ohnehin fast inhaltslos wären. Jedoch ist es durchaus vorstellbar, dass bei anderen Webanwendungen komplexere Vorgänge zum Sammeln der Benutzerdaten notwendig sind, was selbstverständlich durch die bisher vorgestellten Möglichkeiten modelliert werden kann.

### 4.2.5 Philoponellas Aktivitätsdiagramme zu den Navigationsvorgängen

In Philoponella gibt es fünf Vorgänge, welche keine Veränderung des Systeminhalts bewirken und damit zum Stereotyp *«navigation»* gehören. Die Aktivitäten des Stereotyps *«navigation»* beschreiben die reine Ausgabe der Daten zusammen mit der Navigation und

sind sich sehr ähnlich. Als Beispiel ist in Abbildung 4.8 die als erstes beschriebene Aktivität Browse LinkInfos bereits dargestellt.

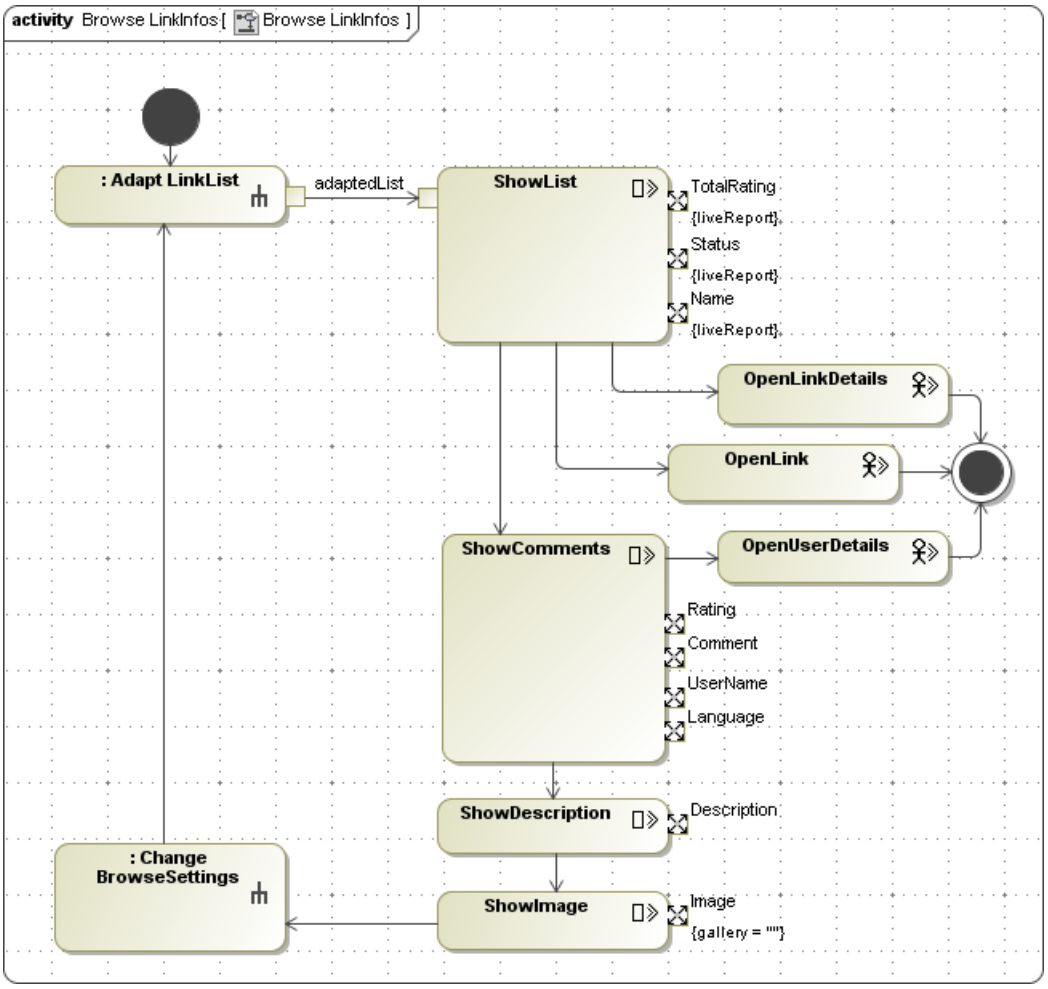


Abbildung 4.8: Das Aktivitätsdiagramm zu Philoponellas Anwendungsfall Browse LinkInfos

Zunächst gibt es hier **Browse LinkInfos** zur Darstellung der Suche interessanter Seiten. Hier kommt zu Beginn die Aktivität der Adaption Adapt LinkList, welche die Anpassung der Liste an den jeweiligen Benutzer modelliert. Der Ausgabepin dieser Aktion entspricht der mit dem Aktivitätsende verbundenen Ausgabepin in Adapt LinkList und ist daher gleich benannt. Danach folgen die Aktionen ShowList, ShowComments, ShowDescription und ShowImage vom Typ *«systemData»*, welche alle einen Teil der dargestellten Informationen der Ansicht repräsentieren. Die dazugehörigen Pins des Stereotyps *«outputElement»* dienen dabei dazu die dargestellten Details zu definieren. Wie schon früher sind manche von ihnen durch gesetzte Tags mit RIA-Eigenschaften ausgestattet. Außerdem sind ShowList und ShowComments mit Aktionen des Typs *«userData»* verbunden. Dies soll die möglichen Übergänge in Verbindung zum entsprechenden Teil der Ansicht modellieren. So kann man

von der Liste der Seiteninformationen entweder direkt auf die Seite oder zur Ansicht der Seitendetails, daher ist ShowList mit den Aktionen OpenLink und OpenLinkDetails verbunden. Ebenso kann man die Benutzerdetails zu einem Verfasser eines Kommentars betrachten, was durch den Übergang von ShowComments nach OpenUserDetails dargestellt ist. Nur Aktionen des Stereotyps *«userAction»* sind mit dem Aktivitätsende verbunden. Das soll repräsentieren, dass nur durch eine Änderung der Ansicht diese Ansicht verlassen wird.

Zum Schluss von Browse LinkInfos gibt es noch die Aktion zur Aktivität **Change BrowseSettings**, welche einen Vorgang, nämlich das bereits besprochene Verändern der Suchparameter, beschreibt durch den die Ansicht zwar modifiziert wird, aber nicht zu einer anderen wechselt. Daher mündet diese Aktion nicht im Aktivitätsende, sondern ist wieder mit der ersten Aktion verbunden, wodurch ein Zyklus entsteht. Diese Aktivität ist sehr einfach, aber wie später zu sehen sein wird, besitzt sie mehr Ähnlichkeit mit einem Prozess als mit den anderen Navigationsvorgängen und ist wegen dieser Besonderheit nochmal extra durch die Abbildung 4.9 hervorgehoben. Der Grund dafür, dass diese Aktivität trotz der hohen Ähnlichkeit nicht zum Stereotyp *«process»* gezählt wird, ist die Tatsache, dass der Benutzer zwar die Ansicht modifizieren kann, jedoch keine der Benutzereingaben eine Veränderung innerhalb des Systems bewirkt. Zu Beginn dieser Aktivität gibt der Benutzer verschiedene Daten ein, was durch die Aktion SetParameters mit den dazugehörenden Pins des Typs *«inputElement»* dargestellt wird. Diese Pins sind mit einfachen Pins der Systemaktion ApplyParameters verbunden, was die Übergabe der eingegebenen Parameter an das System und die Bearbeitung dieser darstellt. Der mit dem Aktivitätsende verbundene Pin newList repräsentiert dabei die an die neuen Parameter angepasste Liste, welche als Ergebnis dieses Vorgangs für den Benutzer zu sehen ist. Außerdem ist zu beachten, dass das durch den Pin Searchfield repräsentierte Eingabefeld über die RIA-Eigenschaft *«autoCompletion»* verfügt, was durch eine entsprechend gesetzte Eigenschaft des Pins modelliert wird.

Der nächste Vorgang ist **View LinkDetails** und dient der Repräsentation der Detailansicht zu einer Adresse. Auch dieser besitzt zu Beginn eine Aktion zur Darstellung von Adaptivität. Hier ist es Change LinkDetails und sie unterscheidet sich von dem vorherigen Fall dadurch, dass sie mehrere Daten liefert. Aber auch diese Ausgabepins tragen die gleichen Namen wie die mit dem Aktivitätsende verbundene Ausgabepins in Change LinkDetails. Danach folgen wiederum mehrere Aktionen zur Darstellung verschiedener Teile der Präsentation dieser Ansicht mit den dazugehörenden Ausgabepins vom Typ *«outputElement»* und abgeschlossen wird die Aktivität durch die beiden Benutzeraktionen OpenUserDetails und OpenLink, welche bereits besprochene Wechsel der Benutzeransicht darstellen. Auffällig ist hier vielleicht noch, dass durch die Aktionsübergänge kein Zyklus entsteht. Dies liegt daran, dass es keinen Vorgang gibt, der zu einer Änderung der Ansicht führt.

Weiterhin gibt es hier noch die Aktivität zur Ausgabe der Benutzerdaten: **View**

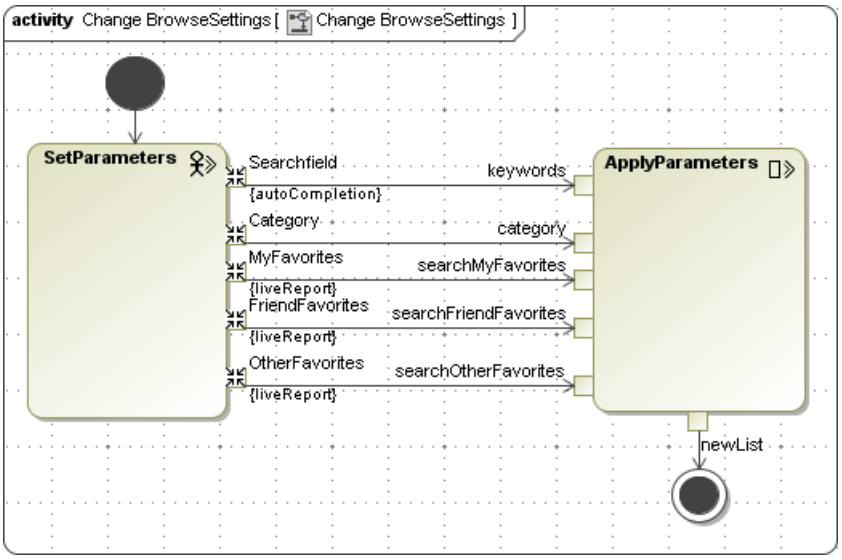


Abbildung 4.9: Das Aktivitätsdiagramm zum Anwendungsfall Change BrowseSettings

**UserDetails**, welche sich ebenfalls nur gering von den anderen Aktivitäten des Typs *«navigation»* unterscheiden. Allerdings beginnt sie nicht mit einer Aktion zur Darstellung der Ausgabe, da die Systemaktion ShowDetails, welche die Darstellung der Grundinformationen über einen Benutzer wie seinen Namen und beherrschten Sprachen modelliert und diese Daten nicht von einer anderen Aktivität abhängen, kann diese Aktion auch zu Anfang der Aktivität stehen. Erst dann folgt die Aktion zur Aktivität Change UserDetails, die zur Beschreibung der Anpassungen dieser Ansicht dient und auf welche mehrere diesmal von den Anpassungen abhängige Aktionen zur Ausgabe folgen. Schließlich wird auch View UserDetails durch mehrere Aktionen vom Typ *«userAction»* abgeschlossen und bildet keinen Zyklus.

Die zum letzten Anwendungsfall des Typs *«navigation»* gehörendes Aktivitätsdiagramm trägt die Bezeichnung **View PersonalData** und ist fast komplett mit dem bereits vorher beschriebenen Vorgang View UserDetails identisch, da beide die Darstellung von Benutzerdaten modellieren. Auch wenn der Unterschied darin liegt, dass beim ersten Mal ein Benutzer seine eigenen Daten betrachtet und beim anderen die Daten von Fremden, so werden dennoch die gleichen Aktionen bei diesem Vorgang ausgeführt. Die Unterschiede ergeben sich also hauptsächlich dadurch, dass der Benutzer bei seinen eigenen Daten alles zu sehen kriegt, während bei fremden Daten dieser nur den freigegeben Bereich einsehen kann. Lediglich die Ansicht der erhaltenen Nachrichten unterscheidet die beiden Aktivitäten, da man bei anderen Benutzern überhaupt keine Nachrichten einsehen kann und wird durch die Aktion ShowMessage modelliert.

### 4.2.6 Philoponellas Aktivitätsdiagramme der Prozesse

Die letzte Gruppe ist die der zum Stereotyp *«process»* gehörenden Vorgänge. Diese kommen in Verbindung mit allen drei Akteuren vor, jedoch im Bereich für angemeldete Benutzer befinden sich die meisten von ihnen, da in Philoponella Änderungen des Systeminhalts fast ausschließlich registrierten Benutzern vorbehalten ist.

Zu Beginn kommen hier aber zwei Prozesse, welche vom System benötigt werden. Bei **Remove UnusedLink** gibt es sogar nur die Systemaktion RemoveLink mit dem Pin unusedLink, welches das entfernen einer übergebenen Linkinformation darstellt. Dazu kommt noch das Diagramm zu **Check LinkStatus**, worin die Überprüfung des Verbindungsstatus für eine Seite durchgeführt wird. Dazu wird zuerst die Verbindung zur in link übergebenen Adresse in CheckConnection geprüft und übermittelt diesen Status dann an die Aktivität SaveConnection, woraufhin diese das Datum der letzten Überprüfung und die daraus ermittelte Dauer der Inaktivität in der Linkinformation abspeichert, was hier durch den Typenlosen Objektknoten LinkInfo dargestellt wird.

Etwas komplizierter und eine graphische Darstellung in Abbildung 4.10 wert ist die Aktivität **Register**. Mit ihr soll die Registrierung eines neuen Benutzers modeliert werden. Sie beginnt mit der Eingabe von Benutzerdaten, welche um Inkonsistenzen zu vermeiden vom System möglichst vor der Weiterverarbeitung ausgewertet werden sollen. Daher wird dies durch die Aktivität EnterData mit dem neuen Stereotyp *«validatedAction»* modelliert. Außerdem ist es für den Benutzer von Vorteil, dass er auf Fehler bei der Eingabe möglichst schnell hingewiesen wird damit er diese auch gleich korrigieren kann. Die Pins Name, Email und Password, welche die Eingabefelder für die kritischen Benutzerdaten Name, Emailadresse und Passwort modellieren, besitzen aus diesem Grund die Eigenschaft *«liveValidation»*. Alle Pins dieser Aktion sind Verbunden mit entsprechenden, typlosen Pins der Aktion CreateUser. Diese Nachfolgeraktion zur modellierung des Abspeicherns des Benutzers durch das System ist vom Stereotyp *«confirmedAction»*, da der Benutzer nach der Eingabe seiner Daten noch einmal die Registrierung bestätigen soll. Außerdem enthält diese Aktion auch gleich benannte Ausgabepins, welche in dem Objektknoten User münden, was die endgültige Erzeugung des Benutzers mit den angegebenen Daten symbolisieren soll.

Die letzte Aktivität aus diesem Bereich beschreibt das Anmelden eines bereits registrierten Benutzers und trägt die Bezeichnung **Login**. Zunächst ist hier auffällig, dass die Benutzereingaben repräsentierende Aktion EnterData vom Typ *«userAction»* und nicht *«validatedAction»* ist, obwohl die Benutzerdaten selbstverständlich vom System ausgewertet werden. Dadurch soll der Unterschied zu anderen Auswertungen im System hervorgehoben werden, da beim Anmelden der Benutzer aus Sicherheitsgründen möglichst keine Warnungen bzw. Mitteilungen über seine Fehler bekommen sollte. Hier wird nur ein Erfolg oder ein

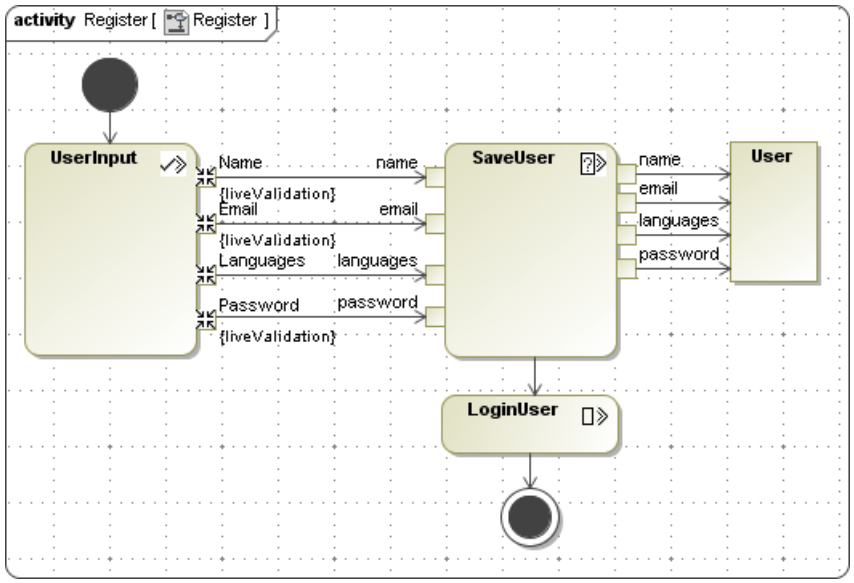


Abbildung 4.10: Das Aktivitätsdiagramm zu Philoponellas Anwendungsfall Register

Fehlschlag registriert. Außerdem folgt auf das einloggen in das System die komplexe Aktion Change Interface. Diese ist eine Aktivität aus dem Bereich Adaption und passt hier rein, da diese Anpassung sofort auf ein erfolgreiches Anmelden folgen sollte.

Schließlich kommt nun die bereits erwähnte größte Gruppe der Anwendungsfälle des Stereotyps *«process»*: die von registrierten Benutzern eingeleitete Prozesse. Hier gibt es zunächst die ganz simple Aktivität **Logout**, welcher das Abmelden des Benutzers durch das System beschreibt, nachdem dieser die Aktivität eingeleitet hat. Der Vorgang der kompletten Entfernung des Benutzers aus dem System ist ebenfalls sehr einfach. Dieser trägt die Bezeichnung **Unregister** und besteht aus der bestätigten Löschaktion DeleteUser gefolgt vom abmelden des Benutzers. Zu beachten wäre, dass es nicht vollständig korrekt ist, dass das Abmelden des Benutzers erst nach dem Löschen geschieht, sondern als Teil des Löschvorgangs. Trotzdem ist es sinnvoll hervorzuheben, dass der Benutzer auch abgemeldet wird.

Die beiden Vorgänge **Change Settings** und **Send Message** zur Darstellung der Änderung von Benutzerdaten und das Abschicken einer Nachricht sind ebenfalls recht einfach und ähneln den bereits vorgestellten. Hier unterscheiden sich der Ablauf allein in den Daten. Die Aktionen beschränken sich bei beiden auf die Benutzereingabe, welche vom System validiert und bei Erfolg abgespeichert wird. Auch die sechs Aktivitäten zur Verwaltung der Freundes- und der Favoritenliste, welche bei der Freundesliste die Namen **Add Friend**, **Update Friend** und **Delete Friend** und bei der Favoritenliste die Namen **Add Favorite**, **Update Favorite** und **Delete Favorite** tragen, sind identisch dazu aufgebaut.

Genauso verhält es sich mit den Aktivitäten **Create Category** und **Create LinkInfo**,

außer dass bei diesen die Systemaktion vom Typ *«confirmedAction»* ist um ein versehentliches Absenden der Daten zu vermeiden. Da die Aktivität Create LinkInfo das Erstellen einer neuen Seiteninformation darstellt und dazu bei bedarf das Erstellen einer neuen Kategorie gehört, das in Create Category modelliert wird, gibt es innerhalb dieser Aktivität die entsprechende komplexe Aktion Create Category welche über in beide Richtungen navigierbare Verbindung zur Aktion zur Darstellung der Dateneingabe verknüpft ist. Auch die Aktivität Create Category unterscheidet sich von den bisher vorgestellten nur dadurch, dass zum Schluss noch die Aktion UploadFile zur Repräsentation des Uploadvorgangs ausgeführt wird.

Zu aller Letzt kommt noch die in Abbildung 4.11 gezeigte Aktivität zur Beschreibung der Erstellung eines neuen oder des Verändern eines bereits vorhandenen Kommentars. Diese trägt die Bezeichnung **Add Comment** und ist wieder eine etwas detailliertere Betrachtung wert. Ausnahmsweise beginnt diese nämlich mit der Systemaktion InsertOldComment, welche über mehrere Eingabepins mit dem Objektknoten Comment verbunden ist. Das modelliert die Tatsache, dass bevor der Benutzer einen Kommentar eingeben kann das System überprüft, ob ein Kommentar zur selben Seite bereits existiert und falls dies der Fall ist diesen in die Eingabefelder einfüllt. Der letzte Teil wird dargestellt durch die von InsertOldComment ausgehenden, typlosen Pins oldRating, oldText und oldLanguage, welche zu den Eingabepins Rating, Text und Language des Stereotyps *«inputElement»* führen. Diese gehören zu der Aktion SetData, welche nun endlich die Benutzereingaben modellieren soll. Darauf folgt der bereits bekannte Vorgang des Speicherns der Daten, nachdem die Änderungen bestätigt wurden.

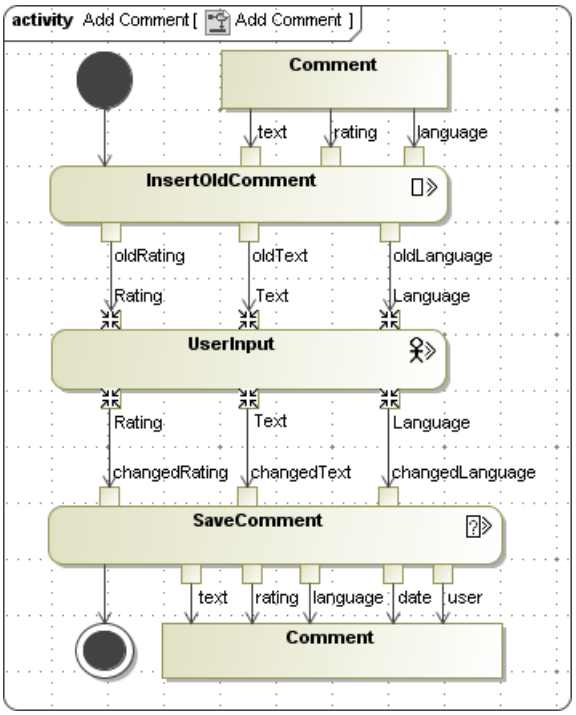


Abbildung 4.11: Das Aktivitätsdiagramm zu Philoponellas Anwendungsfall Add Comment

## 4.3 Prozessmodelierung

Da Aktivitätsdiagramme im Gegensatz zu Anwendungsfällen nicht nur in der Anforderungsanalysephase, sondern auch bei der Prozessmodellierung verwendet werden, können die Änderungen bisher vorgestellten Änderung theoretisch auch dafür verwendet werden. Allerdings sind die beiden neuen Stereotypen der Eingabe- und Ausgabepins zur Modellierung von Benutzeroberflächenelementen gedacht und sollten nicht innerhalb von Aktivitätsdiagrammen von Prozessen verwendet werden, da dies die beiden Bereiche Präsentation- und Prozessmodellierung vermischen würde, was wiederum das Prinzip der Trennung der Verantwortungsbereiche (Separation of Concerns) verletzt. Die beiden Stereotypen für die Aktionen hingegen fassen nur Aktionen zusammen und sollen die Größe eines Diagramms reduzieren. Daher gibt es keine Probleme, wenn man *«validatedAction»* und *«confirmedAction»* innerhalb von Aktivitätsdiagrammen der Prozessmodellierung ebenfalls einsetzt. Auch wenn es nicht überall zu Vereinfachungen führt, so gibt es doch einige Diagramme, welche dadurch kleiner und übersichtlicher werden. Wie sie sich dabei verändern soll hier kurz an einigen Beispielen aus Philoponella dargestellt werden.

### 4.3.1 Überprüfen von Benutzereingaben

Ein einfaches Beispiel für das Überprüfen der Benutzereingaben ist zunächst mal der Prozess SendMessage. Wie bereits erwähnt modelliert dieser das Versenden der Nachrichten zwischen den Benutzern. Dabei muss der Empfängername vom System auf ihre Gültigkeit hin überprüft werden. Bisher wurde dies durch die Aktion EnterData vom Typ *«userAction»* gefolgt von ValidateRecipient mit dem Stereotyp *«systemAction»* dargestellt. Auf diese folgte außerdem noch ein Entscheidungsknoten um den Verlauf nach der Auswertung zu bestimmen. Das ganze wird in der neuen Version des Aktivitätsdiagramms vereinfacht, indem die Aktion EnterData den Stereotyp *«validatedAction»* bekommt und die Aktion ValidateRecipient entfernt wird. Der Entscheidungsknoten kann ebenfalls entfernt werden, da *«validatedAction»* beinhaltet, dass die nachfolgende Aktion erst nach erfolgreicher Validierung ausgeführt werden kann. Abschließend wird das vereinfachte Diagramm in Abbildung 4.12 präsentiert, während das Originaldiagramm auf Seite 44 in Abbildung 3.6 zu finden ist.

### 4.3.2 Bestätigen der Systemvorgänge

Ein weiteres, in Abbildung 4.13 präsentiertes Beispiel ist AddComment, welcher das Erstellen oder Ändern eines Kommentars zu einer Seite darstellt. Die ursprüngliche Version ist in Abbildung 3.7 auf Seite 46 gezeigt. Der interessante Teil beginnt hierbei mit der Aktion UserInput des Stereotyps *«userAction»* und wird gefolgt von der Aktion ConfirmChanges

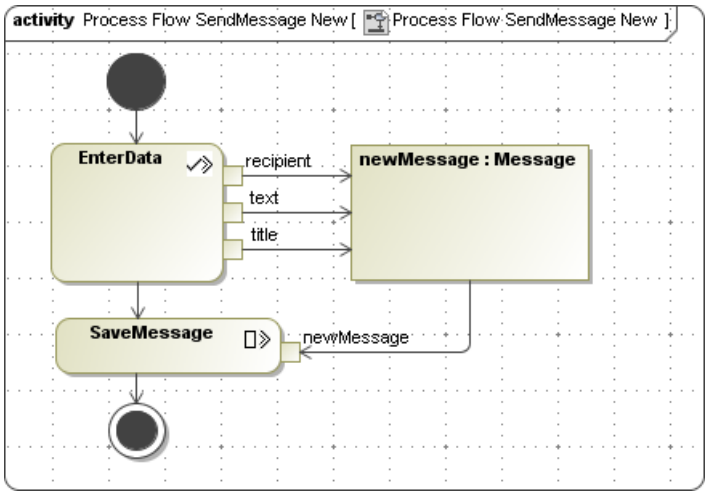


Abbildung 4.12: Das vereinfachte Prozessflussdiagramm zu Philoponellas Prozess SendMessage

des selben Typs. Dies soll das Eintragen der Daten und die Bestätigung einen eventuell bereits vorhandenen Kommentar zu überschreiben darstellen. Um zu modellieren, dass die Aktion SaveComment nur dann als nächstes ausgeführt wird, wenn der Benutzer dies auch bestätigt hat, folgt auf ist ConfirmChanges ein Entscheidungsknoten, welcher einen Pfad durch SaveComment und einen um diese Aktion herum beschreibt. Auch hier kann das Aktivitätsdiagramm vereinfacht werden indem zunächst eine Aktion einen neuen Stereotyp bekommt. In diesem Diagramm wird dazu der Stereotyp von SaveComment aus *«systemAction»* zu *«confirmedAction»*. Dadurch kann man nun die Aktion ConfirmChanges weglassen, da diese bereits in SaveComment integriert wurde. Analog zum ersten Beispiel kann man sich auch dadurch den Entscheidungsknoten sparen, denn *«confirmedAction»* stellt eine durch eine Benutzerbestätigung bedingte Abarbeitung der Aktion dar.

### 4.3.3 Aktivitäten mit unterschiedlichen Aktionen

Mehrere weitere Prozesse in Philoponella besitzen sowohl eine Validierung der Benutzereingaben als auch von einer Bestätigung abhängige Systemaktionen. Die Aktivitätsdiagramme für CreateCategory und ChangeSettings lassen sich auf die gleiche Weise vereinfachen, wie die beiden vorherigen Beispiele. Dies bedeutet, dass die beiden Aktionen zur Benutzereingabe und deren Auswertung zu einer Aktion des Stereotyps *«validatedAction»* zusammengelegt werden. Ebenso wird die Bestätigung durch den Benutzer und die dazugehörende Systemaktion zu einer Aktion des Typs *«confirmedAction»* zusammengefasst. Wie auch schon vorher fallen dabei die bisher benötigten Entscheidungsknoten weg.

Eigentlich funktioniert es genauso bei den Prozessen CreateLinkInfo und Register, jedoch ist die Besonderheit hier, dass die Benutzerbestätigung neben der Ausführung der Aktion auch

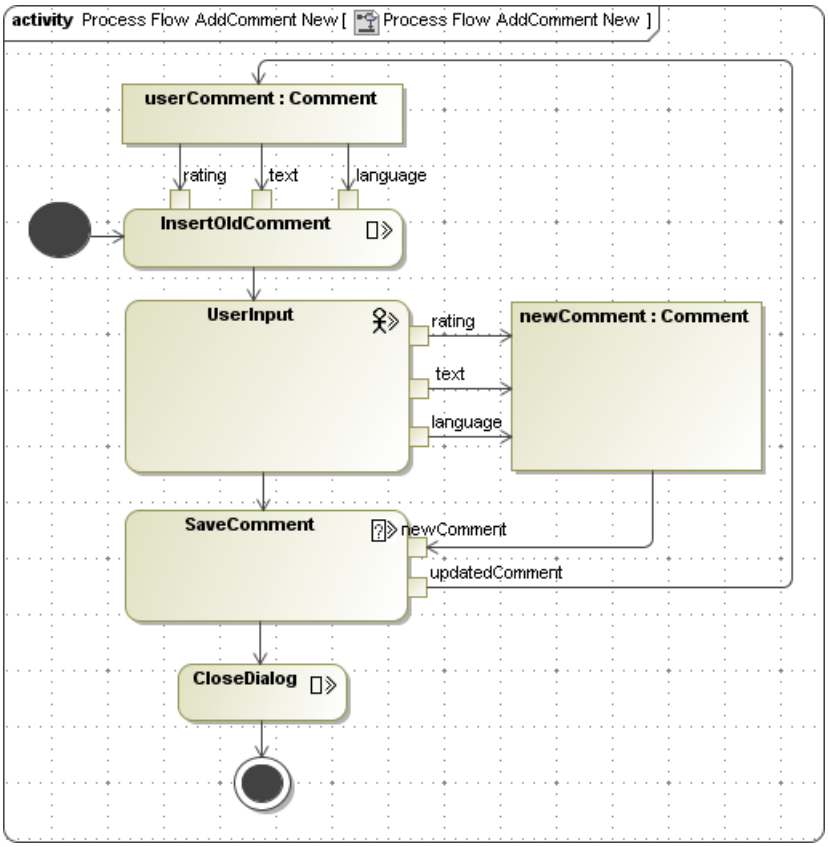


Abbildung 4.13: Das vereinfachte Prozessflussdiagramm zu Philoponellas Prozess AddComment

einen anderen Pfad im Aktivitätsdiagramm bewirkt. Um dies zu modellieren kann man von der entsprechenden *«confirmedAction»* zwei Verbindungen zu den beiden Anfangsaktivitäten der unterschiedlichen Pfade ziehen, welche jeweils mit einer Wächterbedingung ausgestattet werden, die ausdrückt, dass der eine Pfad bei Bestätigung und der andere bei Abbruch des Benutzers eingeschlagen werden soll. In Philoponella sind diese Bedingungen *«canceled»* und *«confirmed»*. Abbildung 4.14 zeigt schließlich auch dazu ein Beispiel in Form des Prozessflussdiagramms zu Register, welches mit der Abbildung 3.4 auf Seite 42 verglichen werden kann.

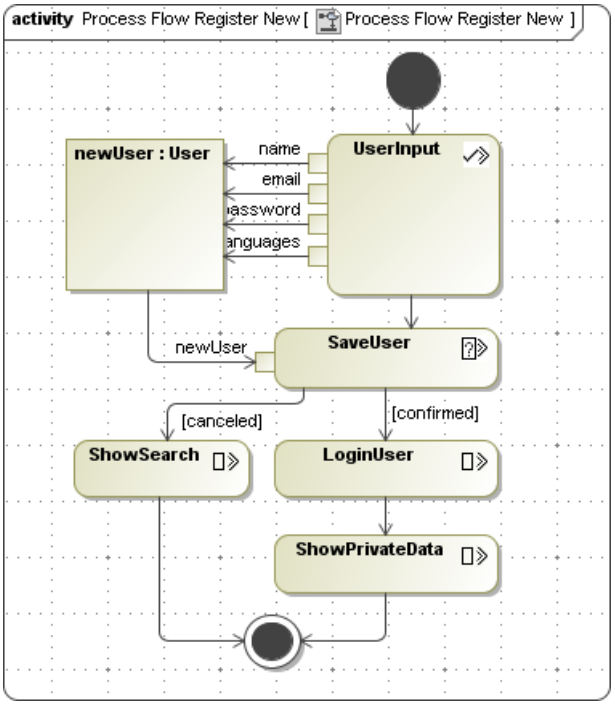


Abbildung 4.14: Das vereinfachte Prozessflussdiagramm zu Philoponellas Prozess Register

# 5 Transformation der Modelle

Ein Hauptgrund für die Erweiterung der Anforderungsanalysemodellierung in UWE ist die Möglichkeit automatisch die nachfolgenden Modelle der Analysephase daraus generieren zu können. Daher werden in diesem Kapitel die bisher erarbeiteten Regeln zur Transformation zusammen mit dem daraus resultierendem Ergebnis für Philoponella und den noch nicht gelösten Probleme beschrieben, wobei die Transformationsregeln dabei allerdings nur informell in deutscher Sprache beschrieben werden.

Bei der Transformation gibt es dabei zwei Herangehensweisen: Zum einen kann durch eine Transformation ein komplettes Modell von Grund auf generiert werden und zum anderen können kleinere Transformationen herangezogen werden um Teile des Modells wie Klassen oder Objekte in ein bereits vorhandenes Modell einzufügen. Dabei wird meist der eine Vorgang anhand des anderen skizziert, um den Zusammenhang und die Unterschiede dieser beiden Methoden hervorzuheben.

Natürlich kann das Ergebnis einer solchen Transformation niemals vollständig den von Hand erstellten Modellen gleichen, da eine Voraussetzung dafür wäre, dass bereits alle Information in den Modellen der Anforderungsanalyse enthalten sind, wodurch allerdings die Anforderungsanalyse bereits den Platz der Modellierung einnehmen würde und diese überflüssig machen würde. Dennoch können solche automatischen Transformationen vor allem zu Beginn der Modellierung eine Hilfe sein und das Erzeugen der Diagramme beschleunigen.

Außerdem werden hier wie üblich die vier Aspekte eines Modells in UWE (Inhalt, Navigation, Prozesse und Präsentation) einzeln betrachtet. Dabei werden für jeder Teil folgende vier Abschnitte beschrieben: vollständige Transformation, Ergebnis für Philoponella, Transformationen zur Gewinnung einzelner Elemente und Zusammenfassung.

Zu erwähnen wäre außerdem noch, dass Transformationen aus dem Modell der Anforderungsanalyse zur Adaption hier noch nicht betrachtet werden können, da die Adaption derzeit noch nicht als ein Teil von UWE implementiert wurde.

## 5.1 Erzeugung des Inhalts

Als erstes wird mal wieder das Inhaltsmodell einer Webanwendung betrachtet. Die hierfür benötigten Daten kann man vollständig aus den Aktivitätsdiagrammen des Anforderungsanalyse gewinnen, da hier die Objekte und die verschiedenen Daten in vorm von Pins gespeichert werden.

### 5.1.1 Einfügen von Klassen

Die Transformationen für das Inhaltsmodell sind wie das Modell selbst recht einfach. Natürlich bietet es sich hier an, den Inhalt durch hinzufügen neuer Klassen zu erweitern. Der Vorgang entspricht dabei der vollständigen Transformation, wobei allerdings nur eine einzelne, ausgewählte Klasse betrachtet wird. Dabei soll dem Benutzer zunächst eine Liste möglicher Klassen aus der Anforderungsanalyse präsentiert werden. Gewonnen wird diese durch betrachten der Objektknoten der Aktivitätsdiagramme. Ist eine Klasse ausgewählt worden, dann gibt es drei denkbare Möglichkeiten wie detailliert diese Klasse generiert werden können.

Die erste Möglichkeit mit dem Titel **EmptyContentClass** ist einfach eine leere Klasse mit dem entsprechenden Namen zu erzeugen. Da dies nur einen geringen Nutzen darstellt gibt es als zweite Alternative **AttributedContentClass**, welche auch eine Erzeugung der Attribute beinhaltet. Diese werden gewonnen, indem die Aus- und Eingabepins der Aktivitätsdiagramme betrachtet, welche mit den Objekten verbunden sind, die sich mit der Klasse den Namen teilen.

Die letzte Option bietet zudem noch die Verbindung der Klassen, wofür ebenfalls die Pins herangezogen werden und trägt die Bezeichnung **ContentClassWithAssociations**. Trägt ein solcher Pin den gleichen Namen wie eine vorher erstellte Klasse, so wird beim Inhalt eine Assoziation zwischen dieser und der Klasse, welche aus dem Objekt, der mit diesem Pin verbunden ist entstanden ist erstellt. Ähnlich verhält es sich mit Pinnamen, welche den Namen einer Klasse enthalten, nur dass hierbei die Assoziation den Namen des Pins übernimmt, nachdem aus diesem der Klassenname und ein eventuell nachfolgendes, Mehrzahl darstellendes 's' entfernt wurde. Bei Pins ohne eine entsprechende Klasse im Inhaltsmodell wird wie im zweiten Fall dieser als Attribut übernommen.

Da die letzten beiden Methode daran scheitern können, dass die Klassen noch nicht alle erstellt wurden, sollte es außerdem eine Methode geben, eine Klasse automatisch mit Attributen und Assoziationen zu füllen. Hierbei kann man ebenfalls unterscheiden, ob man nur die Attribute haben will oder auch die Assoziationen. Diese beiden Regeln, die auch als Teil der beiden zu Letzt beschriebenen Regeln aufgefasst werden können, tragen die

Namen **AddAttributesToContentClass**, falls nur die Attribute gewonnen werden können, oder **AddAssociationsToContentClass**, wenn ebenfalls die Assoziationen generiert werden sollen.

Insgesamt ergeben sich für das Inhaltsmodell fünf verschiedene Transformationsregeln, mit denen man das Inhaltsmodell automatisch erweitern kann. Diese sind nochmal in Tabelle 5.1 kurz zusammengefasst.

| **Transformationsregel** | **Eigenschaften** | |
|---|---|---|
| **EmptyContentClass** | **Quelle:** | Objektknoten |
| | **Ziel:** | Klasse |
| | **Zielkontext:** | Inhaltsmodell |
| | **Definition:** | Für jeden Objektknoten wird eine Klasse mit dem gleichen Namen erzeugt. |
| **AttributedContentClass** | **Quelle:** | Objektknoten und Pins |
| | **Ziel:** | Klasse mit Attributen |
| | **Zielkontext:** | Inhaltsmodell |
| | **Definition:** | Komposition von **EmptyContentClass** und **AddAttributesToContentClass**. |
| **ContentClassWithAssociations** | **Quelle:** | Objektknoten und Pins |
| | **Ziel:** | Klasse mit Attributen und Assoziationen |
| | **Zielkontext:** | Inhaltsmodell |
| | **Definition:** | Komposition von **EmptyContentClass** und **AddAssociationsToContentClass** ausgeführt. |
| **AddAttributesToContentClass** | **Quelle:** | Pins |
| | **Ziel:** | Attribute einer Klasse |
| | **Zielkontext:** | Klasse des Inhaltsmodells |
| | **Definition:** | Für jeden Pin des Objektknotens ein gleichnamiges Attribut in dieser Klasse erzeugt. |
| **AddAssociationsToContentClass** | **Quelle:** | Pins |
| | **Ziel:** | Attribute einer Klasse |
| | **Zielkontext:** | Klasse des Inhaltsmodells |
| | **Definition:** | Für jeden Pin des Objektknotens ein gleichnamiges Attribut in dieser Klasse oder eine Verbindung zur Klasse mit dem Pinnamen erzeugt. |

Tabelle 5.1: Transformationsregeln zur Ergänzung des Inhaltsmodells

### 5.1.2 Vollständige Generierung des Modells

Die vollständige Transformation des Inhaltsmodells ergibt sich nun als eine einfache Komposition mehrerer vorhin beschriebener Schritte. Dabei werden im ersten Schritt die leeren Klassen wie oben durch die Regel **EmptyContentClass** beschrieben erzeugt und im zweiten Durchgang füllt man diese mit Attributen und Assoziationen **AddAssociationsToContentClass**. Der einzige Unterschied besteht hier darin, dass statt einer einzelnen Klasse, diese Regeln automatisch für alle möglichen Elemente automatisch aufgerufen werden.

### 5.1.3 Der transformierte Inhalt von Philoponella

Das Ergebnis dieser Transformation zeigt bei Philoponella große Ähnlichkeit zum bereits vorgestellten Modell. Zur besseren Übersicht ist hier das vollständige, beschriebene Diagramm in Abbildung 5.1 abgebildet, wobei das Originalmodell in Abbildung **??** auf Seite **??** dargestellt wird.

Zunächst enthält es ebenfalls die Klassen User, LinkInfo, Message, Comment und Image. Auch die Klasse Category ist in beiden Modellen enthalten, wobei diese im Originalmodell allerdings etwas anders benannt ist. Allerdings sind die beiden Klassen Description und Keyword unter den Tisch gefallen, da sie keine so wichtige Rolle spielen und in der Anforderungsanalyse nirgends explizit dargestellt wurden. Ebenfalls fällt sofort auf, dass viele der wichtigen Attribute nach der Transformation bereits vorhanden sind, es aber wie erwartet einige Abweichungen in den Details gibt.

Was Attribute angeht ist die komplexeste Klasse hierbei eindeutig User, daher wird diese hier zuerst betrachtet. Das erste und größte Problem darin sind die erzeugten Attribute friends, favorites und keywords, welche eigentlich Assoziationen darstellen sollten. Weiterhin fehlen einige Attribute, welche jedoch für die Grundfunktionen der Anwendung unwichtig sind. Besonders erwähnenswert sind außerdem die beiden Attribute showIgnoredFriends und showIgnoredFavorites, welche im Originalmodell die Namen showIgnoredUsers und showIgnoredLinks tragen. In den Diagrammen der Anforderungsanalyse wurde bewusst auf diese Namen verzichtet, da es bei der Transformation zu falschen Assoziation geführt hätte.

Ähnlich verhält es sich bei den Klasse LinkInfo, Message und Image. Auch hier fehlen einige Attribute und Assoziationen und die erzeugten Attribute author, recipient und descriptions sollten Assoziationen sein, wobei dies für die ersten beiden nicht aus den Namen erkennbar sind und beim letzten Fall noch nicht einmal die entsprechende Klasse existiert.

Interessanter Weise ist die Klasse Comment tatsächlich bis auf einige Umbenennungen vollständig und korrekt aus der Transformation hervorgegangen. Allerdings kann diese als ein Spezialfall der Klasse Description betrachtet werden und wird ebenso im Inhaltmodell repräsentiert, was nicht zuletzt wegen dem Fehlen dieser Klasse in der Anforderungsanalysemodellen nicht ersichtlich ist.

Auch Category ist bis auf den Namen fast richtig erstellt worden. Besonders zu erwähnen wäre hier das Attribut id, welches nicht aus dem Modell der Anforderungsanalyse gewonnen werden kann bzw. extra bei der Modellierung des Inhalts eingefügt werden muss. Dies ergibt sich daraus, dass die Objekte dieser Klasse eindeutig zu identifizieren sein müssen. Jedoch dürfen verschiedene Objekte den gleichen Namen tragen, wodurch das einzige in Frage kommende Attribut für die Modellierung dieser Eigenschaft wegfällt und ein zusätzliches Attribut notwendig macht.

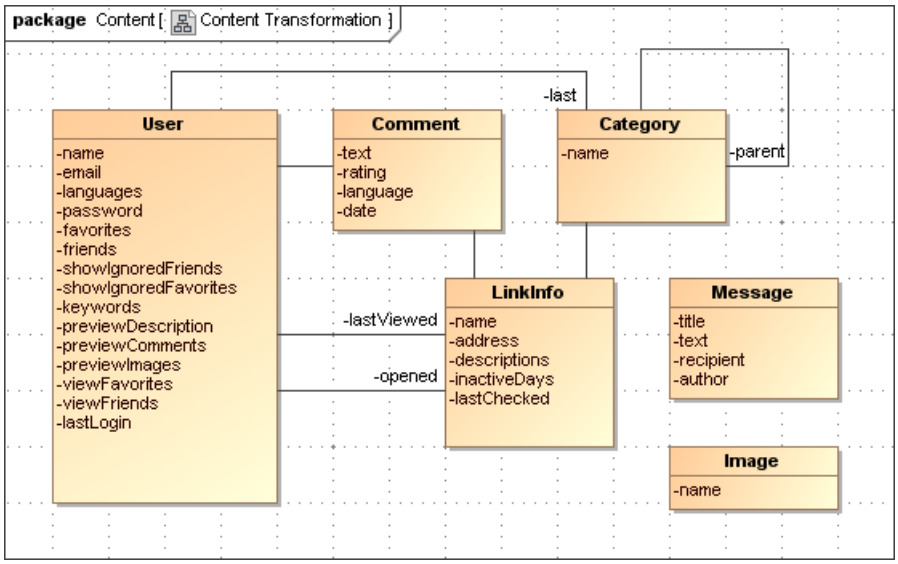


Abbildung 5.1: Philoponellas Inhaltsmodell erzeugt durch Transformation

### 5.1.4 Zusammenfassung

Zusammenfassend lässt sich zu dieser Transformation sagen, dass das Inhaltsmodell sowohl durch eine komplette Transformation sowie durch einzelne Transformationsregeln sehr genau erzeugt werden kann, wenn die Objekte und die damit verbundenen Daten sorgfältig in der Anforderungsanalysephase modelliert und die Namen dafür passend ausgewählt wurden. Vor allem für die Erzeugung richtiger Assoziationen an Stelle von Attributen ist die Namensgebung entscheidend, jedoch ist diese aber in dieser Entwicklungsphase für die beteiligten Personen nicht immer bereits ersichtlich oder auch einfach nicht verständlich. Auch die unwichtigeren, also nur für die Randfunktionen einer Anwendung benötigten Daten müssen meist nach der Transformation in dem entstandenen Modell ergänzt werden, da diese während der Anforderungsanalyse oft nicht beachtet werden können.

## 5.2 Bestimmung der Navigationsstruktur

Als nächstes wird die Generierung des Navigationsmodells betrachtet. Die für die Erzeugung dieser Struktur notwendigen Daten liefern hauptsächlich die Anwendungsfälle aus dem entsprechenden Diagramm. Allerdings ist die Menge an Informationen für diesen Abschnitt sehr begrenzt, was dazu führt, dass für manche Stereotypen deren Objekte bei der Transformation nicht berücksichtigt werden können.

### 5.2.1 Erweiterung der Navigationsstruktur

Die wichtigste Hilfe gibt es im Navigationsmodell durch die automatische Generierung der beiden Klassen *«navigationClass»* und *«procesClass»*. Diese können zunächst wie beim Inhalt allein erzeugt werden, wozu hier aber alle Anwendungsfälle durchlaufen und ihr Stereotyp betrachtet werden, wobei hier jetzt die beiden neu eingeführten Stereotypen *«navigation»* und *«process»* zum Einsatz kommen. Ein Anwendungsfall des Typs *«navigation»* wird dabei in der Transformationsregel **NavigationClass** in eine *«navigationClass»* im Navigationsmodell übersetzt, während bei der Regel **ProcessClass** ein Anwendungsfall vom Typ *«process»* zu *«processClass»*-Elementen werden. Die beiden Stereotypen *«adaptation»* und *«observation»* hingegen sollen die Modellierung der Adaptation ermöglichen und werden daher bei dieser Transformation nicht weiter beachtet. Zur Benennung der so entstandenen Klassen kann man einfach die Namen der Benutzerfälle hernehmen, nachdem aus diesen die Leerzeichen entfernt wurden.

Ebenfalls wie beim Inhaltsmodell gibt es hier die Möglichkeit zu den automatisch generierten Klassen auch ihre Beziehungen zu erzeugen, indem man die Assoziationen unter den Anwendungsfällen auswertet. Hierbei gibt es ebenfalls die Möglichkeit diese gleich beim Erstellen der Klasse hinzuzufügen, was durch die Regel **NavigationClassWithAssociations** beschrieben wird oder auch nachträglich durch die Transformation **AddAssociationsToNavigationClass** zu ergänzen. Da der Vorgang beide Male gleich ist, kann die erste Regel als die Kombination von **NavigationClass** und **AddAssociationsToNavigationClass** angesehen werden, bei der man den Anwendungsfall betrachtet, aus dem die Klasse entstanden ist und nacheinander alle mit diesem verbundenen Anwendungsfälle, welche mit einer Klasse der Navigation gleich benannt sind.

Sind hierbei zwei Anwendungsfälle des Stereotyps *«navigation»* verbunden, so wird auch eine ungerichtete *«navigationLink»*-Verbindung zwischen den beiden aus den Anwendungsfällen erzeugten Klassen erstellt. Ist hingegen zumindest einer der Anwendungsfälle vom Typ *«process»*, dann spielt auch die Art der Beziehung eine Rolle. Wenn diese eine Einschließen-Beziehung(*«include»*) ist, so wird auch im Navigationsmodell eine gleichgerichtete Verbindung erstellt, welche natürlich den Stereotyp *«processLink»* bekommt. Bei der Erweitern-Beziehung(*«extend»*) hingegen wird auch eine gerichtete Beziehung des Typs *«processLink»* erzeugt, jedoch zeigt diese diesmal in die entgegengesetzte Richtung.

Der gleiche Vorgang ist natürlich auch für Prozessklassen möglich, was durch die beiden Transformationsregeln **ProcessClass** und **AddAssociationsToProcessClass** beschrieben wird. Da hier allerdings immer eine Prozessklasse beteiligt ist, fällt hier die Option zum erstellen eines *«navigationLink»* weg.

Eine letzte Hilfe ist das Einfügen des Stereotyps *«menu»*. Diese Option steht bereits

bei der Auswahl einer Verbindung des Navigationsmodells zur Verfügung, könnte aber auch automatisch bei der Erzeugung einer *«navigationClass»* mit seinen Verbindungen hinzugefügt werden, falls es mehrere ausgehende Assoziationen gibt, was die Transformationsregel **NavigationClassWithMenu** erledigt. Dabei wird eine kleine Regel mit dem Namen **CreateMenu** eingeführt, nach der zwischen *«menu»* und *«navigationClass»* eine Kompositionsbeziehung erstellt und alle von der Navigationsklasse ausgehenden Beziehungen werden zur neu erzeugten Menüklasse verschoben. So kann **NavigationClassWithMenu** auch als ein Hintereinanderausführung der Transformationsregeln **NavigationClass**, **AddAssociationsToNavigationClass** und **CreateMenu** betrachtet werden.

Zusammenfassend gibt es in diesem Modell fünf Möglichkeiten: Die einfache Erzeugung der beiden Klassenarten, dazu das zusätzliche Hinzufügen der Assoziationen und als letze Option kann auch noch wie im vorherigen Absatz beschrieben ein Menü erstellt werden. Dies ist auch nochmal im 5.2 dargestellt.

### 5.2.2 Vollständige Generierung des Modells

Der vollständige Transformationsvorgang beginnt erstmal unabhängig von den bisherigen Transformationsregeln damit, dass die Navigationsklasse Home erzeugt wird. Diese repräsentiert den Startpunkt jeder Webanwendung und wird nicht explizit im Modell der Anforderungsanalyse dargestellt. Um außerdem ihre Sonderstellung auszuzeichnen bekommt sie die Eigenschaft *«isHome»* und *«isLandmark»*.

Für den nächsten Schritt werden analog zur den ersten beiden Transformationsregeln alle Anwendungsfälle durchlaufen und ihr Stereotyp betrachtet. Daraufhin werden ebenfalls genauso die Assoziationen zu den erzeugten Klassen hinzugefügt. Das bedeutet, dass je nach Stereotyp des Anwendungsfalls die Regel **NavigationClassWithAssociations**, **ProcessClassWithAssociations** oder überhaupt keine aufgerufen wird.

Nachdem so die Grundstruktur des Navigationsmodells erzeugt wurde, fehlen aber noch diverse Klassen und Verbindungen, von denen einige hier ergänzt werden können. Hierfür gibt es in den ergänzenden Transformationen kein Gegenstück, da eine manuelle Erzeugung der nächsten beiden Schritte nicht durch automatisierte Hilfen merklich erleichtert werden kann.

So gibt es Klassen, die nicht von der Navigationsklasse Home aus erreicht werden können, was natürlich nicht vorkommen darf. Daher werden für alle solchen Klassen eine Verbindung zur Klasse Home erstellt, wobei diese abhängig vom Stereotyp der Klasse entweder ein *«navigationLink»* oder ein *«processLink»* sind.

Ein ähnliches Problem stellen Klassen des Typs *«processClass»* dar, welche keine von diesen ausgehende Assoziation besitzen. Auch wenn es Ausnahmen gibt, so erwartet man wohl

| Transformationsregel | Eigenschaften | |
|---|---|---|
| **NavigationClass** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse des Stereotyps *«navigationClass»* |
| | **Zielkontext:** | Navigationsmodell |
| | **Definition:** | Für jeden Anwendungsfall wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **ProcessClass** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse des Stereotyps *«processClass»* |
| | **Zielkontext:** | Navigationsmodell |
| | **Definition:** | Für jeden Anwendungsfall wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **NavigationClassWithAssociations** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse des Stereotyps *«navigationClass»* |
| | **Zielkontext:** | Navigationsmodell |
| | **Definition:** | Komposition von **NavigationClass** und **AddAssociationsToNavigationClass**. |
| **NavigationClassWithMenu** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse mit Attributen und Assoziationen |
| | **Definition:** | Komposition von **NavigationClass**, **AddAssociationsToNavigationClass** und **CreateMenu** aufgerufen. |
| **ProcessClassWithAssociations** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse mit Attributen und Assoziationen |
| | **Zielkontext:** | Navigationsmodell |
| | **Definition:** | Komposition von **ProcessClass** und **AddAssociationsToProcessClass** aufgerufen. |
| **AddAssociationsToNavigationClass** | **Quelle:** | Anwendungsfall des Typs *«navigation»* und seine Beziehungen |
| | **Ziel:** | Assoziationen einer Klasse |
| | **Zielkontext:** | *«navigationClass»*-Element |
| | **Definition:** | Für jeden Anwendungsfall, welcher eine Verbindung zum ausgewählten Anwendungsfall besitzt wird eine passende Verbindung erzeugt. |
| **AddAssociationsToProcessClass** | **Quelle:** | Anwendungsfall des Typs *«process»* und seine Beziehungen |
| | **Ziel:** | Assoziationen einer Klasse |
| | **Zielkontext:** | *«processClass»*-Element |
| | **Definition:** | Für jeden Anwendungsfall, welcher eine Verbindung zum ausgewählten Anwendungsfall besitzt wird eine passende Verbindung erzeugt. |
| **CreateMenu** | **Quelle:** | Klasse des Typs *«navigationClass»* |
| | **Ziel:** | *«menu»*-Element |
| | **Zielkontext:** | *«navigationClass»*-Element |
| | **Definition:** | Falls die Klasse mehrere ausgehende Verbindungen besitzt, wird ein Menüobjekt hinzugefügt. |

Tabelle 5.2: Transformationsregeln zur Ergänzung des Navigationsmodells

in der Regel, dass die Webanwendung nach der Ausführung des durch diese Klasse modellierten Prozesses zurück zur vor der Ausführung bestehenden Ansicht kehrt. Daher sollen von diesen Klassen aus die eingehenden Assoziationen rückwärts durchlaufen werden. Ist die Klasse am anderen Ende vom Typ *«navigationClass»*, so wird eine neue gerichtete Assoziation von der Prozessklasse zu dieser hergestellt. Falls dadurch aber ebenfalls eine Prozessklasse erreicht wird, so wird der Vorgang so lange wiederholt, bis man auf eine Navigationsklasse stößt.

Schließlich gibt es noch den Stereotyp *«menu»*, welche im letzten Schritt wie bei der Transformationsregel **CreateMenu** ergänzt wird. Da die dazu gehörenden Klassen überall dort eingefügt werden sollen, wo eine Klasse des Typs *«navigationClass»* mit mehr als einer ausgehenden Verbindung steht, werden auch nur diese Betrachtet.

### 5.2.3 Das transformierte Navigationsmodell von Philoponella

Auch hier zeigt die Anwendung dieser Transformation in Philoponella viel Ähnlichkeit zu der manuell erstellten Navigationsstruktur. Jedoch gibt es vor allem bei den Beziehungen zwischen den Klassen mehrere Unterschiede. Zunächst fällt auf, dass viele Klassen unterschiedlich benannt sind, was allerdings nur ein geringeres Problem darstellt.

So trägt im Bereich, welcher die Klassen um die Linkinformationen behandelt die zentrale Navigationsklasse im Originalmodell den Namen LinkInfo und im durch eine Transformation entstandenen Modell die Bezeichnung ViewLinkDetails. Erstaunlicher Weise sind die Namen bei den Klassen des Stereotyps *«processClass»* in diesem Bereich alle gleich. Bei den Verbindungen zwischen den verschiedenen Klassen gibt es dabei auch viele Gemeinsamkeiten und nur zwei Unterschiede. Zunächst ist der Prozess CheckLinkStatus durch die Transformation so modelliert, dass er sowohl von dieser Navigationsklasse startet und endet, wohingegen es im anderen Modell von der Klasse ListIndex ausgeht. Dadurch soll modelliert werden, dass dieser Prozess beim Auswählen der Detailansicht für einen Link aus einer Suchliste ausgeführt werden. Bei der Transformation geht diese Information jedoch verloren.

Der zweite Unterschied ist die Modellierung der beiden Vorgänge CreateLinkInfo und CreateCategory. Um zu verdeutlichen, dass diese Aktivitäten nur einem angemeldeten Benutzer zur Verfügung stehen sind diese vom Menü PersonalMenu der persönlichen Ansicht aus erreichbar. Diese Zusammenhang kann nicht so einfach aus dem Benutzerfalldiagramm der Anforderungsanalyse gewonnen werden und daher ist hier der Ausgangspunkt dieser Prozesse das Menü HomeMenu der Startansicht. Außerdem soll der Prozess CreateCategory innerhalb von CreateLinkInfo ablaufen, was dadurch ausgedrückt werden sollte, dass alle Verbindungen von CreateCategory diesen mit CreateLinkInfo verbinden. Jedoch geht auch dies durch die Transformation verloren und es entsteht nur eine einfache Liste.

Als nächstes soll hier die einfache Gruppe der Klassen um die Ansicht der Details

anderer Benutzer betrachtet werden. Im Originalmodell heißt diese UserDetails und ViewUserDetails nach der Transformation. Hier fehlt die entsprechende Klasse des Typs *«menu»*, wobei dies nur eine Art Folgefehler ist, da hier mehrere Klassen nicht im Transformationsmodell verlorengegangen sind und nur die Prozessklasse AddFriend übrig ist, was eine Menüklasse überflüssig macht. Die nicht bei der Transformation übernommenen Klassen sind FriendFavoriteIndex und FriendFriendIndex, welche beide zum Stereotyp *«index»* gehören. Leider können diese noch nicht automatisch generiert werden.

Aus diesem Grund fehlen auch solche Klassen im dritten Abschnitt des Navigationssystems, welcher die Ansicht und Manipulation eigener Daten eines angemeldeten Benutzers darstellt. Hier sind die meisten Klassen des Stereotyps *«processClass»* ebenfalls korrekt erzeugt worden, aber es gibt einen Fehler. Ähnlich wie bei der Erzeugung der Seiteninformationen und Kategorien gibt es auch ein Problem beim Modellieren der An- und Abmeldung der von Benutzern. Während die beiden Prozesse Register und Login den Übergang von einem Gast zu einem nicht angemeldeten Benutzer darstellen, bilden Unregister und Logout das dazugehörende Gegenstück. Auch diese Information ist im Anforderungsanalysemodell nicht erfassbar, weshalb die zu diesen Prozessen gehörenden nur Verbindungen zur Klasse Home oder untereinander besitzen. Genauso entsteht bei der Transformation die fehlerhafte Darstellung, dass das An- und Abmelden nur innerhalb der entsprechenden Registrierungsvorgänge möglich ist, da hier Login nur von Register und Logout nur Unregister erreichbar sind. Diese sollten aber natürlich auch eigenständig ausgeführt werden können, was eine von Home ausgehende Verbindung zu diesen benötigen würde.

Der letzte Abschnitt, welcher den Ausgangspunkt der Anwendung und die Suche nach Seiteninformationen modellieren soll hat neben den bereits beschriebenen fehlerhaften Verbindungen zu diversen Prozessen noch zwei weitere Unterschiede zum Originalmodell. Zunächst wird die Darstellung der Listen mit Informationen nach der Transformation als eine Klasse des Typs *«navigationClass»* modelliert, während es im manuell erstellten Modell nur durch ein *«index»* repräsentiert wird. Nun ist eine zusätzliches *«navigationClass»*-Element nicht unbedingt falsch, aber dennoch wird auch eine Klasse *«index»* benötigt. Der zweite Unterschied ist ähnlich geartet und hat mit der Modellierung der Suche zu tun. Die Transformation aus dem Anforderungsmodell generiert dabei ebenfalls nur eine Klasse ChangeBrowseSettings des Stereotyps *«navigationClass»*. Allerdings wäre hier die Modellierung aus dem Originalmodell durch den Stereotyp *«query»* günstiger. Schließlich ist auch die Information verloren gegangen, dass die Detailansicht für die in der Liste dargestellten Links von hier aus erreichbar sein soll.

Dieses hier beschriebene Modell ist nochmal in Abbildung 5.2 dargestellt. Außerdem kann es mit dem manuell erstellten Modell auf Seite 37, Abbildung 3.2 verglichen werden.

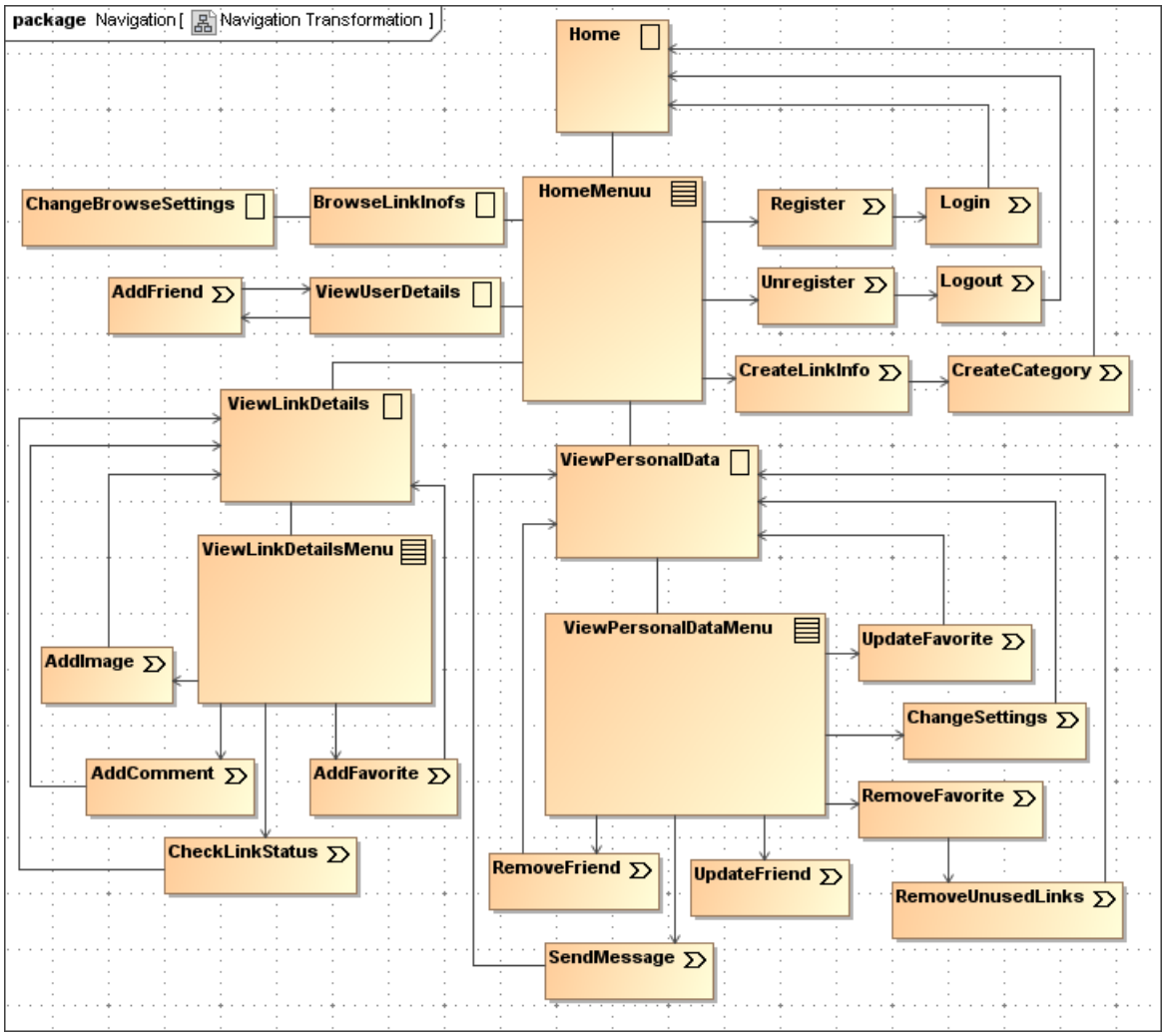


Abbildung 5.2: Philoponellas Navigationsmodell erzeugt durch Transformation

### 5.2.4 Zusammenfassung

Alles in Allem lässt sich zu dieser Transformation sagen, dass auch wenn dies keine vollkommen fehlerfreie Generierung des Navigationsmodells darstellt, das Ergebnis zumindest in den entscheidenden Punkten große Ähnlichkeit zu dem Originalmodell aufweist. So sind die Klassen der Stereotypen *«navigationClass»* und *«processClass»* bis auf einige Bezeichner alle richtig erzeugt worden. Es stellt auch eine große Vereinfachung dar, diese angepasst an die Anforderungsanalyse erstellen zu können. Auch können so viele der Beziehungen richtig ergänzt werden. Das größte Problem stellt die Schwierigkeit zur automatischen Generierung der Stereotypen *«index»*, *«query»* und *«guidedTour»* dar, für welche noch keine passenden Regeln vorhanden sind, da diese Informationen sich nicht automatisch aus der Anforderungsanalyse gewinnen lassen.

## 5.3 Überführung der Anwendungsfälle in Prozesse

Die nächste Transformation die hier betrachtet werden soll ist die Erzeugung der Prozesse. Obwohl hier zwei Arten von Diagrammen erzeugt werden, kann diese Gruppe der Transformationen ziemlich einfach erklärt werden. Der Grund hierfür liegt darin, dass sowohl die verschiedenen Prozesse für die Prozessstruktur aus den Anwendungsfällen, als auch deren Ausformulierung durch die Aktivitätsdiagramme für die Prozessflussdiagramme fast direkt abgelesen werden können. Jedoch auch hier lassen sich nicht alle Informationen automatisch aus der Anforderungsanalyse gewinnen.

### 5.3.1 Transformation von Prozessen

Innerhalb eines Prozessstrukturdiagramms gibt es zunächst die gleiche Möglichkeit wie beim Navigationsmodell die Klassen vom Typ *«processClass»* zu erzeugen, indem man einfach eine Liste der Möglichen Klassen aus den Anwendungsfällen des Stereotyps *«process»* gewinnt. So kann hier die Transformationsregeln **ProcessClass** einfach aus den Transformationen für das Navigationsmodell übernommen werden.

Auch hier gibt es die Möglichkeit einige Beziehungen automatisch generieren zu lassen, was wiederum gleichzeitig mit der Prozessklasse oder auch nachträglich geschehen kann. Im Prozessstrukturdiagramm lässt sich die Generalisierungsbeziehung für zwei Klassen aus der Assoziation *«include»* im Anwendungsfalldiagramm folgern, da dadurch der Umstand dargestellt werden soll, dass der einschließende Prozess auf jeden Fall die Funktionalität des anderen Prozesses beinhaltet. Falls dies bei einer bestehenden Klasse geschieht, wird es durch die Regel **AddGeneralizationsToProcessClass** beschrieben oder wenn dies gleich bei der Erzeugung der Klasse geschehen soll, heißt die Regel **ProcessClassWithGeneralization**. Die erweiternde Beziehung aus dem Anwendungsfalldiagramm lässt sich allerdings nicht sinnvoll zu einer Beziehung übersetzen, da sie nur das mögliche Auftreten des Vorgangs andeutet und somit keine klare Strukturregel darstellt.

Bei der Erzeugung der Prozessflussdiagramme gibt es auch eine Option diese automatisch durch die Regel **AddActivityDiagramToProcessClass** zu einer bestehenden Klasse generieren zu lassen. Soll dies wie oben gleichzeitig bei der Erzeugung der Klasse gemacht werden, so beschreibt dies **ProcessClassWithActivityDiagram**.

Man beginnt hierbei damit den entsprechenden Anwendungsfall zur ausgewählten Prozessklasse zu bestimmen und sein Aktivitätsdiagramm in das Prozessflussdiagramm der Klasse zu kopieren. Dies ist möglich, da beide Aktivitätsdiagramme darstellen und diese außerdem den Ablauf zum selben Vorgang beschreiben sollen. Es gibt jedoch zwei Unterschiede zwischen einem Diagramm für die Anforderungsanalyse und einem Prozessflussdiagramm, die

nach dem Kopieren angepasst werden müssen.

Als erstes dienen die beiden Stereotypen *«inputElement»* und *«outputElement»* der Eingabe- und Ausgabepins dazu, während der Anforderungsanalyse einen Teil der Präsentationsstruktur festzuhalten. Da die Beschreibung der Präsentation aber einen eigenen Teil der Modellierung darstellt, haben diese Stereotypen nichts innerhalb der Prozessdarstellung verloren. Daher werden alle Vorkommen dieser aus dem Aktivitätsdiagramm einfach entfernt.

Beim zweiten Schritt wird der Umstand beseitigt, dass die Objektknoten in den so erstellten Prozessflussdiagrammen keiner Klasse zugeordnet sind. Hier können die Klassen anhand der Namen der Objektknoten bestimmt werden, was allerdings nur bei der Generierung der Inhaltsklassen aus den Modellen der Anforderungsanalyse problemlos abläuft. Kann keine passende Klasse gefunden werden, dann wird den Objektknoten keine Klasse zugewiesen. Zwar wäre es auch denkbar eine neue Klasse dafür im Inhaltsmodell zu erzeugen, aber wenn dies erwünscht worden wäre, dann wäre diese wohl im Inhaltsmodell bereits generiert worden.

Um zusätzlich noch die Gefahr von Verwechslungen zu vermeiden, sollten die verschieden Objektknoten umbenannt werden, wenn mehr als zwei Objekte des selben Typs in einem Aktivitätsdiagramm vorkommen. Dazu kann man bei einem Objekt, welcher nur mit Ausgabepins verbunden ist den Präfix `new` und bei nur mit Eingabepins verbundenen Objektknoten den Präfix `old` hinzufügen. Außerdem liegt es nahe die Objektknoten nach ihrer Klasse zu nummerieren.

Als letztes werden überdies die Aktionen, welche den gleichen Namen wie ein Prozesses besitzen in komplexe, durch eben diese Prozesse ausformulierten Aktionen umgewandelt.

Also lässt sich für diesen Teil der Modellierung festhalten, dass es analog zum Navigationsmodell zunächst zwei Möglichkeiten gibt eine Prozessklasse zu erzeugen. Weiterhin können die Assoziationen zu einem Prozess nachträglich erstellt werden. Aber am wichtigsten ist hier wohl die Option ein Prozessflussdiagramm zu einer Prozessklasse automatisch aus der Anforderungsanalyse ableiten zu können. Alle Möglichkeiten sind auch hier in der nachfolgenden Tabelle (Tabelle 5.3) zusammengefasst.

### 5.3.2 Generierung aller Prozesse

Zu Beginn kommt hier die Generierung des Prozessstrukturdiagramms. Dieses kann wie im vorherigen Abschnitt beschrieben zunächst durch Erzeugung aller möglichen Prozessklassen erstellt werden, wobei im zweiten Schritt die Assoziationen hinzugefügt werden.

Bei den Aktivitätsdiagrammen läuft die Transformation ähnlich einfach ab. Hier wird einfach für jede Prozessklasse ein Prozessflussdiagramm aus den Aktivitätsdiagrammen der Anforderungsanalyse erzeugt, was ebenfalls im vorhergehenden Abschnitt ausführlich

| Transformationsregel | Eigenschaften | |
|---|---|---|
| **ProcessClass** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse des Stereotyps *«processClass»* |
| | **Zielkontext:** | Prozessmodell |
| | **Definition:** | Für jeden Anwendungsfall wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **ProcessClassWithGeneralization** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse mit Attributen und Assoziationen |
| | **Zielkontext:** | Prozessmodell |
| | **Definition:** | Komposition von **ProcessClass** und **AddGeneralizationsToProcessClass** aufgerufen. |
| **ProcessClassWithActivityDiagram** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse mit Attributen und Assoziationen |
| | **Zielkontext:** | Prozessmodell |
| | **Definition:** | Komposition von **ProcessClass** und **AddActivityDiagramToProcessClass** aufgerufen. |
| **AddGeneralizationsToProcessClass** | **Quelle:** | Anwendungsfall des Typs *«process»* und seine Beziehungen |
| | **Ziel:** | Assoziationen einer Klasse |
| | **Zielkontext:** | *«processClass»*-Element |
| | **Definition:** | Für jeden Anwendungsfall, welcher eine Verbindung zum ausgewählten Anwendungsfall besitzt wird eine Generalisierung erzeugt. |
| **AddActivityDiagramToProcessClass** | **Quelle:** | Das Aktivitätsdiagramm des *«process»*-Anwendungsfalls |
| | **Ziel:** | Prozessflussdiagramm |
| | **Zielkontext:** | *«processClass»*-Element |
| | **Definition:** | Das zum Anwendungsfall gehörende Aktivitätsdiagramm wird in ein Prozessflussdiagramm umgewandelt. |

Tabelle 5.3: Transformationsregeln zur Ergänzung der Prozessstruktur

dargestellt wurde.

### 5.3.3 Die transformierten Prozesse von Philoponella

Beim Betrachten des Prozessstrukturdiagramms für Philoponella zusammen mit der automatisch generierten Version wird sofort deutlich, dass diese beiden Diagramme zwar sehr ähnlich sind, jedoch über einen sehr wichtigen Unterschied verfügen: Die Prozesse AddFriend,UpdateFriend,RemoveFriend,AddFavorite, UpdateFavorite und RemoveFavorite sind alle fast identisch und erben deswegen von der abstrakten Prozessklasse UpdateElement, welche ihr Verhalten im allgemeinen Fall modelliert. Solche Analogien sind jedoch nur ersichtlich, wenn man die diversen Aktivitätsdiagramme direkt miteinander vergleicht. Bei einer automatischen Umwandlung kann dies allerdings zu sehr vielen und komplizierten Vergleichen führen, was eine Transformation zu umständlich machen würde und vermutlich bereits an einer unterschiedlichen Benennung der Aktionen scheitern würde. Außerdem gibt es im Originalmodell noch eine Kompositionsbeziehung zwischen den beiden Prozessklassen CreateLinkInfo und CreateCategory. Sie kann ebenfalls nicht automatisch aus den Diagrammen der Anforderungsanalyse gewonnen werden. Allerdings kann diese Beziehung auch gegebenenfalls

weggelassen werden, da sie nur andeuten soll, dass der Prozess CreateCategory nur innerhalb von CreateLinkInfo vorkommt. Obwohl es im Vergleich zu anderen Diagrammen ziemlich einfach ist, ist das aus der vollständigen Transformation entstandene Prozessstrukturdiagramm in Abbildung 5.3 gezeigt und kann dadurch leichter mit dem von Hand erstellten Modell in Abbildung 3.3 auf Seite 41 verglichen werden.

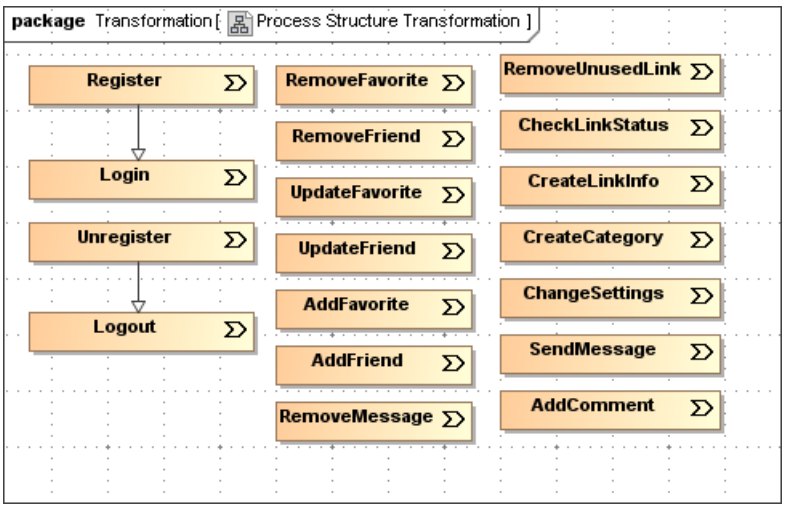


Abbildung 5.3: Philoponellas Processstruktur erzeugt durch vollständige Transformation

Die Prozessflussdiagramme bilden wohl den umfangreichsten Teil der Modellierung, da es in der Regel ein Aktivitätsdiagramm pro Prozess gibt. Innerhalb dieser Diagramme gibt es auf den ersten Blick sehr viele Unterschiede, jedoch sind viele davon keine Fehler in der Transformation, sondern resultieren einfach aus einem unterschiedlichen Stil bei dem Erstellen der verschiedenen Diagramme. Dies verdeutlicht noch einmal sehr schön, dass es nicht zwingend nur ein richtiges Modell für eine Anwendung geben kann.

Den Anfang bilden auch hier die Diagramme zu den Prozessen der An- und Abmeldung der Benutzer. Der Prozess Login ist zunächst so einfach, dass es nicht überrascht, dass beide Modelle gleich sind. Anders sieht es bei Unregister aus. Zunächst sind die beiden Aktionen vertauscht, was allerdings kein Fehler ist, da bereits gesagt wurde, dass die Reihenfolge der beiden Aktionen keine Rolle spielt. Außerdem gibt es in der automatisch generierten Version keinen Objektknoten zur Darstellung des entfernten Benutzers. Dies ist aber nicht zwingend ein Fehler, da man darüber streiten kann, ob ein entferntes Objekt extra dargestellt werden sollte oder nicht.

Der Prozess Register zeigt sehr deutlich zwei Punkte, wie ein Aktivitätsdiagramm unterschiedlich modelliert werden kann. Zum einen sind die Ausgabe- und Eingabepins im Originaldiagramm mit einem Objektknoten verbunden, während sie im aus der Transformation hervorgegangen Modell direkt die Aktionen miteinander verbinden und erst zum Schluss an ein Objekt weitergeleitet werden. Dies sind beides legitime Varianten der Darstellung

und ihre Äquivalenz wird dadurch verdeutlicht, dass die Pins die gleichen Bezeichnungen tragen und damit einen identischen Datenaustausch modellieren sollen. Weiterhin werden im Originalmodell die Änderungen der Ansicht als einzelne Aktionsknoten repräsentiert, aber im anderen Diagramm kommen diese nicht vor. Auch hier kann man nicht genau sagen, welche Methode die Bessere ist, da man darüber streiten kann, ob die Änderung der Ansicht noch Teil des Prozesses ist. Die selben Unterschiede gibt es auch zwischen den beiden Diagrammen zum Prozess Login. Lediglich die Aktion CheckData fehlt nach der Transformation wirklich, da sie im Originalmodell extra eingefügt wurde, um die Überprüfung der Daten nochmal hervorzuheben, was während der Anforderungsanalyse als nicht so wichtig erschien. Da diese beiden Prozessflussdiagramme ein gutes Beispiel für die Transformationsergebnisse darstellen sind diese in Abbildung 5.4 und Abbildung 5.5 präsentiert. Auch zu diesen beiden gibt es die Originalmodelle in den Abbildungen 3.4 und **??** auf den Seiten 42 und **??**.

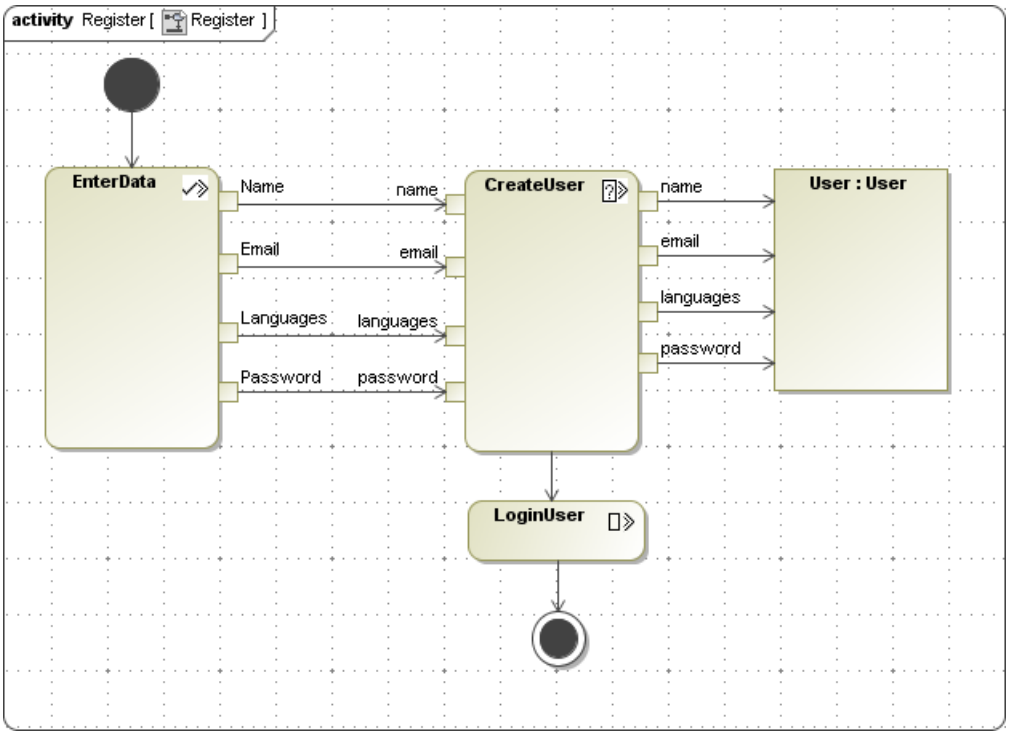


Abbildung 5.4: Philoponellas Prozess Register erzeugt durch Transformation

Auch die Prozesse AddComment, SendMessage, CreateLinkInfo, CreateCategory und ChangeSettings, sowie alle von UpdateElement erbenden Prozesse unterscheiden sich lediglich durch die vorher beschriebenen unterschiedlichen Modellierungsweisen. Nur bei AddComment, welcher als ein weiteres Beispiel in Abbildung 5.6 dargestellt wird, sollte vielleicht noch erwähnt werden, dass im Originaldiagramm den Ausgangspunkt wie auch das Ende der Aktivität hier der selbe Objektknoten des Typs *«Comment»* darstellt. Dies soll verdeutlichen, dass das Objekt am Ende des Vorgangs mit den neuen Daten überschrieben wird, während diese

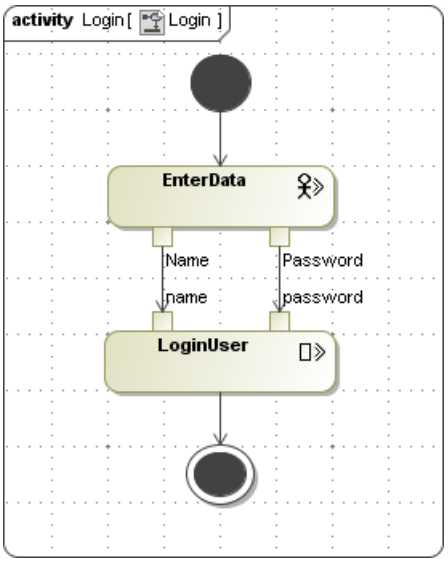


Abbildung 5.5: Philoponellas Prozess Login erzeugt durch Transformation

Information bei der automatischen Generierung verlorengeht. Außerdem befindet sich das manuell zu diesem Prozess erstellte Modell in Abbildung 3.7 auf Seite 46.

### 5.3.4 Zusammenfassung

Abschließend lässt sich zu diesem Teil der Transformation sagen, dass sie recht unkompliziert und fast fehlerfrei abläuft. Der Modellierungsstil aus den Aktivitätsdiagrammen der Anforderungsanalyse wird in die neu erstellten Prozessflussdiagramme übernommen und stellt nur bei direktem Vergleich zwei unterschiedlich erstellter Diagramme ein Problem dar. Die größte Schwierigkeit in diesem Teil liegt aber darin, dass ähnliche Prozesse nicht automatisch erkannt werden können und eventuell von Hand nach modelliert werden müssen.

## 5.4 Skizzieren des Präsentationsmodells

Das letzte Modell, der als Transformation erzeugt werden soll ist das Präsentationsmodell. Dieses Modell ist recht komplex und verschachtelt, was sich auch in den Transformationsregeln wiederspiegelt.

### 5.4.1 Hinzufügen von Präsentationsgruppen

Die Erzeugung neuer Elemente im Präsentationsmodell aus der Anforderungsanalyse beschränkt sich erstaunlicher Weise auf zwei Optionen (Präsentationsgruppe oder Formular),

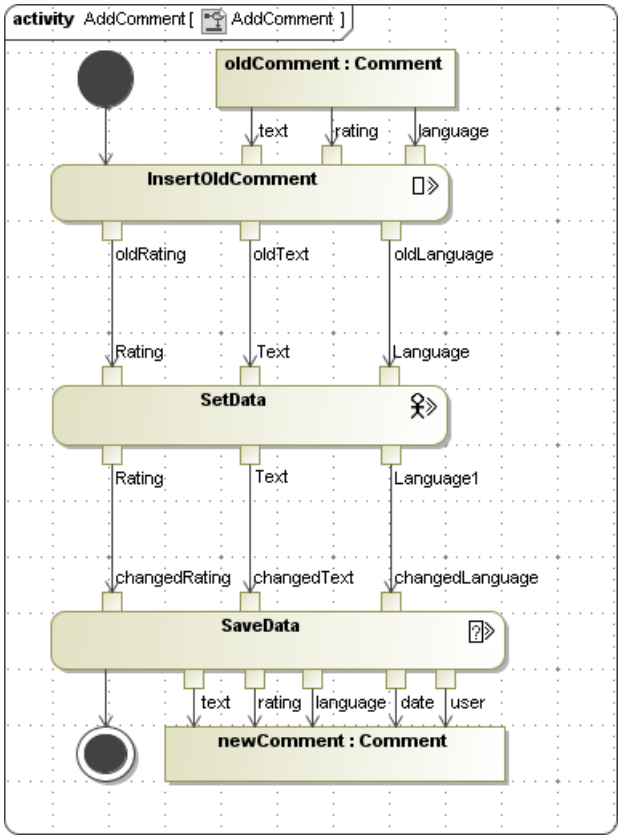


Abbildung 5.6: Philoponellas Prozess AddComment erzeugt durch Transformation

welche auch noch sehr ähnlich sind. Jedoch sind diese Transformationen recht komplex und haben mehr noch als alle bisher beschriebenen das Problem, dass sie die Namen aus der Anforderungsanalyse übernehmen. Dies ist zwar kein Fehler, führt jedoch zu seltsamen Namen für Objekte der Präsentation. Außerdem gibt es für die Erzeugung dieser beiden Elemente verschiedene Optionen, was die Anzahl der Regeln stark vergrößert.

Zunächst gibt es hier die Möglichkeiten Objekte ohne darin enthaltene Präsentationselemente zu erzeugen, wofür es drei Transformationsregeln gibt. Bei den ersten liegt dabei der Unterschied im Stereotypen des Anwendungsfalls. Ist ein solcher Anwendungsfall vom Typ *«navigation»*, dann wird in der Regel **EmptyPresentationGroup** dafür ein Objekt mit dem Stereotyp *«presentationGroup»* in das Präsentationsmodell eingetragen. Falls der Benutzerfall hingegen dem Typ *«process»* besitzt, dann erzeugt die Regel **EmptyInputForm** für ihn ein *«inputForm»*-Objekt. Da die beiden Stereotypen *«adaptation»* und *«observation»* zur Modellierung der Adaptivität dient und diese ohne Benutzerinteraktion innerhalb des Systems abläuft, tragen diese nichts zur Generierung des Präsentationsmodells bei.

Ebenso sollen alle Elemente der Typen *«navigation»* oder *«process»* aus dem Modell herausgehalten werden, falls diese in ihren zugehörigen Aktivitätsdiagrammen keine Aus- oder Eingabepins mit den beiden neu eingeführten Stereotypen *«inputElement»* und *«outputElement»* enthalten. Dies kommt hauptsächlich bei den systeminternen Prozessen vor und soll das Präsentationsmodell klein halten, indem leere Gruppenelemente dort herausgehalten werden.

Die letzte Regel mit der Bezeichnung **EmptyInnerPresentationGroup** generiert dabei die ebenfalls ein Element des Typs *«presentationGroup»*. Allerdings liegt der Unterschied zur ersten Regel darin, dass hier eine Gruppe nicht für den Anwendungsfall erzeugt wird, sondern für eine Aktion des zum Anwendungsfall gehörenden Aktivitätsdiagramms. Dabei sollen jedoch nur bestimmte Aktionen betrachtet werden, die sowohl den Stereotyp *«systemAction»* besitzen und über mindestens einen Pin des Typs *«outputElement»* verfügen.

Nun können diese Gruppen noch auf verschiedene Weisen automatisch mit Inhalt gefüllt werden, wobei die aus Anwendungsfällen gewonnenen Präsentationsgruppen zunächst mit weiteren Gruppen gefüllt werden können. Dies beschreibt die Transformationsregel **FillPresentationGroupFlat** und bewirkt, dass für die gleichen Aktionen wie im vorherigen Fall Präsentationsgruppen innerhalb dieser generiert werden. Auch hier kann dies gleichzeitig mit dem erstellen der oberen Präsentationsgruppe gemacht werden, wofür die Regel **FlatPresentationGroup** steht.

Außerdem kann sowohl für ein *«inputForm»* als auch für eine *«presentationGroup»* der Inhalt auch so weit es möglich ist generiert werden. Dabei wird das zum Anwendungsfall gehörende Aktivitätsdiagramm betrachtet oder wenn die Präsentationsgruppe einer einzelnen Aktion entspricht, so betrachtet man nur diese Aktion des Aktivitätsdiagramms. Enthält so

ein Diagramm eine Aktion, die *«inputElement»*- oder *«outputElement»*-Pins, so wird in die entsprechende Gruppe oder das Formular des Präsentationsmodells ein internes Element mit dem Namen des Pins eingetragen.

Weiterhin wird für jeden Pin des Typs *«outputElement»* ein *«text»*-Objekt in diese Gruppe eingefügt. Zwar stellt *«outputElement»* nicht zwingend einen Text dar, sondern kann auch ein Bild oder ein Multimediaobjekt darstellen. Diese Information steht während der Anforderungsanalyse nicht zwangsläufig zur Verfügung, weshalb die Objekte bei der Übersetzung erstmal auf den einfachsten und wohl häufigsten Typ eingestellt werden.

Analog ist der Ablauf auch bei *«inputElement»*. Diese repräsentiert alle Elemente der Benutzeroberfläche dar, welche eine Eingabe vom Benutzer ermöglichen und wird daher in ein Objekt mit dem Stereotyp *«textField»* übersetzt, da dieser zwar nicht immer die komfortabelste Möglichkeit bietet, aber dafür zur Eingabe aller nur denkbaren Daten herangezogen werden kann.

Sollte das Aktivitätsdiagramm einen oder mehrere Aktionsknoten des Typs *«userAction»* enthalten, welche mit dem Aktivitätsende verbunden sind, so wird für jede solche Benutzeraktion ein *«button»*-Element eingefügt, entweder in die aus der Vorgängeraktion entstandenen Gruppe oder falls eine solche nicht existiert dann in die Gruppe aus dem Anwendungsfall.

Auch die beiden neuen Stereotypen der Aktionen *«confirmedAction»* und *«validatedAction»* tragen zur Erzeugung des Präsentationsmodells bei. *«validatedAction»* stellt eine Validierung der Benutzereingaben durch das System dar und benötigt somit eine Ausgabe der Fehler an den Benutzer, was durch *«text»* möglich ist. Somit wird für jeden Anwendungsfall, dessen Aktivitätsdiagramm mindestens eine solche Aktion enthält das Textelement Validation eingefügt. Bei den Aktionen des Stereotyps *«confirmedAction»* ist es genau umgekehrt: Das System braucht eine Bestätigung durch den Benutzer um Fortfahren zu können. Aber dem Benutzer muss auch die Möglichkeit gegeben werden den Vorgang abzubrechen, da eine Bestätigung ohne eine Alternative ziemlich nutzlos ist. Daher wird *«confirmedAction»* in zwei Elemente des Typs *«button»* übersetzt, von denen einer den Namen Confirm und der andere Cancel bekommt. Falls ein Formular nach diesem Schritt immer noch keinen Element vom Typ *«button»* enthält wird zusätzlich ein Knopf eingefügt, welcher den Namen der Gruppe trägt.

Zu Letzt wird für jede erweiternde Beziehung zwischen zwei Anwendungsfällen, die zu Gruppen im Präsentationsmodell geführt haben ein Knopf in die Gruppe eingefügt, welcher den Startpunkt der Beziehung darstellt. Dieser bekommt den Namen des anderen Gruppenelements und soll eine Möglichkeit darstellen, diese Gruppe anzeigen zu lassen. Bei einer einschließenden Beziehung hingegen kann einfach die Gruppe direkt eingetragen werden.

| Transformationsregel | Eigenschaften | |
|---|---|---|
| **EmptyPresentationGroup** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse des Stereotyps *«presentationGroup»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Für den Anwendungsfall wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **EmptyInputForm** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse des Stereotyps *«inputForm»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Für den Anwendungsfall wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **EmptyInnerPresentationGroup** | **Quelle:** | Aktion eines *«process»*-Anwendungsfalls |
| | **Ziel:** | Klasse des Stereotyps *«presentationGroup»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Für die Aktion des Anwendungsfalls wird eine Klasse mit dem gleichen Namen erzeugt und entsprechendem Stereotyp erzeugt. |
| **FlatPresentationGroup** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse des Stereotyps *«presentationGroup»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Komposition von **EmptyPresentationGroup** und **FillPresentationGroupFlat**. |
| **FullPresentationGroup** | **Quelle:** | Anwendungsfall des Typs *«navigation»* |
| | **Ziel:** | Klasse des Stereotyps *«presentationGroup»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Komposition von **EmptyPresentationGroup** und **FillPresentationGroup**. |
| **FullInputForm** | **Quelle:** | Anwendungsfall des Typs *«process»* |
| | **Ziel:** | Klasse des Stereotyps *«inputForm»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Komposition von **EmptyInputForm** und **FillInputForm**. |
| **FullInnerPresentationGroup** | **Quelle:** | Aktion eines *«process»*-Anwendungsfalls |
| | **Ziel:** | Klasse des Stereotyps *«presentationGroup»* |
| | **Zielkontext:** | Präsentationsmodell |
| | **Definition:** | Komposition von **EmptyInnerPresentationGroup** und **FillInnerPresentationGroup** aufgerufen. |
| **FillPresentationGroupFlat** | **Quelle:** | Das Aktivitätsdiagramm des *«navigation»*-Anwendungsfalls |
| | **Ziel:** | Prozessflussdiagramm |
| | **Zielkontext:** | *«presentationGroup»*-Element |
| | **Definition:** | Für alle relevanten Aktionen des Aktivitätsdiagramms wird ein entsprechendes Präsentationselement in diese eingefügt. |
| **FillPresentationGroup** | **Quelle:** | Anwendungsfall des Typs *«navigation»* und seine Beziehungen |
| | **Ziel:** | Assoziationen einer Klasse |
| | **Zielkontext:** | *«presentationGroup»*-Element |
| | **Definition:** | Für alle relevanten Aktionen des Aktivitätsdiagramms wird ein entsprechendes Präsentationselement in diese eingefügt und für sie wird die Regel **FillInnerPresentationGroup** aufgerufen. |

Tabelle 5.4: Transformationsregeln zur Ergänzung des Präsentationsmodells

| Transformationsregel | Eigenschaften | |
|---|---|---|
| **FillInputForm** | **Quelle:** | Das Aktivitätsdiagramm des *«process»*-Anwendungsfalls |
| | **Ziel:** | Prozessflussdiagramm |
| | **Zielkontext:** | *«inputForm»*-Element |
| | **Definition:** | Für alle relevanten Pins des Aktivitätsdiagramms wird ein entsprechendes Präsentationselement in diese eingefügt. |
| **FillInnerPresentationGroup** | **Quelle:** | Aktion des *«navigation»*-Anwendungsfalls |
| | **Ziel:** | Prozessflussdiagramm |
| | **Zielkontext:** | *«presentationGroup»*-Element |
| | **Definition:** | Für alle relevanten Pins dieser Aktion wird ein entsprechendes Präsentationselement in diese eingefügt. |

Tabelle 5.5: Fortsetzung der Transformationsregeln zur Ergänzung des Präsentationsmodells

### 5.4.2 Vollständige Generierung des Modells

Der vollständige Tranformationsprozess beginnt in diesem Abschnitt zunächst damit, dass ein Objekt des Stereotyps *«presentationPage»* mit dem Namen Home erzeugt wird. Dieser soll den Ausgangspunkt der Anwendung darstellen und wird für gewöhnlich nicht explizit in der Anforderungsanalyse modelliert.

Als nächstes werden die Anwendungsfälle des entsprechenden Diagramms übersetzt, was analog zu den Transformationen **FlatPresentationGroup**, **FullInputForm** und **FullInnerPresentationGroup** abläuft. Hier gibt es nun das Problem mit den zwei unterschiedlichen Möglichkeiten der Erzeugung der Präsentationsgruppen, wobei für diese vollständige Transformation des Präsentationsmodells die Methode mit separat generierten inneren Präsentationsgruppen ausgewählt wurde, da diese meist ein flacheres und kleineres Diagramm zur Folge hat.

Bleiben am Ende Gruppen im Modell, welche weder gleichnamigen Elemente des Typs *«button»* noch des selben Typs in anderen Gruppen haben, dann wird für sie ein Knopf in das Element Home eingefügt. Außerdem wird ein Element mit den Namen HomeAlternatives des Stereotyps *«iteratedPresentationGroup»* in Home erstellt und alle bisherigen Elemente auf oberster Ebene der Hierarchie werden in dieses eingetragen, falls sie nicht Teil einer anderen Gruppe sind.

### 5.4.3 Das transformierte Präsentationsmodell von Philoponella

Wenn man die primären Objekte der Präsentation betrachtet, dann fällt einem auf, dass logischer Weise beide Modelle über ein Objekt des Stereotyps *«presentationPage»* verfügen. Aber bereits bei *«presentationGroup»*-Objekten gibt es selbst wenn man von unterschiedlicher Benennung absieht große Unterschiede. So gibt es im Originalmodell das Element NavigationBar, was jedoch nur den Zweck erfüllt den Knoten Home zu entlasten und die Objekte zur Benutzerinteraktion von den Darstellungsgruppen zu trennen. Darauf

wurde beim Erstellen der Anforderungsanalyse keine Rücksicht genommen, weshalb eine entsprechende Gruppe auch im transformierten Modell fehlt.

Auch die Gruppen Input, SelectCategory, Comment und ListElement haben einen ähnlichen Sinn, da sie Ein- und Ausgabeelemente gruppieren, welche in dieser Form in mehreren anderen Gruppen vorkommen. Dabei haben die letzteren beiden ein Gegenstück im transformierten Modell, wobei diese hier jedoch die Namen ShowComments und ShowList tragen. Für die anderen beiden gibt es keinen Partner, was am Ende des Abschnitts beschrieben wird.

Beim Anwendungsfall ChangeBrowseSettings verhält es sich genau umgekehrt. Dieser soll die Einstellungen der Suchparameter für die Liste der Seiten darstellen und wurde nur in der Anforderungsanalyse aber nicht im Originalmodell hervorgehoben. Daher erscheinen die so dargestellten Präsentationselemente auch nur im durch die Transformation erstellten Modell in der gleichnamigen Präsentationsgruppe und werden dort somit von den diversen mit dieser Liste zusammenhängenden Ausgaben separiert.

Eine weitere Änderung durch die Transformation bei den Elementen des Stereotyps *«presentationGroup»* sind die beiden Objekte ViewUserDetails und ViewPersonalDetails. Diese stellen die beiden Ansichten zum Betrachten der Benutzerinformationen dar. Da zwischen der Darstellung der eigenen Daten und den Daten anderer Benutzer sehr viele Ähnlichkeiten bestehen, sind diese im Originalmodell zur Gruppe UserDetails zusammengefasst. Auch solche Ähnlichkeiten werden bei der Transformation nicht berücksichtigt.

Bei den aus Prozessvorgängen entstandenen Formularen gibt es mehr Übereinstimmungen, wenn man von der unterschiedlichen Benennung des Elements zur Änderung der persönlichen Daten, welche im Originalmodell PersonalDetails heißt und nach der Transformation den Namen ChangeSettings erhält. Andere Objekte des Typs *«inputForm»* im transformierten Modell sind AddImage, Login und Unregister, welche alle ziemlich kleine Objektgruppen darstellen und daher im manuell erstellten Modell direkt in andere Gruppen integriert wurden.

Die größeren Unterschiede ergeben sich jedoch beim Präsentationsmodell in den inneren Elementen bzw. aus Aktionen entstandenen Gruppen, wobei diese nur selten tatsächliche Fehler darstellen sondern sind meist auf Umstrukturierungen zurückzuführen. So fällt bereits bei Home auf, dass die von der Transformation erzeugten Version deutlich mehr Objekte enthält. Dies hängt zum einen natürlich mit der bereits erwähnten Auslagerung der Interaktionselemente in die Gruppe NavigationBar im Originalmodell, was zumindest die für die An- und Abmeldungsvorgänge und die Registrierung benötigten Knöpfe aus der Seite entfernt. Die anderen Unterschiede in Home resultieren daraus, dass durch die abweichende Modellierung auch die Anzahl und auch manche Namen der Gruppenelemente veschieden ist.

Im Bereich des Elements LinkSearch bzw. BrowseLinkInfos gibt es in der automatisch erstellten Version die bereits erwähnte Untergruppe ShowList, welche bis auf den Namen

und das Fehlen der Bilder mit der Gruppe ListElement identisch ist. Jedoch ist diese im Originalmodell in ein Element des Stereotyps *«iteratedPresentationGroup»* eingebettet, was die Darstellung einer Liste von Elementen repräsentieren soll. Diese Information ist jedoch nicht aus der Anforderungsanalyse ersichtlich. Die drei weiteren Gruppen ShowComments, ShowDescription und ShowImage sind abgesehen von der fehlenden Differenzierung der Ausgabeelemente innerhalb der Anforderungsanalyse mit den Gruppen des manuell erstellten Modells identisch. Nur die Werte *«carrousel»* und *«collapse»* müssten manuell nachgereicht werden und beim transformierten Modell wird die Darstellung des Benutzernamens und die Möglichkeit die Detailansicht für diesen Benutzer zu betrachten getrennt modelliert, während dies beim Originalmodell in einem Knopfelement vereint wird.

Bei der Ansicht der Seitendetails ergeben sich im transformierten Modell ebenfalls die drei Gruppen ShowComments, ShowDescription und ShowImage. Das Originalmodell ist da etwas einfacher, da die beiden letzteren Gruppen keinen weiteren Zweck besitzen und daher ihre Elemente direkt eingetragen wurden. Auch wurden hier die Knöpfe gruppiert, was allerdings nur für eine bessere Übersicht von Nutzen ist und daher nicht wirklich im anderen Modell vermisst wird. Genau umgekehrt verhält es sich mit den Seitendetails, welche nach der Transformation in der Gruppe ShowDetails zusammengefasst ist. Zu Letzt gibt es in dieser Gruppe wie auch bei der Benutzeransicht die Gruppe zur Darstellung neu hinzugekommener Seiten. Dies unterscheidet sich in den Modellen ziemlich stark, jedoch liegt hier das Problem darin, dass beim manuellen Erstellen des Präsentationsmodells unter anderem aus Bequemlichkeit von der Vorgabe der Anforderungsanalyse abgewichen und die selbe Darstellung wie bei den Listenelementen gewählt wurde.

Die Benutzerdetails haben neben den bereits erwähnten Gruppe zur Darstellung neuer Seiten, noch beide die Gruppe der Freunde und die Gruppe der Favoriten, welche sich in der Namensgebung der inneren Elemente unterscheiden. Außerdem ist im Originalmodell in jedem Eintrag FriendGroup ein zusätzliches Element SendMessage eingefügt, welcher die direkte Absendung einer Nachricht an diesen Benutzer darstellt und nicht in der Anforderungsanalyse modelliert wurde. Auch wurden im Originalmodell die Möglichkeiten den Status der Freunde und Favoriten zu ändern nicht direkt modelliert, da dies durch Verschieben der Elemente innerhalb der Ansicht erreicht werden sollte. Im anderen Modell wurde dies durch die vier Eingabefeldelemente UpdateFriend, UpdateFavorite, RemoveFriend und RemoveFavorite repräsentiert, was allerdings ebenfalls nicht besonders gelungen ist. Bei der Darstellung der beiden Details Name und Language und den Aktionsauswahlelementen AddFriend, SendMessage und ChangeSettings gibt es schließlich nur das Problem, dass die Stereotypen durch die Transformation nicht immer passend ausgewählt wurden.

Bei den Formularen gibt es zunächst mal wieder das Problem, dass alle Eingabeelemente den Typ *«textInput»* tragen. Zusätzlich gibt es bei der Registrierung und beim Ändern

persönlicher Einstellungen den Unterschied bei der Abfrage des Passworts. Im Originalmodell existieren dafür zwei Eingabefelder, womit sichergestellt werden soll, dass der Benutzer sich beim Passwort nicht vertippt hat. Diese Information steht in den Diagrammen der Anforderungsanalyse nicht zur Verfügung, weshalb es hier nur ein einzelnes Eingabefeld dafür gibt.

Weiterhin gibt es im Originalmodell die beiden Formulare Input und SelectCategory, wobei das erste die typischen Elemente eines Formulars zusammenfassen und beim zweiten eine bessere Möglichkeit zur Auswahl der Kategorie dargestellt wird. Vor allem beim letzten Fall sind diese Darstellungsinformationen zu detailliert und sind nicht in der Anforderungsanalyse enthalten. Sonst enthalten die Formulare in der transformierten Version alle Informationen, die auch im Originalmodell vertreten sind. Richtig falsch ist nur AddImage, da es das Hochladen einer Datei darstellen und daher nicht als ein Formular, sondern als ein Element des Typs *«fileUpload»* modelliert werden soll.

Wegen seiner Größe ist das transformierte Modell zur besseren Übersicht auf die beiden Abbildungen 5.7 und 5.8 aufgeteilt. Auch das Originalmodell zur Präsentation ist sehr umfangreich und deswegen auf die beiden Abbildungen **??** und **??** auf den Seiten **??** und **??** verteilt.

### 5.4.4 Zusammenfassung

Wie hier dargestellt unterscheidet sich das Präsentationsmodell nach der Transformation zwar sehr stark von der manuellen Version, allerdings stellt dies nur selten einen groben Fehler dar. So kann das Fehlen der Stereotypen wie *«image»* oder *«selection»* leicht manuell angepasst werden, sobald feststeht, wo diese sinnvoller als die automatisch erstellten Stereotypen sind. Ein wirklich ernstes Problem stellt bei der Transformation nur das Fehlen der Informationen, ob eine Liste oder nur ein einzelnes Element dargestellt wird.

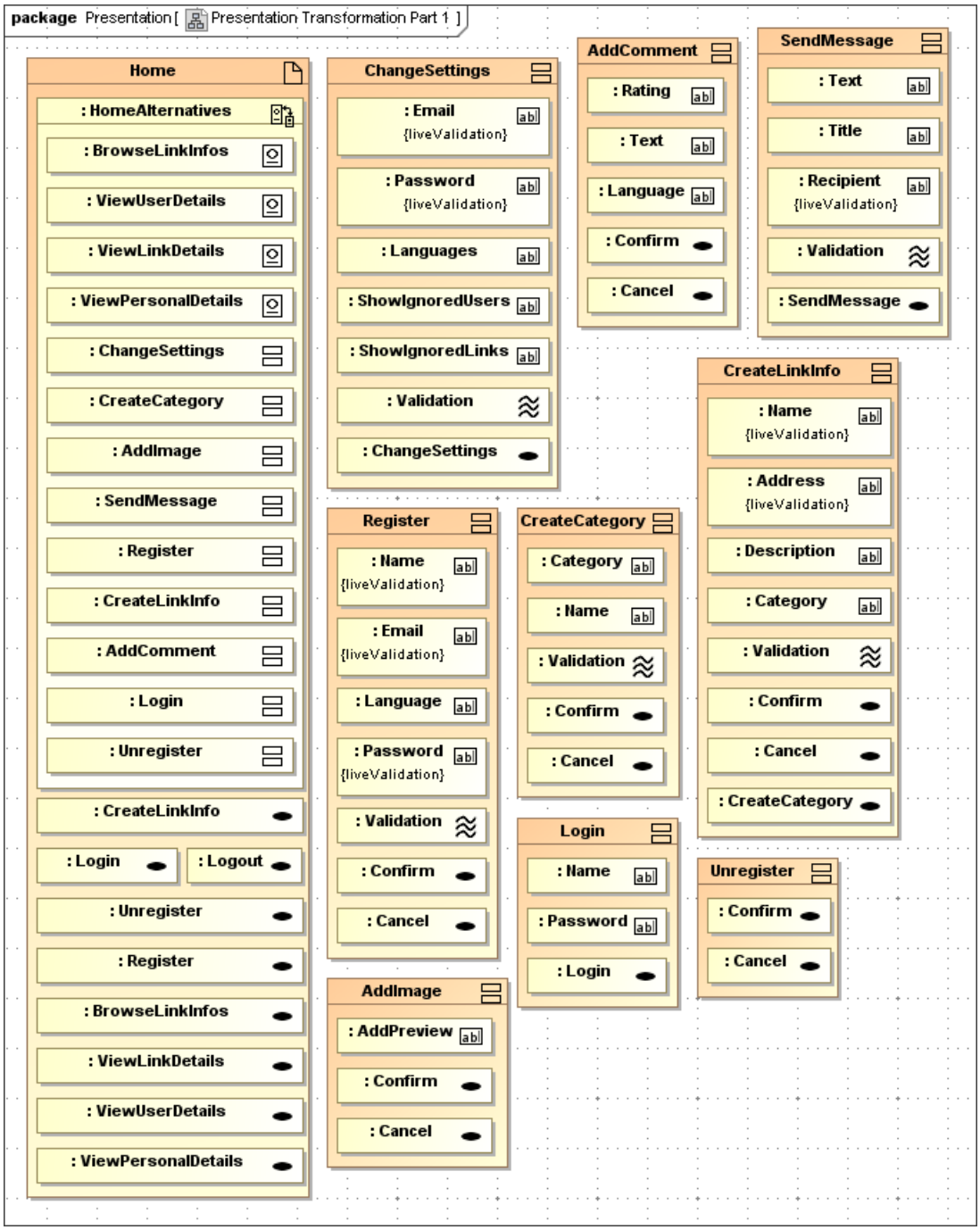


Abbildung 5.7: Philoponellas Präsentationsmodell erzeugt durch Transformation Teil 1

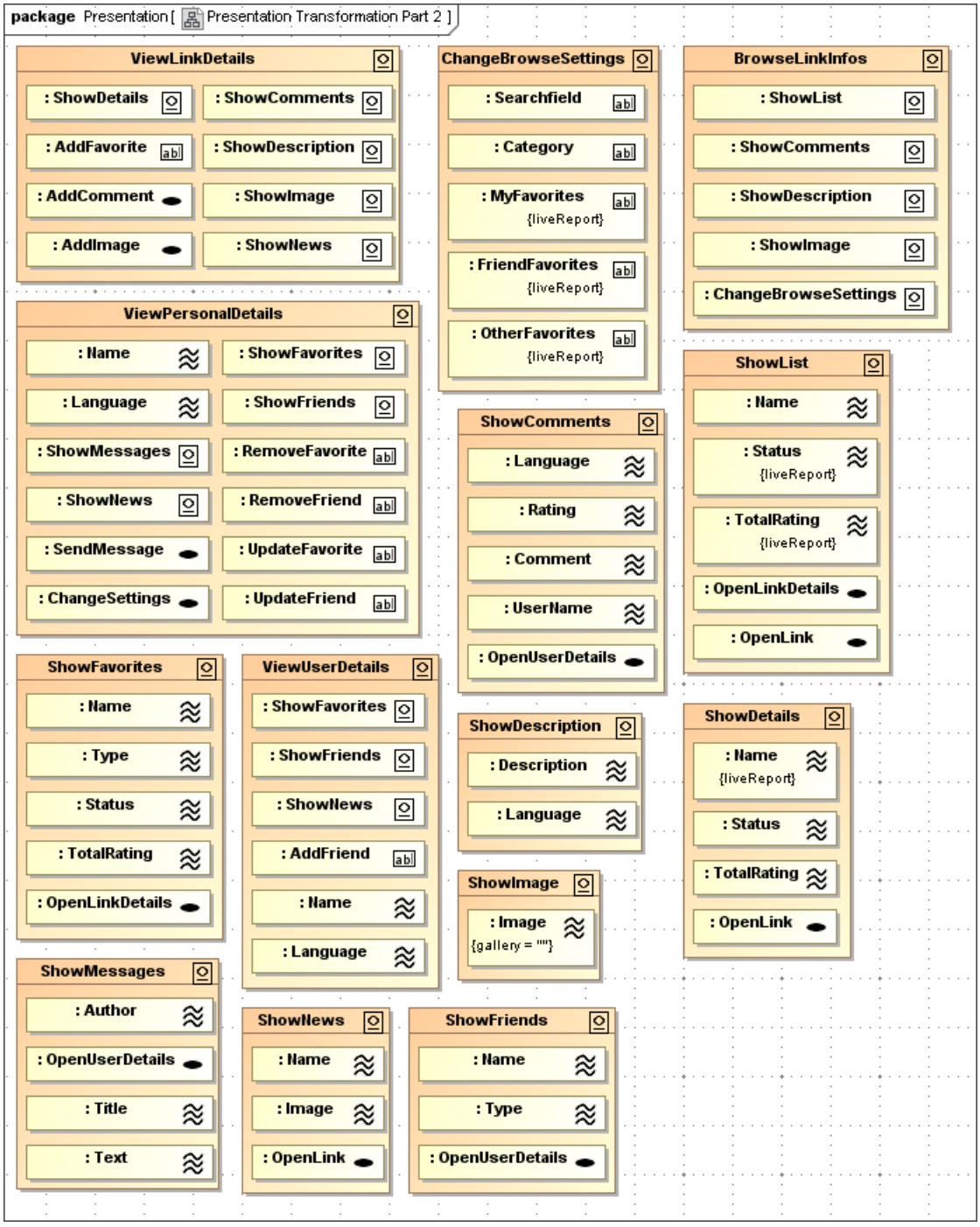


Abbildung 5.8: Philoponellas Präsentationsmodell erzeugt durch Transformation Teil 2

# 6 Implementierung der Beispielanwendung

Als letztes Kapitel dieser Arbeit folgt noch eine Beschreibung der Implementierung von Philoponella. Zur Implementierung dieser Webanwendung wird dabei das Google Web Toolkit (GWT) verwendet, welches im nachfolgenden Abschnitt näher beschrieben wird. Außerdem soll diese Kapitel die Schwierigkeiten und Möglichkeiten bei der Umsetzung der Modelle durch GWT anhand dieses Beispiels zumindest teilweise aufzeigen.

## 6.1 Google Web Toolkit

Als sich die Webanwendungen von simplen Textseiten zu komplexen Applikationen mit einer kaum noch von Desktopanwendungen zu unterscheidenden Benutzeroberfläche entwickelten, stellten sich die bisherigen Möglichkeiten zur Erstellung solcher Websysteme als zu umständlich und kompliziert heraus. Neben dem Versuch durch das Bereitstellen von aufwändigeren Entwicklungsumgebungen, welche viele Elemente einer solchen Webanwendung automatisch anpassten, entstanden auch viele sogenannte Frameworks.

Dabei ist ein Framework eine Art Programmiergerüst innerhalb dessen die Programmierer ihre Software erstellen und wodurch der Aufwand einer solchen Implementierung reduziert werden soll. Es besteht aus einer Reihe von Klassenbibliotheken„ welches durch Vererbung oder durch Delegation an die Klassen des Frameworks und durch einen vordefinierten Einstiegspunkt eine spezielle Anwendung realisiert. Durch die Bereitstellung der Grundfunktionalitäten für einen Typ von Anwendungen erleichtert ein Framework die Programmierung, reduziert die Fehlerquote, verbessert die Übersichtlichkeit und vereinfacht ebenfalls die Wiederverwendbarkeit des Quelltexts.

Das Google Web Toolkit ist ein Beispiel für ein solches Framework, wobei sich das

GWT mit der Erstellung von Webanwendungen beschäftigt. Im Gegensatz zu den meisten anderen Frameworks, welche sich mit dem selben Gebiet beschäftigen und dem Programmierer Bibliotheken in JavaScript bereitstellen, geht das GWT einen eigenen Weg: Es verwendet als Programmiersprache Java. Aus dieser Vorgehensweise ergeben sich mehrere Vorteile wie die größere Anzahl an Entwicklungsumgebungen und ein höherer Bekanntheitsgrad von Java im Vergleich zu JavaScript. Außerdem besitzt die Programmiersprache Java diverse Konstrukte, welche sie leistungsfähiger und robuster während der Implementierung machen, wie das Konzept der Objektorientierung oder die festen Datentypen.

Da aber das Ergebnis einer mit GWT implementierten Anwendung eine klassische auf HTML und JavaScript basierende Webapplikation sein soll, kommt nach einem erfolgreichen Erstellen der Software in Java der eigentliche Kern des GWT ins Spiel: Der Java-to-JavaScript-Compiler. Dieser übersetzt den ihm übergebenen Javaquelltext in einen Satz von HTML-, CSS- und natürlich auch JavaScript-Dateien. Weiterhin rühmt sich GWT damit, dass eine so entstandene Webanwendung einer handgeschriebenen Version weder in Größe noch in Performanz nachstehen soll. Genauso bemüht sich das GWT Quelltext zu erzeugen, welcher auf allen gängigen Browsern eine möglichst einheitliche Ausgabe und Verhalten produzieren soll, wozu GWT sogar nicht davor zurückschreckt die hohe Fehlertoleranz von Browsersoftware auszunutzen und teilweise fehlerhaften Quelltext abliefert, solange dieser das gewünschte Ergebnis bei der Ausgabe erzielt.

Die Klassenbibliotheken des Google Web Toolkits enthalten hauptsächlich die Definitionen von sogenannten Widgets und Panels, wobei Widgets Komponenten der Benutzeroberfläche einer Webanwendung wie Knöpfe, Auswahlfelder aber auch komplexe, aus anderen zusammengesetzte Elemente und Panels Elemente zur Aufnahme und Anordnung solcher Komponenten darstellen. Weiterhin stellt es selbstverständlich Klassen zur Kommunikation bereit, wobei hier das Absenden von XMLHttpRequests und die Remote Procedure Calls zur Verfügung stehen.

Auch wenn das GWT keinen eigenen Teil für die Entwicklung der Serverseite eines Websystems liefert und theoretisch davon unabhängig arbeiten kann, so empfiehlt es sich doch zur Implementierung des Servers sogenannte Servlets zu verwenden. Als Servlets werden dabei Javaklassen bezeichnet, deren Instanzen innerhalb eines Webservers laufen und Anfragen eines Clients bearbeiten und beantworten, was den Vorteil mit sich bringt, dass sowohl der Client als auch der Server in Java entwickelt werden können.

## 6.2 Die Datenbasis und das Inhaltsmodell

Der erste Teil einer Webanwendung, welcher hier betrachtet werden soll bilden die Daten des Systems. Um diese im System festzuhalten gibt es zwei Möglichkeiten. Zunächst gibt es die weit verbreitete Methode einer Datenbank, welcher meist in Form von Relationen die Daten einer Anwendungen festhält. Diese Methode hat den Vorteil, dass auch sehr große Datenmengen vergleichsweise gut gehandhabt werden können und vor allem die Auswertung von aufwändigen Anfragen sehr effizient ablaufen kann. Da die Datenbanken für genau diese Aufgaben entwickelt und optimiert wurden, ist dies nicht sonderlich überraschend. Jedoch spielen diese Vorteile bei kleineren Datenmengen kaum ein Rolle, weshalb auch bei Philoponella auf die Einbindung einer Datenbank verzichtet wurde.

Da Philoponella nur einen Prototypen darstellt, werden die Daten darin durch Objekte normaler Javaklassen festgehalten und zur dauerhaften Sicherung in Dateien gespeichert. Dies erleichterte die Entwicklung und machte außerdem die gesamte Anwendung einheitlich nur auf Java basierend. Sollte die Datenmenge irgendwann zu groß werden, so lässt sich außerdem die Version relativ leicht in eine datenbankbasierende Anwendung überführen, indem man die relevanten Methoden so modifiziert, dass sie anstelle der Betrachtung der Objekte eine Anfrage an den Datenbankserver abschicken und die Antwort entsprechend auswerten oder weiterleiten.

Darüber hinaus lässt sich der Aufbau dieses Teils der Anwendung, welcher im Javapaket org.philoponella.database zusammengefasst ist, durch diese Herangehensweise sehr gut aus dem Inhaltsmodell gewinnen. Zunächst entstehen aus jeder Klasse des Inhaltsmodells eine Klasse im Quelltext Philoponellas. Jedoch gibt es in Philoponella auch noch einige zusätzliche Klassen, wovon die meisten sich aus den attributierten Assoziationen des Inhaltsmodells ergeben. Diese fassen die Attribute dieser Verbindung und die dadurch erreichbare Klasse zusammen. Ihre Namen können entweder aus dem Namen der Assoziation des Inhaltsmodells gewonnen werden, oder wie es im Falle der Klasse UserKeyword geschehen ist durch die Zusammensetzung der beiden Klassennamen gewonnen werden, falls keine Bezeichner für die Assoziation festgelegt wurden.

Auch die Vererbungen lassen sich direkt übernehmen, wie es bei Description und Commentar geschehen ist. Allerdings gibt es noch die Klasse ListElement, welche kein Vorbild im Inhaltsmodell besitzt. Diese Klasse entstand aus der Beobachtung, dass sowohl Friend als auch Favorite große Gemeinsamkeiten besitzen. Solche gemeinsame Attribute und Methoden werden in ListElement zusammengefasst, von der die anderen beiden dann erben. Obwohl dieser Umstand bereits zur Modellierungszeit bekannt war, entstanden die beiden Klassen aus einer Assoziation, welche mit Attributen versehen wurde. Da es keine Verbindungen zwischen Assoziationen gibt, gab es auch keine Möglichkeit dies im Modell festzuhalten.

Schließlich gibt es noch die Aufzählungsstruktur ListElementType und die Klasse DataBase, welche ebenfalls keine Vorbilder im Inhaltsmodell besitzen. Bei ListElementType ergab sich dies daraus, dass während der Modellierung keine Notwendigkeit sichtbar war, die dazugehörigen Attribute vom Typ her dieser Aufzählung zuzuordnen, da ein Integerwert hier den selben Zweck erfüllt. Die Klasse DataBase hingegen hat den Zweck die gesamte, so entstandene Struktur der Javaklassen zusammenzufassen und eventuell auch diese für die anderen Schichten der Anwendung zu kapseln.

Auch bei den Attributen gibt es viele Gemeinsamkeiten und nur wenige Abweichungen. So gibt es in der Klasse User alle Attribute des Inhaltsmodells, wobei noch drei weitere Werte hinzukommen. Zunächst gibt es die Liste der Nachrichten acMessages. Dieser Unterschied ergibt sich daraus, dass anstelle von zwei Verbindungen von der Nachrichtenklasse zum Benutzer, die Verbindung recipient so implementiert wurde, dass die zu einem Benutzer gehörenden Nachrichten direkt innerhalb seines Objekts abgelegt werden. Die beiden anderen Attribute iLastView und sAddress, wobei das erste zur Speicherung der Ansicht und das Andere zur eindeutigen Bestimmung der angemeldeten Benutzer dient, sind nicht im Modell festgehalten, da ihre Notwendigkeit erst bei der Implementierung erkannt wurde, was bei der Entwicklung einer Anwendung durchaus vorkommen kann. Durch die Einführung von iLastView wurden außerdem die beiden Attribute iViewFriends und iViewFavorites überflüssig und sind nicht mehr in User enthalten.

Ähnlich verhält es sich bei der Klasse LinkInfo, welche die zusätzlichen Attribute bActive und iChanged enthält. Diese ergaben sich aus einer während der Implementierung beschlossenen Veränderung der Darstellung des Aktivitätszustands und des Zeitpunkts des letzten Zugriffs, weshalb auch die beiden im Inhaltsmodell dafür verwendeten Attribute lastChecked und inactiveDays entfernt wurden.

Abgesehen davon, dass die Assoziation recipient aus bereits dargelegten Gründen nicht in die Klasse Message übernommen wurde, belaufen sich die Abweichungen der kleineren Klassen in diesem Abschnitt allein auf abweichende Benennungen. Dies gilt auch für die Javaklassen, welche aus mit Attributen versehenen Beziehungen des Inhaltsmodells entstanden sind.

## 6.3 Implementierung der Serverseite

Der serverseitige Teil von Philoponella wird in Form von Servlets implementiert, wobei jede Art von Anfrage durch eine eigene Servletklasse behandelt wird. Zwar wurde zunächst auch erwogen, die Seite des Servers durch ein einzelnes Servlet bearbeiten zu lassen, jedoch hat dies einige gravierende Nachteile. So ist die Übersicht und somit die Fehleranfälligkeit und Wartbarkeit bei vielen kleinen Servlets deutlich besser und auch die Größe der versendeten

Daten sinkt, da ein Teil des Overheads wegfällt. Allgemein lassen sich zwei Arten von Servlets unterscheiden. Als erstes gibt es die Servlets, die allein den Zweck haben die Anfragen zu beantworten und nur Änderungen an der Datenbasis vornehmen, falls dies für die Adaption der Anwendung notwendig ist und die zweite Gruppe bilden dann die von Prozessen abgeleiteten Servlets, welche dann genau das Verhalten implementieren, also auch Änderungen an der Datenbasis vornehmen. Beide Hälften der Serverseite sind im Paket org.philoponella.server enthalten, werden hier aber getrennt behandelt.

### 6.3.1 Aus Prozessen generierte Servlets

Vergleicht man die Servletklassen von Philoponella, welche Änderungen an der Datenbasis vornehmen, fällt einem ziemlich schnell auf, dass sich viele Prozessnamen innerhalb dieser Klassen wiederfinden. Allerdings gibt es auch hier einige Unterschiede.

So fehlen die Servlets für die beiden Prozesse RemoveUnusedLink und CheckLinkStatus, da diese Prozesse Abläufe allein auf der Seite des Servers darstellen und daher nicht direkt vom aus Client ausgelöst werden können. Vielmehr laufen diese als ein Teil anderer Prozesse ab und können daher die Funktionalität eines Servlets verzichten.

Weiterhin gibt es im Prozessmodell von Philoponella diverse Klassen zur Behandlung der Freunde- und Favoritenliste, wie zum Beispiel AddFriend und RemoveFavorite. Für diese fehlen die entsprechenden Servlets innerhalb der Implementierung. Stattdessen finden sich im Serverpaket die beiden Klassen SetFriend und SetFavorite, welche die Funktionalität der verschiedenen Prozesse zu einer Liste vereinen. Dies ist möglich, da sich die Abläufe innerhalb der Prozesse sich nur geringfügig unterscheiden, was bereits bei der Modellierung festgestellt wurde. Erwähnenswert wäre hier auch noch, dass diese Prozesse im Modell alle von dem abstrakten Prozess UpdateElement erben, dieser jedoch in der Implementierung fehlt, da durch die vorher beschriebene, andere Zusammenfassung der Vorgänge sein Nutzen wegfällt.

Ähnlich verhält es sich mit dem Prozess RemoveMessage, welcher ebenfalls durch das Servlet SetMessage ersetzt wurde, da sich durch die Drag&Drop-Funktionalität des entsprechenden Panels im Präsentationsmodell die Möglichkeit für den Benutzer ergibt die Nachrichten umzusortieren. Diese beiden Vorgänge lassen sich interessanterweise zusammenfassen, indem man beim Löschen einer Nachricht als Zielposition einfach eine nicht erlaubten Wert angibt. Der Prozess SendMessage hingegen benötigt ganz andere Datenangaben, wodurch es auch weiterhin durch ein separates Servlet implementiert wird.

Bei den Prozessen zum An- und Abmelden der Benutzer gibt es für jeden eine äquivalente Servletklasse, allerdings ist die Generalisierung nicht durch einer Vererbung implementiert, was bei diesen Servlet wegen ihre geringen Komplexität nicht notwendig erschien. Statt dessen benutzt das Servlet Register zusätzlich zu den Methoden zum Einlogen aus der Datenbasis,

welche auch in Login Anwendung finden, noch eigens für die zusätzliche Funktionalität dieses Prozesses bereitgestellte Prozeduren. Ebenso verhält es sich mit Logout und Unregister.

Die Komposition zwischen den beiden Klassen der Prozessstruktur CreateLinkInfo und CreateCategory ist ebenfalls nicht auf der Serverseite zu finden. Da diese Verbindung symbolisieren soll, dass die Kategorienerstellung als Teil von der Erstellung von Seiteninformationen ablaufen kann, wird diese durch eine entsprechende Option auf der Seite des Clients implementiert.

Schließlich gibt es noch die beiden Servletklassen ChangeSettings und AddComment, welche jedoch einfach aus den entsprechenden Prozessen gewonnen wurden und hier nur wegen der Vollständigkeit erwähnt werden.

### 6.3.2 Servlets für Anfragen

Die zweite Hälfte der Anfragen haben den primären Zweck Daten an den Client zu liefern und bei Bedarf adaptives Verhalten zu implementieren. Zu dieser Kategorie gehören in Philoponella alle Servlets, welche mit dem Präfix Get beginnen. Natürlich haben sie wegen ihrer Art keine Gegenstücke in Philoponellas Prozessmodell, jedoch fällt eine andere Ähnlichkeit zum Modell auf. So gibt es viele Servlets, welche die gleichen Daten liefern wie eine *«presentationGroup»* und sich oft sogar im Namen ähneln.

Bei der Präsentationsgruppe UserDetails gibt zunächst das entsprechende Servlet mit dem Namen GetUserDetails, welche die allgemeinen Details über einen Benutzer an den Client übermittelt. Jedoch gibt im Zusammenhang mit Benutzerdetails noch weitere Daten, welche separat angefordert werden.

Diese entsprechen ebenfalls Elementen des Stereotyps *«presentationGroup»* und repräsentieren Daten, welche enger mit einander zusammenhängen und bei Philoponella interessanter Weise immer Listen darstellen. So gibt es das Servlet GetFavorites zum Anfordern der Favoritenliste, GetFriends für die Liste der Freunde, GetNews, welcher der Gruppe NewLinks im Präsentationsmodell entspricht und schließlich das Servlet GetMessages für die Auflistung der Nachrichten. Die Notwendigkeit sie einzeln anzufordern ergibt sich dadurch, dass dem Client die Möglichkeit gegeben werden soll, bei einer möglichen Veränderung einer Liste diese anzufordern, ohne sämtliche Daten eines Benutzers anfordern zu müssen, wodurch der Datentransfer deutlich reduziert werden kann. Weiterhin erfüllen diese Servlets noch zwei weitere wichtige Funktionen, von denen zuerst die Aussortierung von Listenelementen steht, welche nicht vom aktuellen Benutzer gesehen werden dürfen. Dies geschieht bei der Liste der Favoriten und der Freunde. Außerdem werden die Datenlisten vor dem Abschicken auch noch nach Relevanz sortiert bzw. aussortiert und markiert um dem Client eine Anpassung an den Benutzer zu ermöglichen.

Innerhalb der Gruppe UserDetails gibt es auch noch die beiden Elemente PersonalDetails und OtherDetails, welche jedoch keine entsprechende Servletklasse besitzen, da sie keine Präsentationselemente zur Ausgabe von Daten enthalten oder diese bereit innerhalb einer anderen Gruppe liegen.

Genauso verhält es sich mit der Gruppe LinkDetails, deren Daten durch das Servlet GetLinkDetails ausgeliefert werden und der Gruppe LinkSearch mit dem entsprechenden Servlet GetLinkSearch. Bei beiden Gruppen gibt es das bereits bekannte Element NewLinks, was durch das bereits eingeführte Servlet GetNews abgehandelt wird. Hier zeigt sich noch ein Nutzen der Aufteilung der gesendeten Datenmengen, da die selbe Anfrage an drei Stellen eingesetzt werden kann um einen Teil der benötigten Daten zu erhalten. Darüber hinaus gibt es in LinkDetails noch die Gruppe LinkButtons, die jedoch keine relevanten Präsentationselemente enthält und daher bei den Servlets nicht vorkommt.

LinkSearch enthält die drei Untergruppen Comment, ImagePreview und DescriptionPreview. Wie schon im Kapitel über die Modellierung von Philoponella beschrieben, dienen diese dazu unterschiedliche Details zu den Seiteninformationen darzustellen.

Die erste kommt auch bei der Darstellung der Detailansicht einer Linkinformation zur Anwendung und wird durch das Servlet GetComments ergänzt. In der Implementierung von Philoponella gibt es allerdings auch das Servlet GetComment von dem man zwar annehmen könnte, dass diese Klasse hier die passende ist. Jedoch ist das Element Comment mit der Eigenschaft *carrousel* ausgestattet, was andeutet, dass es hier um eine Liste von Elementen handelt, wodurch die Benennung des dazugehörigen Servlets passender erscheint. Das Servlet GetComment hingegen dient dazu einzelne Kommentare anzufordern, was bei der Erstellung von Kommentaren nützlich ist, da somit das entsprechende Formular mit den alten Daten gefüllt werden kann. Da dies keiner Gruppe des Präsentationsmodells entspricht und auch keiner entsprechen kann, stellt dieses Servlet eine Ausnahme vom bisherigen Vorgehen dar.

Ein ähnliches Problem gibt es bei der Darstellung der Bilder, wofür das Servlet GetImages zuständig ist. Hier gibt es allerdings auch noch das Servlet GetImage, was ein einzelnes Bild liefert, wohingegen GetImages nur eine Liste mit Bildernamen ausgibt. Die Aufteilung dieser Aufgabe in zwei Servlets ergibt sich dadurch, dass Bilder eine deutlich größere Datenmenge darstellen als dies bei den anderen Daten der Fall war. Daher wäre es eine zu hohe Belastung alle Bilder einer Liste auf einmal zu erhalten, vor allem, da diese nur einzeln angezeigt werden sollen.

Außerdem ist noch zu erwähnen, dass obwohl Bilder auch bei der Detailansicht der Seiteninformationen angezeigt werden sollen, jedoch nicht innerhalb einer Gruppe, sondern nur durch ein einzelnes Darstellungselement modelliert wurden. Ebenso verhält es sich bei der Präsentation der Beschreibungen. In der Suchlistenansicht sind diese Details repräsentierende Elmente nur deswegen in eine Gruppe eingefügt, da sie dadurch mit der RIA-Eigenschaft

*collapse* ausgestattet werden können.

Schließlich gibt es in diesem Abschnitt noch zwei weitere Servlets zu betrachten: GetSelectCategory und GetKeywords. Das erste entspricht ebenfalls einer Präsentationsgruppe und dient der Auswahl einer Kategorie. Es ist auch die letzte Präsentationsgruppe, welche einem Servlet entspricht.

Interessanter ist jedoch das letzte Servlet. Dieses hat keine entsprechende Gruppe im Präsentationsmodell, noch wird die Darstellung der durch sie gelieferten Daten irgendwo modelliert. Seine Notwendigkeit ist versteckt in der RIA-Eigenschaft *autoSuggestion* des Suchfeldelements. Da dadurch einem Benutzer eine Liste mit passenden Suchbegriffen präsentiert werden soll, muss der Client eine solche vom Server anfordern können.

## 6.4 Implementierung des Clients

Schließlich wird nun die Implementierung des Clients von Philoponella zu betrachten. Hier kommen schließlich die speziellen Bibliotheken des Google Web Toolkits zum Einsatz, welche die Übersetzung des Javaquelltextes in die Dateien der Webanwendung ermöglichen.

### 6.4.1 Umsetzung der Gruppenobjekte des Präsentationsmodells

Da die Klassen des Clients primär der Darstellung der Benutzeroberfläche dienen, basieren sie Hauptsächlich auf dem Präsentationsmodell.

Zunächst gibt es in GWT diverse Klassen des Typs Panel, welche eine rechteckige Fläche innerhalb der Ausgabe darstellen und zur Gruppierung und Anordnung anderer Elemente dienen. Daher erscheint es logisch zunächst diese mit den Gruppen im Präsentatiosmodell zu vergleichen.

Aber zuvor gibt es noch die Einstiegsklasse Home, die sich aus direkt aus dem Element Home des Präsentationsmodells ergibt. Dabei entspricht dem Stereotyp *«presentationPage»* beim GWT die Klasse MainEntryPoint von der auch die Klasse Home erbt. Wie auch im Präsentationsmodell enthält diese Klasse alle anderen Elemente der Benutzeroberfläche.

Beim Stereotyp *«presentationGroup»* gibt es zunächst viele Gemeinsamkeiten, aber auch einige Unterschiede. So gibt es in der Implementierung die Klasse NavigationClass, welche der gleichnamigen Gruppe im Modell entspricht. Jedoch enthält das Modell die beiden Gruppen Registered und Guest. Diese sind nicht Teil der Implementierung, da sie im Modell den Zweck erfüllen die beiden Gruppen von dargestellten Eingabeelementen zu Trennen. In einer Javaklasse besteht dazu allerdings keine Notwendigkeit, da die Elemente der

Benutzeroberfläche genausogut einzeln hinzugefügt oder entfernt werden können.

Bei der nächsten Gruppe mit dem Namen LinkSearch und auch bei allen darin enthaltenen Gruppen gibt es wieder keine Abweichungen zwischen dem Präsentationsmodell und der Implementierung. Das Ergebnis ist außerdem in Abbildung 6.1, welche die Suchansicht des Prototyps darstellt zu sehen.

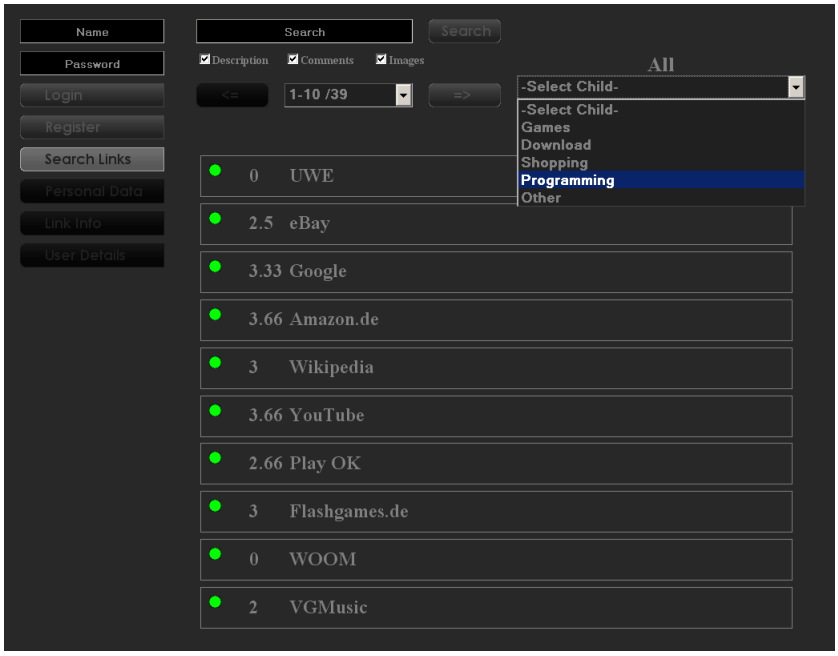


Abbildung 6.1: Suchansicht des Philoponellaprototyps

Fast gleich verhält es sich mit den Gruppen LinkDetails und UserDetails. Die einzigen Abweichungen bilden hier LinkButtons und OtherDetails, wobei diese beide die Knöpfe gruppieren, die nur einem angemeldeten bzw. nicht angemeldeten Benutzer zur Verfügung stehen sollen. Wie auch schon vorher kann dies bei der Javaklasse problemlos weggelassen werden.

Das letzte fehlende Element bildet das Eingabeformular PersonalDetails. Das Problem bei der Implementierung ergab sich hier dadurch, dass es bereits eine Gruppe mit diesem Namen gab und diese in die Implementierung auch schon übernommen wurde. Daher wurde die dem Formular entsprechende Klasse in ChangeDetails umbenannt, wobei sich hier sonst keine Unterschiede ergaben.

Auf der anderen Seite gibt es auch noch einige Klassen, welche weder einer Präsentationsgruppe oder einem Eingabeformular entsprechen. Zunächst wurde im Präsentationsmodell zur Darstellung neuer Linkinformationen, der Favoriten und auch der Elemente der Suchlisten die Präsentationsgruppe ListElement verwendet. Innerhalb der Implementierung wurde jedoch die unterschiedlichen Klasse NewElement, FavoriteElement und ListElement verwendet, da zwar

alle drei über die selben Darstellungselemente verfügen, aber die Funktionalität dahinter sich stets unterscheidet. Zwar wäre es denkbar gewesen alle Möglichkeiten in eine einzige Klasse zu packen, dies wurde aber zugunsten von kleinen, übersichtlichen Klassen verworfen. Bei größeren Klassen, welche viele Gemeinsamkeiten besitzen wäre es in einem solchen Fall außerdem sinnvoll eine Grundklasse zu konstruieren und die konkreten Klassen davon erben zu lassen.

Die beiden Javaklassen Keywords und DropController ergeben sich aus den im Modell verwendeten RIA-Eigenschaften. So stellt Keywords ein Panel dar, welches entsprechend der Eigenschaft *autoSuggestion* ein Panel implementiert aus dem der Benutzer die vorgeschlagenen Eingaben auswählen kann. Die Eigenschaft *drag&drop* hingegen führt dazu, dass eine Klasse notwendig wird, welche die Drop-Ereignisse richtig weiterverarbeitet, weshalb die Klasse DropController notwendig wird.

Schließlich kommt es immer wieder mal vor, dass einer Gruppe Teil mehrerer anderer Gruppen wird. Wenn beide Gruppen in einer Javaklasse umgesetzt werden und das erste Panel mit seinem Elternpanel kommunizieren können soll, ist es sinnvoll ein Interface zu definieren, welches die Elternpanels implementieren. So sind zum Beispiel die beiden Interfaces InputInteface für die Klasse Input und SelectCategoryInterface für das Panel SelectCategory entstanden.

Abschließend kommt in diesem Abschnitt noch einmal ein Screenshot des Prototyps in Abbildung 6.2. Dieses zeigt die Sicht auf eine Seiteninformation.

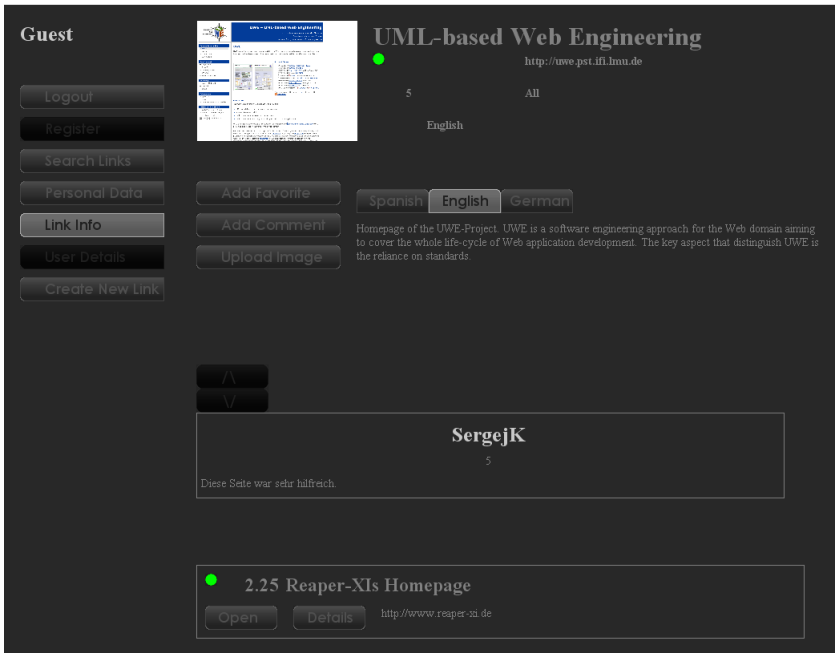


Abbildung 6.2: Informationsansicht des Philoponellaprototyps

### 6.4.2 Umsetzung der Benutzerinteraktionselemente

Für die Implementierung der Elemente des Präsentationsmodells, welche der Darstellung von Daten oder ihre Eingabe ermöglichen, gibt es in GWT sogenannte Widgets. Dabei gibt es ein oder mehrere Widgets, welche einem Stereotyp der Präsentation entsprechen. In Tabelle 6.1 sind die Stereotypen des Präsentationsmodells zusammen mit aus ihnen entstehenden Widgets in Philoponella zusammengefasst.

| **Stereotyp des Präsentationsmodell** | **Entsprechende Widgets in Philoponella** |
|---|---|
| *«text»* | Label, HTML |
| *«image»* | Image |
| *«button»* | Button, PushButton |
| *«mediaObject»* | — |
| *«textInput»* | TextBox, PasswordTextBox, TextArea, RichTextArea |
| *«selection»* | CheckBox, RadioButton, ToggleButton, ListBox |
| *«fileUpload»* | FileUpload |
| *«anchor»* | — |

Tabelle 6.1: In Philoponella benutze Widgets des GWT

Als erster Stereotyp wird hier *«text»* vorgestellt. In Philoponella wurden die Elemente dieses Typs hauptsächlich in zwei Widgets umgesetzt, je nachdem welche Art von Text dargestellt werden sollte. Ist es nur ein Name oder ein paar Begriffe, die angezeigt werden sollen, so wird die Klasse Label verwendet. Wird hingegen ein ganzer Text variabler Länge gebraucht, erfüllt das Widget HTML den Zweck, da diese jede Art von Text, welche wie der Name schon andeutet auch durch HTML formatiert werden kann.

Ein weiterer Stereotyp zur Darstellung ist *«image»*. Dieser hat in GWT ein gleichnamiges Äquivalent, welcher auch in Philoponella dafür benutzt wird. Da damit alle Bilder dargestellt werden können, war eine weitere Klasse nicht notwendig.

Da keine Objekte des Typs *«mediaObject»* in Phioponella zum Einsatz kommen und die dafür benötigte Klasse sehr stark vom Objekt abhängt, werden hier keine entsprechende Widgets aufgelistet.

Auf der Seite der Eingabeelemente gibt es vier Stereotypen, von denen *«button»* der einfachste ist. Auch zu diesem gibt es eine gleichnamige Klasse in GWT, welche einen einfachen Knopf darstellt. Will man diesen graphisch etwas verschönern oder individuell anpassen, so gibt es neben der Änderung des Stils auch noch die Möglichkeit das Widget PushButton zu benutzen. Neben allen Möglichkeiten von Button ermöglicht es einem auch für die Knopfoberfläche eigene Bilder zu verwenden.

Für die Eingabe von Text werden in Philoponella drei Widgets verwendet. Als einfachsten Fall gibt es TextBox. Dieser ermöglicht die Eingabe von einzeiligem Text und eignet sich daher nur für kurze Eingaben. Ein Sonderfall davon stellt die Klasse PasswordTextBox, dieser verschleiert die eingegebenen Daten, weshalb es für die Eingabe von Passwörtern benutzt wird.

Wird schließlich die Eingabe eines langen Texts benötigt, dann sind diese beiden Widgets nicht geeignet und in Philoponella kommt das Widget TextArea zum Einsatz. Außerdem wäre an dieser Stelle eine weitere Klasse erwähnenswert, obwohl diese in Philoponella nicht zum Einsatz kommt. Es ist das Widget RichTextArea, welcher gegenüber TextArea den Vorteil besitzt, dass es auch Textformatierungen zulässt. Zwar wäre das in Philoponella auch denkbar, jedoch wäre es nur eine Spielerei ohne wirklichen Gewinn für die Anwendung.

Weiterhin gibt es den Stereotyp *«selection»*, welcher eine Auswahl repräsentiert. Hier gibt es ebenfalls verschiedene Möglichkeiten dies umzusetzen. Eine der klassischen Methoden dafür wäre es die Widgets CheckBox und RadioButton zu verwenden. Ebenfalls geeignet ist die Klasse ToggleButton, welche die selbe Funktionalität ermöglicht, jedoch die gleichen Vorteile von PushButton besitzt. Bei größeren Mengen an Auswahlmöglichkeiten bietet sich jedoch das Widget ListBox an, da es in der kollabierten Form deutlich weniger Platz verbraucht als die anderen Alternativen.

Auch beim Stereotyp *«fileUpload»* gibt es eine sehr direkte Umsetzung, indem man das Widget FileUpload dafür einsetzt. Dieses stellt einen kompletten Dialog zur Auswahl eines oder mehreren Dateien dar und ist daher alles, was die Implementierung dieses Stereotyps benötigt.

Als letzter Stereotyp gibt es im Präsentationsmodell noch das Element *«anchor»*. Allerdings kann dafür je nach Bedarf fast jedes Widget verwendet werden, wenn es die dafür notwendigen Ereignisse behandelt.

Diese einfachen Umsetzungsregeln funktionieren für Philoponella so gut, dass es kaum Abweichungen davon gibt. Auch die meisten Eigenschaften einer RIA lassen sich mit diesen Klassen und passenden Javafunktionen und richtiger Ereignisbehandlung umsetzen. Die einzige Ausnahme bilden in Philoponella die Listen für Freunde und Favoriten. Diese werden zunächst innerhalb eines Tree-Widgets gehalten, was eine leichte Unterteilung in Kategorien ermöglicht. Außerdem wird für die Drag&Drop-Funktionalität die bereits vorgestellte Klasse DropController benötigt. Dieses Widget kann in der Abbildung 6.3 betrachtet werden, da diese die Favoritenliste innerhalb der persönlichen Sicht abbildet.

## 6.5 Fazit

Abschließend lässt sich sagen, dass die Implementierung einer Webanwendung mit dem Google Web Toolkit leicht zu erlernen und ohne größere Probleme abläuft. Dies gilt vor allem bei Vorkenntnissen in Java. Auch die Unterstützung durch diverse Entwicklungstools macht die Sache deutlich leichter. Aus den UWE-Modellen lässt sich auch fast automatisch wie in diesem Kapitel beschrieben eine grobe Struktur der Anwendung generieren.

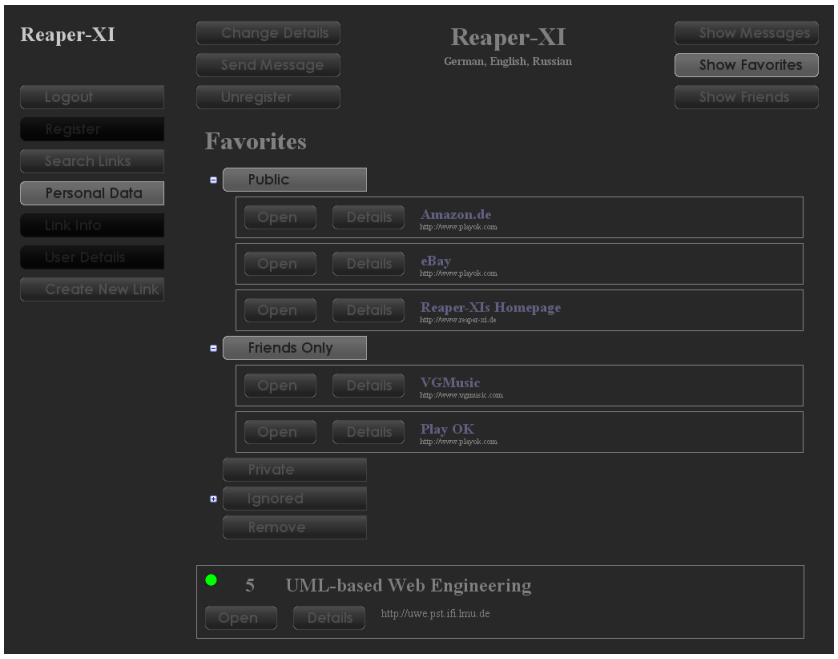


Abbildung 6.3: Persönliche Sicht des Philoponellaprototyps

Jedoch gibt es auch zwei Nachteile, welche den Einsatz des GWT erschweren. Zum einen ist es sehr kompliziert die Anwendung bei Fehlern zu debuggen, da die Fehler innerhalb der JavaScript-Dateien geschehen, diese aber nicht so leicht zu den entsprechenden Stellen im Javaquelltext zurückverfolgt werden können. Das andere Problem von GWT ist die Geschwindigkeit der Übersetzung. Diese war bei der Implementierung von Philoponella zumindest im Vergleich zur Kompilierung von normalen Javaprogrammen sehr viel langsamer, da der Quelltext zunächst normal kompiliert werden muss und danach nochmal in JavaScript- und HTML-Dateien übersetzt werden muss. Diese haben zwar keinen Einfluss auf die Entwicklung, welche auch weiterhin vergleichsweise einfach abläuft, jedoch fallen diese ständigen kleinen Verzögerung einem schnell unangenehm auf.

# 7 Abschluss

Nachdem in den vorhergehenden Kapiteln die verschiedenen Teile der Arbeit detailiert beschrieben wurden, ist nun an der Zeit anschließend die Ergebnisse zusammenzufassen und die Möglichkeiten, die sich für anschließende Arbeiten ergeben zu betrachten.

## 7.1 Ergebnisse

Im Rahmen dieser Diplomarbeit wurde zunächst eine Webanwendung namens Philoponella entwickelt, welche ein soziales Netzwerk zum Speichern und zum Austausch von Favoriten und Informationen zu diesen ermöglicht. Dabei verfügt Philoponella über adaptive Elemente und über RIA-Eigenschaften, die vor allem für die Entwicklung der UWE-Erweiterung und von MagicUWE von nutzen waren.

Auf diesen Modellen aufbauend wurde das Profil von UWE im Bereich der Anforderungsanalyse erweitert. Im Bereich der groben Anforderungsanalyse lässt sich durch die neuen Stereotypen für Anwendungsfälle jetzt zum einen die Struktur der Webanwendung etwas genauer festhalten und zum anderen gibt es nun hier die Möglichkeit die adaptiven Vorgänge zu definieren. Weiter im Detail wurden bei der Anforderungsanalyse die Aktivitätsdiagramme um einige Stereotypen erweitert, welche zum einen die Diagramme vereinfachen sollen und darüber hinaus auch die Darstellung von Präsentationselementen und den damit verbundenen RIA-Eigenschaften ermöglichen.

Schließlich wurde das Plugin des Entwicklungstools MagicDraw mit dem Titel MagicUWE zur Verwendung dieser Stereotypen angepasst und um eine Reihe von Modelltransformationen erweitert. Mit ihnen ist es nun möglich alle derzeit vorhandenen Modelle einer Webanwendung, also den Inhalt, die Navigation, die Prozesse und die Präsentation automatisch aus der dazugehörigen Anforderungsanalyse zu generieren. Für bereits bestehende Modelle wurden Optionen bereitgestellt diese um Elemente zu erweitern, die aus den Anwendungsfällen oder den dazugehörenden Aktivitätsdiagrammen durch Transformationen gewonnen werden, womit

auch eine schnelle Möglichkeit bereitsteht Änderungen der Anforderungen in die Modelle zu übertragen.

## 7.2 Ausblick

Auch wenn diese Arbeit einen ersten Schritt auf dem Weg einer detailierten Modellierung der Anforderungsanalyse der Webanwendungen darstellt, so gibt es mit Sicherheit noch Raum zur Verbesserung. Die hier eingeführten Stereotypen der Anwendungsfälle, Aktionen und Pins könnten in der Praxis getestet werden und bei Bedarf modifiziert oder durch besser passende Ersetz werden. Außerdem gibt es Elemente einer Webanwendung, welche trotz aller Versuche nur unzureichend innerhalb der Anforderungsanalyse erfasst werden können. Daher könnte auch ein weiterer Versuch diese zu integrieren nicht schaden.

Im Bereich der Modelltransformationen gibt es bestimmt auch noch Verbesserungsmöglichkeiten. Auch diese könnten einem Praxistest unterzogen werden und auftretende Fehler korrigiert werden. Außerdem wurde innerhalb dieser Arbeit die Anordnung der Elemente eines durch Transformationen generierten Diagramms vernachlässigt. Genauso sind die Namen der transformierten Elemente manchmal nicht sonderlich gelungen. Diese beiden Aspekte sind vielleicht nur von geringer Bedeutung, würden jedoch trotzdem zu einer schöneren und einfacheren Modellierung beitragen.

Man könnte diesen Gedanken noch weiter spinnen und MagicUWE bzw. MagicDraw um eine Funktion erweitern, welche die Elemente innerhalb eines Diagramms modifiziert. Damit könnte man ohne viel Zutun seine Diagramme verbessern, indem man die Überschneidungen so gut es möglich ist beseitigt, die Diagrammelemente platzsparend anordnet und vielleicht sogar die Lesbarkeit des Diagramms optimiert.

Zuletzt sollte natürlich nicht unbeachtet bleiben, dass die Modellierung der Adaptivität in der Phase der Anforderungsanalyse möglich ist, diese jedoch noch in kein Modellierungstool integriert ist. Daher fehlen auch die Modelltransformationen, welche ein Modell der Adaptivität automatisch aus der Anforderungsanalyse generieren. Eine weitere Arbeit könnte genau diese beiden Mängel beseitigen, was die Modellierung mit UWE noch weiter aufwerten würde.

# Literaturverzeichnis

[1] Andrews, M.: Story of a Servlet: An Instant Tutorial, Oracle, Sun Developer Network (SDN), URL: java.sun.com/products/servlet/articles/tutorial/

[2] Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling Adaptivity with Ascpects, Lowe, D., Gaedke, M., Proc. 5th Int. Conf. Web Engineering (ICWE'05), LNCS 3579, S. 406 – 4166, Springer, Berlin, 2005

[3] Bendel, O., Hauske, S.: E-Learning: Das Wörterbuch, Sauerländer, Aarau, 2004

[4] Busch, M., Koch, N.: Rich Internet Applications. State-of-the-Art, Technical Report, Ludwig-Maximilians-Universität München, 2009

[5] Busch, M., Koch, N.: MagicUWE - A CASE Tool Plugin for Modeling Web Applications, Proc. 9. International Conference Web Engineering (ICWE'09), LNCS, Vol 5648, S. 505 - 508, Springer, Berlin, 2009

[6] Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., White, B.: Web Engineegring, Journal of Web Engineering, Vol. 1, Rinton, 2002

[7] Dumke, R., Lother, M., Wille, C., Zbrog, F.: Web Engineering, Person Studium, München, 2003

[8] Escalona, M., J., Koch, N.: Metamodelling the Requirements of Web Systems, Filipe, J., Cordeiro, J., Pedrosa, V., Web Information Systems and Technologies: Int. Conferences WEBIST 2005 and WEBIST 2006. Revised Selected Papers, LNBIP, Vol. 1, S. 267 – 280, 2007

[9] Google Inc.: Google Web Toolkit 1.5 - Developer's Guide, Google Code, URL: code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=google-web-toolkit-doc-1-5, 2008

[10] Hunter, J., Crawford, W.: Java Servlet Programming, O'Reilly, Köln, 2001

[11] Kappel, G., Pröll, B., Reich, S., Retschitzegger, W.: Web Engineering. Systematische Entwicklung von Webanwendungen, Dpunkt, Heidelberg, 2003

[12] Klein, H.: Rich Internet Applications - Ein Überblick, Create or Die, URL: createordie.de/cod/artikel/Rich-Internet-Applications-%26ndash%3B-Ein-Ueberblick-1791.html, 2008

[13] Knapp, A., Zhang, G.: Model Transformations for Integrating and Validating Web Application Models, Mayr, H., Breu, R., Proc. Modellierung 2006, Lecture Notes Informatics, Vol. P-82, S. 115 - 128, Gesellschaft für Informatik, 2006

[14] Koch, N.: Classification of Model Transformation Techniques used in UML-based Web Engineering, IET Software Journal, Vol. 1, No. 3, Institution of Engineering and Technology, 2007

[15] Koch, N.: Software Engineering for Adaptive Hypermedia Systems, Ludwig-Maximilians-Universität München, 2000

[16] Koch, N.: Transformation Techniques in the Model-Driven Development Process of UWE, Proc. 2. Model-Driven Web Engineering Workshop, ACM Vol. 155, Palo Alto, USA, 2006

[17] Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based Web Engineering: An Approach based on Standards, Rossi, G., Pastor, O., Schwabe, D., Olsina L., Kapitel 7, S. 157 – 191, Springer, Berlin, 2008

[18] Koch, N., Kraus, A.: The Expressive Power of UML-based Web Engineering, Second International Workshop on Web-oriented Software Technology, 2002

[19] Koch, N., Kraus, A., Cachero, C., Meliá, S.: Integration of Business Processes in Web Application Models, Journal of Web Engineering, Vol. 3, Rinton, 2004

[20] Koch, N., Pigerl, M., Zhang, G., Morozova, T.: Patterns for the Model-based Developments of RIAs, Proc. 9. Int. Conf. Web Engineering (ICWE'09), LNCS, Vol. 5648, S. 283 – 291, Springer, Berlin, 2009

[21] Koch, N., Rossi, G., Vallecillo, A.: Model-Driven Web Engineering, Proc. 1. International Workshop on Model-Driven Web Engineering, University of Wollongong, Sydney, 2005

[22] Koch, N., Zhang, G., Escalona, M.: Model Transformations from Requirement of Web System Design, Proc. 6. International Conference on Web Engineering, ACM, S. 281 - 288, Palo Alto, USA, 2006

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

[24] Kroiß, C.: Modellbasierte Generierung von Web-Anwendungen mit UWE, Ludwig-Maximilians-Universität München, 2008

[25] Kroiß, C., Koch, N.: UWE Metamodel and Profile: User Guide and Reference, Technical Report 0802, Ludwig-Maximilians-Universität München, 2008

[26] Macromedia, Inc.: Entwicklung von Rich-Internet-Anwendungen mit Macromedia MX, Adobe Developer Connection, URL: www.adobe.com/de/devnet/mx_ whitepaper.pdf, 2002

[27] Marinschek, M., Radinger, W., Breuer, M., Sowa, H.: Google Web Toolkit, Pfeilschnelle Ajax-Anwendungen in Java, Dpunkt, Heidelberg, 2008

[28] Moore, D., Budd, R., Benson, E.: Professional Rich Internet Applications: AJAX and Beyond, John Wiley & Sons, Hoboken, 2007

[29] Moreno, N., Melia, S., Koch, N., Vallecillo, A.: Addressing New Concerns in Model-Driven Web Engineering Approaches, Proc. 9. Int. Conference on Web Information Systems Engineering, S. 426 - 442, Auckland, New Zealand, Springer, 2008

[30] Morozova, T.: Modellierung und Generierung von Web 2.0 Anwendungen, Ludwig-Maximilians-Universität Münch

[31] No Magic, Inc.: MagicDraw – Open API version 16.9 user guide, No Magic, Inc. 2010, URL: www.magicdraw.com/files/manuals/MagicDraw OpenAPI UserGuide.pdf

[32] No Magic, Inc.: MagicDraw - User's Manual version 16.9, No Magic, Inc. 2010, URL: www.magicdraw.com/files/manuals/MagicDraw UserManual.pdf

[33] Object Management Group: About the Object Management Group, Object Management Group, URL: www.omg.org/gettingstarted/gettingstartedindex.htm

[34] Object Management Group: Unified Modeling Language (UML), URL: www.uml.org

[35] Scherer, S.: Entwicklung von Rich Internet Applications: Evaluierung von UWE anhand einer Fallstudie, Ludwig-Maximilians-Universität München, 2010

[36] Schulte, A.: Rich Internet Applications: Best Practices vom Core bis zum Desctop, Entwickler.Press, München, 2009

[37] Schwinger, W., Koch, N. : Modeling Web Applications, Kappel, G., Pröll, B., Reich, S., Retschitzegger W. Web Engineering: Systematic Development of Web Applications, Kapitel 3, S. 39 – 64, John Wiley, 2006en, 2008

[38] Seemann, M.: Das Google Web Toolkit: GWT, O'Reilly, Köln, 2008

[39] Steyer, R.: Ajax Frameworks, Addison-Wesley, München, 2008

[40] Steyer, R.: Das Google Web Toolkit, Entwickler.Press, München, 2007

[41] TechTarget: Rich Internet Applications, SearchSOA.com, URL: searchsoa.techtarget.com/sDefinition/0,,sid26_gci1273937,00.html, 2007

[42] Zeppenfeld, K., Wolfers, R.: Generative Software-Entwicklung mit der Model Driven Architecture, Spektrum Akademischer Verlag, 2005

[43] Zhang, G.: Towards Aspect-Oriented Class Diagrams, Proc. 12. Asia-Pacific Software Engineering Conf., S. 763 - 768, IEEE, 2005

# Abbildungsverzeichnis

# Tabellenverzeichnis